
Introduction to Applied Cryptography

Internet Security [1] VU

Paolo Milani Comparetti

pmilani@seclab.tuwien.ac.at

Clemens Kolbitsch

ck@seclab.tuwien.ac.at

Announcements

- Challenge 4 (Fakebook XSS) ended today
 - 30 people have completed it
 - First was JOHANNES "Demon Reflective" FERNER (6 hours)
- Challenge 5 will be announced today
 - programming challenge in Java
 - code "skeleton" is provided, to make it a bit faster for you
 - deadline in 2 weeks (26.05)
- Next lecture (and start of last challenge) in two weeks:
 - Buffer overflows (26.05)
 - Language security (02.06)
- Announcing written exam: 16.06
 - EI 10 Fritz Paschke HS, Elektrotechnisches Institutsgebäude.
 - 12:00 - 13:30 c.t. (that is, we start at 12:15, 75 minutes)

Introduction to Cryptography

Cryptography

(One) definition of cryptography

Mathematical techniques related to aspects of information security such as:

- confidentiality
 - keep content of information from all but authorized entities
- integrity
 - protect information from unauthorized alteration
- authentication
 - identification of data or communicating entities
- non-repudiation
 - prevent entity from denying previous commitments or actions

Some History

- Hebrew scholars
 - Atbash - mono-alphabetic substitution (reverse of Hebrew alphabet)
- Roman
 - Caesar cipher - mono-alphabetic substitution (letters are shifted by fixed offset)
- Alberti (1465)
 - poly-alphabetic substitution
- Enigma (World War 2)
 - complex poly-alphabetic substitution implemented with electro-mechanical device

Encryption

- Alphabet of definition A
 - finite set of symbols, e.g., binary alphabet $\{0,1\}$
- Message space M
 - set that contains strings from symbols of an alphabet A_1
 - elements of M are called **plaintext messages**
- Ciphertext space C
 - set that contains strings from symbols of an alphabet A_2
 - elements of C are called **ciphertext messages**
- Key space K
 - each element $e \in K$ uniquely determines bijective mapping $E_e: M \rightarrow C$ (called **encryption function**)
 - each element $d \in K$ uniquely determines bijective mapping $D_d: C \rightarrow M$ (called **decryption function**)

Cryptographic Primitives

- Unkeyed primitives
 - hash functions
 - (real) random sequences (get randomness from hardware)
 - ...
- Symmetric-key primitives
 - symmetric key ciphers
 - block ciphers
 - stream ciphers
 - signatures
 - pseudo-random sequences
 - ...
- Public-key primitives
 - public-key ciphers
 - signatures
 - ...

Cryptographic Primitives

- Algorithms are publicly known
 - only the selection of the keys remains secret
- Guarantee certain properties against a **computationally bounded** adversary
 - only brute force attacks are possible (i.e: try all possible keys)
 - if there is a faster attack, the primitive is broken
- Can be treated as a black box
 - compose multiple primitives to obtain a system with required properties
 - formal reasoning to prove the properties of the whole system, given the properties of each primitive

Security of Cryptographic Primitives

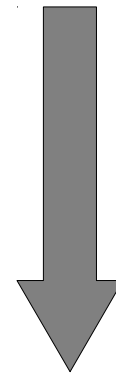
- Computational Security
 - secure against a **computationally bounded** adversary
 - attacker using currently known attack techniques
 - there may be better attacks!
- Provable security
 - secure against a **computationally bounded** adversary
 - mathematical proof exists that breaking the primitive is as hard as solving some known hard problem
 - e.g.: breaking RSA is **as hard** as factorization of large numbers
- Unconditional Security
 - secure against **any** adversary
 - also called perfect security
 - the ciphertext gives no information on the plaintext
 - e.g: one time pad

Cryptanalysis

- Study of techniques to defeat cryptographic primitives
 - frequency analysis
 - linear cryptanalysis
 - differential cryptanalysis
 - outside the scope of this class
- For most primitives, there is no mathematical proof that a cryptographic primitive cannot be broken
 - only that it cannot be broken with known cryptanalysis techniques
 - widely analyzed: many smart people have tried to break it
 - do not use yet untested algorithms!

Cryptanalysis

- Different models of attacker
- Ciphertext only
 - attacker only knows c
 - *e.g: attacker intercepts encrypted message*
- Known plaintext
 - attacker knows m and c
 - *e.g: attacker can obtain some m,c pairs*
- Chosen plaintext
 - attacker can choose m and obtain c
 - *e.g: attacker has an encryption black box*
- Assume the worst!
 - evaluate encryption systems against chosen plaintext attacks



More
powerful
attacker

Symmetric-Key Cryptography

Symmetric-key Encryption

- Consider an encryption scheme with key pair (e,d)
 - scheme is called a symmetric-key scheme if it is “relatively” easy to obtain d when e is know
 - often $e = d$
- Block cipher
 - break up plaintext into strings (blocks) of fixed length t
 - encrypt one block at a time
 - uses *substitution* and *transposition (permutation)* techniques
- Stream Cipher
 - special case of block cipher with block length $t = 1$
 - however, substitution technique can change for every block
 - key stream (e_1, e_2, e_3, \dots)

Block Ciphers

- *Simple (mono-alphabetic) substitution cipher*
 - for each symbol $m_k \in A$ of the plaintext, substitute another symbol $e(m_k)$ according to the permutation p defined by the key e
 - $E_e(m) = (p(m_1), p(m_2), p(m_3), \dots)$
- Example
 - p : map each letter to another letter based on a table (the table is the key)

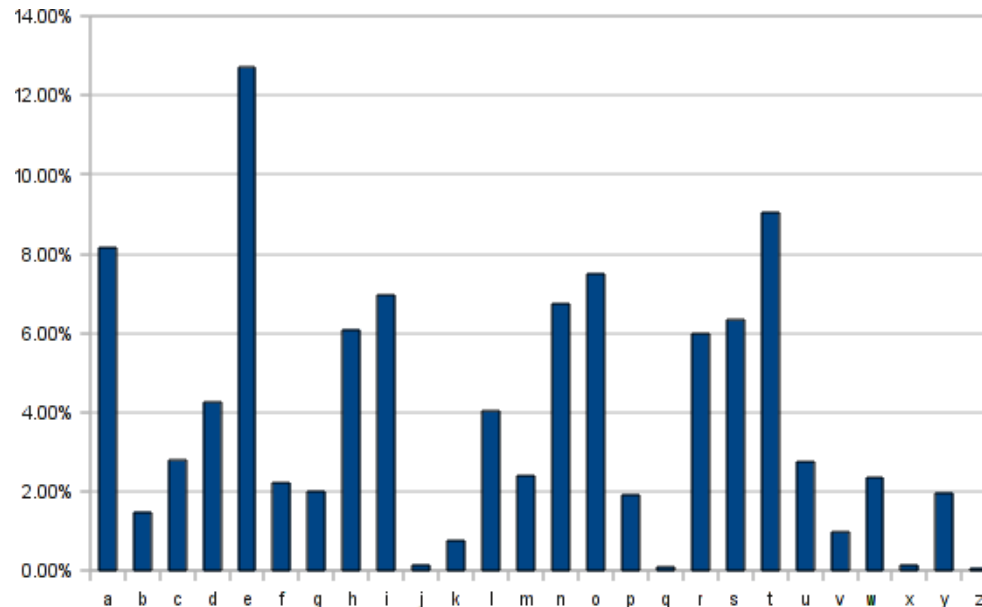
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
D	Z	F	G	H	I	Q	M	L	K	N	O	P	J	R	S	T	U	V	W	X	Y	Z	A	E	C

plaintext : THISC IPHER IS CER TAINL YNOTS ECURE

ciphertext: WMLVF LSMHU LVFHU WDLQO EJRWV HFXUH

Simple Substitution Cipher

- Easy to break by considering the frequency of symbols
- Some letters occur more often in the English language
 - most frequent symbol is likely to be **E!**



