CS 4173/5173
COMPUTER SECURITY

Diffie-Hellman Key Exchange and Digital Signature Standard (DSS)

PRIMITIVE ROOT

- If we fix n, and change a in $a^m \bmod n$ for $m = 1, 2, 3, 4, \ldots$
- Example: n=19. Then, a=2 is called a primitive root

a	a^2	a^3	a^4	a^5	a^6	a^7	a^8	a^9	a^{10}	a^{11}	a^{12}	a^{13}	a^{14}	a^{15}	a^{16}	a^{17}	a^{18}	← order
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	4	8	16	13	7	14	9	18	17	15	11	3	6	12	5	10	1	18
4	16	7	9	17	11	6	5	1	4	16	7	9	17	11	6	5	1	9
7	11	1	7	11	1	7	11	1	7	11	1	7	11	1	7	11	1	3
8	7	18	11	12	1	8	7	18	11	12	1	8	7	18	11	12	1	6
9	5	7	6	16	11	4	17	1	9	5	7	6	16	11	4	17	1	9
18	1	18	1	18	1	18	1	18	1	18	1	18	1	18	1	18	1	2

LOGARITHMS

- Example: a=2 is a primitive root of p=19.
  it is straightforward to get $b = a^i \bmod p$

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
b	2	4	8	16	13	7	14	9	18	17	15	11	3	6	12	5	10

- How to get the discrete logarithm i from b; e.g., $\text{ind}_{2,19}(9)$

DIFFICULTIES IN MODULAR ARITHMETIC

- Factoring large numbers
- Computing Totient function
  - Need factoring first
- Obtaining primitive roots
- Discrete logarithm

- Public key cryptography design should leverage all these difficulties!

REVIEW RSA

Encryption: $c = m^e \bmod n, m < n$

Decryption: $m = c^d \bmod n$

e – public key

d – private key

Signing: $s = m^d \bmod n, m < n$

Verification: $m = s^e \bmod n$

5

SECURITY IN RSA

- Key generations: (*e, d, n, p, q*)
  - Given e, why it is hard to get d?

- Encryption: $\boldsymbol{c = m^e \bmod n}$
  - Given *C,* why it is hard to get *m?*

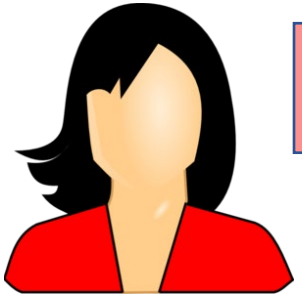
- Signing: $\boldsymbol{s = m^d \bmod n}$
  - Given *s,* why it is hard to get *d?*

RSA

- Encryption:
  - Encrypt with public key, decrypt with private key
- Authentication:
  - Sign with private key, verify with public key
- Key Negotiation:
  - Digital envelope
  - RSA based negotiation
  - Alice and Bob must know each other's public key

