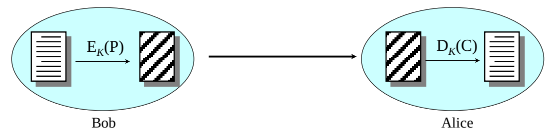
Secure Communication

Symmetric cryptography

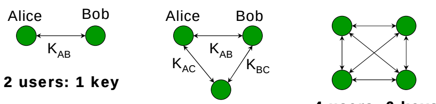
- Both parties must agree on a secret key, *K*
- message is encrypted, sent, decrypted at other side



- Key distribution must be secret
  - otherwise messages can be decrypted
  - users can be impersonated

Key explosion

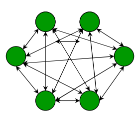
- Each pair of users needs a separate key for secure communication



2 users: 1 key

3 users: 3 keys

4 users: 6 keys



6 users: 15 keys

100 users: 4950 keys

1000 users: 399500 keys

n users: $\frac{n(n-1)}{2}$ keys

Key distribution

Secure key distribution is the biggest problem with symmetric cryptography

Key exchange

How can you communicate securely with someone you've never met?

- Whit Diffie - idea for a *public key* algorithm
- goal: sender can create two sets of keys: one public and one private
- sender sends data encrypted with the receiver's public key
- receiver can decrypt data with her private key
- challenge: can this be done securely?
  - Knowledge of public key should not allow derivation of private key

Diffie-Hellman exponential key exchange

Key distribution algorithm

- first algorithm to use public/private keys
- *not* public key encryption
- based on difficulty of computing discrete logarithms in a finite field compared with ease of calculating exponentiation

allows us to negotiate a secret **session key** without fear of eavesdroppers

Diffie-Hellman exponential key exchange

- All arithmetic performed in field of integers modulo some large number
- Both parties agree on
  - a **large prime number *p***
  - and a number $\alpha < p$
- Each party generates a public/private key pair

private key for user *i*: X_i

public key for user *i*: $Y_i = \alpha^{X_i} \bmod p$

Diffie-Hellman exponential key exchange

- Alice has secret key X_A
- Alice has public key Y_A
- Alice computes $K = Y_B^{X_A} \bmod p$
- Bob has secret key X_B
- Bob has public key Y_B

K = (Bob's public key) $^{\textit{(Alice's private key)}}$ ***mod p***

Diffie-Hellman exponential key exchange

- Alice has secret key X_A
- Alice has public key Y_A
- Alice computes $K = Y_B^{X_A} \bmod p$
- Bob has secret key X_B
- Bob has public key Y_B
- Bob computes $K' = Y_A^{X_B} \bmod p$

K' = (Alice's public key) $^{\textit{(Bob's private key)}}$ ***mod p***

Diffie-Hellman exponential key exchange

- Alice has secret key X_A
- Alice has public key Y_A
- Alice computes $K = Y_B^{X_A} \bmod p$
- expanding:

$$K = Y_B^{X_A} \bmod p$$
$$= (\alpha^{X_B} \bmod p)^{X_A} \bmod p$$
$$= \alpha^{X_B X_A} \bmod p$$

- Bob has secret key X_B
- Bob has public key Y_B
- Bob computes $K' = Y_A^{X_B} \bmod p$
- expanding:

$$K' = Y_A^{X_B} \bmod p$$
$$= (\alpha^{X_A} \bmod p)^{X_B} \bmod p$$
$$= \alpha^{X_A X_B} \bmod p$$

K = K'

K is a **common key**, known *only* to Bob and Alice

Diffie-Hellman example

Suppose $p = 31667$, $\alpha = 7$

- Alice picks $X_A = 18$
- Alice's public key is: $Y_A = 7^{18} \bmod 31667 = 6780$
- $K = 22184^{18} \bmod 31667$ **K = 14265**
- Bob picks $X_B = 27$
- Bob's public key is: $Y_B = 7^{27} \bmod 31667 = 22184$
- $K = 6780^{27} \bmod 31667$ **K = 14265**

Key distribution problem is solved!

