Security and Privacy in Wireless Networks

Mohammad Hossein Manshaei
manshaei@gmail.com

Introduction to Cryptographic Algorithms and Protocols

Appendix A

Security and Cooperation in Wireless Networks

secowinet.epfl.ch

Some materials are derived from those available on the Web site of the book "Computer Networking", by Kurose and Ross, PEARSON

Contents

- Asymmetric-key Encryption
- Message Authentication Codes and Hash Function
- Digital Signature
- Session Key Establishment Protocols
- Pseudo-Random Generator
- Advanced Authentication Techniques

Public Key Cryptography

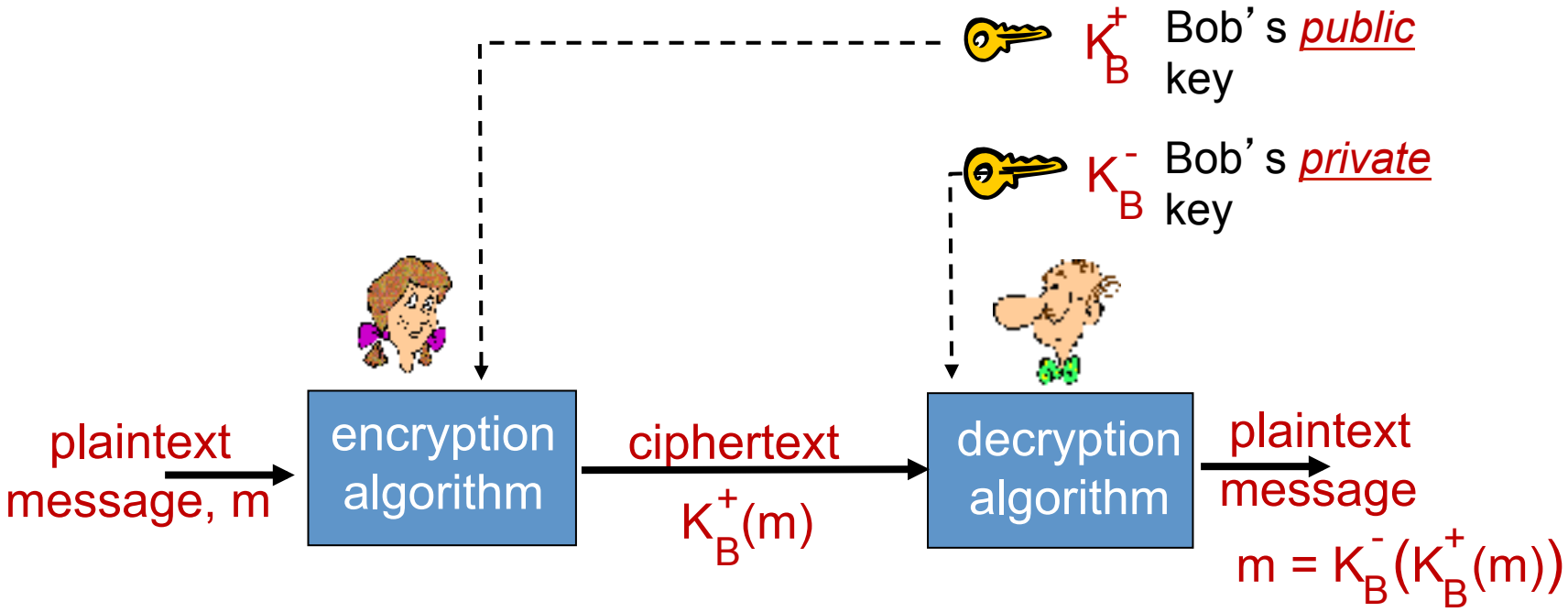
symmetric key crypto

- requires sender, receiver know shared secret key
- Q: how to agree on key in first place (particularly if never "met")?

public key crypto

- radically different approach [Diffie-Hellman76, RSA78]
- sender, receiver do *not* share secret key
- *public* encryption key known to *all*
- *private* decryption key known only to receiver

Public Key Cryptography



Public Key Encryption Algorithms

Requirements:

1. Need $K_B^+(\cdot)$ and $K_B^-(\cdot)$ such that

$$K_B^-(K_B^+(m)) = m$$

2. Given public key K_B^+, it should be impossible to compute private key K_B^-

RSA: Rivest, Shamir, Adelson algorithm

Prerequisite: Modular Arithmetic

- x mod n = remainder of x when divide by n
- Facts:
  - [(a mod n) + (b mod n)] mod n = (a+b) mod n
  - [(a mod n) - (b mod n)] mod n = (a-b) mod n
  - [(a mod n) * (b mod n)] mod n = (a*b) mod n
- Thus
  - $(a \bmod n)^d \bmod n = a^d \bmod n$
- Example: x=14, n=10, d=2:
  $(x \bmod n)^d \bmod n = 4^2 \bmod 10 = 6$
  $x^d = 14^2 = 196 \quad x^d \bmod 10 = 6$

RSA: Getting Ready

- Message: just a bit pattern
- Bit pattern can be uniquely represented by an integer number
- Thus, encrypting a message is equivalent to encrypting a number.

Example:

- m= 10010001 . This message is uniquely represented by the decimal number 145.
- To encrypt m, we encrypt the corresponding number, which gives a new number (the ciphertext).

