

To Octave library you must upload file **concat.m**

%Concatenate two numbers return string type

```
%>>concat(2,3)
```

```
%>> ans = 23
```

```
function out = concat(a,b)
```

```
a = num2str(a);
```

```
b = num2str(b);
```

```
out = strcat(a,b);
```

```
end
```

Midterm exam will be held in April 3-rd at 19:00.

During the MidTerm Exam you must solve 2 problems in

<https://imimsociety.net/en/14-cryptography>

namely: DH-KAP, MIM Attack.

Register to the site in the similar way as you are registering in eShop.

After that you will receive 10 Eur virtual money to purchase the problems.

Please purchase only one problem at time and after solving it purchase the next one.

Course Works (CW) list is presented in my Google drive

<https://docs.google.com/document/d/1IFjPGziqO7EiMPFtwiBI28gfBrv8hnBo/edit?usp=sharing&ouid=111502255533491874828&rtpof=true&sd=true>

Please choose topic and label it by the first letter of surname dot name, e.g. S.Name.

For some of topics the group project realization can take place.

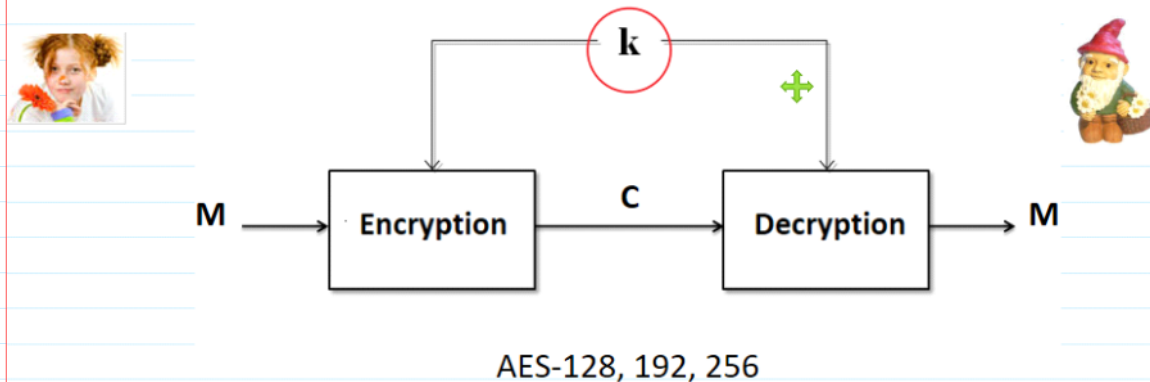
Requirements for CW you can find in

<http://crypto.fmf.ktu.lt/xdownload/>

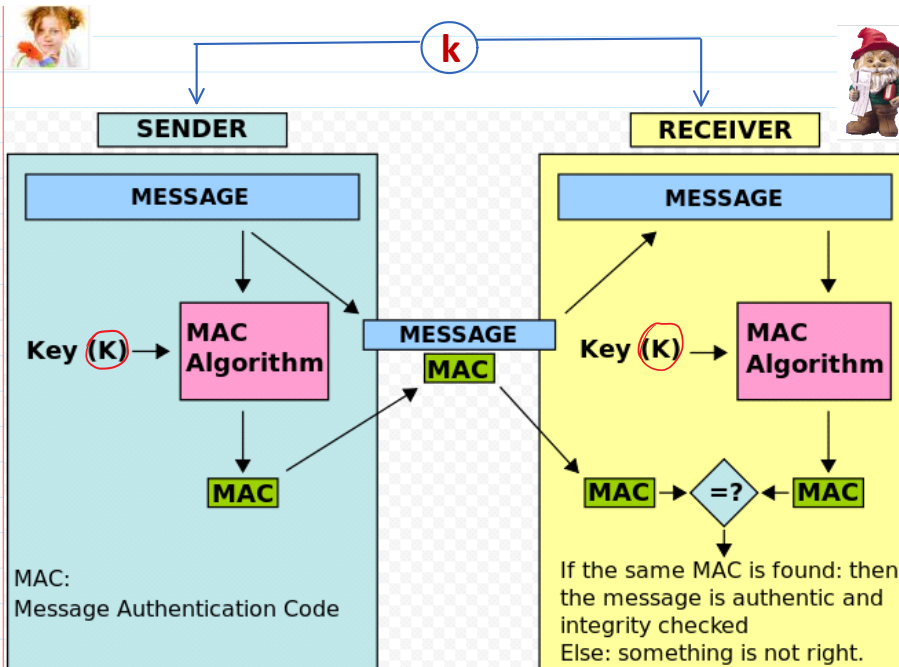
in files Course_Work

Symmetric - Secret Key CryptoSystems

Symmetric encryption



HMAC based e-signature



Asymmetric - Public Key CryptoSystems - PKCS

