## Lecture 11: October 30, 2001

- Midterm solved
- Review
- Signature Schemes
- OWFs



## El Gamal and Discrete Logs

The discrete logarithm problem in Zp
(where p - large prime, at least 512 bits)

*Eve knows:*
$p, b, y$
$p - prime$
$b \in Z_p^* - generator, base$
$y \in Z_p^* - query\ element$
$FIND\ x \in [0, p-2]\ \ni\ b^x \equiv y \bmod p$

Factoids:
- no known poly algorithms
- basis for many crypto methods
- conjectured to be a good OWF; why?

# El Gamal PK cryptosystem (83)

$p$ – large prime

$b$ – base, primitive element, generator

$x$ – private exponent

$y$ – public residue; $y \equiv b^x \bmod p$

$P = Z_p^*$

$C = Z_p^* \times Z_p^*$

publics : $p, b, y$

secrets : $x$

*Encryption*:

1. generate random $r \in Z_{p-1}$
2. compute: $k = b^r \bmod p$
3. compute: $c = my^r \bmod p = mb^{xr} \bmod p$
4. ciphertext = $\{k,c\}$

*Decryption*:

1. compute $k^x \bmod p$
2. compute $(k^x)^{-1} \bmod p$
3. $m' = (k^x)^{-1} c = b^{-rx} mb^{xr} \bmod p = m$



# Merkle-Hellman Knapsack PKCS

Both are NP-c

Subset sum decision problem

$S = \{s_1,...,s_n\}$ – sizes
$T$ – target sum

Does there exist a binary vector
$X = \{x_1,...,x_n\}$
such that :
$\sum S \times X = T$

Subset sum search problem

$S = \{s_1,...,s_n\}$ – sizes
$T$ – target sum

Compute a binary vector
$X = \{x_1,...,x_n\}$
such that :
$\sum S \times X = T$

$S = \{s_1,...,s_n\}$ superincreasing iff $\forall i$
$s_i > s_1 + ... + s_{i-1}$ → Trivial to solve...

## Merkle-Hellman (contd)

$S = \{s_1,...,s_n\}$ – *sizes (superincreasing)*
$p$ – *prime*, $p > \sum S$
$a < p$
$C = \{c_1,...,c_n\}$
$c_i = as_i \bmod p$

*publics* : $C$
*secrets* : $p,a,S$

*Encryption* :
*cleartext* $X = \{x_1,...,x_n\}$
$e_k(X) = \sum XC$

*Decryption* :
*ciphertext* $Y$
1. $T = a^{-1}Y \bmod p$
2. *solve SSP for* $S,T -> X$



## Digital Signatures

**A signature scheme:**

***(P,A,K,Sign,Verify)***

***P* - plaintext (msgs)**

***A* - signatures**

***K* - keys**

***Sign* - signing function: (*P*K)->A***

***Verify* - verification function: *(P*A*K)->{0,1}***

## RSA Signature Scheme

$Let\ n = pq\ where\ p \neq q - (large)\ primes$

$e, d \in_R Z_n \ \ and\ e = d^{-1}\ and\ ed \equiv 1\ mod\ \Phi(n)$

$\Phi(n) = (p-1)(q-1)$

$Secrets: p, q, d$

$Publics: n, e$

$Signing: message = m$
$Sign(x): y = m^d\ mod\ n$

$Verification: signature = y$
$Verify(y, m): (m = y^e)???$



## RSA Signature Scheme (contd)

The good:
- •Verification can be made cheap
- •Same as decryption function
- •Security based on RSA encryption
- •Signing is harder but #verify-s > 1…
- •Deterministic

The bad:
- •Recall that RSA is malleable: signatures can be "massaged"
- •Phony "random" signatures
  - •compute Y=RSA(e,X)=$X^e$ mod n
  - •X is a signature of Y because $Y^d$=X mod n

The ugly:
- •Signing requires integrity!
- •How to sign multiple blocks?
- •Deterministic

## El Gamal Signature Scheme

$p$ – large prime
$b$ – base, generator
$x$ – private exponent
$y$ – public residue ; $y \equiv b^x \bmod p$
$P = Z_p^*$
$A = Z_p^* \times Z_p^*$
publics : $p, b, y$
secrets : $x$

Signing :
1. generate random $r \in Z_{p-1}$
2. compute : $k = b^r \bmod p$
3. compute : $c = (m - xk)r^{-1} \bmod p - 1$
4. signatur $e = \{k, c\}$