USING RSA FOR KEY NEGOTIATION

Alice



Public key: Ka,p
Private key: Ka,i

Generate random number R1

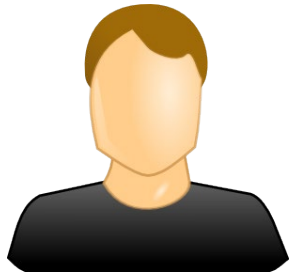
Send R1 encrypted using Kb,p

1) Get R1,
2) Generate random number R2,
3) get key as K = H(*R1*⊕*R2*),

Send R2 encrypted using Ka,p

1) get R2
2) get symmetric key as K = H(*R1*⊕*R2*),

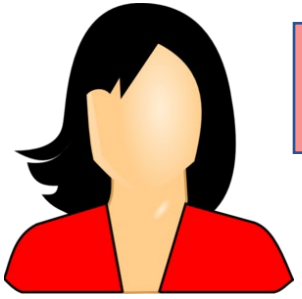
Bob



Public key: Kb,p
Private key: Kb,i

DIGITAL ENVELOPE

Alice



Knows the public key: Kp

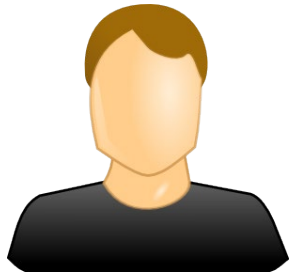
Generate a session key Ks

Encrypt message M using Ks
Encrypted Ks using Kp

Send a response using Ks

All remaining communication is based on Ks

Bob



Public key: Kp
Private key: Ki

1. Use Ki to decrypt Ks
2. Use Ks to decrypt M

OTHER SCENARIO

- What if Alice and Bob do not have their public/private keys, but want to communicate secretly?

SOLUTION

- Alice and Bob
  - generate a pair of public and private key individually
    - Public key <*e*, *n*> in RSA
    - Private key <*d*, *n*> in RSA
  - Send their public keys <*e*, *n*> to each other

CS 4173/5173
COMPUTER SECURITY

Diffie-Hellman Key Exchange

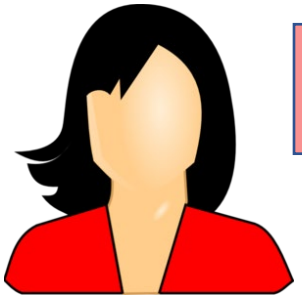
DIFFIE-HELLMAN PROTOCOL

- 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 (at least 1024 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 $T_A^{S_B} \bmod p$

USING D-H FOR KEY NEGOTIATION

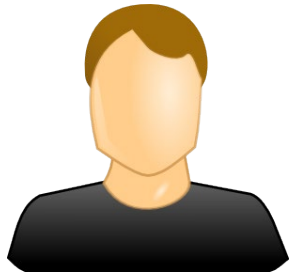


Alice

No public/private key

Public knowledge g, p

Bob



No public/private key

Generate random number S_A

Generate random number S_B

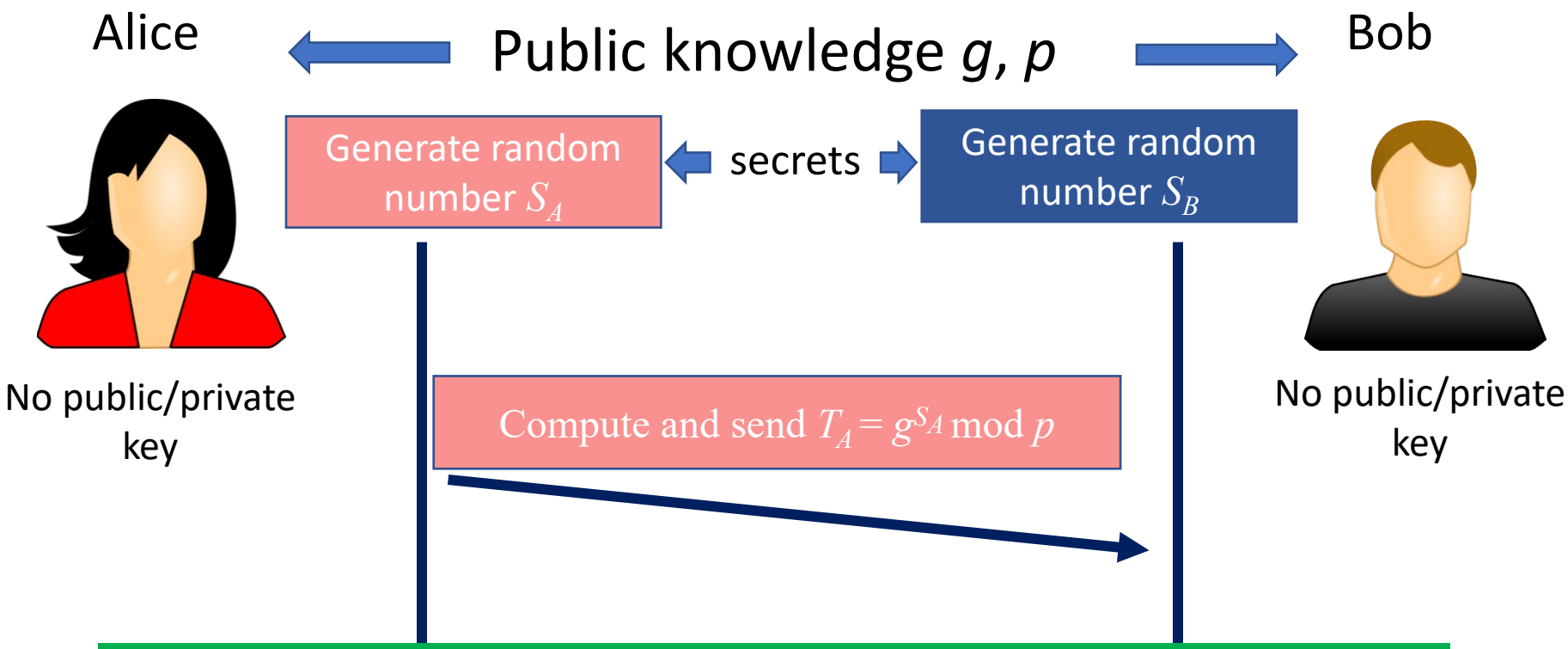
Compute and send $T_A = g^{S_A} \bmod p$