- User maintains private key
- Publishes public key in database ("phonebook")
- Communication begins with key exchange to establish a common key
- Common key can be used to encrypt a **session key**
  - increase difficulty of breaking common key by reducing the amount of data we encrypt with it
  - session key is valid *only* for one communication session

RSA

- Ron Rivest, Adi Shamir, Leonard Adleman created a true public key encryption algorithm in 1977
- Each user generates two keys
  - private key (kept secret)
  - public key
- Data encrypted with the private key can only be decrypted with the corresponding public key
  - *integrity, authentication*
- Data encrypted with the public key can only be decrypted with the corresponding private key
  - *secure communication*
- difficulty of algorithm based on the difficulty of factoring large numbers
  - keys are functions of a pair of large (~200 digits) prime numbers

RSA algorithm

Generate keys

- choose two random large prime numbers p, q
- Compute the product $n=pq$
- randomly choose the encryption key, e, such that
  e and $(p-1)(q-1)$ are relatively prime
- use the extended Euclidean algorithm to compute the decryption key, d:
  $ed = 1 \bmod ((p-1)(q-1))$
  $d = e^{-1} \bmod ((p-1)(q-1))$
- discard p, q

RSA algorithm

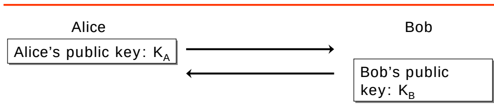
- encrypt
  - divide data into numerical blocks < n
  - encrypt each block:
    $c = m^e \bmod n$
- decrypt:
  $m = c^d \bmod n$

Communication with public key algorithms

Different keys for encrypting and decrypting

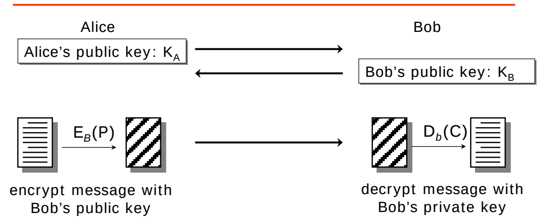
- no need to worry about key distribution

Communication with public key algorithms

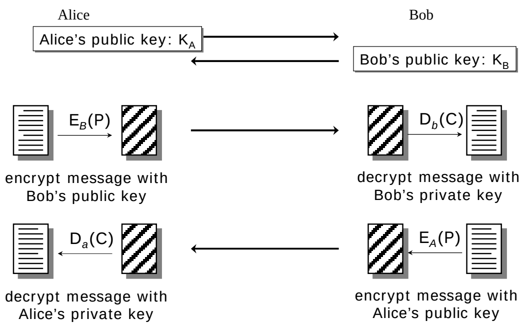


exchange public keys
(or look up in a directory/DB)

Communication with public key algorithms



Communication with public key algorithms



Public key woes

Public key cryptography is great but:

– RSA about 100 times slower than DES in software, 1000 times slower in HW

– Vulnerable to chosen plaintext attack
  - if you know the data is one of *n* messages, just encrypt each message with the recipient's public key and compare

– It's a good idea to reduce the amount of data encrypted with any given key
  - but generating RSA keys is computationally very time consuming

Hybrid cryptosystems

Use public key cryptography to encrypt a randomly generated symmetric key

session key

Communication with a hybrid cryptosystem

Alice — Bob

← Bob's public key: K_B

Get recipient's public key
(or fetch from directory/database)

Communication with a hybrid cryptosystem

Alice — Bob

← Bob's public key: K_B

Pick random session key, *K*

Encrypt session key
with Bob's public key

$E_B(K)$ → $K = D_b(E_B(K))$

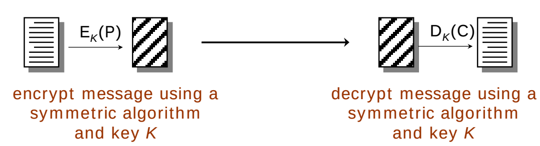
Bob decrypts *K* with
his private key

Communication with a hybrid cryptosystem

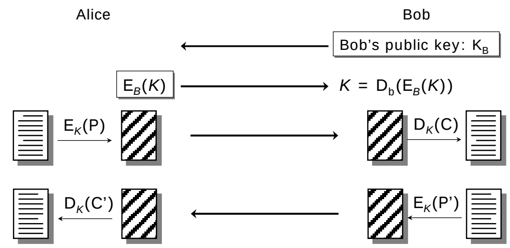
Alice — Bob

← Bob's public key: K_B

$E_B(K)$ → $K = D_b(E_B(K))$



Communication with a hybrid cryptosystem



decrypt message using a symmetric algorithm and key *K*

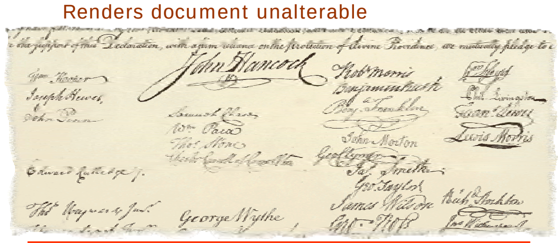
encrypt message using a symmetric algorithm and key *K*

