CIS 6930/4930 Computer and Network Security

Topic 5.2 Public Key Cryptography

Diffie-Hellman Key Exchange

Diffie-Hellman Protocol

- For negotiating a shared secret key using only public communication
- Does not provide authentication of communicating parties
- What's involved?
  - p is a large prime number (about 512 bits)
  - g is a primitive root of p, and $g < p$
  - p and g are publicly known

D-H Key Exchange Protocol

Alice	Bob
Publishes g and p	Reads g and p
Picks random number S_A *(and keeps private)*	Picks random number S_B *(and keeps private)*
Computes $T_A = g^{S_A} \bmod p$	Computes $T_B = g^{S_B} \bmod p$
Sends T_A to Bob,	Sends T_B to Alice,
Computes $T_B{}^{S_A} \bmod p$ =	Computes $T_A{}^{S_B} \bmod p$

Key Exchange (Cont'd)

Alice and Bob have now both computed the same secret $g^{S_A S_B}$ mod p, which can then be used as the shared secret key K

S_A is the discrete logarithm of g^{S_A} mod p and

S_B is the discrete logarithm of g^{S_B} mod p

D-H Example

- Let p = 353, g = 3
- Let random numbers be S_A = 97, S_B = 233
- Alice computes T_A = ___ mod __ = 40 = g^{S_A} mod p
- Bob computes T_B = ___ mod ___ = 248 = g^{S_B} mod p
- They exchange T_A and T_B
- Alice computes K = __ mod __ = **160** = $T_B^{S_A}$ mod p
- Bob computes K = __ mod ___ = **160** = $T_A^{S_B}$ mod p

D-H Example

- Let p = 353, g = 3
- Let random numbers be S_A = 97, S_B = 233
- Alice computes $T_A = 3^{97}$ mod 353 = 40 = g^{S_A} mod p
- Bob computes $T_B = 3^{233}$ mod 353 = 248 = g^{S_B} mod p
- They exchange T_A and T_B
- Alice computes $K = 248^{97}$ mod 353 = **160** = $T_B^{S_A}$ mod p
- Bob computes $K = 40^{233}$ mod 353 = **160** = $T_A^{S_B}$ mod p

Why is This Secure?

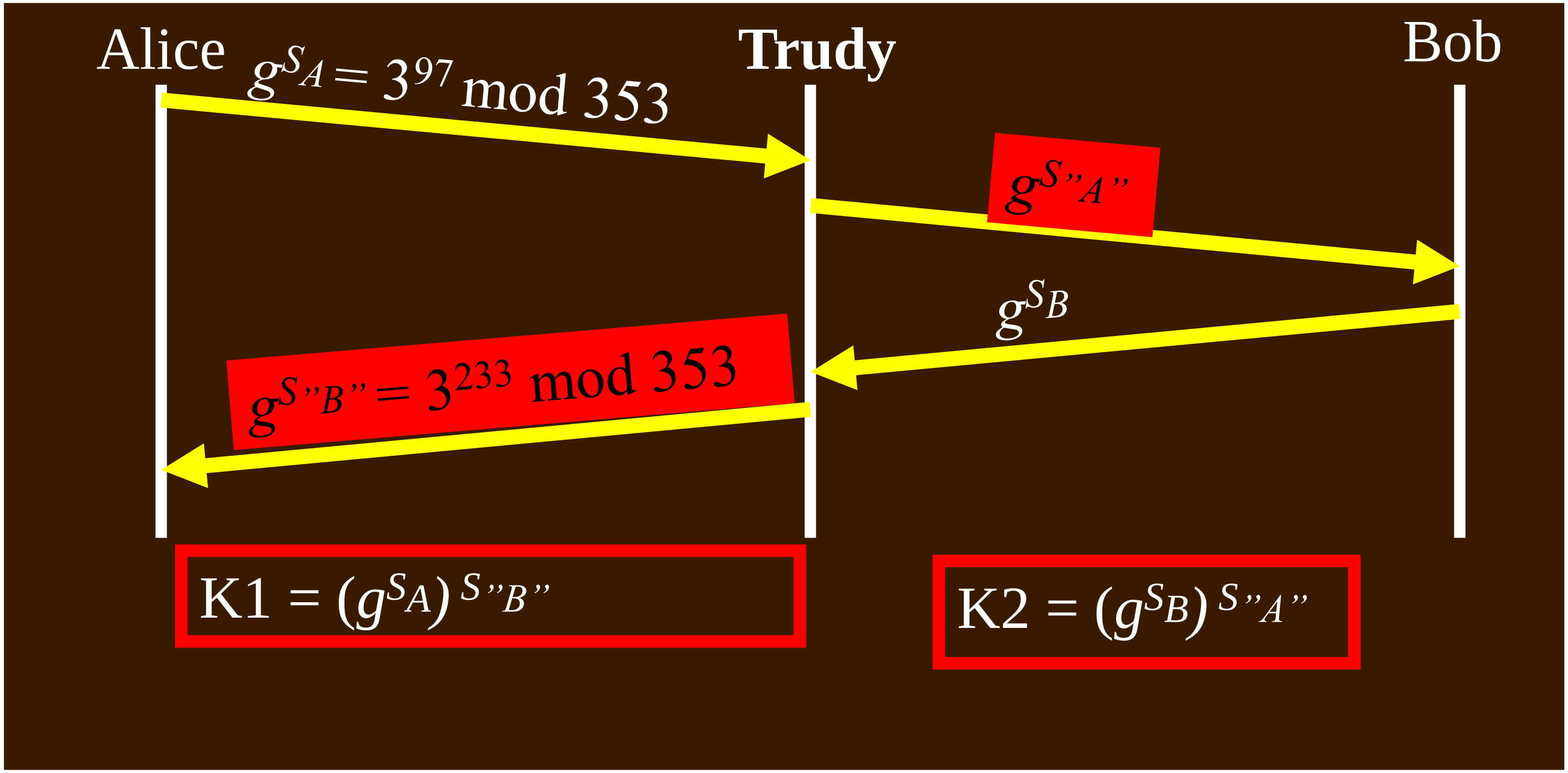
- Discrete log problem is hard:
  - given a^x mod b, a, and b, it is computationally infeasible to compute x

