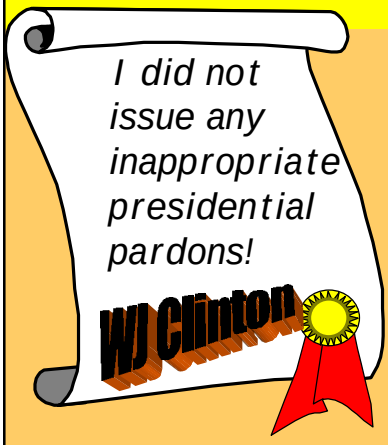
## Lecture 11: October 31, 2001

- Hash functions
- DSS
- DH Key Distribution



## Digital Signatures




- Integrity
- Authentication
- Non-repudiation
- Timestamping
- Causality
- Authorization

## Hash Functions

- Many applications; in and out of crypto
- Make signatures practical
- Provide integrity and authentication
- Reduction of input into fixed-size digest
- |domain|>>|range|
- Need to be very, very fast and secure…



## Properties of a Good Hash Function

- fixed-size digest
  - for any |x|, the size of f(x) is constant
- easy computation
  - f(x) is easy to compute
- one-wayness
  - given y, finding x such that f(x)=y is hard
- collision-resistance (weak)
  - given x, finding z such that f(x)=f(z) is hard
- collision-resistance (strong)
  - finding **any** x,z such that f(x)=f(z) is hard

## The Birthday Paradox

- 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



## The Birthday Paradox

What is it good for?

Test of strong collision-resistance: for a well-designed hash function with n-bit output, it must hold that:

finding any pair $(x, y)$ э $f(x) = f(y)$
takes $2^{n/2}$ trials

## 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



## Discrete-log hash function

Suppose Alice computes (x1,x1',x2,x2') such that

$$h(x1,x2) = h(x1',x2')$$

Or

$$a^{x1} b^{x2} \bmod p = a^{x1'} b^{x2'} \bmod p$$

then, $x1+x2*x \bmod (p-1) = x1' + x2'*x \bmod (p-1)$

$(x2-x2')x \bmod (p-1) = x1'-x1 \bmod (p-1)$

## More on hash functions

- Given a secure finite-input H() we can construct a secure arbitrary-input H*() by "sort of" hashing the hashes...

- Secure hash functions can be built out of cryptosystems where keys are known, e.g.:
  - Hi = E(Hi-1,Xi) XOR Xi

  or
  - Hi = E(Hi-1,Xi XOR Hi-1) XOR Xi
- But, **truly useful** hash functions must be FAST!
- Examples: MD4 & MD5 (Rivest), SHA/SHS (NIST)
- Use only: |, &, >>, +, ~, XOR
- SW-optimized…
- Security is based on prudent design, not math...



## MD5

- MD5 (Rivest, 1992): 128-bit hash, 512-bit blocks (similar to MD4, 1990)
- (MD = Message Digest)
- Simplified versions have been cryptanalyzed, but not MD5 itself
- But: strong **collision resistance only 64-bit**
- Not really long enough nowadays
- Like DES: now being phased out

## SHA

- Secure Hash Algorithm SHA (or SHA-1): NIST, NSA, 1995
- 160-bit hash, 512-bit blocks
- Used in DSS (Digital Signature Standard
- 80-bit strong collision resistance



## RIPE-MD

- RIPE-MD developed in Europe (1996-7), funded by EC
- RIPEMD-160: 160-bit hash, 512-bit blocks (same as SHA-1)
- Comparable to SHA-1 in speed, security

- Both SHA and RIPE-MD are roughly half the speed of MD5
- American standard is SHA-1 (for now)
- SHA-256, SHA-384, SHA-512 match key lengths in AES

## Schnorr Signature

$Z_p^* =< b >$

$g = b^{(p-1)/q}$

$G_q =< g >= \{1, g, g^2, ..., g^{q-1}\}$

$x \in_R Z_q$

$y \equiv g^x \bmod p$

$P = Z_p^*$

$A = Z_p^* \times Z_p^*$

*publics* : $p, b, g, y$

*secrets* : $x$

*Signing* :

1. *generate* $r \in_R Z_{p-1}$
2. *compute* : $e = H(m \| g^r)$
3. *compute* : $s = (r - xe) \bmod q$

*signature* $= \{e, s\}$

*Verifying* :

$e = H(m \| g^s y^e)$ ???