Block Ciphers

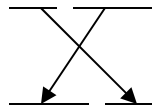
- *Poly-alphabetic substitution (Vigenere) cipher*
 - for each symbol $m_k \in A$ of the plaintext, substitute another symbol $e(m_k)$ according to one of several permutations p_i defined by the key e
 - for two permutations p_1 and p_2 : $E_e(m) = (p_1(m_1), p_2(m_2), p_1(m_3), \dots)$
- Example
 - using three permutations (mappings)
 - p_1 : map to letter that is three positions to the right
 - p_2 : map to letter that is seven positions to the right
 - p_3 : map to letter that is ten positions to the right

plaintext : THISC IPHER IS CER TAINL YNOTS ECURE
ciphertext: WOSVJ SSOOU PCFLB WHSQS IQVDV LMXYO

Block Ciphers

- *Transposition cipher*
 - for each block of symbols $(m_1, \dots, m_t) \in A$ of the plaintext, the key e defines a permutation on the set $\{1, \dots, t\} = \{p(1), p(2), \dots, p(t)\}$
 - $E_e(m) = (m_{p(1)}, m_{p(2)}, \dots, m_{p(t)})$
- Example
 - $t = 5$, permutation is $\{3, 4, 5, 1, 2\}$

plaintext : THISC IPHER ISCER TAINL YNOTS ECURE



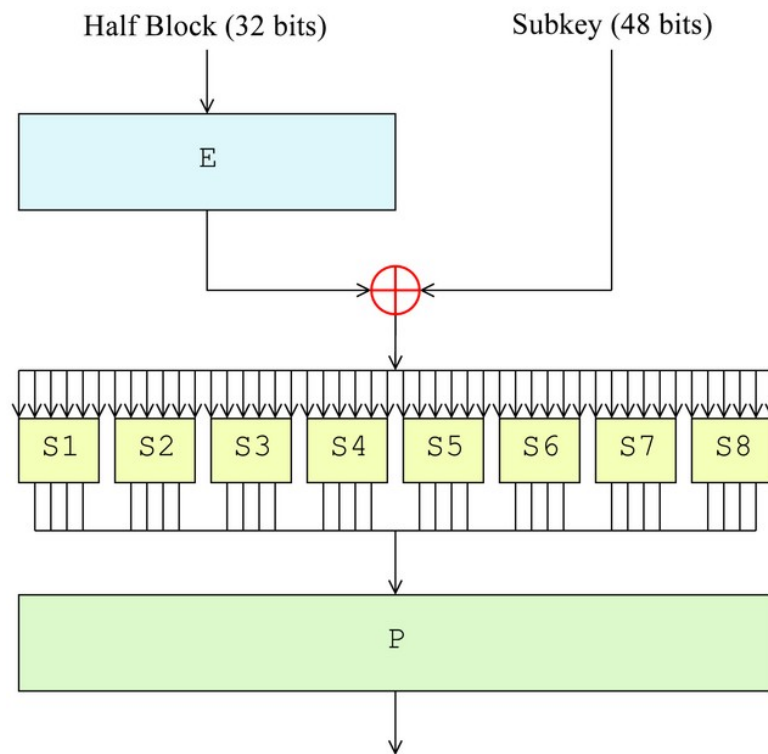
ciphertext: ISCTH HERIP CERIS INLTA OTSYN UREEC

Block Ciphers

- *Product cipher*
 - combination of substitution and transposition (permutation)
 - often organized in multiple rounds of alternating techniques called a SPN (substitution-permutation-network) or Feistel network
 - aims to achieve *confusion* and *diffusion*
- Confusion
 - refers to making the relationship between the key and the ciphertext as complex and involved as possible (achieved via substitution)
- Diffusion
 - refers to the property that redundancy in the statistics of the plaintext is dissipated in the statistics of the ciphertext (via transposition)

Block Ciphers

- Many block ciphers are based on the SPN design
- Data Encryption Standard (DES) is most well-known
 - 64 bit block size
 - 56 bit keys
 - 16 rounds
 - $S_1 - S_8$
 - S-Boxes
 - non-linear mapping
 - P
 - permutation network

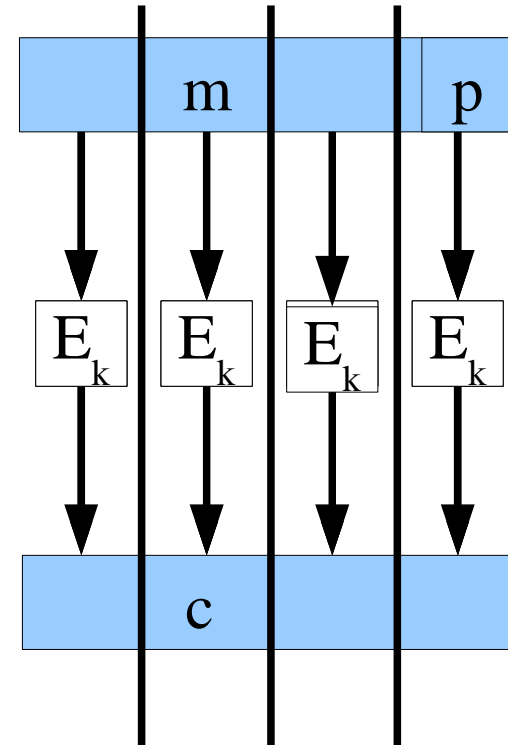


Encryption Modes for Block Ciphers

- Block cipher encrypts blocks of fixed size
 - DES: 56 bit key, 64 bit block
 - AES: 128,192, or 256 key, 128 bit block
 - ...
- How do I encrypt large, variable length messages?
 - Electronic Code Book (ECB)
 - Cipher Block Chaining (CBC)
 - ...

Electronic Code Book

- Pad message with random data so its length is a multiple of the block size
- Split message into blocks
- Encrypt each block separately



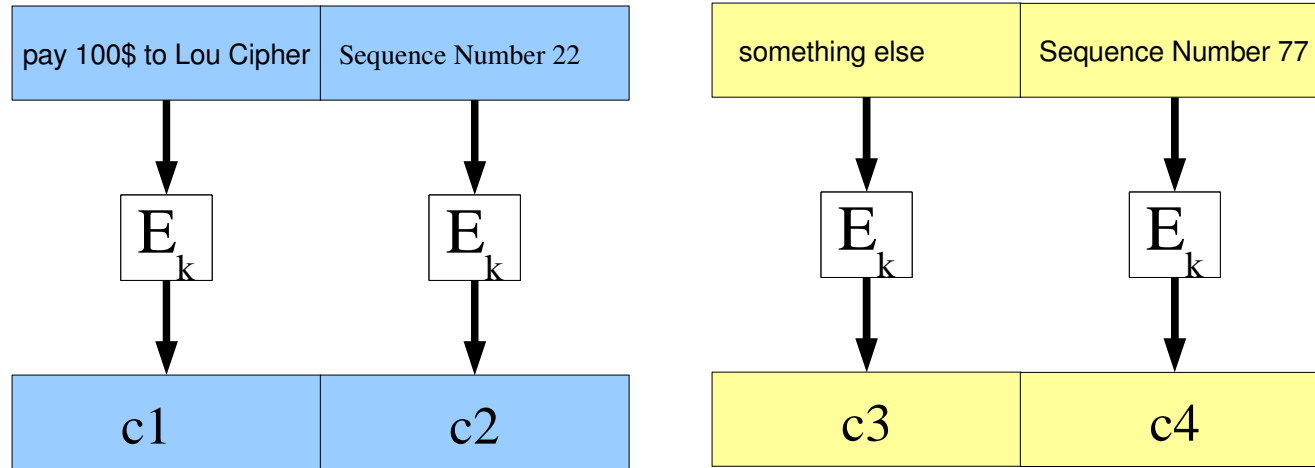
Electronic Code Book

- Each block encrypted independently of other blocks
 - ECB does not hide data patterns (repetitions)
- Vulnerable to block insertion and deletion
 - attacker can combine and reorder individual blocks from different messages into a new message

