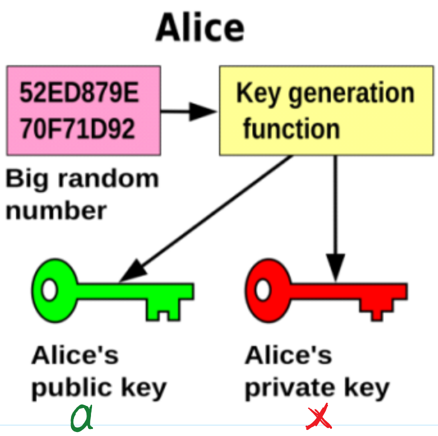
PKCS - Public Key Crypto System: 1.Key generation $PP = (p, g)$



① $p = 2q+1$; p, q - are primes

$2 \mid p-1$ & $q \mid p-1$

p - strong prime

>> genstrongprime(l)

$Z_p^* = \{1, 2, 3, \dots, p-1\}$ *mod p

② g is generator iff

$g^2 \neq 1 \mod p$ & $g^q \neq 1 \mod p$

Public parameters = (p, g) = PP $+, -, *$ mod $(p-1)$

A: $x \in_R Z_{p-1}$; $PrK_A = (x)$; $a = g^x \mod p$: $PuK_A = (a)$; $Z_{p-1} = \{0, 1, 2, \dots, p-2\}$

$x \leftarrow rand$: mod $(p-1)$

$1 < m < p$: message to be encrypted : $m \in Z_p^*$.

$PuK_A = a = g^x \mod p$ $c = Enc(PuK_A, m) = (E, D)$

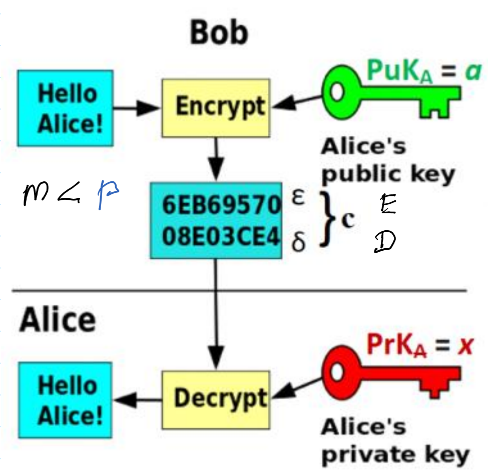
ElGamal Encryption

Zether: Towards Privacy in a Smart Contract World
Financial Cryptography and Data ..., 2020 - Springer

From <https://scholar.google.com/scholar?hl=en&as_sdt=0%2C5&q=Zether%3A+Towards+Privacy+in+a+Smart+Contract+World&btnG=>

Benedikt Bunz1, Shashank Agrawal2, Mahdi Zamani3, and **Dan Boneh**4

1Stanford University, benedikt@cs.stanford.edu
2Visa Research, shaagraw@visa.com
3Visa Research, mzamani@visa.com
4**Stanford University**, dabo@cs.stanford.edu



Ctrl/F --> ElGamal --> Exact maths **21**

B: intends to encrypt message M to A.

$F_{encod}(M) = m$

$m \in Z_p^*$; $r \xleftarrow{rand} Z_{p-1}$;

$E = m * a^r \bmod p$; $D = g^r \bmod p \Rightarrow c = (E, D)$

$\mathcal{B}:$ $\xrightarrow{c=(E,D)}$ $\mathcal{A}:$ $PrK_A = (x)$

$D^{-x \bmod (p-1)} \bmod p$

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

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

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

$\underbrace{-x \bmod (p-1) = (p-1-x)}$

Ex. $27 \bmod 54 = 27$

$27 \bmod 23 = 4 \neq 27$

1. $D^{-x} \bmod p = (g^r)^{-x} \bmod p =$

$= g^{-rx} \bmod p$

2. $m = E * D^{-x} = m * a^r * g^{-rx} =$

$= m * (g^x)^r * g^{-rx} \bmod p =$

$= m * g^{xr} * g^{-xr} \bmod p =$

$= m * g^0 \bmod p = m \bmod p = m$

since $1 < m < p$

Additively inverse element -***x*** to element ***x*** modulo ***p***-1.

