6.857 Rivest
3/14/11 L12.1

Admin:

Outline:

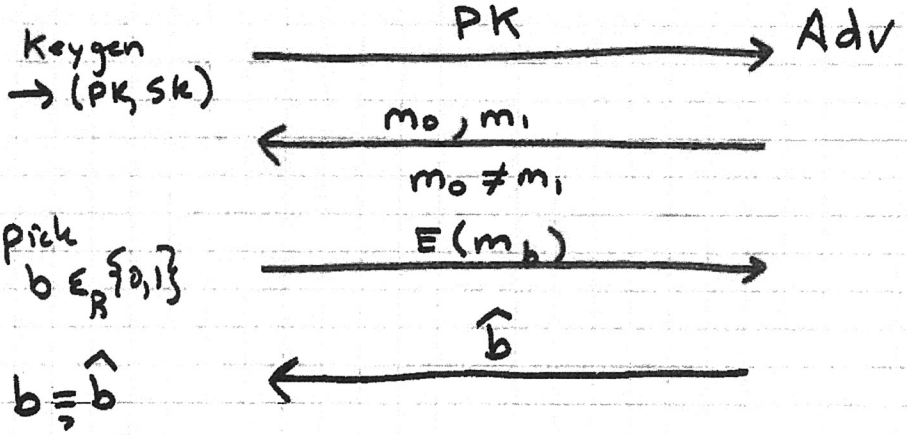
IND-CCA2 review

Cramer-Shoup

Elliptic Curves

Semantic Security

- early def of security for PK enc (Goldwasser/Micali)
- Adversary can't tell $E(m_0)$ from $E(m_1)$
- Game



if yes, adversary wins.

- Scheme is semantically secure if $Pr[\text{Adv wins}] \leq \frac{1}{2} + \text{negligible}$
- (Note: scheme must be randomized to be sem. secure, at least...)
- Is El Gamal semantically secure?
- Recall DDH:

  distinguishing (g^a, g^b, g^c) from (g^a, g^b, g^{ab}) is hard.

  a, b, c random a, b random

  [Note: Boneh presented this as
  four tuple (g, g^a, h, h^d) is $a \stackrel{?}{=} d$
  is the same (g, g^a, g^b, g^{bd}) is $a \stackrel{?}{=} d$

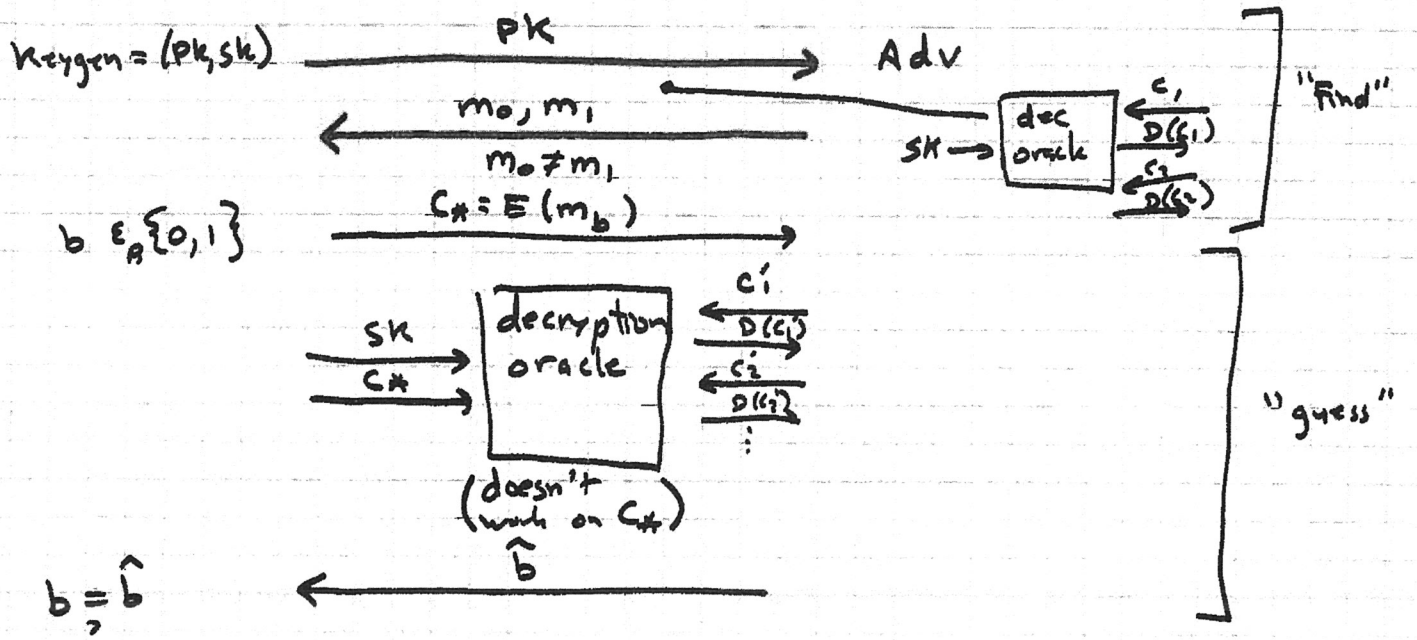
- Theorem (Tsiounis & Yung):

  El Gamal is sem secure $\iff$ DDH holds in G

- Sem. security may not be enough:
  - El Gamal is malleible: Given $E(m) = (g^k, m \cdot y^k)$ easy to produce $E(2m) = (g^k, 2 \cdot m \cdot y^k)$ without knowing m!