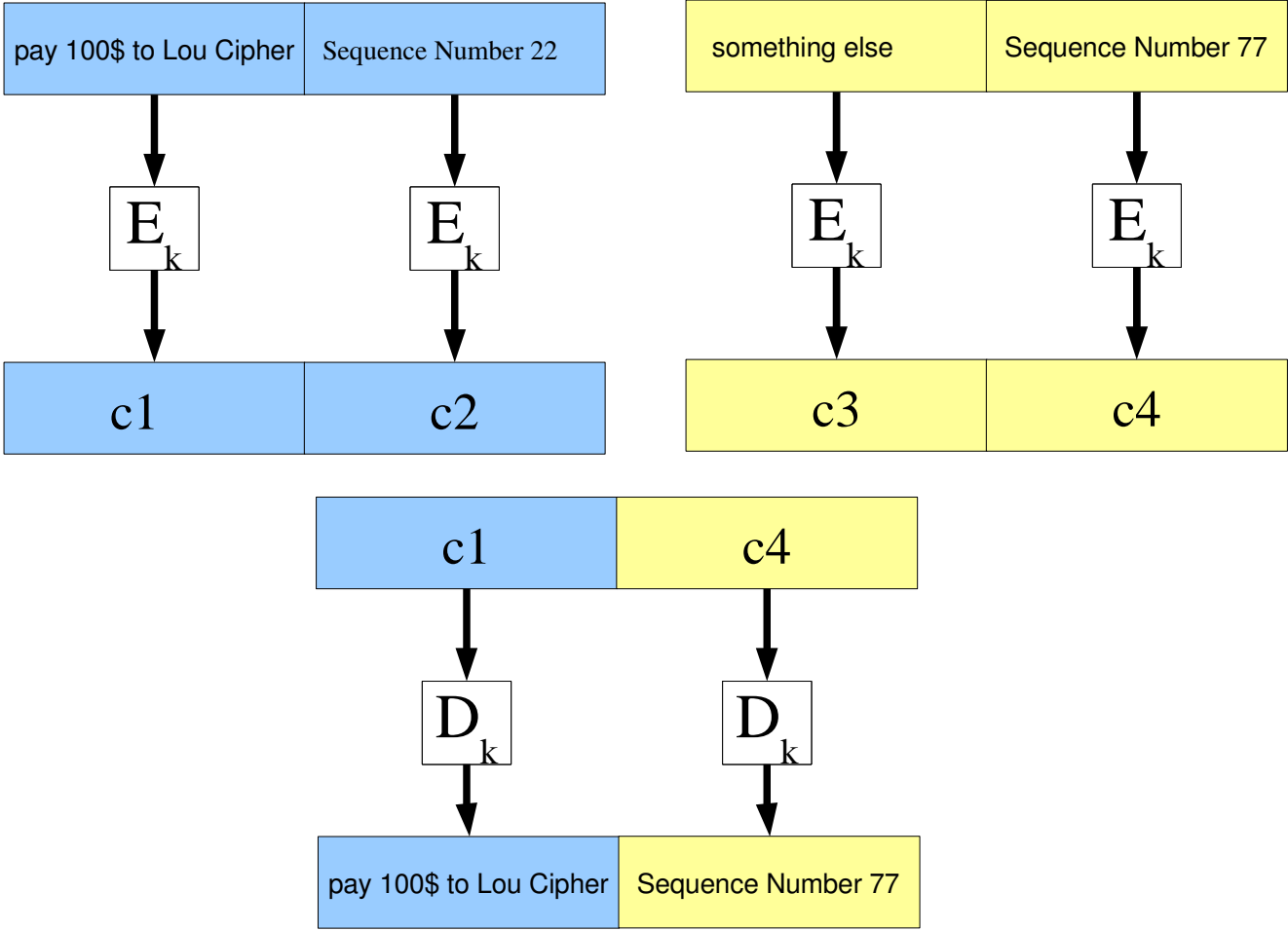
An Example of Block Replay

- Message replay attack:
 - attack a "secure" protocol by simply re-sending (encrypted) messages
 - Example:
 - $m = \text{pay } 100\$$ (to attacker)
 - initiate one legitimate payment
 - sniff $c = E_k(m)$
 - resend c multiple times
- Message replay defenses
 - use some form of sequence number/timestamp
 - reject messages with old sequence number

Block Replay



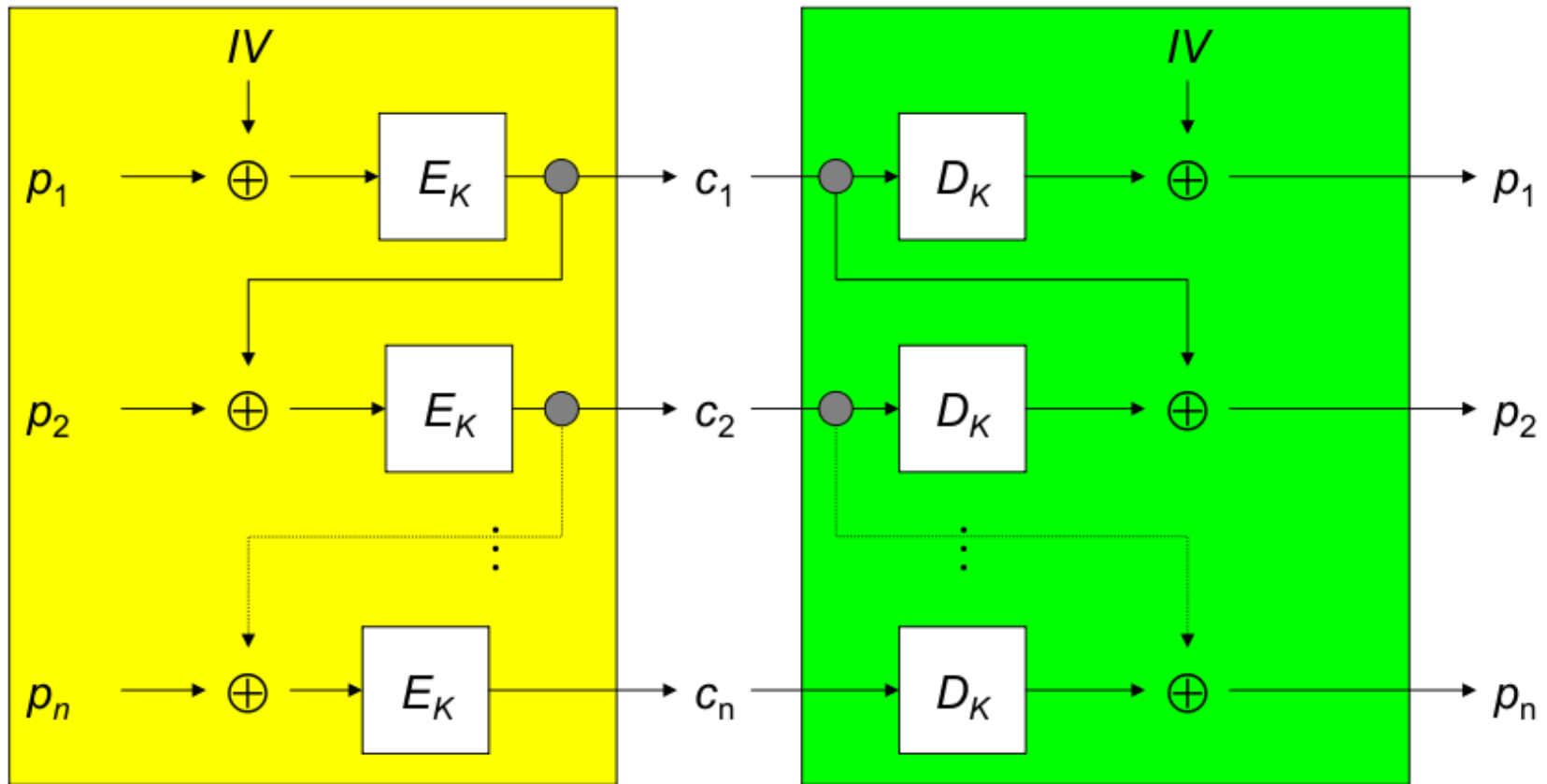
Block Replay



Cipher Block Chaining

- Encryption of one block depends on previous block
- IV: initialization vector
 - sent in plaintext
- For each block of ciphertext c_i :
 - $c_i = E_k(p_i \oplus c_{i-1})$
 - $c_0 = IV$
- For each block of plaintext p_i :
 - $p_i = D_k(c_i) \oplus p_{i-1}$
 - $c_0 = IV$