*notice that* :

$g^s y^e = g^{r-xe} g^{xe} = g^r$



## The Digital Signature Standard (DSS)

- Why DSS?
- RSA issues: patents, malleability, etc.
- A variant of El Gamal
- Originally for |p|=512 bits, now up to 1024
- Optimized for signature size (320- vs. 1024-bit)
- Signing -1 exp, verification - 2 exps
- No attacks thus far

## DSS (contd)

$p$ – 512 – bit prime
$q$ – 160 – bit prime, $(p-1)\%q = 0$
$b$ – base, $b^q \equiv 1 \bmod p \quad (b = d^{(p-1)/q})$
$x$ – private exponent
$y$ – public residue; $y \equiv b^x \bmod p$
$P = Z_p^*, A = Z_q \times Z_q$
publics: $p, q, b, y$ secrets: $x$

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

*Verifying*:
$(b^{mc^{-1}} k^{kc^{-1}} \bmod p) \bmod q = k$ ???
notice that:
$b^{mc^{-1}} k^{kc^{-1}} = b^{mr/(m+xb^r)} (b^x)^{(b^r r/(m+xb^r))}$
$= b^{(mr+xb^r r)/(m+xb^r)} = b^r$



## DSS vs El Gamal

$p$ – 512 – bit prime
$q$ – 160 – bit prime, $(p-1)\%q = 0$
$b$ – base, $b^q \equiv 1 \bmod p \quad (b = d^{(p-1)/q})$
$x$ – private exponent
$y$ – public residue; $y \equiv b^x \bmod p$
$P = Z_p^*, A = Z_q \times Z_q$
publics: $p, q, b, y$ secrets: $x$

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

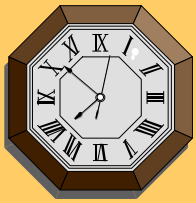
*Verifying*:
$(b^{mc^{-1}} k^{kc^{-1}} \bmod p) \bmod q = k$ ???

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

## The importance of Having a Good Time



- Often, signatures must be time-stamped
- How can Alice trust Bob's time?
- Can Alice trust her own time?
- Consider compromise scenario (Alice looses her key…)
- How to prove time-of-signing:
  - Not before time/date X
  - Not after time/date Y

- Using a Trusted Time-stamping Service



## Key Distribution

- How can two (or more) entities achieve secure communication without a shared key?
- Can they do so efficiently?
- How can a key be established?
- Is there a TTP/TA/KDC/AS/KDS/etc, etc.?
- Who chooses it (key)?
- Who contributes to it?
- How strong/long is it?
- How long can it be used?
- What can it be used for?
- Is there a separate SUPER-SECURE channel for KD?
- Can keys be pre-distributed?
- What about attacks?

## Key pre-distribution: Diffie-Hellman

*System – wide parameters:*
*p – large prime, a – generator in* $Z_p$
*Alice:*
*secret* – $x_a$, *public* – $y_a = a^{x_a} \bmod p$
*Bob:*
*secret* – $x_b$, *public* – $y_b = a^{x_b} \bmod p$

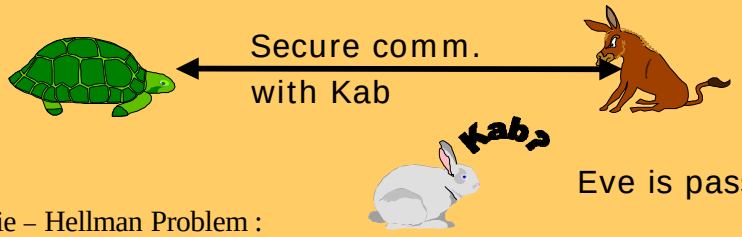
$CERT_u = \{ID_u, y_u, validity, ID_{CA}\}^{CA}$

$Alice: CERT_{bob}$
$Bob: CERT_{alice}$

$K_{ab} = a^{x_a x_b} \bmod p$



## Key pre-distribution: Diffie-Hellman




Eve is passive

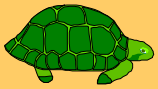
Diffie – Hellman Problem:
*p – large prime, a – generator in* $Z_p$
*Given:*
$y_a = a^{x_a} \bmod p$ *and* $y_b = a^{x_b} \bmod p$
*FIND:* $a^{x_a x_b} \bmod p$

Discrete Log Problem:
*Given:*
$y_a = a^{x_a} \bmod p$
*FIND:* $x_a$

Decision DH Problem:
*p – large prime, a – generator*
*Given:*
$y_a = a^{x_a} \bmod p$, $y_b = a^{x_b} \bmod p$
*Distinguish:*
$K_{ab} = a^{x_a x_b} \bmod p$
*from a random number!*

## Key exchange: Diffie-Hellman

Choose random v



$y_a = a^v \bmod p$

$y_b = a^w \bmod p$

Compute $K_{ab} = (y_b)^u \bmod p$

Choose random w, Compute $K_{ba} = (y_a)^w \bmod p$

Secure comm. with Kab

Eve is not passive!

The "rabbit-in-the-middle-attack"



## Authenticated Key Exchange (STS)

Choose random v

$y_a = a^v \bmod p$

Choose random w, Compute $K_{ba} = (y_a)^w \bmod p$

$y_b = a^w \bmod p$

$SIG_{bob} = \{y_b, y_a\}^{Bob}$

$CERT_{bob}, y_b, SIG_{bob}$

Compute $K_{ab} = (y_b)^u \bmod p$

$SIG_{alice} = \{y_a, y_b\}^{alice}$

$CERT_{alice}, SIG_{alice}$

Eve is not happy!