Wrapping up El Gamal...

Recall DDH: $(g^a, g^b, g^{ab}) \underset{PPT}{\approx} (g^a, g^b, g^c)$

for $a, b, c \in \mathbb{Z}_q$

where $|\langle g \rangle| = q$

Note: DDH not true with $\langle g \rangle = \mathbb{Z}_p^*$!

Assumption has held up so far w/ $g = \gamma^{\frac{p-1}{q}}$ as the generator where q is prime and $p - 1 = Nq$.

Practical issue: plaintext space is...? $\langle \gamma \rangle$

How to encode messages in $\langle \gamma \rangle$?

(Encoding in $\mathbb{Z}_p^*$ is easy enough:

"hi!" $\rightarrow$ (int)'h' + (int)'i' * 256 + (int)'!' * 256^2

(Just use characters to represent an integer in base 256))

Reminder on El Gamal: $SK = a \in_R \mathbb{Z}_q$

$PK = A = \gamma^a$

$E_{PK}(m)$: $(B, A^b m)$ where $b \in_R \mathbb{Z}_q$, $B = \gamma^b$ and $m \in \langle \gamma \rangle$

$\langle \gamma \rangle = \{ \gamma^0, \gamma^1, \gamma^2, \ldots \gamma^{q-1} \}$

How to encode (text) message in $\langle \gamma \rangle$?

Treat $m \in \mathbb{Z}_q$ (base 256 as before)

Then send γ^m?

Annoying for recipient! Alice would have to compute a discrete log to find m x_t

Better idea: don't encode messages in $\langle g \rangle$ at all.

instead use "hashed ElGamal":

Say $H : \mathbb{Z}_p \longrightarrow \{0,1\}^t$.

Now for $m \in \{0,1\}^t$, encrypt like so:

$$E_{PK}(m) = (B, H(A^b) \oplus m) \qquad (PK : A = g^a)$$

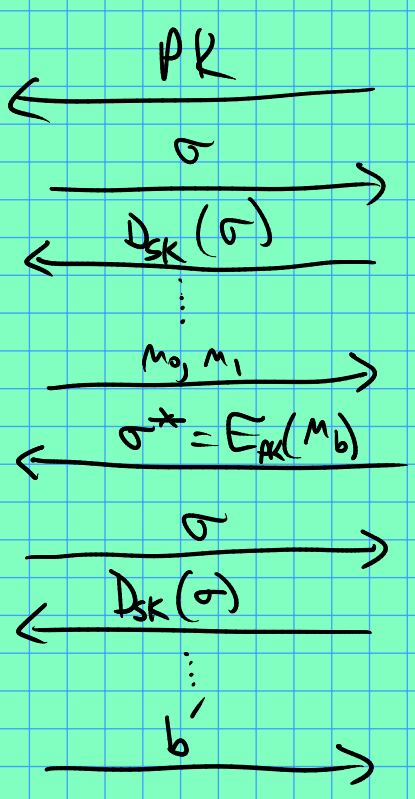
H serves as a "randomness extractor".

(See the "leftover hash lemma" for more.)

Is (vanilla) ElGamal CCA2 secure?

↑ not hashed

No:



Observation about ElGamal:

Say $\sigma = E_{PK}(m)$, $\sigma' = E_{PK}(m')$ $\qquad B = g^b$, $B' = g^{b'}$

$= (B, A^b \cdot m)$ $\quad = (B', A^{b'} \cdot m')$

Note: $\sigma\sigma' = (BB', A^b m A^{b'} m')$

$= (g^b g^{b'}, A^b A^{b'} m m')$

$= (g^{(b+b')}, A^{(b+b')} \cdot (mm')) = E_{PK}(mm')$

$E_{PK}(m) \cdot E_{PK}(m') = E_{PK}(mm')$ ("Homomorphic encryption")

Can use this to win CCA 2 game w/ probability 1.