Cipher Block Chaining



Stream Ciphers

- Block ciphers with $t = 1$
- Different *substitution* for each bit
- Vernam cipher
 - $m_1, m_2, \dots, m_t \in \{0,1\}$
 - $e_1, e_2, \dots, e_t \in \{0,1\}$
 - $c_i = m_i \oplus e_i$
 - when e_i are generated randomly and used only once \rightarrow one-time pad
 - in practice, keystream is often generated from a pseudo-random generator, using a secret seed as the actual key
- RC4
 - used in 802.11 networks for WEP (Wired Equivalent Privacy)

One Time Pad

- t-bit message $m = m_1, m_2, \dots, m_t \in \{0,1\}$
- t-bit key $e = e_1, e_2, \dots, e_t \in \{0,1\}$ chosen randomly
- encryption function: $c = E_e(m)$:
 - $c_i = m_i \oplus e_i \quad 0 \leq i \leq t$
- decryption function $m = D_e(m)$
 - $m_i = c_i \oplus e_i \quad 0 \leq i \leq t$
- attacker can derive key if he knows one m, c pair
 - $e_i = m_i \oplus c_i \quad 0 \leq i \leq t$
- can use key only once

One Time Pad

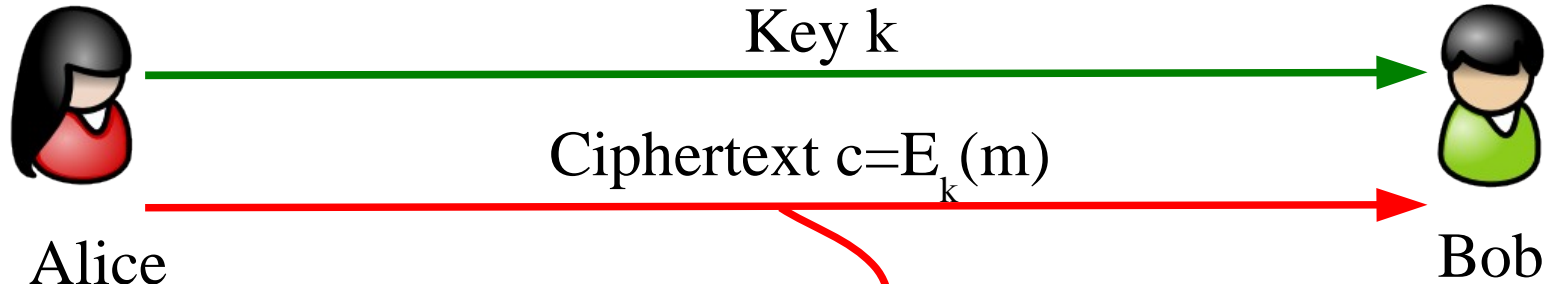
- One Time Pad is a perfect cipher
 - c leaks no information about m
 - $P(m|c)=P(m)$
- One Time Pad is unconditionally secure
 - even against attacker with unbounded computing power
 - any t -bit message is equally likely
- Any message m' can be obtained from c
 - using appropriate key e_i'
 - $e_i' = m_i' \oplus c_i \quad 0 \leq i \leq t$

How to make an insecure protocol out of a perfect cipher

- $m = \text{Pay } 100\$ \text{ to Lou Cipher}$
 - $100 = 0000000001100100$ (in binary, 16 bits)
- $c = m \oplus e$
- Lou cipher flips one bit of the ciphertext
 - $c' = c \oplus x$
 - $x = 1000000000000000$
- decrypted message is:
 - $m' = c \oplus e \oplus x = m \oplus x = 1000000001100100$
 - Pay **32868**\$ to Lou Cipher!

Simple Cryptographic Protocols

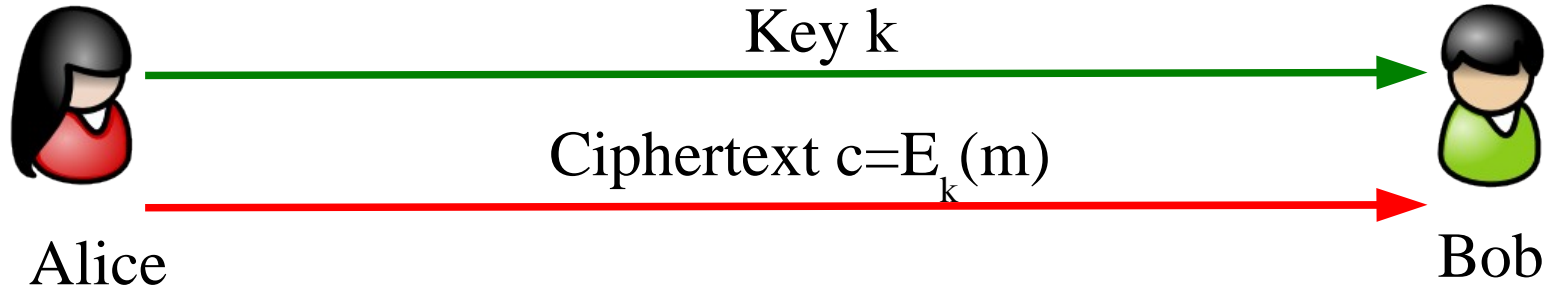
A Simple 2 Party Protocol



- Ciphertext c can be transmitted over **insecure** channel
- Key k must be transmitted over **secure** channel!
 - confidentiality
 - integrity
 - authentication

Lou Cipher

A Simple 2 Party Protocol

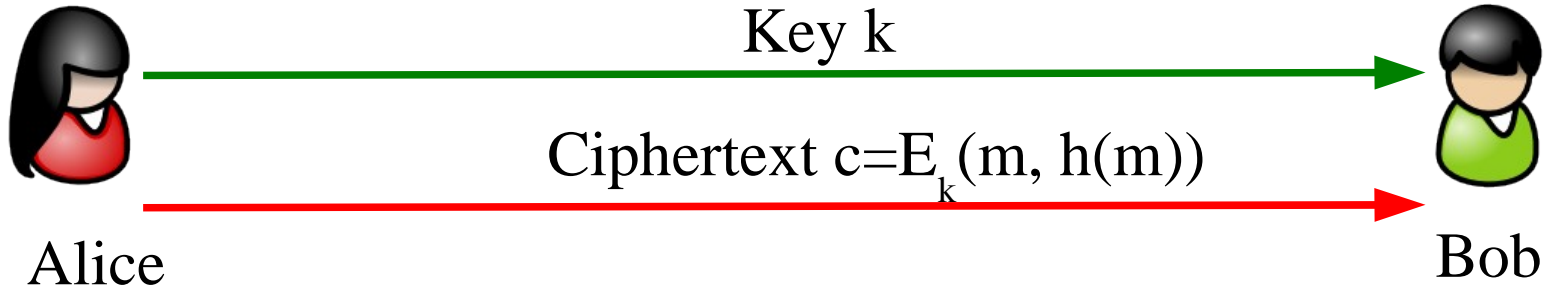


- Alice trusts Bob to keep k secret
- Alice knows only Bob can obtain m
 - confidentiality
- Bob trusts Alice to keep k secret
- Bob computes $m = D_k(c)$
- How does Bob know if m is correct?
- If attacker replaces c with c' , what happens to $m' = D_k(c')$?