Digital Signatures

signatures

We use signatures because a signature is:

- Authentic
- Unforgeable
- Not reusable
- Non repudiatable
- Renders document unalterable



~~Digital~~ signatures

~~We use signatures because a signature is~~

- ~~Authentic~~
- ~~Unforgeable~~
- ~~Not reusable~~
- ~~Non repudiatable~~
- ~~Renders document unalterable~~

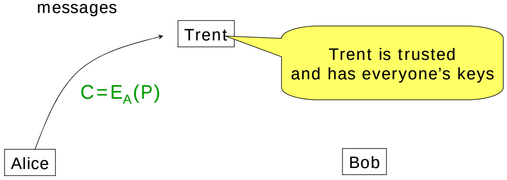
ALL UNTRUE!

Can we do better with **digital signatures?**

Digital signatures - arbitrated protocol

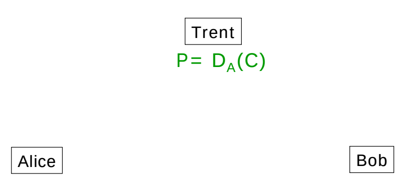
Arbitrated protocol using symmetric encryption

– turn to trusted third party (arbiter) to authenticate messages



Alice encrypts message for *herself* and sends it to Trent

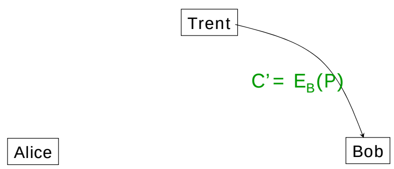
Digital signatures - arbitrated protocol



Trent receives Alice's message and decrypts it with Alice's key

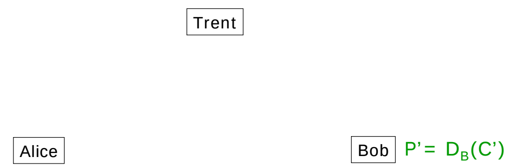
- this authenticates that it came from Alice
- he may choose to log a hash of the message to create a record of the transmission

Digital signatures - arbitrated protocol



Trent now encrypts the message for Bob and sends it to Bob

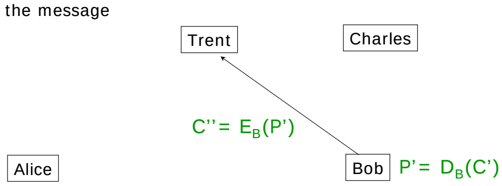
Digital signatures - arbitrated protocol



Bob receives the message and decrypts it
- it *must* have come from Trent since only Trent and Bob have Bob's key
- if the message says it's from Alice, it must be - we trust Trent

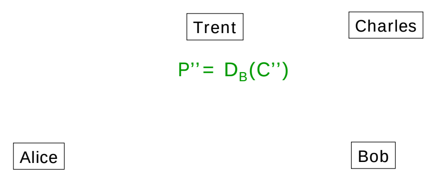
Digital signatures with multiple parties

Bob can forward the message to Charles in the same manner.
Trent can validate stored hash to ensure that Bob did not alter the message



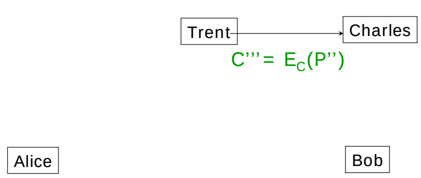
Bob encrypts message with his key and sends it to Trent