Compute and send $T_B = g^{S_B} \bmod p$

Compute $T_B^{S_A} \bmod p = \underline{\mathbf{g^{S_A S_B} \bmod p}}$

Compute $T_A^{S_B} \bmod p = \underline{\mathbf{g^{S_A S_B} \bmod p}}$

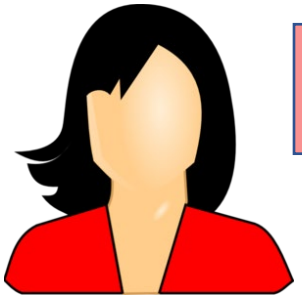
SECURITY ANALYSIS



Can an attacker get S_A from T_A, why??

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

SECURITY ANALYSIS II

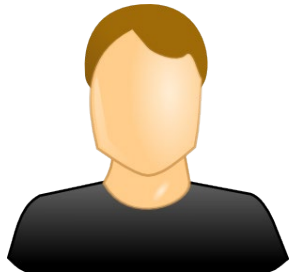


Alice

No public/private key

Public knowledge g, p

Bob



No public/private key

Generate random number S_A

Generate random number S_B

Compute and send $T_A = g^{S_A} \bmod p$

Compute and send $T_B = g^{S_B} \bmod p$

Compute $T_B^{S_A} \bmod p = g^{S_A S_B} \bmod p$

Compute $T_A^{S_B} \bmod p = g^{S_A S_B} \bmod p$

Can an attacker get $g^{S_A S_B} \bmod p$ from T_A and T_B?

SUMMARY

Alice and Bob both compute the same secret $g^{S_A S_B} \bmod p$, which can then be used as the shared secret key K

- Difficulties for the attacker:
- S_A is the discrete logarithm of $g^{S_A} \bmod p$ and
- S_B is the discrete logarithm of $g^{S_B} \bmod p$
- Compute discrete logarithm is computationally difficult!