Cryptographic Hash Functions

- $y=h(x)$
 - easy to compute
 - difficult to invert
- Map variable length x to fixed-length digest y
 - many-to-1 mapping
 - implies collisions: $\exists x, x' \mid h(x)=h(x')$
- Preimage resistant: given y , hard to find x
- 2nd Preimage resistant: given x , hard to find x' such that $h(x)=h(x')$
- Collision Resistant: hard to find x and x' such that $h(x)=h(x')$
- “hard” in computational sense (brute force guessing)

A Simple 2 Party Protocol



- $h(\)$: collision resistant hash function
 - i.e: sha-1
- Bob trusts Alice to keep k secret
- Bob computes:
 $(m, w) = D_k(c)$
- Bob computes $h(m)$ and compares it with w
 - integrity
 - authenticity

Multi-Party Protocols

- Use a single key k shared by all “authorized” users
 - individual users can't be authenticated
 - cannot protect users from one another
 - cannot easily revoke a user's authorization
 - low level of security
 - this is what we do for wireless (WEP, WPA,..)



Bob



Alice



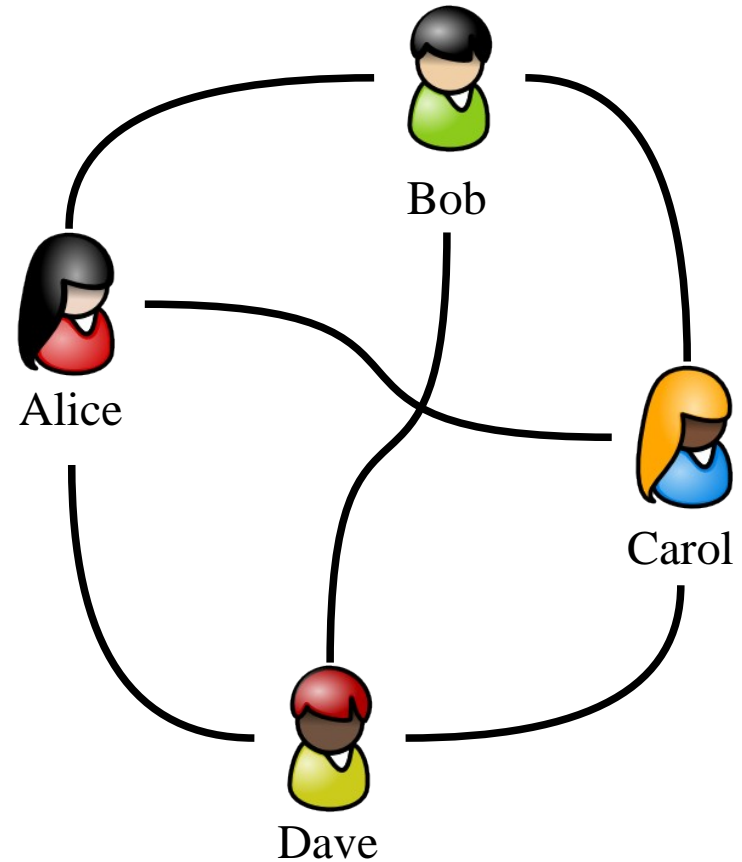
Carol



Dave

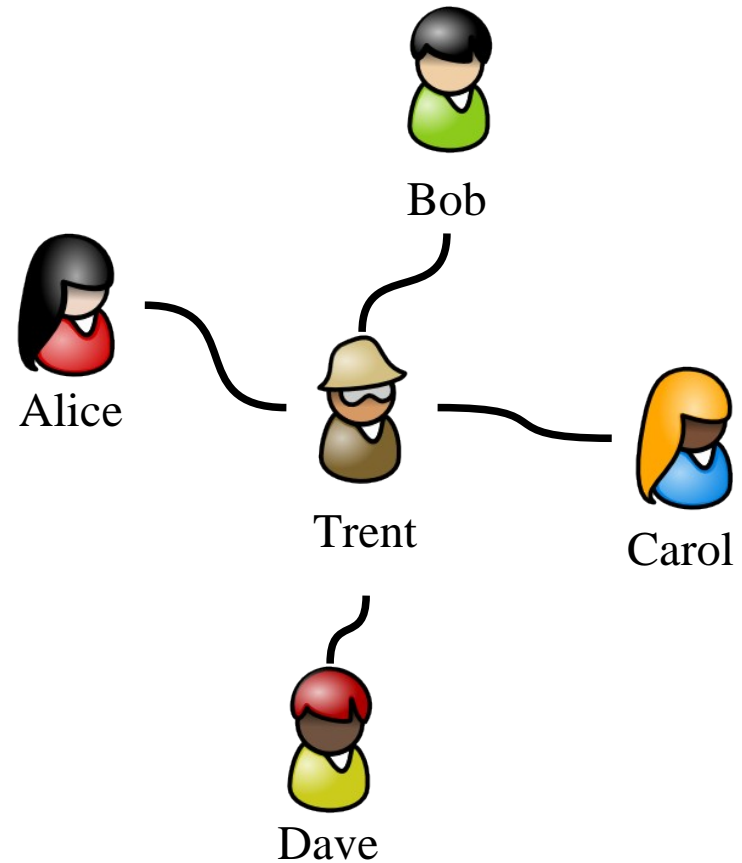
Multi-Party Protocols

- n users
- Each pair of users shares a secret key
- Each user has n-1 keys
- $n*(n-1)/2 \sim n^2$ keys for n users
- adding a user is complex (distribute 1 new key to each user)
- doesn't scale



Trusted Third Party

- n users, 1 authority T
- each user shares a secret key with T
- n keys total
- cryptographic protocol to establish a session key between A and B
 - simplest form: T encrypts and sends key to A and B
- T knows all session keys (can read all messages)



Diffie Hellman Protocol

- Algorithm for point to point key establishment (between 2 peers)
- No previously shared secret!
- An attacker sniffing the wire does not learn key
- Vulnerable to man-in-the-middle attack

- Based on discrete logarithm problem

Discrete Logarithm Problem

- (large) prime number p
- generator g : $1 \leq g < p$
 - $\forall 1 \leq n < p, \exists t$ such that $g^t \bmod p = n$
- Discrete Exponentiation
 - Given g, x , computing $y = g^x \bmod p$ is computationally easy
- Discrete Logarithm
 - Given $g, 1 \leq y \leq p-1$, it is computationally difficult to determine x ($0 \leq x \leq p-2$) such that $y = g^x \bmod p$

Discrete Logarithm Problem

- p represented in n bits ($p < 2^n$)
- Exponentiation (by squaring):
 - compute $g^2, (g^2)^2, \dots$
 - requires at most $2 \times \log_2 p \approx 2n$ multiplications mod p
- Logarithm
 - naive algorithm: try all powers of g mod p
requires at most $p \approx 2^n$ multiplications mod p
 - no polynomial-time algorithm is known

Diffie Hellman Protocol

- A and B want to decide a secret key
- A selects a random number a
- B selects a random number b
- A computes and sends $Y_A = g^a \text{ mod } p$
- B computes and sends $Y_B = g^b \text{ mod } p$

- A computes: $K_{AB} = (Y_B)^a \text{ mod } p = g^{ab} \text{ mod } p$
- B computes: $K_{AB} = (Y_A)^b \text{ mod } p = g^{ab} \text{ mod } p$

Diffie Hellman Protocol

- Passive adversary can see Y_A and Y_B
- Does not know a and b
- Can compute $b = \log_g(Y_B)$
 - this is computationally hard
- then $K_{AB} = (Y_A)^b$
- There is no proof that there is no easier way of computing K_{AB}
 - but no better way is known