RSA: Creating Public/Private Key Pair

1. Choose two large prime numbers *p, q.* (e.g., 1024 bits each)
2. Compute *n* = *pq*, *z* = *(p-1)(q-1)*
3. Choose *e (*with *e<n)* that has no common factors with z (*e, z* are "relatively prime").
4. Choose *d* such that *ed-1* is exactly divisible by *z*. (in other words: *ed* mod *z* = *1*).
5. *Public* key is *(n,e)*. *private* key is *(n,d)*.

K_B^+ K_B^-

RSA: Encryption and Decryption

0. Given (n,e) and (n,d) as computed above

1. To encrypt message m ($<n$), compute

$$c = m^e \bmod n$$

2. To decrypt received bit pattern, c, compute

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

magic happens!

$$m = \underbrace{(m^e \bmod n)}_{c}{}^d \bmod n$$

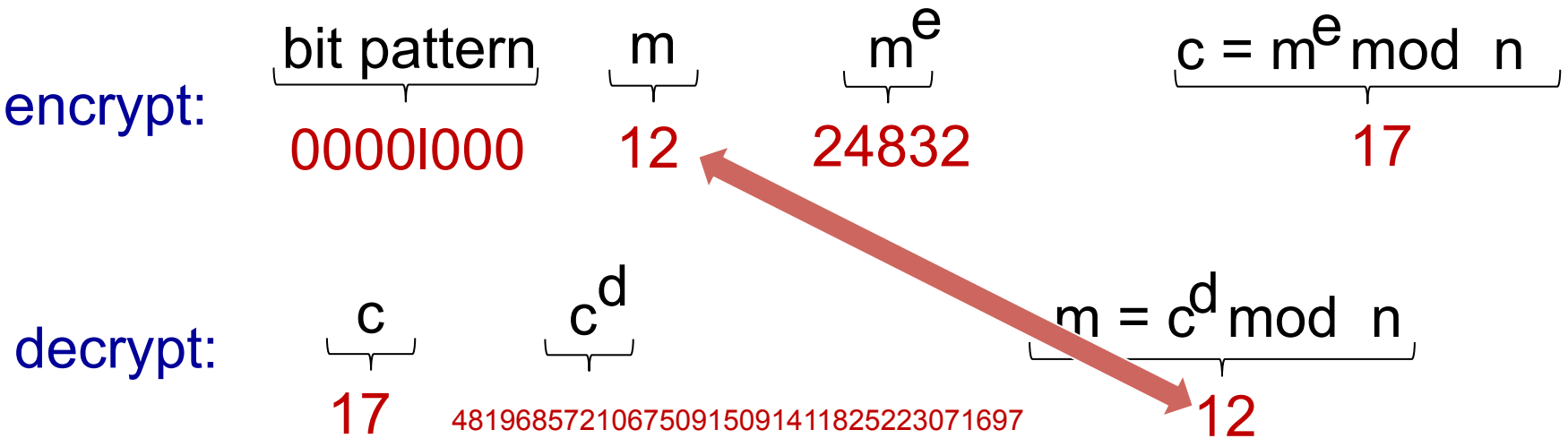
RSA Example:

Bob chooses *p=5, q=7*. Then *n=35, z=24*.

e=5 (so *e, z* relatively prime).

d=29 (so *ed-1* exactly divisible by z).

encrypting 8-bit messages.



Why does RSA work?

- must show that $c^d \bmod n = m$ where $c = m^e \bmod n$
- fact: for any x and y: $x^y \bmod n = x^{(y \bmod z)} \bmod n$
  - where n= pq and z = (p-1)(q-1)
- thus,

$$c^d \bmod n = (m^e \bmod n)^d \bmod n$$
$$= m^{ed} \bmod n$$
$$= m^{(ed \bmod z)} \bmod n$$
$$= m^1 \bmod n$$
$$= m$$

RSA: Another Important Property

The following property will be *very* useful later:

$$\underbrace{K_B^-(K_B^+(m))}_{\text{use public key first, followed by private key}} = m = \underbrace{K_B^+(K_B^-(m))}_{\text{use private key first, followed by public key}}$$

result is the same!

Why $K_B^-(K_B^+(m)) = m = K_B^+(K_B^-(m))$?

follows directly from modular arithmetic:

$$(m^e \bmod n)^d \bmod n = m^{ed} \bmod n$$
$$= m^{de} \bmod n$$
$$= (m^d \bmod n)^e \bmod n$$

Why is RSA secure?

- Suppose you know Bob's public key (n,e). How hard is it to determine d?
- Essentially need to find factors of n without knowing the two factors p and q
  - fact: factoring a big number is hard

Examples for Hard Problems

- **Factoring problem**
  - given a positive integer n, find its prime factors
    - true complexity is unknown
    - it is believed that it does not belong to P

- **Discrete logarithm problem**
  - given a prime p, a generator g of Z_p^*, and an element y in Z_p^*, find the integer x, $0 \le x \le p-2$, such that $g^x \bmod p = y$
    - true complexity is unknown
    - it is believed that it does not belong to P

- **Diffie-Hellman problem**
  - given a prime p, a generator g of Z_p^*, and elements $g^x \bmod p$ and $g^y \bmod p$, find $g^{xy} \bmod p$
    - true complexity is unknown
    - it is believed that it does not belong to P