1. 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 **Kerchhoff 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

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

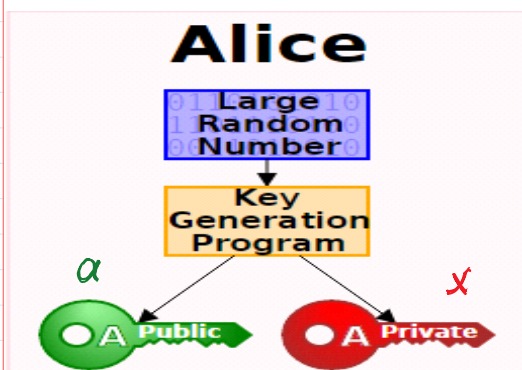
In this case the secrecy of PrK is lost with all negative consequences above. To avoid these undesirable consequences function F must be **one-way function** – OWF. In this case informally OWF is defined in the following way:

1. The computation of its direct value PuK when PrK and F in 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).

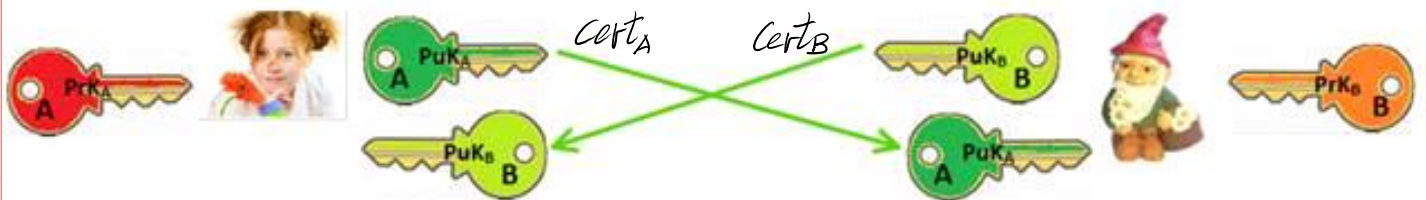
We will consider the same two traditional (canonical) actors in our study, namely Alice and Bob. Everybody is having the corresponding key pair $(\text{PrK}_A, \text{PuK}_A)$ and $(\text{PrK}_B, \text{PuK}_B)$ and are exchanging with their public keys using open communication channel as indicated in figure below.



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

Key generation

- Randomly choose a private key x with $1 < x < p - 1$.
- Compute $a = g^x \bmod p$.
- The public key is $\text{PuK} = a = g^x \bmod p$.
- The private key is $\text{PrK} = x \leftarrow \text{randi}(p-1)$



Asymmetric Signing - Verification

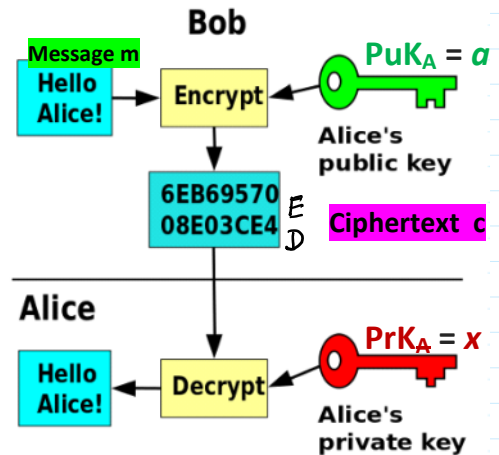
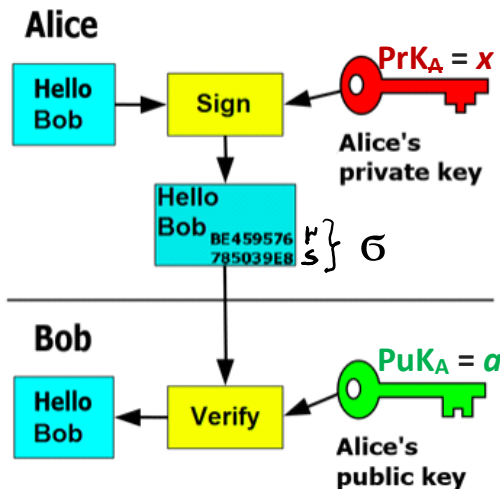
$$\sigma = \text{Sign}(\text{PrK}_A, m)$$