Man in the Middle Attack



a

Alice

c



c

Lou Cipher

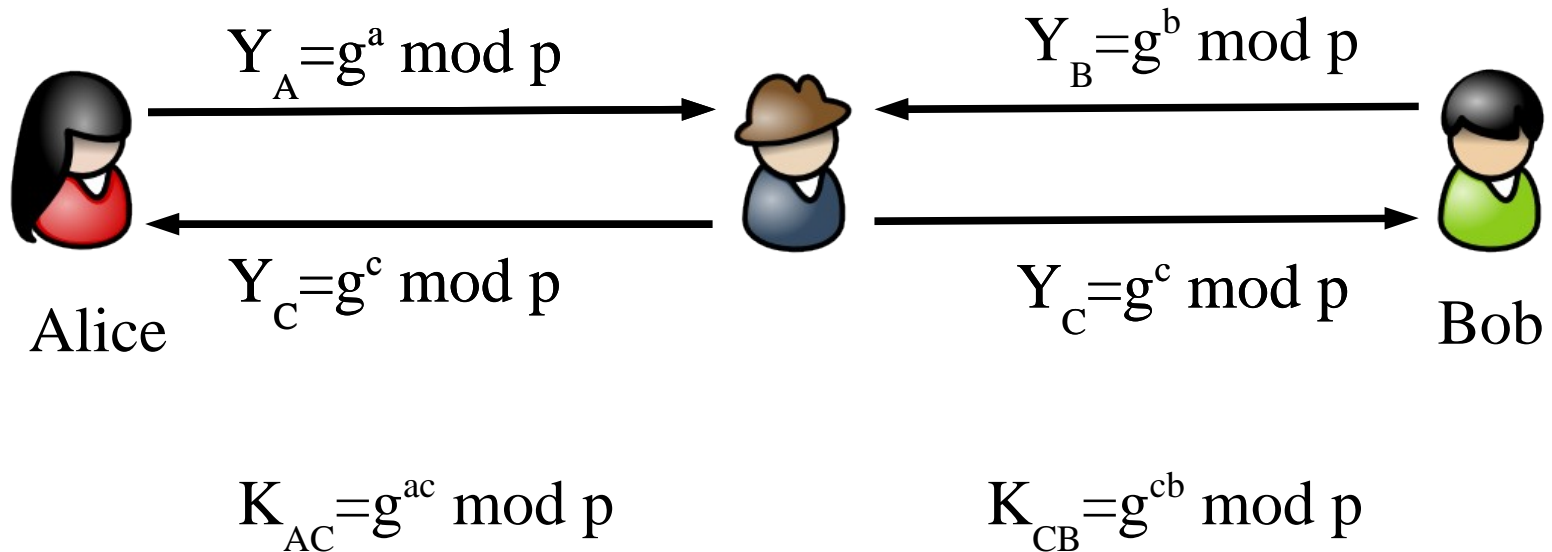
b



Bob

- Alice and Bob do not know with whom they are talking
- An active adversary can perform a man in the middle attack
- General problem: always need a root of trust!

Man in the Middle Attack



Public Key Cryptography

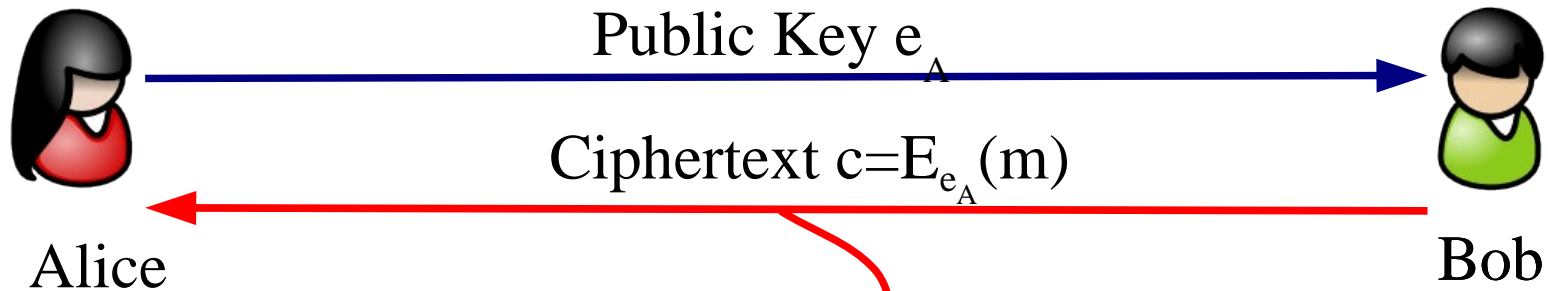
Public-key Cryptography

- Consider an encryption scheme with key pair (e,d)
 - scheme is called a public-key scheme if it is computationally infeasible to determine d when e is known
- decryption key d must be kept secret
- encryption key e can be published
- In public-key schemes, E_e is usually a *trapdoor one-way function* and d is the trapdoor

Public-key Cryptography

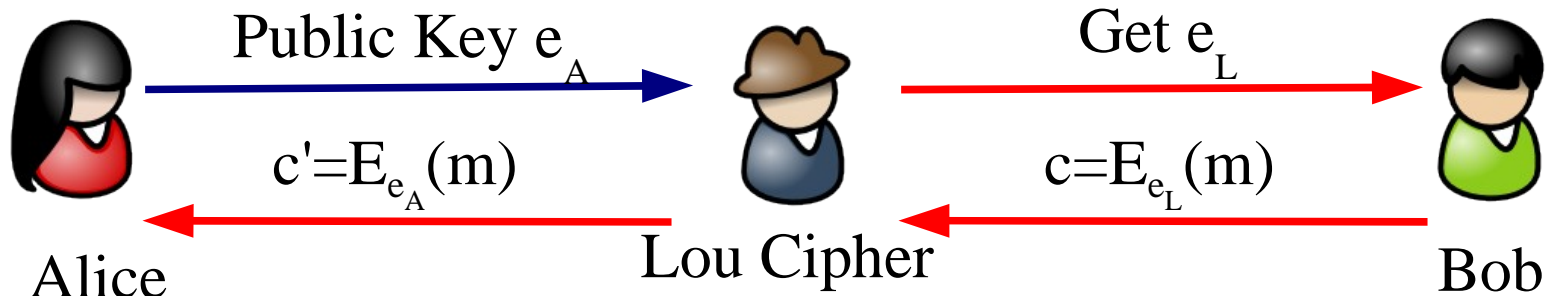
- One-way Functions: easy to compute $y=f(x)$, hard to compute $x=f^{-1}(y)$
 - calculating the exponentiation of an element a in a finite field [$a^p \pmod{p}$]
 - discrete logarithm is hard!
 - multiplication of two large prime numbers [$n = p*q$]
 - factorization is hard!
- Trapdoor one-way functions:
 - easy to compute $c=f_e(m)$, hard to compute $m=f_e^{-1}(c)$
 - easy to compute $f_e^{-1}(c)$ if trapdoor d is known
- RSA (Rivest, Shamir, Adleman) 1978
 - based on intractability of the factorization of $n=p*q$, where p and q are two large prime numbers

2 Party Protocol with Public-key



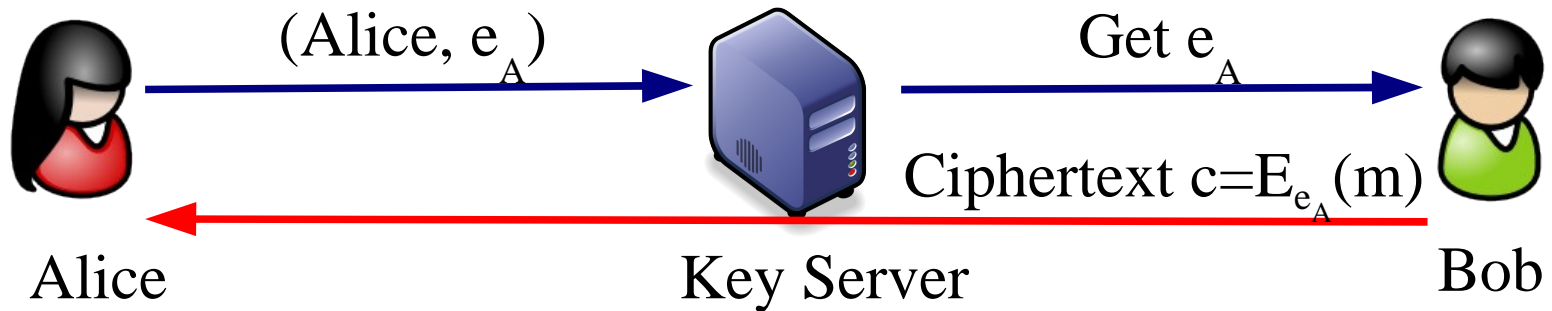
- Ciphertext c can be transmitted over **insecure** channel
- Public key e_A has to be transmitted on a **more secure** channel
 - integrity
 - authenticity
 - confidentiality not required!