D-H EXAMPLE

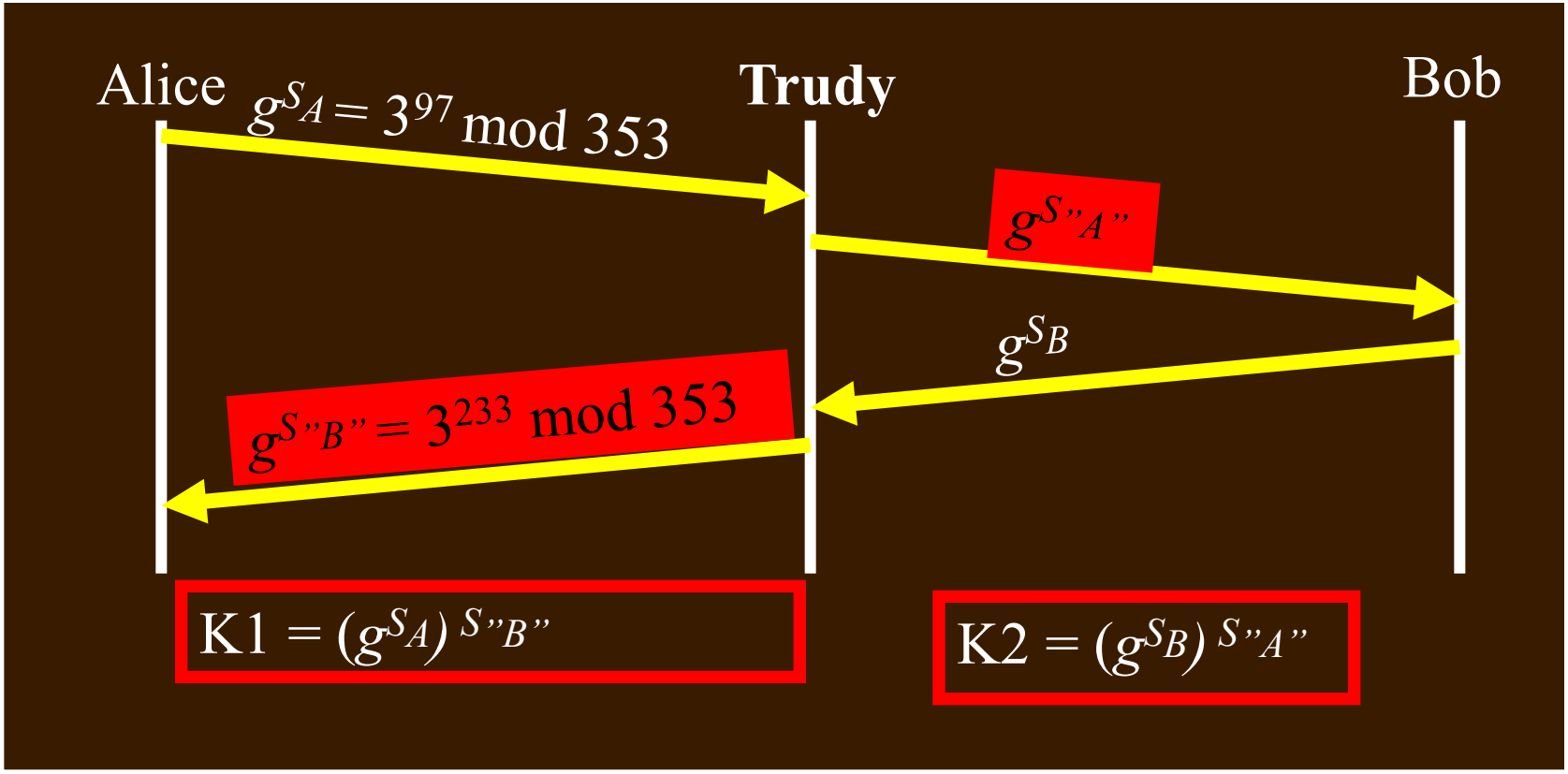
- Let $p = 353$, $g = 3$
- Let random numbers be $S_A = 97$, $S_B = 233$
- Alice computes $T_A = g^{S_A} \bmod p = 3^{97} \bmod 353 = 40$
- Bob computes $T_B = g^{S_B} \bmod p = 3^{233} \bmod 353 = 248$
- They exchange T_A and T_B
- Alice computes $K = T_B^{S_A} \bmod p = 248^{97} \bmod 353 = \mathbf{160}$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

D-H LIMITATIONS

- Expensive exponential operation is required
- 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!
- Vulnerable to man-in-the-middle attack

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???