The Need for Salting

- Let us assume that the adversary observes a ciphertext $c = E_K(m)$
- Let the set of possible plaintexts be M
- If M is small, then the adversary can try to encrypt every message in M with the publicly known key K until **she finds the message m that maps into c**
- The usual way to prevent this attack is to randomize the encryption
  - some random bytes are added to the plaintext message before encryption through the application of the PKCS #1 formatting rules
  - when the message is decrypted, the recipient can recognize and discard these random bytes

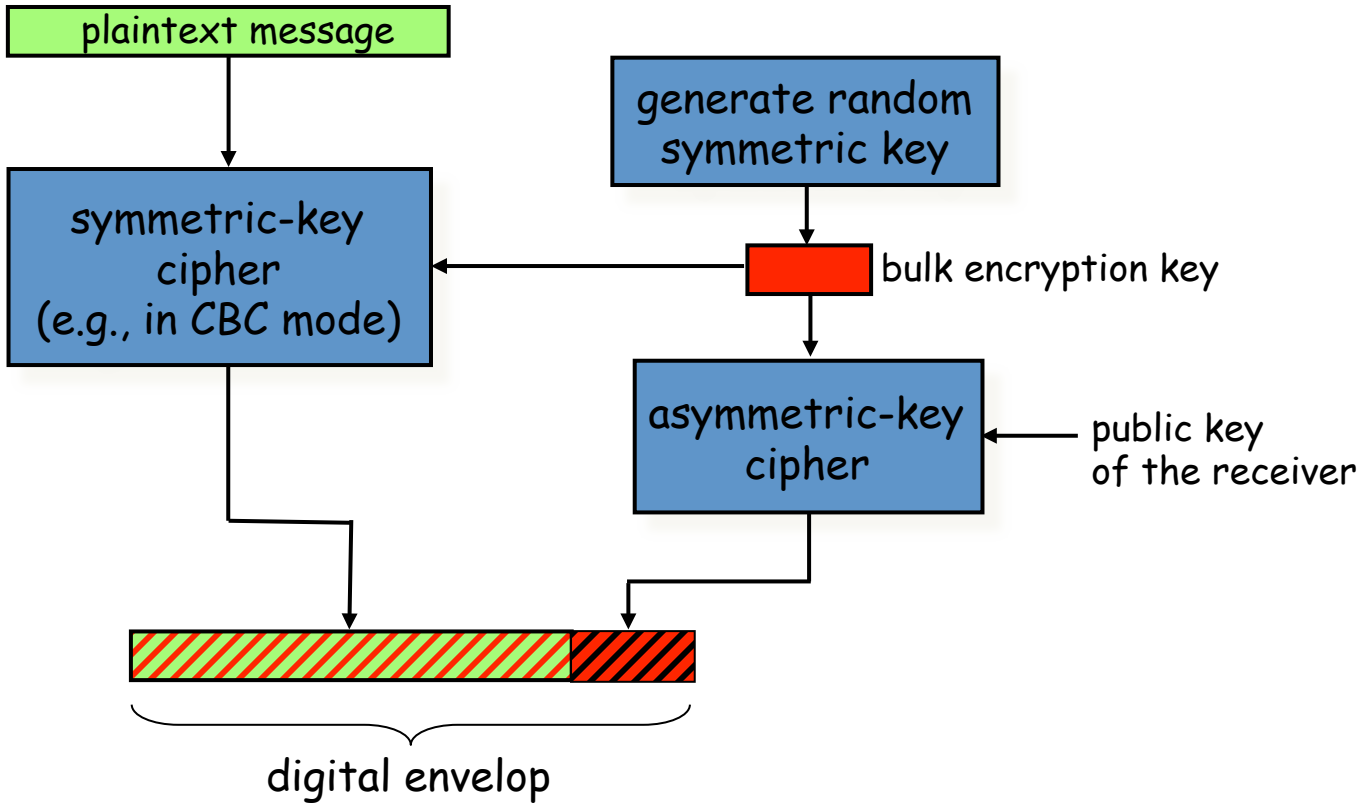
RSA in Practice: Session Keys

- Exponentiation in RSA is computationally intensive
- DES is at least 100 times faster than RSA
- Use public key crypto to establish secure connection, then establish second key – symmetric session key – for encrypting data

Session key, K_S

- Bob and Alice use RSA to exchange a symmetric key K_S
- Once both have K_S, they use symmetric key cryptography

Digital Enveloping



The ElGamal Encryption Scheme

- **Key generation**
  - generate a large random prime p and choose generator g of the multiplicative group Z_p^* = {1, 2, …, p-1}
  - select a random integer a, $1 \le a \le p-2$, and compute $A = g^a \bmod p$
  - the public key is (p, g, A)
  - the private key is a
- **Encryption**
  - represent the message as an integer m in [0, p-1]
  - select a random integer r, $1 \le r \le p-2$, and compute $R = g^r \bmod p$
  - compute $C = m \cdot A^r \bmod p$
  - the ciphertext is the pair (R, C)
- **Decryption**
  - compute $m = C \cdot R^{p-1-a} \bmod p$
- **Proof of decryption**