Man in the Middle



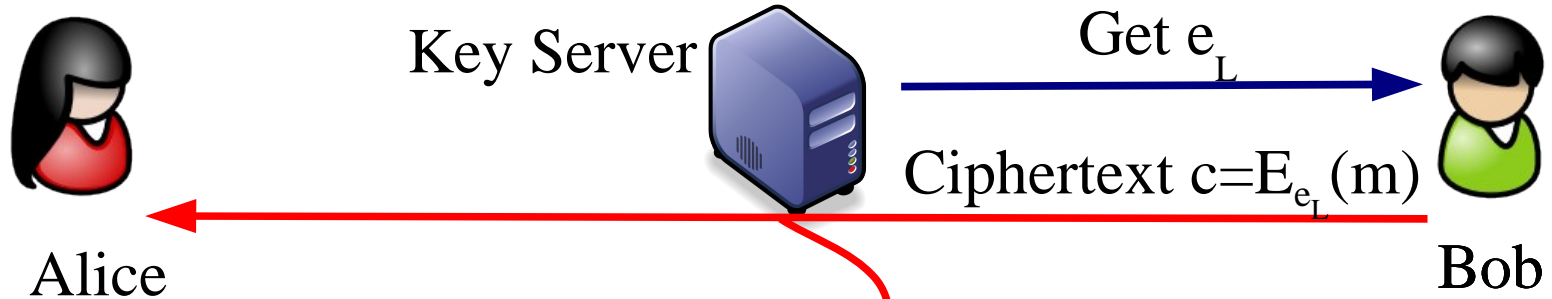
- If public key e_A is also transmitted over **insecure** channel
- Passive adversary still cannot retrieve m
- MITM is possible
 - attacker sends e_L to Bob instead of e_A
 - retrieves $m = D_{d_L}(c)$
 - sends on $c' = E_{e_A}(m)$

2-Party Protocol with Key Server



- Can use a Key Server to publish your public key
 - i.e: pgp (Pretty Good Privacy) servers
- Anyone can retrieve e_A
 - public key does not need to be secret
- Anyone can upload a key to a server
 - no verification of identity!
 - Bob cannot be sure that the key he obtains is really Alice's

Impersonation Attack



(Alice, e_L)



Lou Cipher

- Adversary can upload a public key of his choice as someone else's
- retrieve $m = D_{d_L}(c)$

Digital Signature

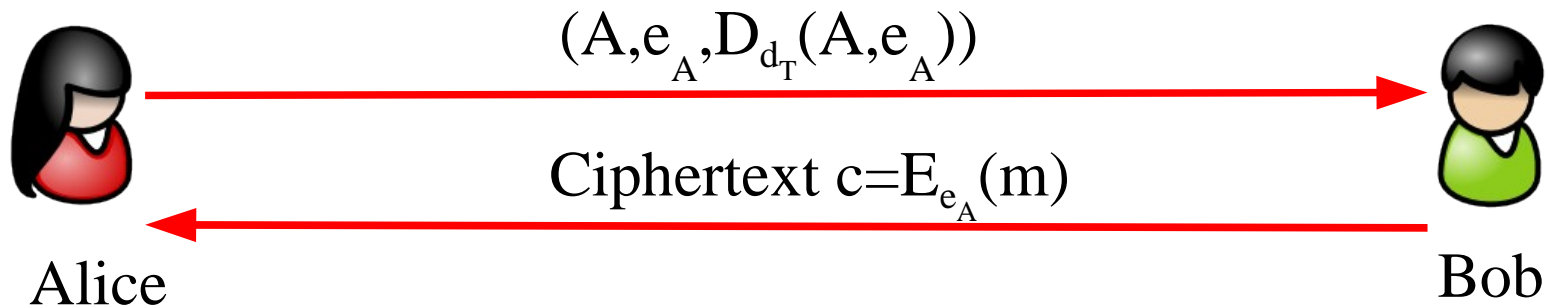
- to sign message m
- “decrypt” it with your private key
- signature $s = D_{d_A}(m)$
 - only A can compute it
- signature is verified by “encrypting” it again
- compare m with $E_{e_A}(s)$
 - anyone can compute it

- Normally a hash of the message is signed instead
 - $s = D_{d_A}(h(m))$

Authenticating Public Keys

- (A, e_A) is signed with a trusted private key d_T
 $(A, e_A, D_{d_T}(A, e_A))$
- Authenticated public keys can be stored on **untrusted** servers, and transmitted over an **untrusted** network
 - impersonation attack is not possible
 - the worst thing an adversary can do is a denial of service (do not provide a signed public key for A)
- But who signs e_T ?
- Different trust models
 - hierarchical trust model
 - peer to peer trust model

2 Party Protocol with Signed Public key

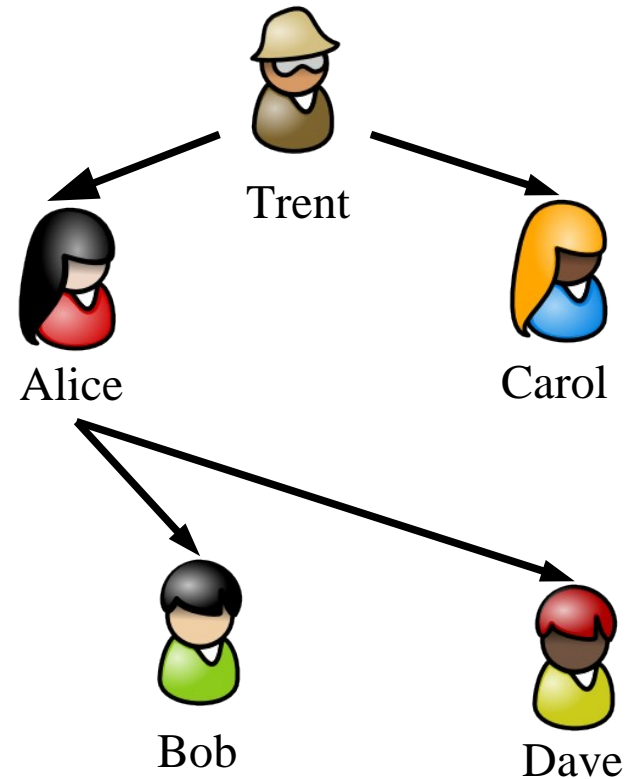


- Public Key e_T is a shared secret
- Confidentiality is maintained against active adversary (MITM)
- Alice sends
 - (A, e_A, s) , with $s = D_{d_T}(A, e_A)$
- Bob verifies that
 - $E_{e_T}(s) = (A, e_A)$