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

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

```
>> mx = p-1-x
>> mod(x+mx, p-1) => 0
>> D_mx = mod_exp(D, mx, p)
```

Homomorphic encryption: cloud computation with encrypted data

$PP = (p, g)$

$\mathcal{B}: PuK_A = a$; $\mathcal{A}: PrK_A = x$; $a = g^x \bmod p$.

Multiplicatively Homomorphic Encryption

$\mathcal{B}:$

m_1, m_2 - two messages to be encrypted: $1 < m_1 * m_2 < p-1$.

m_1 : $r_1 \leftarrow randi(Z_{p-1})$

$E_1 = m_1 * a^{r_1} \bmod p$; $c_1 = (E_1, D_1)$ $\mathcal{A}:$

$m_1: \quad r_1 \leftarrow \mathrm{randi}(\mathbb{Z}_{p-1})$

$$\left.\begin{aligned} E_1 &= m_1 * a^{r_1} \bmod p \\ D_1 &= g^{r_1} \bmod p \end{aligned}\right\} \; c_1 = (E_1, D_1) \longrightarrow \quad A: \; \mathrm{Dec}(x, c_1) = m_1$$

$m_2: \quad r_2 \leftarrow \mathrm{randi}(\mathbb{Z}_p^*)$

$$\left.\begin{aligned} E_2 &= m_2 * a^{r_2} \bmod p \\ D_2 &= g^{r_2} \bmod p \end{aligned}\right\} \; c_2 = (E_2, D_2) \longrightarrow \quad \mathrm{Dec}(x, c_2) = m_2$$

$B: \quad m = m_1 * m_2 \bmod p$

$r = (r_1 + r_2) \bmod (p-1)$

$$m: \left.\begin{aligned} E &= m * a^r \bmod p \\ D &= g^r \bmod p \end{aligned}\right\} \; c = (E, D)$$

$A:$

$$c_1 * c_2 \bmod p = (E_1, D_1) * (E_2, D_2) = (E_1 * E_2, D_1 * D_2) =$$

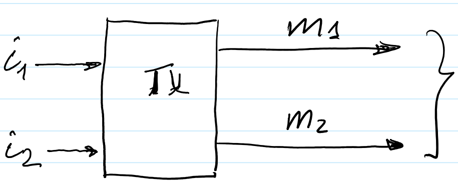
$$= (m_1 * m_2 * a^{r_1} * a^{r_2} \bmod p, \; g^{r_1} * g^{r_2} \bmod p) =$$

$$= (m * a^{(r_1+r_2) \bmod p-1} \bmod p, \; g^{(r_1+r_2) \bmod (p-1)} \bmod p) =$$

$$= (m * a^r \bmod p, \; g^r \bmod p) = c = (E, D)$$

Multiplicative homomorphic encryption means that encryption of multiplication $m_1 * m_2$ of two messages m_1, m_2 is equat to ciphertext c that is equal to the multiplication of two ciphertexts $c_1 * c_2$.

Fintex $\longrightarrow$ Blockchain: incomes = expenses



$$i = i_1 + i_2 = m_1 + m_2 = m \longrightarrow i = m$$

Balance equation

$$\left.\begin{aligned} \mathrm{Enc}(PuK, i_1 + i_2) &= c_i = c_{i1} * c_{i2} \bmod p \\ \mathrm{Enc}(PuK, m_1 + m_2) &= c_m = c_{m1} * c_{m2} \bmod p \end{aligned}\right\}$$

To prove that different c_i and c_m užšifruoja tą pačią sumą ta pačia suma

Additively-multiplicative homomorphic encryption

Property: everyone in the net could verify balance m:

e.g. $c_1 \cdot c_2 = c = Enc^+(a, m_1 + m_2) = Enc^+(a, m)$

Addively-multiplicative homomorphic encryption.

Let $n_1 = g^{m_1} \bmod p$, $n_2 = g^{m_2} \bmod p$ $\Big\}$ $n = n_1 * n_2 \bmod p = g^{m_1} * g^{m_2} \bmod p =$
$= g^{(m_1+m_2) \bmod (p-1)} \bmod p = g^{m \bmod (p-1)} \bmod p$

$m = m_1 + m_2 \bmod (p-1)$.