  $C \cdot R^{p-1-a} \equiv m \cdot A^r \cdot R^{p-1-a} \equiv m \cdot g^{ar} \cdot g^{r(p-1-a)} \equiv m \cdot (g^{p-1})^r \equiv m \pmod p$

Relation to Hard Problems

- Security of the ElGamal scheme is said to be based on the discrete logarithm problem in Z_p^*, although equivalence has not been proven yet

- Recovering m given p, g, A, R, and C is equivalent to solving the Diffie-Hellman problem

Contents

- Asymmetric-key Encryption
- Message Authentication Codes and Hash Function
- Digital Signature
- Session Key Establishment Protocols
- Pseudo-Random Generator
- Advanced Authentication Techniques

Problems and Possible Solutions

AUTHENTICATION

Authentication

Goal: Bob wants Alice to "prove" her identity to him

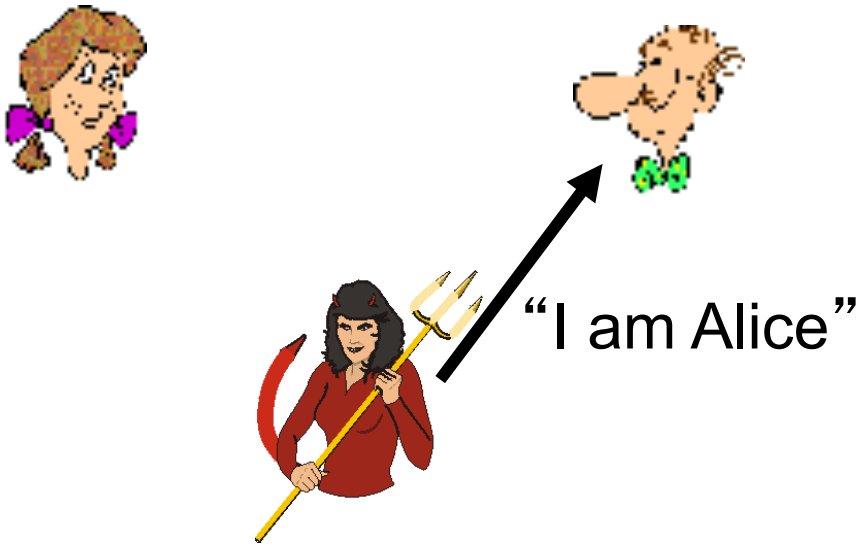
Protocol ap1.0: Alice says "I am Alice"

Failure scenario??

Authentication

Goal: Bob wants Alice to "prove" her identity to him

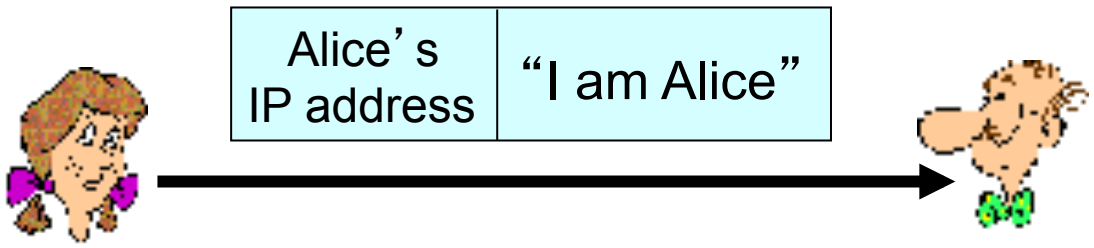
Protocol ap1.0: Alice says "I am Alice"



in a network,
Bob can not "see" Alice,
so Trudy simply declares herself to be Alice

Authentication: Another Try

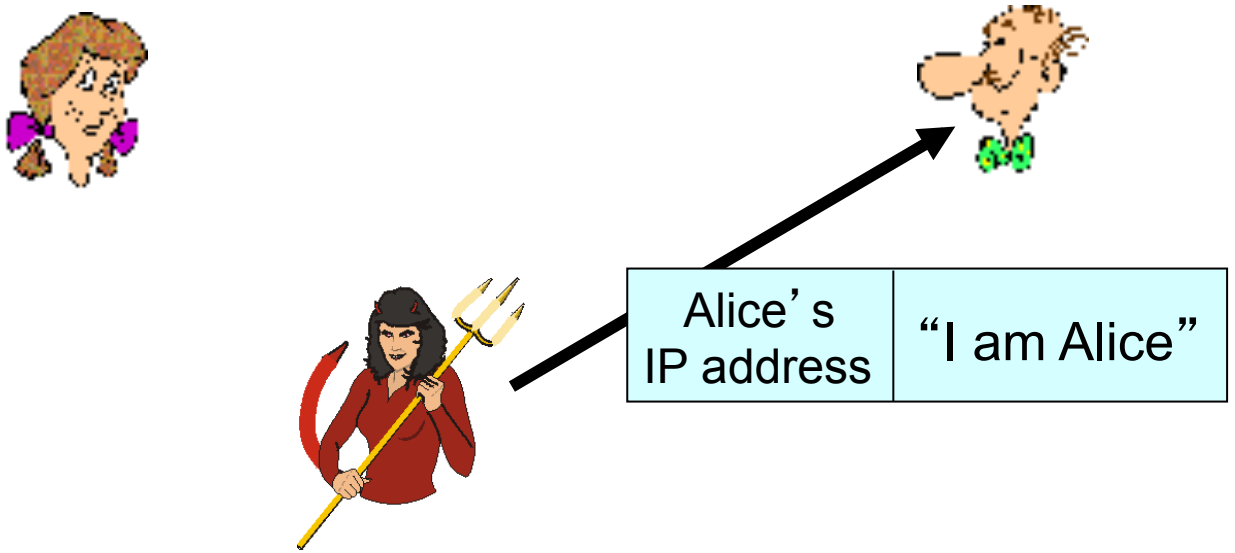
Protocol ap2.0: Alice says "I am Alice" in an IP packet containing her source IP address



Failure scenario??