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

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

$g^q \neq 1 \pmod p$ and $g^2 \neq 1 \pmod p$.

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

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

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

`>> int64(2^28-1)`

`ans = 268435455`

`>> dec2bin(ans)`

`ans = 1111 1111 1111 1111 1111 1111 1111`

$$p \sim 2^{2048} \approx 10^{620}$$

$$p \sim 2^{28} = 268\,435\,455$$

NSA CIA

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

EC Gamal sign. \rightarrow Digital Signature Alg. (DSA)
 \rightarrow Elliptic Curve DSA - ECDSA

NSA

Certicom

Public Parameters $PP = (p, g)$;

Mathematical background:

1. Prime number p defines the set $Z_p^* = \{1, 2, 3, \dots, p-1\}$ that is closed under the multiplication $\ast_{\text{mod } p}$ and since every element z in Z_p^* has its inverse z^{-1} such that $z \ast z^{-1} = 1 \text{ mod } p$ then the division operation $/_{\text{mod } p}$ is defined.

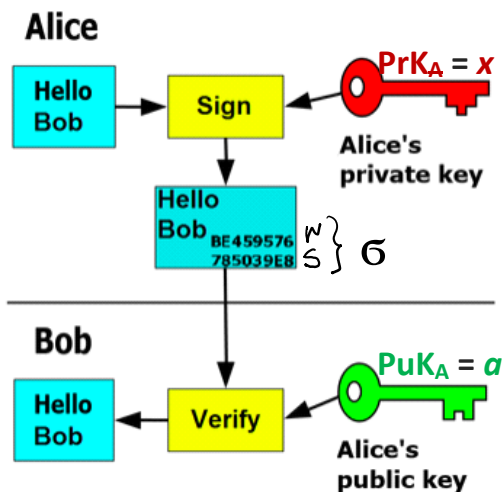
2. **Lagrange theorem.** If p is prime, then for any integer z : $z^{p-1} = 1 \text{ mod } p$.

Corollary: computing expressions mod p for variables with arithmetical expressions in exponents, they are performed mod $(p-1)$.

3. Let $n = p-1$ and i is a positive integer, $i < n$.

If $\text{gcd}(i, n) = 1$, then there exists unique inverse element i^{-1} such that $i \ast i^{-1} = 1 \text{ mod } n$.

E.g., If $\text{gcd}(i, p-1) = 1$, then exists unique inverse element i^{-1} such that $i \ast i^{-1} = 1 \text{ mod } (p-1)$.



Signature creation for message M :

1. Compute decimal h-value $h = H(M)$; $h < p$.
 2. Generate $i = \text{int64}(\text{randi}(p-1)) \% p$ such that $\text{gcd}(i, p-1) = 1$.
 3. Compute $i^{-1} \text{ mod } (p-1)$. You can use the function `>> i_m1 = mulinv(i, p-1);`
 4. Compute $r = g^i \text{ mod } p$.
 5. Compute $s = (h - xr) i^{-1} \text{ mod } (p-1)$.
 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
```