Verifying :
$y^k k^c \bmod p = b^m \bmod p$ ???

notice that :
$y^k k^c = b^{xb^r} (b^r)^{(m/r - xk/r)} = b^{xb^r + m - xb^r} = b^m$



## El Gamal PK Cryptosystem

$p$ – large prime
$b$ – base, primitive element, generator
$x$ – private exponent
$y$ – public residue; $y \equiv b^x \bmod p$
$P = Z_p^*$
$C = Z_p^* \times Z_p^*$
publics : $p, b, y$
secrets : $x$

Encryption :
1. generate random $r \in Z^*_{p-1}$
2. compute : $k = b^r \bmod p$
3. compute : $c = my^r \bmod p = mb^{xr} \bmod p$
4. ciphertext = $\{k,c\}$

Decryption :
1. compute $k^x \bmod p$
2. compute $(k^x)^{-1} \bmod p$
3. $m' = (k^x)^{-1} c = b^{-rx} m b^{xr} \bmod p = m$

## El Gamal Signature Scheme

$p$ – large prime
$b$ – base, generator
$x$ – private exponent
$y$ – public residue; $y \equiv b^x \bmod p$
$P = Z_p^*$
$A = Z_p^* \times Z_p^*$
publics : $p, b, y$
secrets : $x$

Signing :
1. generate random $r \in Z^*_{p-1}$
2. compute : $k = b^r \bmod p$
3. compute : $c = (m - xk)r^{-1} \bmod p - 1$
4. signature = $\{k,c\}$

Verifying :
$y^k k^c \bmod p = b^m \bmod p$ ???

notice that :
$y^k k^c = b^{xb^r} (b^r)^{(m/r - xk/r)} = b^{xb^r + m - xb^r} = b^m$

## El Gamal Sig. Scheme (contd)

The good:
- Signing is cheap(er)
- Designed as a signature function
- Non-deterministic

The bad:

Compare to RSA
- Some attacks possible
- Randomizers cannot be revealed (trace)
- Randomizers cannot be reused

The ugly:
- Signing requires integrity (cf. attacks)
- How to sign multiple blocks?



## El Gamal (Attacks)

Impromptu forgery

*Signing*:

0. *select*: $i, j \in [0, p-2]$
   $gcd(j, p-1) = 1$
1. *compute*: $k = b^i y^j \bmod p$
2. *compute*: $c = -kj^{-1} \bmod p-1$
3. *compute*: $m = -kij^{-1} \bmod p-1$
4. *signature* = {k,c}

*Verifying*:

$y^k k^c \bmod p = b^{xb^{i+xj}} (b^{i+xj})^{-b^i b^{xj}/j} =$
$b^{xb^{i+xj} - ib^{i+xj}/j - xb^{i+xj}} = b^{-b^i y^j i/j} = b^m$!!!

- Oracle forgery (k,c given) (see p. 209)

- No chosen-message forgery!
- *m* determined by forged *k* and *c*

## Hash Functions

- Many applications; in and out of crypto
- Make signatures practical
- Provide integrity and authentication (later)
- Reduction of input into digest
- |domain|>>|range|
- Need to be very, very fast
  and secure…
  - collision-resistance (weak)
  - collision-resistance (strong)
  - one-way-ness (e.g., fingerprint)



## The Birthday Paradox

- the hash function: Birthday(person)=y
- y ranges over Y=[1…365], let |Y|=n
- how many people do we need to 'hash'
  to have a collision?

What is the probability of selecting at random
k random and DISTINCT numbers from Y?

P0=1*(1-1/n)*(1-2/n)*…*(1-(k-1)/n)) == e^(k(1-k)/n)
P1=1-P0 ---> at least one collistion
Say, P1 is at least 0.5… solve for k

k == 1.17 * SQRT(n)
k = 22.3 for n=365

## Discrete-log hash function

- Chaum et al.
- Provably secure based on discrete log problem

- p, q=(p-1)/2 --- primes
- let n=|p|
- a, b --- generators in Zp

- secret x, where $a^x = b \bmod p$
- h (x1,x2) = $a^{x1}\ b^{x2} \bmod p$
- where x1,x2 in [0,…,q-1]

- Given just one collision, x can be computed!!!
- Expensive, not very reductive:
  Hashes two (n-1)-bit numbers to one n-bit number