Choose any $m_0 \neq m_1 \in M (= \langle g \rangle)$.

Upon receipt $\sigma^* = E_{PK}(m_b)$, $b \in \{0,1\}$

form $\sigma = E_{PK}(1) = E_{PK}(g^0)$ using fresh randomness.

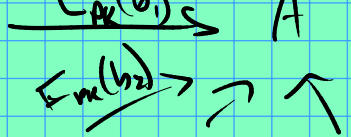
Now set $\tilde{\sigma} = \sigma^* \sigma$. Note: $\tilde{\sigma}$ decrypts to same message as σ^*.

Submit $\tilde{\sigma}$ as a decryption query.

Profit.

Why is CCA2 even important?

Canonical example: silent auctions / bidding.



Say $E(m)\,E(m') = E(m+m')$

then could produce a slightly higher bid to

$\sigma = E(b)$ by forming $\sigma' = \sigma\, E(1)$

$= E(b+1)$

A few words about the project.

First look at symmetric encryption (e.g. AES).

Assuming AES is IND-CPA, how to get CCA2 security? Idea: add a MAC to "neutralize" the decryption queries:

New key is $k = (k_{AES}, k_{mac})$.

$$E_k(m) = (C_1, C_2)$$

$$C_1 = E_{k_{AES}}(m) \qquad C_2 = MAC_{k_{MAC}}(C_1)$$

$D_k(C)$: Parse $C = (C_1, C_2)$.

If $MAC_{k_{MAC}}(C_1) \neq C_2$, output "$\perp$"

Else decrypt as normal:

output $m = D_{k_{AES}}(C_1)$.

What about public key encryption?

"Key Encapsulation Method" (KEM).

Broad Strokes: use any public key scheme to communicate a symmetric key:

choose random k for a symmetric key scheme,

then send $\left(E_{PK}(k),\ AES_k(m)\right)$

k: 256 bits!

Real version: $\left(E_{PK}(x),\ (C_1, C_2)\right)$

$C_1 = AES_{k_{AES}}(m)$, $C_2 = MAC_{k_{MAC}}(C_1)$

where $KDF(x) = k_{AES} \| k_{MAC}$

Chinese Remainder Theorem (CRT)

$n_1, n_2, \ldots n_m$

$x \equiv y_1 \mod n_1$

$x \equiv y_2 \mod n_2$

$\vdots$

Algebraic View: say $n = pq$.

$\mathbb{Z}_n \simeq \mathbb{Z}_p \times \mathbb{Z}_q = \{(a,b) \mid a \in \mathbb{Z}_p,\ b \in \mathbb{Z}_q\}$

Arithmetic in $\mathbb{Z}_n$ "looks like" that of $\mathbb{Z}_p \times \mathbb{Z}_q$.

$[(a,b) + (a',b') = (a+a' \bmod p,\ b+b' \bmod q)]$

More formally, there is a bijective function

$\psi : \mathbb{Z}_n \longrightarrow \mathbb{Z}_p \times \mathbb{Z}_q$

s.t. $\forall\ x,y \in \mathbb{Z}_n$ ① $\psi(x+y) = \psi(x) + \psi(y)$

② $\psi(xy) = \psi(x)\psi(y)$

③ $\psi(0) = (0,0)$

④ $\psi(1) = (1,1)$

$\psi(x) + \psi(y) = \psi(x+y)$

$\psi(x) \cdot \psi(y) = \psi(xy)$

$\mathbb{Z}_n \mid \mathbb{Z}_p \times \mathbb{Z}_q$

Example: say $n = 6 = 2 \cdot 3$, with $2 = p$ and $3 = q$.

$$\mathbb{Z}_n \xrightarrow{\ \psi\ } \mathbb{Z}_p \times \mathbb{Z}_q$$

- $0 \to (0, 0)$
- $1 \to (1, 1)$
- $2 \to (0, 2)$
- $3 \to (1, 0)$
- $4 \to (0, 1)$
- $5 \to (1, 2)$

$$\psi : x \longmapsto (x \bmod p,\ x \bmod q)$$

$$\psi(2+3) = (0,2) + (1,0)$$

$$\| \qquad\qquad \| \qquad \checkmark$$

$$\psi(5) = (1,2)$$

Application: Speed up RSA <u>decryption!</u>

Say p, q are k bit integers.