Digital signatures with multiple parties



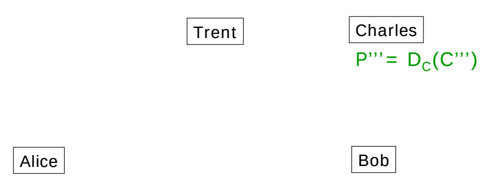
Trent decrypts the message
- knows it must be from Bob
- looks up ID to match original hash from Alice's message
- validates that the message has not been modified
- adds a "signed by Bob" indicator to the message

Digital signatures with multiple parties



Trent encrypts the new message for Charles

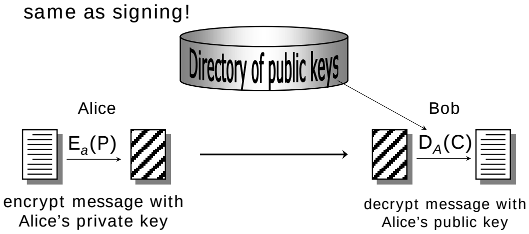
Digital signatures with multiple parties



Charles decrypts the message
- knows the message must have come from Trent
- trusts Trent's assertion that the message originated with Alice and was forwarded through Bob

Digital signatures - public key cryptography

Encrypting a message with a private key is the same as signing!



Digital signatures - public key cryptography

- What if Alice was sending Bob binary data?
  - Bob might have a hard time knowing whether the decryption was successful or not
- Public key encryption is considerably slower than symmetric encryption
  - what if the message is very large?
- What if we don't want to hide the message, yet want a valid signature?

Digital signatures - public key cryptography

- Create a **hash** of the message
- **Encrypt the hash** and send it with the message
- Validate the hash by decrypting it and comparing it with the hash of the received message

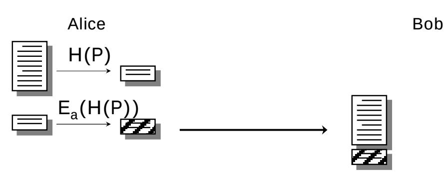
Digital signatures - public key cryptography

Alice generates a hash of the message

Digital signatures - public key cryptography

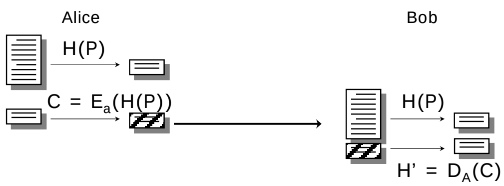
Alice encrypts the hash with her private key

Digital signatures - public key cryptography



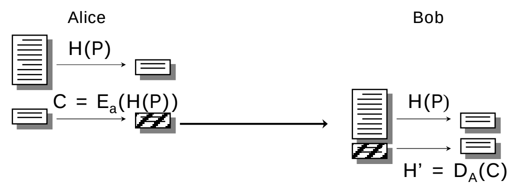
Alice sends Bob the message and the encrypted hash

Digital signatures - public key cryptography



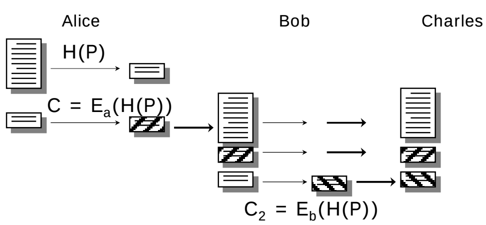
1. Bob decrypts the has using Alice's public key
2. Bob computes the hash of the message sent by Alice

Digital signatures - public key cryptography



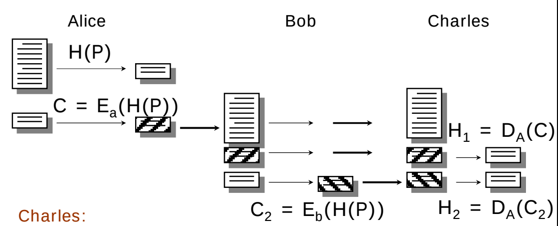
If the hashes match
- the encrypted hash *must* have been generated by Alice
- the signature is valid

Digital signatures - multiple signers



Bob generates a hash (same as Alice's) and encrypts it with his private key
- sends Charles:
  {message, Alice's encrypted hash, Bob's encrypted hash}

Digital signatures - multiple signers

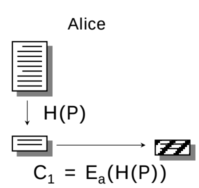


Charles:
- generates a hash of the message: H(P)
- decrypts Alice's encrypted hash with Alice's public key
  - validates Alice's signature
- decrypts Bob's encrypted hash with Bob's public key
  - validates Bob's signature