```
>> pm1=p-1
```

```
pm1 = 268435018
```

```
>> 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
```

3. Signature creation

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

- Compute $h = H(M)$.
- Choose a random i such that $1 < i < p-1$ and $\text{gcd}(i, p-1) = 1$.
- Compute $i^{-1} \text{ mod } (p-1)$: $i^{-1} \text{ mod } (p-1)$ 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

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

- Compute $r = g^i \text{ mod } p$

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

Signature $\sigma = (r, s)$

4. 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 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

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

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

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

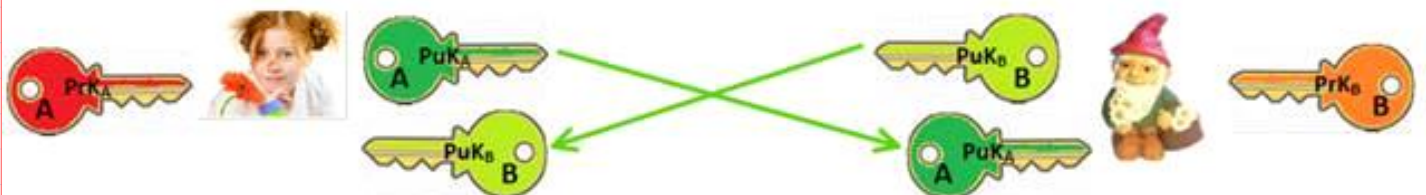
Till this place

Asymmetric Encryption-Decryption: El-Gamal Encryption-Decryption

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

Let message m needs to be encrypted, e.g. $m = 111222$.

$$\Rightarrow m < p \Rightarrow m \bmod p = m.$$



$A: i \xrightarrow{PK_A = a} B: \text{is able to encrypt } m \text{ to } A: m < p$
 $B: t \leftarrow \text{randi}(\mathbb{I}_p^*)$
 $E = m \cdot a^t \bmod p$
 $D = g^t \bmod p$

$C = (E, D) \rightarrow A: \text{is able to decrypt } C = (E, D) \text{ using her } PK_A = x.$

$(-x) \bmod (p-1) = (0 - x) \bmod (p-1) = (p-1-x) \bmod (p-1)$

1. $D^{-x \bmod (p-1)} \bmod p$
 2. $E \cdot D^{-x \bmod p} = m$

$D^{-x} \bmod p$ computation using Fermat theorem:
 If p is prime, then for any integer a holds $a^{p-1} = 1 \bmod p$.

$$D^{p-1} = 1 \bmod p \quad / \cdot D^{-x}$$

$$D^{p-1} \cdot D^{-x} = 1 \cdot D^{-x} \bmod p \Rightarrow D^{p-1-x} = D^{-x} \bmod p$$

$$D^{-x} \bmod p = D^{p-1-x} \bmod p$$

Correctness

$$\text{Enc}_{PK_A}(m, t) = C = (E, D) = (E = m \cdot a^t \bmod p; D = g^t \bmod p)$$

$$\begin{aligned}
 \text{Dec}_{PK_A}(C) &= E \cdot D^{-x} \bmod p = m \cdot a^t \cdot (g^t)^{-x} \bmod p = \\
 &= m \cdot \underbrace{(g^x)^t}_{a^t} \cdot g^{-tx} = m \cdot g^{xt} \cdot g^{-tx} = m \cdot g^{xt-tx} \bmod p = m \cdot g^0 \bmod p = \\
 &= m \cdot 1 \bmod p = m \bmod p = m
 \end{aligned}$$

Since $m < p$

If $m > p \rightarrow m \bmod p \neq m$; $27 \bmod 5 = 2 \neq 27$.

If $m < p \rightarrow m \bmod p = m$; $19 \bmod 31 = 19$.

Decryption is correct if $m < p$.

ASCII

$$\frac{2048}{8}$$

= 256 char.

ElGamal encryption is probabilistic: encryption of the same message m two times yields the different ciphertexts

ElGamal encryption is probabilistic. encryption of the same message m two times yields the different ciphertexts c_1 and c_2 .

1-st encryption:

$$\begin{aligned} t_1 &\leftarrow \text{randi}(\mathcal{Z}_p^*) \\ E_1 &= m \cdot a^{t_1} \bmod p \\ D_1 &= g^{t_1} \bmod p \end{aligned} \left. \vphantom{\begin{aligned} t_1 &\leftarrow \text{randi}(\mathcal{Z}_p^*) \\ E_1 &= m \cdot a^{t_1} \bmod p \\ D_1 &= g^{t_1} \bmod p \end{aligned}} \right\} C_1 = (E_1, D_1)$$