D-H Limitations

- Expensive exponential operation is required
  - possible timing attacks??
- Algorithm is useful for key negotiation only
  - i.e., not for public key encryption/verification
- Not for user authentication
  - In fact, you can negotiate a key with a complete stranger!

Man-In-The-Middle Attack

- Trudy impersonates as Alice to Bob, and also impersonates as Bob to Alice



Man-In-The-Middle Attack (Cont’d)

- Now, Alice thinks K1 is the shared key, and Bob thinks K2 is the shared key
- Trudy intercepts messages from Alice to Bob, and
  - decrypts (using K1), substitutes her own message, and encrypts for Bob (using K2)
  - likewise, intercepts and substitutes messages from Bob to Alice
- Solution???

Authenticating D-H Messages

- That is, you know who you're negotiating with, and that the messages haven't been modified
- Requires that communicating parties already share something
- Then use shared information to enable authentication

Using D-H in "Phone Book" Mode

1. Alice and Bob each chooses a secret number, generate T_A and T_B
2. Alice and Bob *publish* T_A, $T_{B,}$ i.e., Alice can get Bob's T_B at any time, Bob can get Alice's T_A at any time
3. Alice and Bob can then generate a shared key without communicating
   – but, they must be using the same *p* and *g*

- Essential requirement: reliability of the published values (no one can substitute false values)

Encryption Using D-H?

- How to do key establishment + message encryption in one step
- Everyone computes and publishes their own individual $<p_i, g_i, T_i>$, where $T_i = g_i^{S_i} \bmod p_i$
- For Alice to communicate with Bob…
  1. Alice picks a random secret S_A
  2. Alice computes $g_B^{S_A} \bmod p_B$
  3. Alice uses $K_{AB} = T_B^{S_A} \bmod p_B$ to encrypt the message
  4. Alice sends encrypted message along with (unencrypted) $g_B^{S_A} \bmod p_B$

Encryption (Cont'd)

- For Bob to decipher the encrypted message from Alice
  1. Bob computes $K_{AB} = (g_B{}^{s_A})^{s_B} \bmod p_B$
  2. Bob decrypts message using K_{AB}

Example

- Bob publishes $\langle p_B, g_B, T_B \rangle$ = <401, 5, 51> and keeps secret $S_B = 58$
- Steps
  1. Alice picks a random secret $S_A = 17$
  2. Alice computes $g_B^{S_A} \bmod p_B$ = ___ mod ___ = 173
  3. Alice uses $K_{AB} = T_B^{S_A} \bmod p_B$ =
     ___ mod ___ = **360** to encrypt message M
  4. Alice sends encrypted message along with (unencrypted) $g_B^{S_A} \bmod p_B = 173$
  5. Bob computes $K_{AB} = (g_B^{S_A})^{S_B} \bmod p_B$ =
     ___ mod ___ = **360**
  6. Bob decrypts message M using K_{AB}

Example

- Bob publishes $<p_B, g_B, T_B>$ = <401, 5, 51> and keeps secret $S_B = 58$
- Steps
  1. Alice picks a random secret $S_A = 17$
  2. Alice computes $g_B^{S_A} \bmod p_B = 5^{17} \bmod 401 = 173$
  3. Alice uses $K_{AB} = T_B^{S_A} \bmod p_B =$ $51^{17} \bmod 401 =$ **360** to encrypt message M
  4. Alice sends encrypted message along with (unencrypted) $g_B^{S_A} \bmod p_B = 173$
  5. Bob computes $K_{AB} = (g_B^{S_A})^{S_B} \bmod p_B =$ $173^{58} \bmod 401 =$ **360**
  6. Bob decrypts message M using K_{AB}