Secure and authenticated messaging

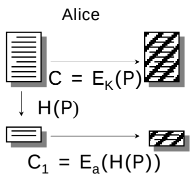
If we want secrecy of the message
- combine encryption with a digital signature
- use a session key: pick a random key, *K*, to encrypt the message with a symmetric algorithm
- encrypt *K* with the public key of each recipient
- for signing, encrypt the hash of the message with sender's private key

Secure and authenticated messaging



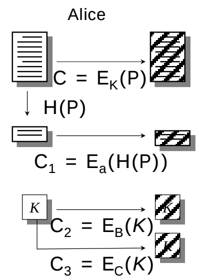
Alice generates a digital signature by encrypting the message digest with her private key.

Secure and authenticated messaging



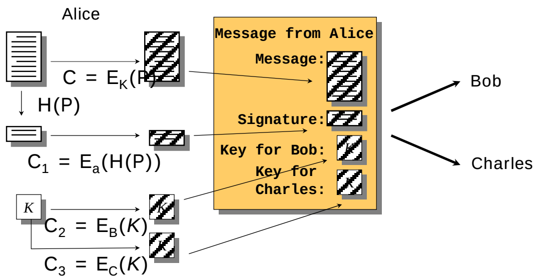
Alice picks a random key, *K*, and encrypts the message (P) with it using a symmetric algorithm.

Secure and authenticated messaging



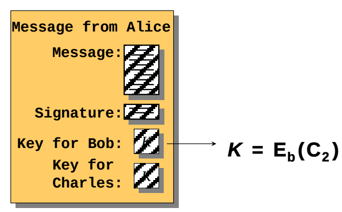
Alice encrypts the session key for each recipient of this message: Bob and Charles using their public keys.

Secure and authenticated messaging



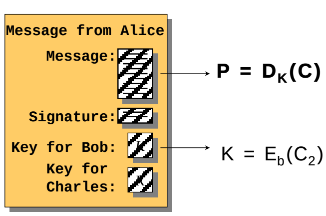
The aggregate message is sent to Bob and Charles

Secure and authenticated messaging



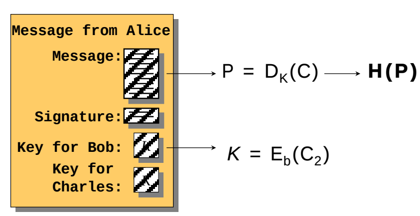
Bob receives the message:
- extracts key by decrypting it with his private key

Secure and authenticated messaging



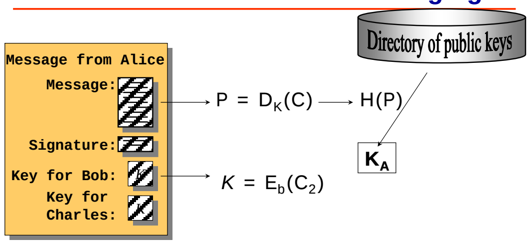
Bob decrypts the message using *K*

Secure and authenticated messaging



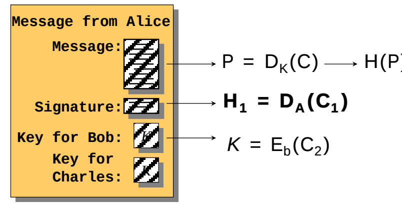
Bob computes the hash of the message

Secure and authenticated messaging



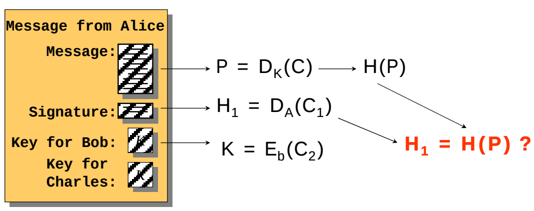
Bob looks up Alice's public key

Secure and authenticated messaging



Bob decrypts Alice's signature using Alice's public key

Secure and authenticated messaging



Bob validates Alice's signature

Cryptographic toolbox

- Symmetric encryption
- Public key encryption
- One-way hash functions
- Random number generators
  - Nonces, session keys
- Message authentication codes
  - Made from hash functions
- Digital signatures
  - Commonly: encrypted hash functions

The end