$r_1 \neq r_2$

$C_1 \neq C_2$

2-nd encryption

$$\begin{aligned} t_2 &\leftarrow \text{randi}(\mathcal{Z}_p^*) \\ E_2 &= m \cdot a^{t_2} \bmod p \\ D_2 &= g^{t_2} \bmod p \end{aligned} \left. \vphantom{\begin{aligned} t_2 &\leftarrow \text{randi}(\mathcal{Z}_p^*) \\ E_2 &= m \cdot a^{t_2} \bmod p \\ D_2 &= g^{t_2} \bmod p \end{aligned}} \right\} C_2 = (E_2, D_2)$$

Necessity of probabilistic encryption.

Encrypting a message with textbook RSA always yields the same ciphertext, and so we actually obtain that any deterministic scheme must be insecure for multiple encryptions.

Tavern episode

Key agreement protocol using ElGamal encryption

How to encrypt large data file: Hybrid enc-dec method.

- Parties must agree on common symmetric secret k for symmetric block cipher, e.g. AES-128, 192, 256 bits.

A: 1) $k \leftarrow \text{randi}(2^{256})$

2) $\text{Enc}(\text{PrK}_B, k) = c = (E, D)$

2) M - large file to be encrypted

$$E_k(M) = \text{AES}_k(M) = G$$

3) Signs ciphertext G

$$3.1) h = H(G)$$

$$3.2) \text{Sign}(\text{PrK}_A, h) = \sigma = (r, s)$$

B:

1.1. Verify if PrK_A is valid

1.2. Verify if σ is valid

$$h' = H(G)$$

$$\text{Ver}(\text{PrK}_A, \sigma, h') = \text{True}$$

$$2. \text{Dec}(\text{PrK}_B, c) = k$$

$$3. D_k(G) = \text{AES}_k(G) = M.$$

A was using so called encrypt-and-sign (E-&-S) paradigm.

(E-&-S) paradigm is recommended to prevent so called Chosen Ciphertext Attacks - CCA: it is most strong attack but most complex in realization.

Till this place

Homomorphic property of ElGamal encryption

Let we have 2 messages m_1, m_2 to be encrypted

$$r_1 \leftarrow \text{randi}(\mathbb{Z}_p^*)$$

$$E_1 = m_1 \cdot a^{r_1} \bmod p$$

$$D_1 = g^{r_1} \bmod p$$

$$r_2 \leftarrow \text{randi}(\mathbb{Z}_p^*)$$

$$E_2 = m_2 \cdot a^{r_2} \bmod p$$

$$D_2 = g^{r_2} \bmod p$$

$$\text{Enc}_{a, r_1, r_2}(m_1 \cdot m_2) = c_{12} = (E_{12}, D_{12})$$

$$E_{12} = m_1 \cdot m_2 \cdot a^{r_1 + r_2} \bmod p = \underbrace{(m_1 \cdot a^{r_1} \bmod p)}_{E_1} \cdot \underbrace{(m_2 \cdot a^{r_2} \bmod p)}_{E_2} \bmod p$$

$$E_{12} = E_1 \cdot E_2 \bmod p$$

$$D_{12} = g^{r_1 + r_2} \bmod p = \underbrace{(g^{r_1} \bmod p)}_{D_1} \cdot \underbrace{(g^{r_2} \bmod p)}_{D_2} \bmod p$$

$$D_{12} = D_1 \cdot D_2 \bmod p$$

$$\boxed{\text{Enc}_{a, r_1, r_2}(m_1 \cdot m_2) = c_1 \cdot c_2 \bmod p}$$

Multiplicative isomorphism

Multiplicatively additive isomorphism

$\text{Enc}(m_1 + m_2) = c_1 + c_2 \nleftrightarrow$ Pascal Paillier encryption.

One special encryption is instead of m_1, m_2 encryption to encrypt messages $n_1 = g^{m_1}$, $n_2 = g^{m_2}$

$$\text{Enc}(m_1 + m_2) = c_1 \circ c_2$$

Homomorphic encryption: cloud computation with encrypted data.

Paillier encryption scheme is additively-multiplicative homomorphic and has a potentially nice applications in blockchain, public procurement, auctions, gamblings and etc.

$$\text{Enc}(\text{Puk}, m_1 + m_2) = c_1 \circ c_2.$$