But $p \sim 2^{2048} \longleftrightarrow 10^{700}$ $\longrightarrow$ $i_1, i_2, m_1, m_2,$ etc. $<< 10^{700}$

Therefore $m_1 + m_2 \bmod (p-1)$ = always $= m_1 + m_2 = m$. $27 \bmod 1175 = 27$

Since $DEF(m_1) = g^{m_1} \bmod p$ is 1-to-1 mapping:
for one m_1 corresponds one $DEF(m_1)$, then
$DEF(m_1 + m_2) = DEF(m) = n_1 * n_2 \bmod p = n \bmod p = g^m \bmod p$.

B: $n = n_1 * n_2 \bmod p$.

n_1: $E_1 = n_1 * a^{r_1} \bmod p$, $D_1 = g^{r_1} \bmod p$ $\Big\}$ $c_1 = (E_1, D_1) \longrightarrow$ A: $Dec^+(x, c_1) = n_1$

n_2: $E_2 = n_2 * a^{r_2} \bmod p$, $D_2 = g^{r_2} \bmod p$ $\Big\}$ $c_2 = (E_2, D_2) \longrightarrow$ $Dec^+(x, c_2) = n_2$

$n = n_1 * n_2 \bmod p$; $r = (r_1 + r_2) \bmod (p-1)$.

n: $E = n * a^r \bmod p$, $D = g^r \bmod p$ $\Big\}$ $c = (E, D) \longrightarrow$ $Dec^+(x, c) = n = n_1 * n_2 \bmod p$.

A: must find m_1 from equation $g^{m_1} \bmod p = n_1$
m_2 $\qquad g^{m_2} \bmod p = n_2$

Net: must verify balance

If p is secure $p \sim 2^{2048} \approx 10^{700}$, the find m_1, m_2, in general, is infeasible.

But! If $m_1, m_2 \sim 10^9$, then m_1, m_2 could be found total scan procedure: search numbers from 1 to 10^9.
Sine A knows what sums should be received she simply verifies if $g^{m_1} \bmod p = n_1$
& $g^{m_2} \bmod p = n_2$.

Till this place

$m \in Z_p^*$; $r \in Z_{p-1}$; $\xRightarrow{\text{Ecryption}}$ $E \in Z_p^*$; $D = Z_p^*$

If $p = 11 \rightarrow$ $\left.\begin{array}{l} Z_p^* = \{1, 2, 3, \ldots, 10\} \\ Z_{p-1} = \{0, 1, 2, \ldots, 9\} \end{array}\right\} |Z_p^*| = |Z_{p-1}|$

$\mathrm{Enc}(m, r) = (E, D)$

$\mathrm{Enc} : Z_p^* \times Z_{p-1} \xleftrightarrow[\text{isomorphism}]{\text{one-to-one}} Z_p^* \times Z_p^*$

m_1, m_2 : must be encrypted using $r_1 \xleftarrow{\text{rand}} Z_{p-1}$ & $r_2 \xleftarrow{\text{rand}} Z_{p-1}$

$m = m_1 * m_2$ $\qquad r = r_1 + r_2$

$\mathrm{Enc}(m) = c = (E, D)$: $\begin{cases} E = m_1 * m_2 * g^{r_1 + r_2} \bmod p \\ D = g^{r_1 + r_2} \end{cases}$

$E = m \cdot g^r$; $E = g^r$.

Additively Homomorphic Encryption

ElGamal encryption. ElGamal encryption is a public key encryption scheme secure under the DDH assumption. A random number from $\mathbb{Z}_p^\star$, say x, acts as a private key, and $y = g^x$ is the public key corresponding to that. To encrypt an integer b, it is first mapped to one or more group elements. If $b \in \mathbb{Z}_p$, then a simple mapping would be to just raise g to b. Now, a ciphertext for b is given by $(g^b y^r, g^r)$ where $r \xleftarrow{\$} \mathbb{Z}_p^\star$. With knowledge of x, one can divide $g^b y^r$ by $(g^r)^x$ to recover g^b. However, g^b needs to be brute-forced to compute b.