Authentication: Another Try

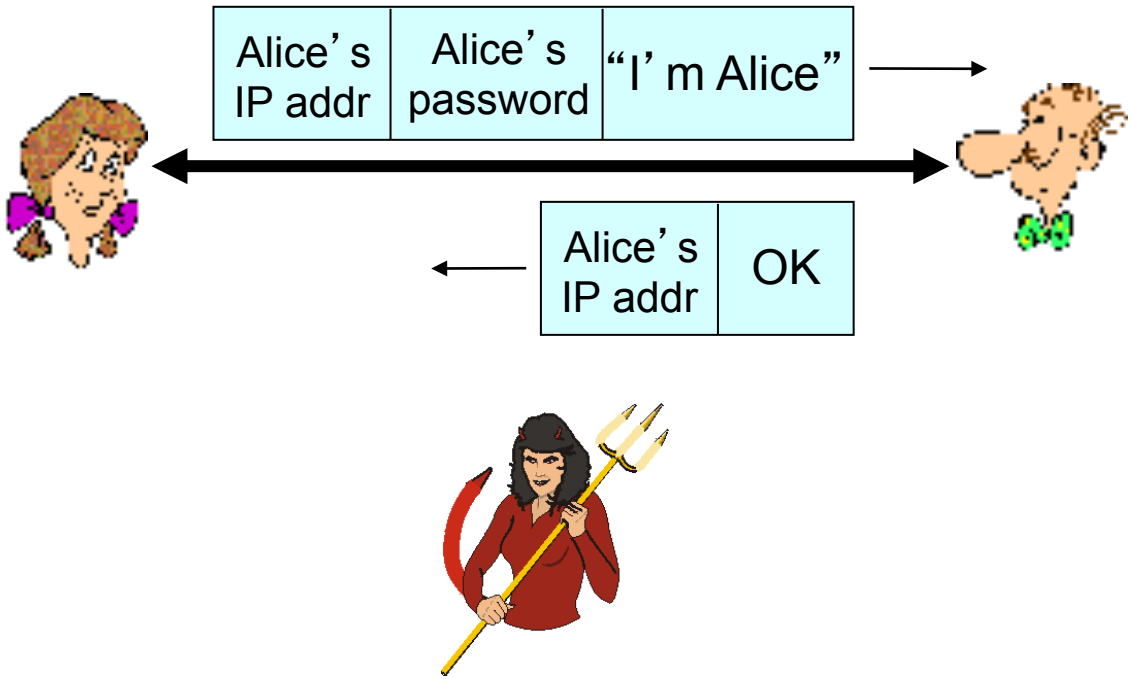
Protocol ap2.0: Alice says "I am Alice" in an IP packet containing her source IP address



Trudy can create a packet "spoofing" Alice's address

Authentication: Another Try

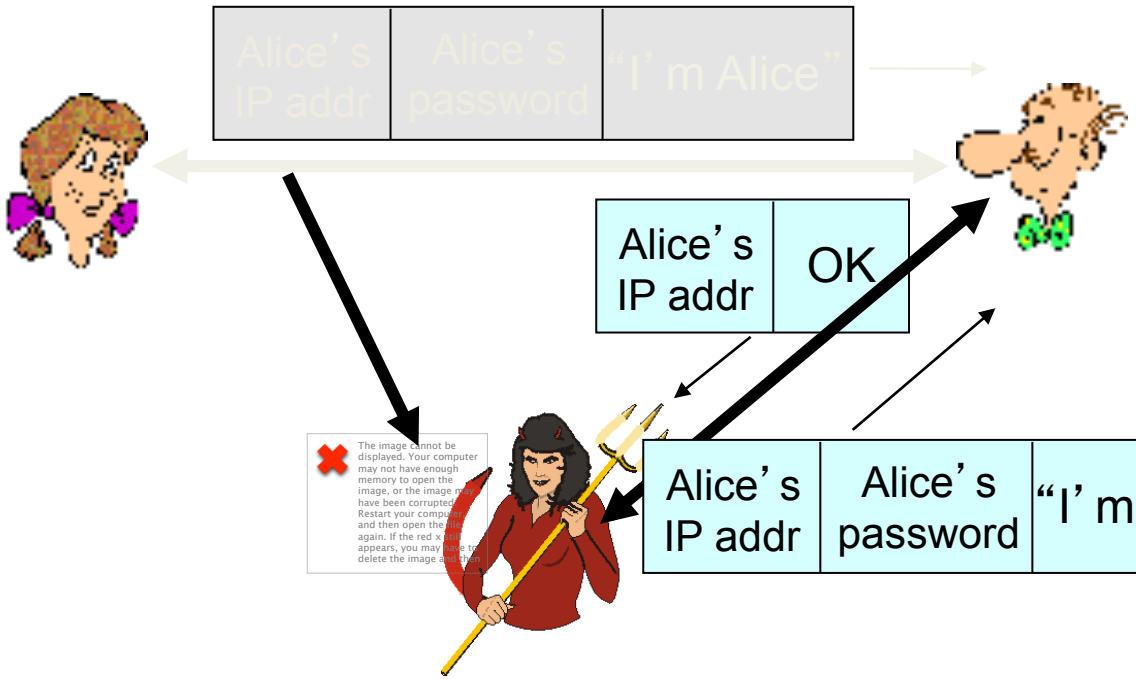
Protocol ap3.0: Alice says "I am Alice" and sends her secret password to "prove" it.



