El Gamal

Setup: $\mathbb{Z}/p\mathbb{Z}$, g = primitive root

Alice

message: $0 < m < p$

h

Encryption:

choose random $0 < k < p-1$

Compute $r = g^k$

$t = h^k m$

$\xrightarrow{(r,t)}$

Bob

Key generation: choose random $0 < a < p-1$

compute $h = g^a$

[Public Key: h
Private Key: a]

$\xleftarrow{h}$

(r, t)

Decryption: compute $t r^{-a}$

works because

$$t r^{-a} = h^k m (g^k)^{-a} = g^{ak} m g^{-ak} = m$$

Setup: $\mathbb{Z}/p\mathbb{Z}$, g primitive root

DLP
(Discrete Log Prob)
Given g^x
Find x

CDHP
(Computational Diffie-Hellman Prob)
Given g^x, g^y
Find g^{xy}

Break El Gamal
Given $g^a, g^k, h^k m$
Find m

Setup: $\mathbb{Z}/p\mathbb{Z}$, g primitive root

DLP
(Discrete Log Prob)
Given g^x
Find x

CDHP
(Computational Diffie-Hellman Prob)
Given g^x, g^y
Find g^{xy}

Break El Gamal
Given $g^a, g^k, h^k m$
Find m

Definition. Problem A "reduces to" Problem B in polynomial time if an algorithm for Prob B can be used polynomially often, along w/ polynomial time other computations, to solve Prob A.

Setup: $\mathbb{Z}/p\mathbb{Z}$, g primitive root

DLP
(Discrete Log Prob)
Given g^x
Find x

CDHP
(Computational Diffie-Hellman Prob)
Given g^x, g^y
Find g^{xy}

Break El Gamal
Given $g^a, g^k, h^k m$
Find m

Definition. Problem A "reduces to" Problem B in polynomial time if an algorithm for Prob B can be used polynomially often, along w/ polynomial time other computations, to solve Prob A.

Definition. Two problems are equivalent in poly time if they reduce to each other in poly time.

Example. CDHP reduces to DLP in polynomial time.

Proof. Suppose A is an algorithm to solve DLP.
Given a CDHP problem, g^x and g^y,
apply A twice to obtain x and y.
Then compute g^{xy}. $\blacksquare$

Definition. Problem A "reduces to" Problem B in polynomial time if an algorithm for Prob B can be used polynomially often, along w/ polynomial time other computations, to solve Prob B.

DLP (Discrete Log Prob)	CDHP (Computational Diffie-Hellman Prob)
Given g^x	Given g^x, g^y
Find x	Find g^{xy}

Definition. Two problems are equivalent in poly time if they reduce to each other in poly time.

Cryptography that relies on a "hard problem":

1. Encryption / Decryption / Key Generation (implementation) is polynomial time.

2. Breaking the system is equivalent to a hard problem for which no poly-time algorithms are known.

Cryptography that relies on a "hard problem":

1. Encryption / Decryption / Key Generation (implementation) is polynomial time.
2. Breaking the system is equivalent to a hard problem for which no poly-time algorithms are known.

Ex. El Gamal

1. Key Gen (Bob): modular expon.
   Encryption (Alice): mod. exp & mult.
   Decryption (Bob): mod. exp & mult.
   poly time
2. Breaking El Gamal is equivalent to CDHP.
   (believed no poly-time alg's exist)

El Gamal Security

El Gamal Security

① $\boxed{h = g^a \xrightarrow{DLP} \text{recover } a}$ breaks El Gamal (a is secret key)

② $\boxed{r = g^k \xrightarrow{DLP} \text{recover } k}$ breaks El Gamal (can compute $m = t h^{-k}$)

El Gamal Security

① $\boxed{h = g^a \xrightarrow{DLP} \text{recover } a}$ breaks El Gamal (a is secret key)

② $\boxed{r = g^k \xrightarrow{DLP} \text{recover } k}$ breaks El Gamal (can compute $m = t h^{-k}$)

③ Don't Re-Use k!

$$(r, t_1) = (r, h^k m_1)$$
$$(r, t_2) = (r, h^k m_2)$$

then $\left(\text{knowledge of } m_1\right) \Rightarrow \left(\begin{array}{c}\text{knowledge} \\ \text{of } h^k \\ = t_1 m_1^{-1}\end{array}\right) \Rightarrow \left(\begin{array}{c}\text{knowledge} \\ \text{of } m_2 \\ = t_2 (h^k)^{-1}\end{array}\right)$

RSA Algorithm

Alice

message $m \pmod n$

encryption:

$$c \equiv m^e \pmod n$$

(n, e)

c

Bob

chooses secret primes p, q
chooses secret d invertible mod
$\varphi(pq) = (p-1)(q-1)$
and its inverse e.

Public Key $(n = pq, e)$ "encrypt. exponent"
Private Key p, q, d "decryption exponent"

decryption:

$$c^d \pmod n$$
$$\equiv m^{ed} \pmod n$$
$$\equiv m^1 \pmod n$$
$$\equiv m$$

what we need to study:

Euclidean Algorithm → modular inversion in general
→ Chinese Remainder Theorem
→ Euler φ function (prove fmla)

Primality Testing (set up key)

Factoring Algorithms (for security)

Primality Testing

Fermat's Little Theorem: Let p be prime. Let $0<a<p$. Then $a^{p-1} \equiv 1 \pmod{p}$.

Fermat Primality Test: Let $n>1$ be an integer.

Choose a random $1<a<n$.

Check if $a^{n-1} \equiv 1 \pmod{n}$.

If NO $\Longrightarrow$ n is composite, we say a is a "Fermat witness"

if YES $\Rightarrow$ n is probably prime, but we don't know.

(If n is composite, but $a^{n-1} \equiv 1 \pmod{n}$, then we say a is a "Fermat liar".)

Runtime: polynomial.

What does "Probably" mean?

A "<u>Fermat pseudoprime to base a</u>" is an composite n s.t. $a^{n-1} \equiv 1 \pmod{n}$.

A "<u>Carmichael number</u>" is a Fermat pseudoprime to all bases a coprime to n.

e.g. 561, 41041, 825265, ... rare!

<u>Fact</u>: If n is not a Carmichael #, but is composite, then at least ½ of the invertible a's are "Fermat witnesses".

exercise

so Prob ≥ ½ that we <u>discover</u> compositeness using F.P.T.

So: run the test for many random a's.