$m \in Z_{p-1} : \ 1 < m < p-1$

$$E^+ = g^m * a^r \bmod p \ ; \quad D^+ = g^r \bmod p$$
$$E = m * a^r \bmod p \ ; \quad D = g^r \bmod p$$
$$\left.\begin{matrix} \\ \end{matrix}\right\} D^+ = D$$

$$C^+ = (E^+, D^+)$$

$B: \xrightarrow{\quad C^+ \quad} A: PrK_A = x$

1. Compute $(D^+)^{-x} * g^{-rx} \bmod p$

2. $E^+ * (D^+)^{-x} \bmod p = g^m$

Decrypted message is in the form of $\tilde{m} = \underbrace{g^m \bmod p}$

$dlog_g(\tilde{m}) = dlog_g(g^m \bmod p) = m$. discrete exp function DEF

If p is large, e.g. $p \sim 2^{2048}$, i.e. $|p| = 2048$ bits

Then computation of $dlog_g(\tilde{m})$ – is infeasible!

Since according to the complexity assumptions of
Discrete Logarithm Function
Discrete Logarithm Assumption – DLA

We argue that this is not an issue. First, as we will see, the Zether smart contract does not need to do this, only the users would do it. Second, users will have a good estimate of ZTH in their accounts because, typically, the transfer amount is known to the receiver. Thus, brute-force computation would occur only rarely. Third, one could represent a large range of values in terms of smaller ranges. For instance, if we want to allow amounts up to 64 bits, we could instead have 2 amounts of 32 bits each, and encrypt each one of them separately. In this paper, for simplicity, we will work with a single range, 1 to MAX, and set MAX to be 2^{32} in the implementation.

$2^{10} = 1024 = 1\,K \ ; \ 2^{20} = 1\,M ; \ 2^{30} = 1\,G \ ; \ 2^{40} = 1\,T \ ; \ 2^{50} = 1\,P$

$\sim 10^3 \quad \sim 10^6 \quad \sim 10^9 \quad \sim 10^{12} \quad \sim 10^{15}$

$$2^{64} = 2^{14} \cdot 2^{50} = 2^{14} \cdot 1\,P$$

$$\sim 8192 \cdot 10^{15}$$

$$\begin{array}{r} 4096 \\ 2 \\ \hline 8192 \end{array}$$

2^{2048} : $\quad 64 << 2048.$

Ethereum crypto currency $\quad 1\,Eth = 10^{18}$ gas

$1\,Eth \sim 400\,\$$

$1\,T\,Eth \equiv 2^{40}$

ElGamal encryption.
ElGamal encryption is a public key encryption scheme secure under the DDH assumption.
A random number from Z_p, say x, acts as a private key, and $a = g^x \bmod p$ the public key corresponding to that.
To encrypt an integer m in Z_{p-1}, it is first mapped to one or more elements of Z_p^*.
If m is in Z_p^*, then a simple mapping would be to just raise g to m.
Now, a ciphertext for m is given by ($g^m a^r$), where r is chosen at random from Z_{p-1}.
With knowledge of x, one can divide $g^m a^r$ by $(g^r)^x$ to recover g^m.

However, gb needs to be brute-forced to compute b.
We argue that this is not an issue. First, as we will see, the Zether smart contract does not need to do this, only the users would do it. Second, users will have a good estimate of ZTH in their accounts because, typically, the transfer amount is known to the receiver. Thus, brute-force computation would occur only rarely. Third, one could represent a large range of values in terms of smaller ranges. For instance, if we want to allow amounts up to 64 bits, we could instead have 2 amounts of 32 bits each, and encrypt each one of them separately. In this paper, for simplicity, we will work with a single range, 1 to MAX, and set MAX to be 232 in the implementation.

$$2^{64} = \underbrace{2^8 \cdot 2^8 \cdot 2^8 \cdot 2^8 \cdot 2^8 \cdot 2^8 \cdot 2^8 \cdot 2^8}$$

$dlog_g(\tilde{m})$ - - - - $\quad dlog_g(\tilde{m}) = m$

$PP = (p, g)$; $\quad |p| \sim 2^8$; $\quad |g| \sim 2^8$ $\quad$ search area

256 $\quad$ 256 ← choices

$|p| = 8$ bits $\quad |g| = 8$ bits

Ethereum: gas - price for computation of smart contract.

Ethereum: gas - price for computation of smart contract.

Search area is 1 - 16