Failure scenario??

Authentication: Another Try

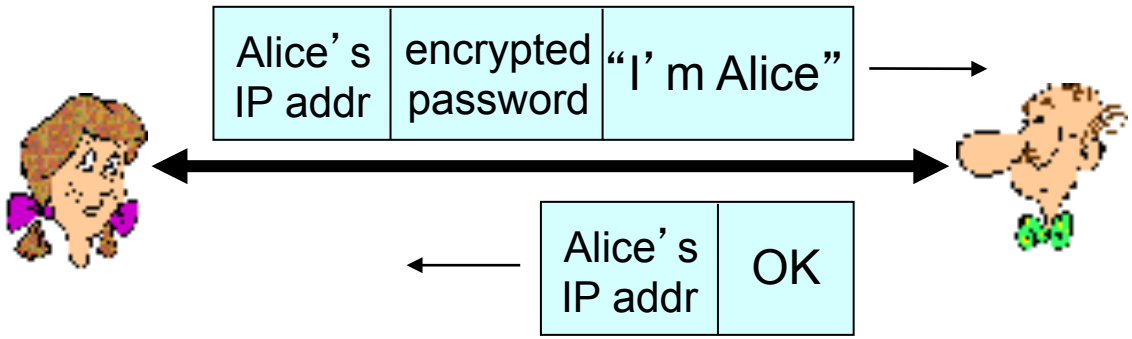
Protocol ap3.0: Alice says "I am Alice" and sends her secret password to "prove" it.



playback attack: Trudy records Alice's packet and later plays it back to Bob

Authentication: Yet Another Try

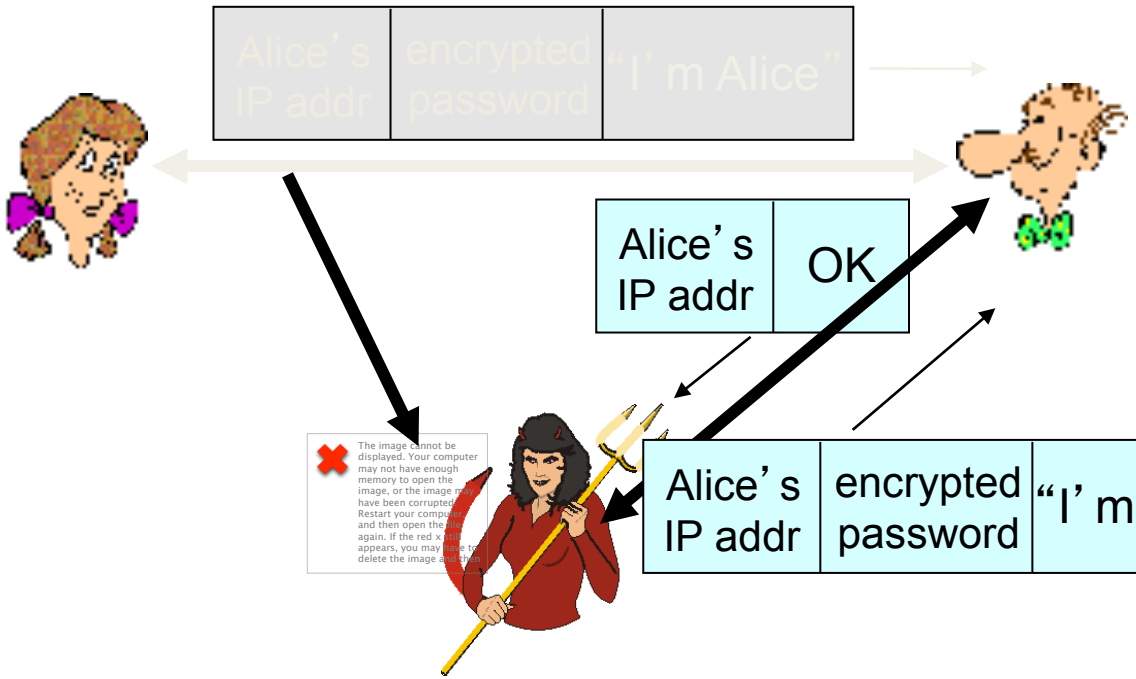
Protocol ap3.1: Alice says "I am Alice" and sends her *encrypted* secret password to "prove" it.



Failure scenario??

Authentication: Yet Another Try

Protocol ap3.1: Alice says "I am Alice" and sends her *encrypted* secret password to "prove" it.