(Imagine I want to outbid you at an auction.

- Also, not IND-CCA2 secure (ACCA) adaptive chosen-ciphertext attack



- Scheme is ACCA secure (IND-CCA2 secure) if
  Prob (Adv wins) $\leq \frac{1}{2}$ + negligible

- El Gamal is not IND-CCA2 secure:
  Given $C_* = (g^k, m \cdot y^k)$
  ask to decrypt $C_*' = (g^k, 2my^k) \Rightarrow 2m \stackrel{\div 2}{\Rightarrow} m$

- El Gamal is homomorphic: $C_1 \in E(m_1) = (g^r, m_1 \cdot y^r)$
  $C_2 \in E(m_2) = (g^s, m_2 \cdot y^s)$
  $C \cdot C_1 = (g^{r+s}, m_1 \cdot m_2 \cdot y^{r+s}) = E(m_1 \cdot m_2$

Cramer-Shoup

IND-CCA2 secure
Can be viewed as elaboration of El Gamal
One of simpler ones. "Plaintext aware"...

Let G_q be group of prime order q (E.g. squares in Z_p^*, where $p = 2q+1$).

Keygen: $g_1, g_2 \in_R G_q$

$x_1, x_2, y_1, y_2, z \in_R Z_q$

H: hash fn mapping $G_q^3 \to Z_q$

$c = g_1^{x_1} g_2^{x_2}$

$d = g_1^{y_1} g_2^{y_2}$

$h = g_1^z$ ← EG

$PK = (g_1, g_2, c, d, h, H)$

$SK = (x_1, x_2, y_1, y_2, z)$

Encrypt (m): $(m \in G_q)$

$r \in_R Z_q$ ← EG

$u_1 = g_1^r$

$u_2 = g_2^r$

$e = h^r \cdot m$ ← EG

$\alpha = H(u_1, u_2, e)$

$v = c^r d^{r\alpha}$

ciphertext $= (u_1, u_2, e, v)$

Decrypt (u_1, u_2, e, v):

$\alpha = H(u_1, u_2, e)$

check: $u_1^{x_1 + y_1 \alpha} \; u_2^{x_2 + y_2 \alpha} \stackrel{?}{=} v$

if not =, **reject**

output: $m = e / u_1^{z}$ ← EG

Note: $u_1^{x_1} u_2^{x_2} = g_1^{r x_1} g_2^{r x_2} = c^r$

$u_1^{y_1} u_2^{y_2} = d^r$

$u_1^{z} = g_1^{rz} = h^r$

Thm: Cramer Shoup is secure against adaptive chosen ciphertext attack (IND-CCA2 secure) if DDH assumption holds in G_q and H satisfies a certain condition ($\simeq$ "target collision resistance").

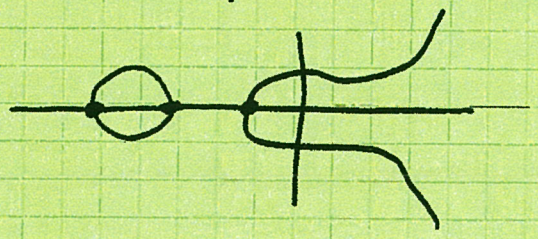
Thus, this strongest notion of security for PK encryption is achievable, albeit at some cost in terms of speed & complexity.

Elliptic curve groups:

Let p be a prime

Let $a, b \in \mathbb{Z}_p$ $\quad [4a^3 + 27b^2 \neq 0 \bmod p]$

Consider eqn **E**: $y^2 = x^3 + ax + b \pmod p$



roots r_1, r_2, r_3

$((r_1 - r_2)(r_1 - r_3)(r_2 - r_3))^2$

$= -(4a^3 + 27b^2)$

So roots are distinct

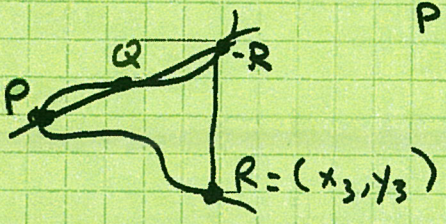
Def: points on curve $E = E(\mathbb{F}_p)$ $\qquad \mathbb{F}_p = GF(p)$