Picking *g* and *p*

- Advisable to change *g* and *p* periodically
  - the longer they are used, the more info available to an attacker
- Advisable not to use same *g* and *p* for everybody

Digital Signature Standard (DSS)

Digital Signature Standard (DSS)

- Useful only for digital signing (no encryption or key exchange)
- Components
  - SHA-1 to generate a hash value (some other hash functions also allowed now)
  - Digital Signature Algorithm (DSA) to generate the digital signature from this hash value
- Designed to be fast for the signer rather than verifier

Digital Signature Algorithm (DSA)

1. Announce public parameters used for signing
   - pick p (a prime with >= 1024 bits)

     ex.: $p = 103$
   - pick q (a 160 bit prime) such that $q \mid (p-1)$

     ex.: $q = 17$ (divides 103 - 1)
   - choose $g \equiv h^{(p-1)/q} \bmod p$, where $1 < h < (p-1)$, such that $g > 1$

     ex.: if $h = 2$, $g = 2^6 \bmod 103 = 64$
   - note: g is of order q mod p

     ex.: powers of 64 mod 103 =
     64 79 9 61 93 81 34 13 8 100 14 72 76 23 30 66 1

     17 values

DSA (Cont'd)

2. User Alice generates a long-term private key x
   - random integer with $0 < x < q$

ex.: $x = 13$

3. Alice generates a long-term public key y
   - $y = g^x \bmod p$

ex.: $y = 64^{13} \bmod 103 = 76$

DSA (Cont'd)

4. Alice randomly picks a per message secret number k such that $0 < k < q$, and generates $k^{-1} \bmod q$

ex.: k = 12, $12^{-1} \bmod 17 = 10$

5. Signing message M

ex.: H(M) = 75

– $r = (g^k \bmod p) \bmod q$

ex.: $r = (64^{12} \bmod 103) \bmod 17 = 4$

– $s = [k^{-1} * (H(M) + x * r)] \bmod q$

ex.: s = [10 * (75 + 13*4)] mod 17 = 12

– transmitted info = M, r, s

ex.: M, 4, 12

Verifying a DSA Signature

- Known : g, p, q, y

ex.: p = 103, q = 17, g = 64, y = 76, H(M) = 75

- Received from signer: M, r, s

ex.: M, **4**, 12

1. $w = (s)^{-1} \bmod q$

ex.: $w = 12^{-1} \bmod 17 = 10$

2. $u_1 = [H(M) * w] \bmod q$

ex.: $u_1 = 75*10 \bmod 17 = 2$

3. $u_2 = (r*w) \bmod q$

ex.: $u_2 = 4*10 \bmod 17 = 6$

4. $v = [(g^{u1}*y^{u2}) \bmod p] \bmod q$

ex.: $v = [(64^2 * 76^6) \bmod 103] \bmod 17 =$ **4**

5. If $v = r$, then the signature is verified

Why Does it Work?

- Correct? The signer computes
- $s = [k^{-1} * (H(m) + x*r)] \bmod q$
- so $k = [s^{-1} * (H(m) + x*r)] \bmod q$

$= [(H(m) + x*r)*s^{-1}] \bmod q$

$= \{[H(m) + x*r] \bmod q\} *(s^{-1} \bmod q)$

$= \{[H(m) + x*r] \bmod q\} *w$

$= [H(m)*w \bmod q] + (x*r*w \bmod q)$

Why Does it Work? (Cont'd)

- $g^k = g^{[H(m)*w] \bmod q} * g^{(x*r*w) \bmod q}$

  $= g^{u1} * g^{(x \bmod q)*(r*w \bmod q)}$

  $= g^{u1} * g^{x*u2}$ $(x < q)$

$r = (g^k \bmod p) \bmod q = [(g^{u1} * g^{x*u2}) \bmod p] \bmod q$

$= [(g^{u1} \bmod p) * (g^{x*u2} \bmod p)] \bmod q$

$= [(g^{u1} \bmod p) * (g^x \bmod p)^{u2}] \bmod q$

$= [(g^{u1} \bmod p) * y^{u2}] \bmod q$

$= [(g^{u1} * y^{u2}) \bmod p)] \bmod q = v$

Is it Secure?

- Given *y*, it is difficult to compute *x*
  - *x* is the discrete log of *y* to the base *g*, mod *p*
- Likewise, given *r*, it is difficult to compute *k*
- Cannot forge a signature without *x*
- Signatures are not repeated (only used once per message) and cannot be replayed

Assessment of DSA

- Slower to verify than RSA, but faster signing than RSA
- Key lengths of 2048 bits and greater are also allowed