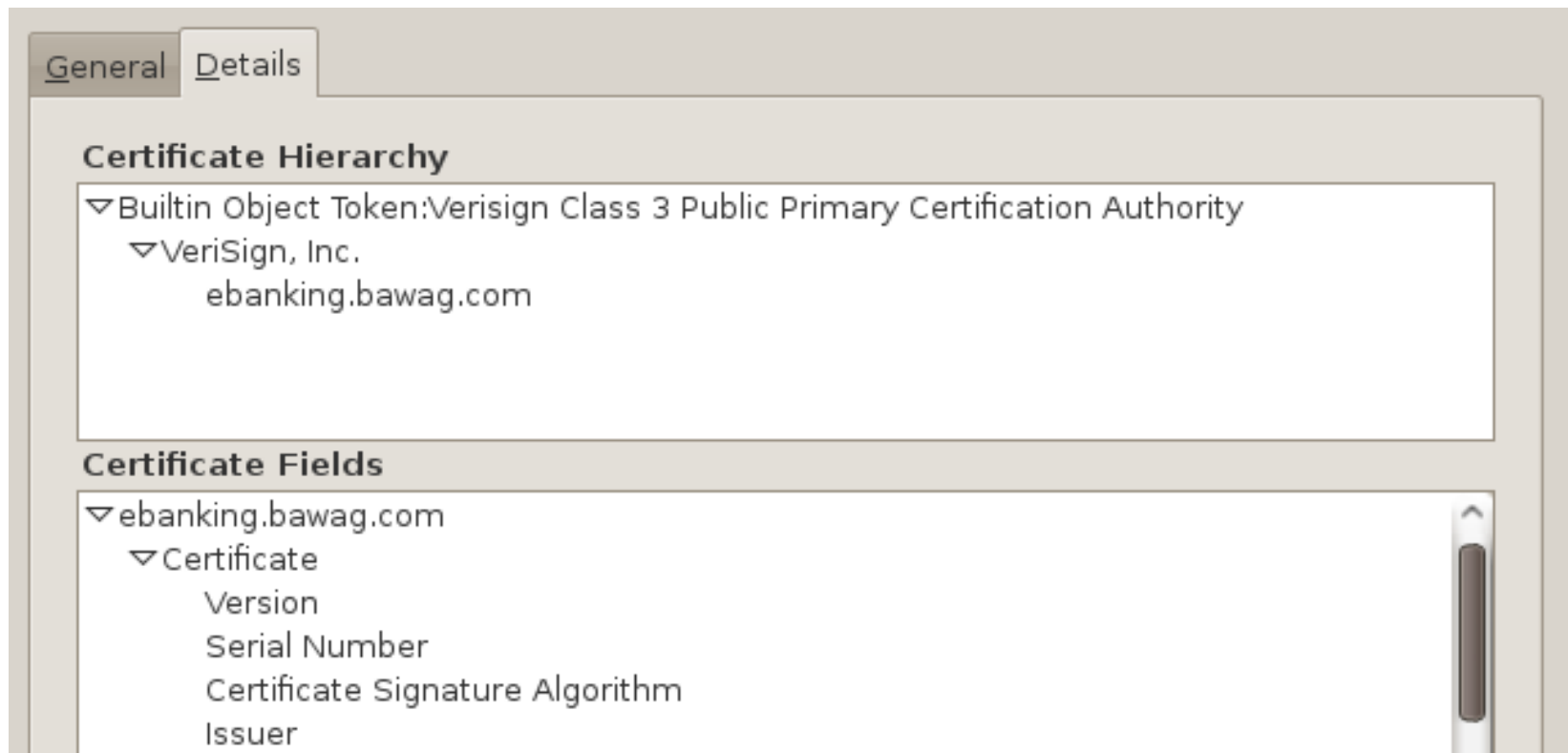
Models of Trust: Hierarchical and Peer-to-Peer

Hierarchical Trust Model

- 1 or more trusted third parties
- Trust model is a tree
 - Trusted Third Party is root of trust tree
 - Root Certification Authorities (CA)
- Public key of TTP has to be distributed
 - bundled with operating system, browser
- TTP delegate trust to intermediate CAs



Hierarchical Trust Model



Public Key Infrastructure

- public-key encryption is slow
 - usually, digital signatures sign a **hash** of the message
 - $s = D_{d_A}(h(A, e_A, \dots))$
 - hash needs to be collision resistant!
- Public Key Infrastructure (PKI) is based on:
 - multiple root Certification Authorities
 - standard formats for certificates (X.509)
- Used in https
 - crucial to the security/viability of online commerce
- X.509 allows the use of md5 as hash function $h(.)$
- collision resistance of md5 hash is known to be broken!

Md5 Collision Attack on Public Key Infrastructure

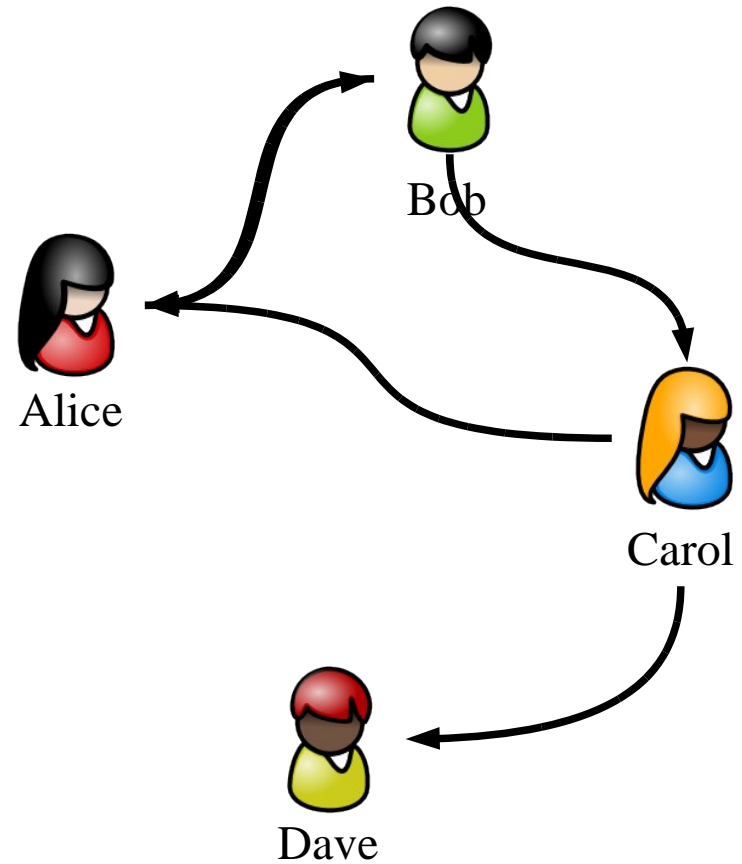
- Create 2 certificates with colliding (same) md5
 - one for my domain: $X = (A, e_A, \dots)$ $A = \text{mysite.com}$
 - one for target domain: $Y = (B, e_B, \dots)$ $B = \text{banking.com}$
 - $h(X) = h(Y)$
- Send X to Certification Authority for signing
 - CA will sign it because mysite.com is really my website
 - signature $s = D_{d_T}(h(X)) = D_{d_T}(h(Y))$
 - also valid for $Y = (B, e_B, \dots)$!
- Impact:
 - can now impersonate website banking.com
 - allow man in the middle attack despite https

Md5 Collision Attack on Public Key Infrastructure

- Real attack is even worse:
 - have CA sign a certificate that identifies you as a CA
 - set appropriate field in $Y = (B, e_B, \dots, CA)$
- Impact:
 - can now impersonate **any** website
 - just sign a certificate for it with e_B
- Defense:
 - CA: do not use md5 in signing process
 - Browser: reject certificates based on md5
- For more information:
 - <http://www.win.tue.nl/hashclash/rogue-ca/>

Peer to Peer Trust Model

- User-centric
- Each user decides whom to trust
- Trust “propagates” in the network
- Used in pgp (pretty good privacy) for email
 - sign email
 - encrypt email
 - sign other people's public keys to indicate your trust



Pretty Good Privacy (PGP)

- Public keys stored in public key servers (e.g.: pgp.mit.edu)
 - (user@email.com, e_{user})
- Different levels of trust for a key
 - implicit trust: in my own key
 - complete trust
 - marginal trust
 - no trust
- Users sign each other's keys
 - a key is valid if it is
 - signed with my own key
 - signed with 1 key I completely trust
 - signed with 2 keys I marginally trust
 - signatures are also uploaded to public key servers

Conclusions

- (Very) brief introduction to cryptography
- Symmetric encryption
 - block ciphers
 - stream ciphers
- Asymmetric encryption (public key)
- Key distribution
 - Diffie-Hellman
 - key distribution with public keys
 - models of trust
- Next lecture (in 2 weeks):
 - buffer overflows!