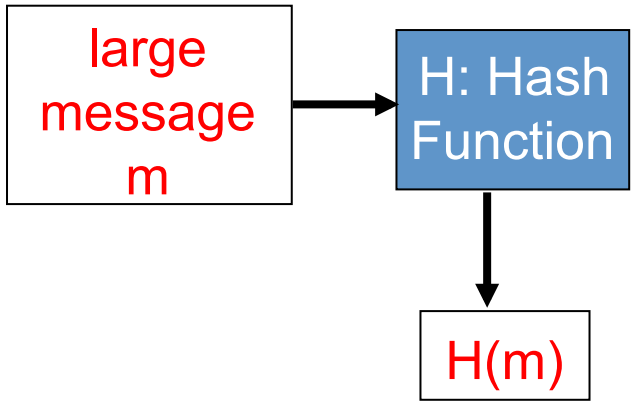
record and playback *still* works!

Message Integrity

- Allows communicating parties to verify that received messages are authentic.
  - Content of message has not been altered
  - Source of message is who/what you think it is
  - Message has not been replayed
  - Sequence of messages is maintained
- Let's first talk about message digests

Message Digests

- Function H( ) that takes as input an arbitrary length message and outputs a fixed-length string: "message signature"
- Note that H( ) is a many-to-1 function → collisions are unavoidable
- H( ) is often called a "hash function"
- however, finding collisions are difficult → the hash value of a message can serve as a compact representative image of the message (similar to fingerprints)



- Desirable properties:
  - Easy to calculate
  - Irreversibility: Can't determine m from H(m)
  - Collision resistance: Computationally difficult to produce m and m' such that H(m) = H(m')
  - Seemingly random output

Desirable Properties of Hash Functions

- **Ease of computation**
  - given an input x, the hash value h(x) of x is easy to compute
- **Weak collision resistance (2nd preimage resistance)**
  - given an input x, it is computationally infeasible to find a second input x' such that h(x') = h(x)
- **Strong collision resistance (collision resistance)**
  - it is computationally infeasible to find any two distinct inputs x and x' such that h(x) = h(x')
- **One-way property (preimage resistance)**
  - given a hash value y (for which no preimage is known), it is computationally infeasible to find any input x s.t. h(x) = y

The Birthday Paradox

- Given a set of N elements, from which we draw k elements randomly (with replacement). What is the probability of encountering at least one repeating element?
- First, compute the probability of no repetition:
  - the first element x_1 can be anything
  - when choosing the second element x_2, the probability of $x_2 \neq x_1$ is 1-1/N
  - when choosing x_3, the probability of $x_3 \neq x_2$ and $x_3 \neq x_1$ is 1-2/N
  - …
  - when choosing the k-th element, the probability of no repetition is 1-(k-1)/N
  - the probability of no repetition is $(1 - 1/N)(1 - 2/N)\ldots(1 – (k-1)/N)$
  - when x is small, $(1-x) \approx e^{-x}$
  - $(1 - 1/N)(1 - 2/N)\ldots(1 – (k-1)/N) = e^{-1/N}e^{-2/N} \ldots e^{-(k-1)/N} = e^{-k(k-1)/2N}$
- The probability of at least one repetition after k drawing is

$$1 – e^{-k(k-1)/2N}$$

The Birthday Paradox (cont'd)

- How many drawings do you need, if you want the probability of at least one repetition to be ε ?
- Solve the following for k:

$$\varepsilon = 1 - e^{-k(k-1)/2N}$$

$$k(k-1) = 2N \ln(1/1-\varepsilon)$$

$$k \approx \text{sqrt}(2N \ln(1/1-\varepsilon))$$

- Examples:
  - $\varepsilon = ½ \rightarrow k \approx 1.177 \text{ sqrt}(N)$
  - $\varepsilon = ¾ \rightarrow k \approx 1.665 \text{ sqrt}(N)$
  - $\varepsilon = 0.9 \rightarrow k \approx 2.146 \text{ sqrt}(N)$
- **Origin of the name "Birthday Paradox":**
  - elements are dates in a year (N = 365)
  - among 1.177 sqrt(365) ≈ 23 randomly selected people, there will be at least two that have the same birthday with probability ½

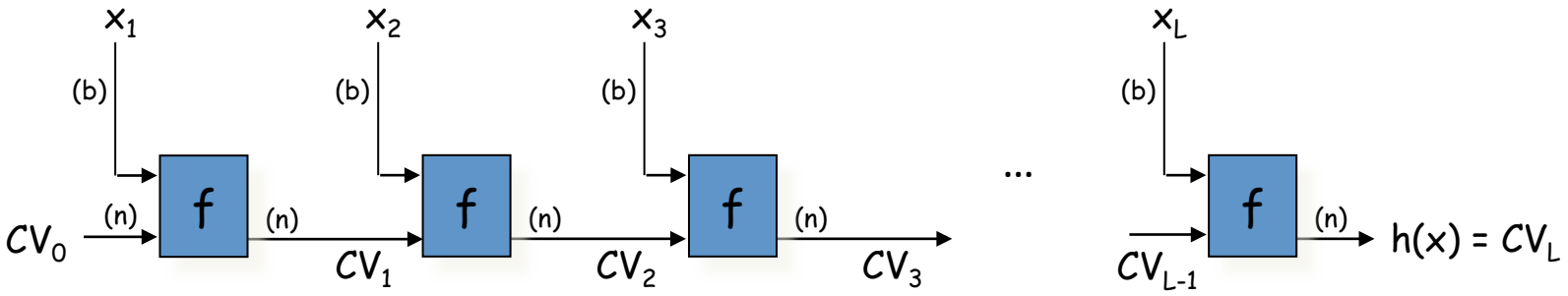
Choosing the Output Size of a Hash Function