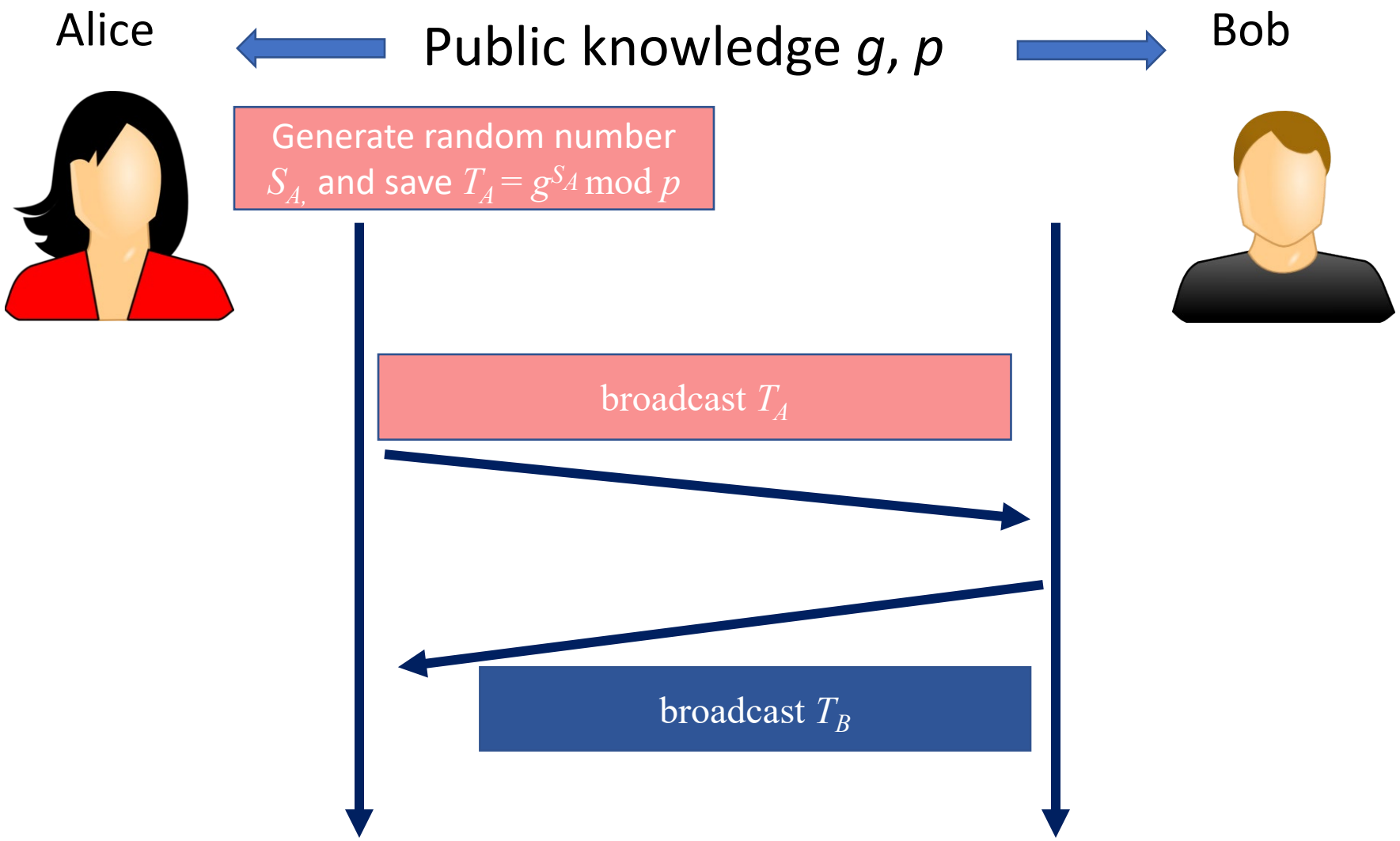
AUTHENTICATING D-H MESSAGES

- 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
  - public key information ➔ RSA authentication
  - some common secret K, ➔ HMAC, CBC-MAC

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*

D-H IN "PHONE BOOK" MODE



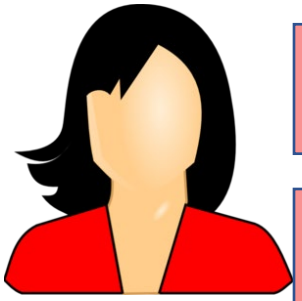
ENCRYPTION USING D-H?