Then the cost of decryption? $c^d \bmod n$

$c, d, n \approx 2k$ bits.

Using long multiplication, cost is $\approx (2k)^3$

$= 8k^3$ steps

If we first compute the CRT form, then what is the cost?

$c \underset{CRT}{\equiv} (c_p, c_q)$ (c mod p, c mod q)

Want $c^d \bmod n \underset{CRT}{\equiv} (c_p^d \bmod p,\ c_q^d \bmod q)$

$|\mathbb{Z}_p^*| = p-1$. $\forall x \in \mathbb{Z}_p^*$, $x^{p-1} = 1 = x^0$

So we can actually compute as $\left(c_p^{d \bmod (p-1)} \bmod p,\ c_q^{d \bmod (q-1)} \bmod q\right)$

Cost: $\approx k^3 + k^3$.

So, only $2k^3 + [\text{cost of } \psi, \psi^{-1}]$

vs $8k^3$ to do it directly in $\mathbb{Z}_n$!

$$\begin{array}{ccc} \mathbb{Z}_n & \xrightarrow{\psi} & \mathbb{Z}_p \times \mathbb{Z}_q \\ \text{decrypt}\downarrow & & \downarrow \text{<arithmetic...> (decryption)} \\ \mathbb{Z}_n & \xleftarrow{\psi^{-1}} & \mathbb{Z}_p \times \mathbb{Z}_q \end{array}$$

Application 2: factoring $n = pq$ is equivalent to computing $\sqrt{x}$ in $\mathbb{Z}_n$ for $x = y^2$.

Say we are given a machine that computes $\sqrt{x}$ on input $x = y^2 \in \mathbb{Z}_n^*$.

How many square roots does x have?

Well, for $x = y^2 \in \mathbb{Z}_p^*$, there are only 2 square roots: $\pm y$.

For $x = y^2 \in \mathbb{Z}_n^*$, say $y \underset{CRT}{\equiv} (y_p, y_q)$, and $x \underset{CRT}{\equiv} (x_p, x_q)$.

$x_p = y_p^2$, $x_q = y_q^2$

Then x_p has 2 square roots in $\mathbb{Z}_p$, as does x_q in $\mathbb{Z}_q$:
$\pm y_p$ and $\pm y_q$.

So, $y' = (\pm y_p, \pm y_q)$ will all satisfy $y^2 = x$.

So x has 4 square roots.

How would this hypothetical machine enable us to factor n?

Say we choose $y \in_R \mathbb{Z}_n^*$, and set $x = y^2$.

On input x, black box will give <u>some</u> $\sqrt{x}$:
$(\pm y_p, \pm y_q)$.

Strategy: say box gives $y' = \sqrt{x}$.
we will compute $\gcd(y + y', n)$ until
we get a result other than 1 or n.

what are the possible values of $\gcd(y+y', n)$?

$\gcd(z, n) \in \{1, p, q, n\}$

Say $y' = y$. Then $y + y' = 2y$ ...
probably $\gcd(2y, n) = 1$

Say $y' = -y$ Then $y + y' = 0$, and $\gcd = n$.

Otherwise either $y' \equiv_{CRT} (-y_p, y_q)$ or
$y' \equiv_{CRT} (y_p, -y_q)$.

Hence $y + y' \equiv_{CRT} (0, 2y_q)$
or $(2y_p, 0)$ in this case.

$y + y' \equiv_{CRT} (0, 2y_q) \Rightarrow \gcd(y+y', n) = p$