$= \{(x,y) : y^2 = x^3 + ax + b\} \cup \{\infty\}$

$|E(\mathbb{F}_p)| = p + 1 + t$ where $|t| \leq 2\sqrt{p}$ $\quad$ [can find "efficiently"]

Can define group law on $E(\mathbb{F}_p)$: (written additively)

- ∞ is the identity $\qquad P + \infty = \infty + P = P$
- $P + Q = R$



$P = (x_1, y_1)$

$Q = (x_2, y_2)$

If $x_1 \neq x_2$: $\begin{cases} x_3 = m^2 - x_1 - x_2 \\ y_3 = m(x_1 - x_3) - y_1 \end{cases}$ $\quad m = \dfrac{y_2 - y_1}{x_2 - x_1}$

If $x_1 = x_2$ & $y_1 \neq y_2$: $P + Q = \infty$

If $P = Q$ & $y_1 = 0$: $P + Q = \infty$

If $P = Q$ & $y_1 \neq 0$: $\begin{cases} x_3 = m^2 - 2x_1 \\ y_3 = m(x_1 - x_3) - y_1 \end{cases}$ $\quad m = \dfrac{3x_1^2 + A}{2y_1}$ ← a

associative!

$P = (x,y) \Rightarrow -P = (x,-y)$ inverses

may or may not be cyclic.

```
09:14:47 notes $ /Applications/sage/sage
Detected SAGE64 flag
Building Sage on OS X in 64-bit mode
----------------------------------------------------------------------
| Sage Version 4.6.2, Release Date: 2011-02-25                       |
| Type notebook() for the GUI, and license() for information.        |
----------------------------------------------------------------------
sage: # some experiments with elliptic curves with sage
sage: # first define a field mod 101
sage: F = Zmod(101)
sage: F
Ring of integers modulo 101
sage: # example of multiplication in F
sage: F(10)*F(11)
9
sage: # define elliptic curve over F
sage: E = EllipticCurve(F,[0,1])
sage: E
Elliptic Curve defined by y^2 = x^3 + 1 over Ring of integers modulo 101
sage: P = E.random_point()
sage: P
(96 : 49 : 1)
sage: # note coordinates are in projective form (X : Y : Z) representing
sage: # point x = X/Y, y = Y/Z, with Z = 0 for point at infinity.
sage: # get another point
sage: Q = E.random_point()
sage: Q
(29 : 94 : 1)
sage: P+Q
(21 : 77 : 1)
sage: # check commutativity
sage: Q+P
(21 : 77 : 1)
sage: # get third point
sage: R = E.random_point()
sage: R
(76 : 58 : 1)
sage: # check associativity
sage: P + (Q+R)
(53 : 2 : 1)
sage: (P+Q)+R
(53 : 2 : 1)
sage: # find size of this group
sage: E.order()
102
sage: # what are factors of 102?
sage: factor(102)
2 * 3 * 17
sage: # so possible orders of elements are 1,2,3,6,17,34,51,102
sage: P.order()
51
sage: Q.order()
51
sage: R.order()
51
sage: # none of P,Q, R are a generator (i.e. have order 102)
sage: # let's find one
sage: R = E.random_point()
```

```
sage: R.order()
102
sage: # bingo
sage: # what does identity look like?
sage: P-P
(0 : 1 : 0)
sage: I = P-P
sage: I
(0 : 1 : 0)
sage: I+P
(96 : 49 : 1)
sage: P+I
(96 : 49 : 1)
sage: -P
(96 : 52 : 1)
sage: # note that inverses just negate Y component, modulo 101
sage: # look at some small powers of generator R
sage: for i in range(15): print i, i*R
....:
0 (0 : 1 : 0)
1 (72 : 85 : 1)
2 (15 : 89 : 1)
3 (9 : 86 : 1)
4 (84 : 21 : 1)
5 (52 : 44 : 1)
6 (87 : 61 : 1)
7 (90 : 65 : 1)
8 (35 : 31 : 1)
9 (10 : 71 : 1)
10 (18 : 51 : 1)
11 (93 : 14 : 1)
12 (4 : 41 : 1)
13 (38 : 38 : 1)
14 (76 : 58 : 1)
sage: # find discrete log of P, base R
sage: R.discrete_log(P)
80
sage: 80*R
(96 : 49 : 1)
sage: 80*R==P
True
sage: # find discrete log of Q, base R
sage: R.discrete_log(Q)
58
sage: # find elements of each possible order
sage: R.order()
102
sage: S = 2*R
sage: S.order()
51
sage: S = 3*R
sage: S.order()
34
sage: S = 6*R
sage: S.order()
17
sage: S = 17*R
sage: S.order()
```

```
6
sage: S = 34*R
sage: S.order()
3
sage: S = 51*R
sage: S.order()
2
sage: S
(100 : 0 : 1)
sage:
```