- 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 number 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) $g_B^{S_A} \bmod p_B$

D-H WITH ENCRYPTION

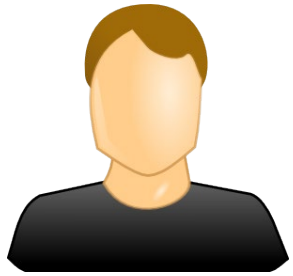
Alice sends a secure message to Bob

Bob



Generate random number S_A, compute $g_B^{S_A} \bmod p_B$

Use $K_{AB} = T_B^{S_A}$ to encrypt the message M



$<p_a, g_a, T_a>$
Known to everyone

$<p_b, g_b, T_b>$
Known to everyone

Send K_{AB} (M) along with $g_B^{S_A} \bmod p_B$

Compute the key K_{AB}
$K_{AB} = (g_B^{S_A})^{S_B} \bmod p_B$
Then, decrypt the message

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

POPULAR VERSION

- ECDHE
  - Elliptic Curve Diffie-Hellman key Exchange
  - Widely used on the Internet

PUBLIC KEY VS. SYMMETRIC KEY

Symmetric key	Public key
Two parties MUST trust each other	Two parties DO NOT need to trust each other
Both share same key (or one key is computable from the other)	Two separate keys: a public and a private key
Typically faster	Typically slower
Examples: DES, RC5, AES, …	Examples: RSA, DSA, ECC…

Best Practice: use public key cryptography to negotiate a symmetric session key

CS 4173/5173
COMPUTER SECURITY

Digital Signature Standard (DSS)

DIGITAL SIGNATURE STANDARD (DSS)

- Useful only for digital signing
  - no encryption
  - no key exchange
- Components
  - SHA-1 to generate a hash value (some other hash functions also allowed now)
    - **The practical cracking of SHA-1 was publicly announced in Feb 2017.**
  - 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)
   - pick q (a 160 bit prime) such that $q \mid (p-1)$
   - choose $g \equiv h^{(p-1)/q} \bmod p$, where $1 < h < (p-1)$, such that $g > 1$
   - note: g is of order q mod p

ex.: $p = 103$

ex.: $q = 17$ (divides 102)

ex.: if $h = 2$, $g = 2^6 \bmod 103 = 64$

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 $k^{-1} \bmod q$

5. Signing message M

   - $r = (g^k \bmod p) \bmod q$
   - $s = [k^{-1}*(H(M)+x*r)] \bmod q$
   - transmitted info = M, r, s

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

ex.: H(M) = 75

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

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

ex.: M, 4, 12

VERIFYING A DSA SIGNATURE

- Known : g, p, q, *y*
- Received from signer: *M*, *r*, *s*

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

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

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 $k \equiv H(m)*s^{-1} + x*r*s^{-1}$
- $\equiv H(m)*w + x*r*w \bmod q$
- And :
- $g^k \equiv g^{H(m)w} * g^{xrw}$
- $\equiv g^{H(m)w} * y^{rw}$
- $\equiv g^{u1} * y^{u2} \bmod p$, and
- $r = (g^k \bmod p) \bmod q = (g^{u1}*y^{u2} \bmod p) \bmod q = v$

$u_1 = [H(M) * w] \bmod q$
$u_2 = (r*w) \bmod q$

ASSESSMENT OF DSA

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

PUBLIC KEY ALGORITHMS

- Public key algorithms covered in this class, and their applications

System	Encryption / Decryption?	Digital Signatures?	Key Exchange?
RSA	Yes	Yes	Yes
Diffie-Hellman			Yes
DSA		Yes	