- The Birthday Paradox have a profound impact on the design of hash functions (and other cryptographic algorithms and protocols)!
  - Let n be the output size of a hash function
  - **Among ~sqrt(2^n) = $2^{n/2}$ randomly chosen messages, with high probability, there will be a collision pair**
  - It is easier to find collisions than to find preimages or 2nd preimages for a given hash value

  → *in order to resist birthday attacks, $2^{n/2}$ should be sufficiently large (e.g., n = 160 bits)*

Iterated Hash Functions

- Input is divided into fixed length blocks $x_1, x_2, \ldots, x_L$
- Last block is padded if necessary
  - Merkle-Damgard strengthening: padding contains the length of the message
- Each input block is processed according to the following scheme

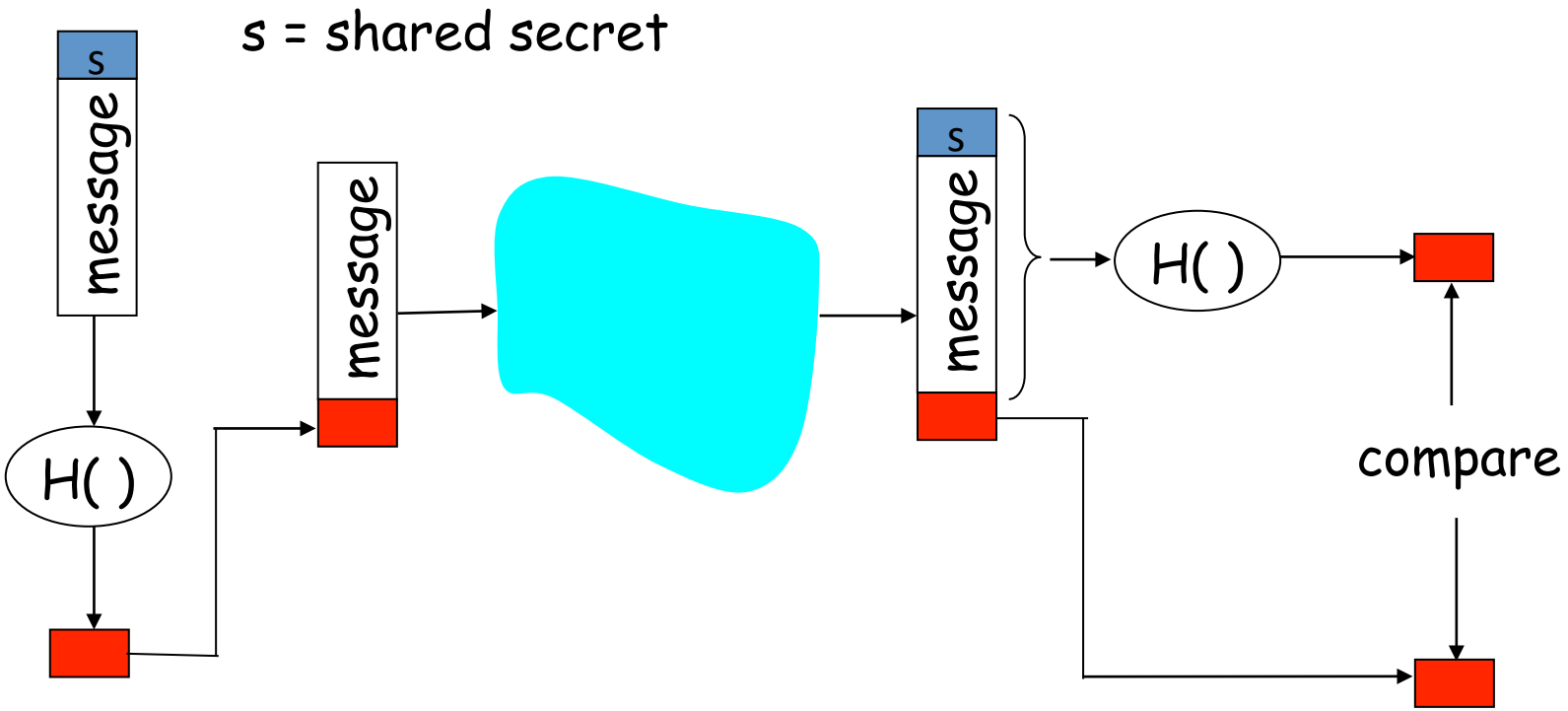


- f is called the compression function
  - can be based on a block cipher, or
  - can be a dedicated compression function

Hash Function Algorithms

- MD5 hash function widely used (RFC 1321)
  - computes 128-bit message digest in 4-step process.
- SHA-1 is also used.
  - US standard [NIST, FIPS PUB 180-1]
  - 160-bit message digest

Message Authentication Code (MAC)



- ***Authenticates sender***
- ***Verifies message integrity***
- No encryption !
- Also called "keyed hash"
- Notation: $MD_m = H(s||m)$; send $m||MD_m$

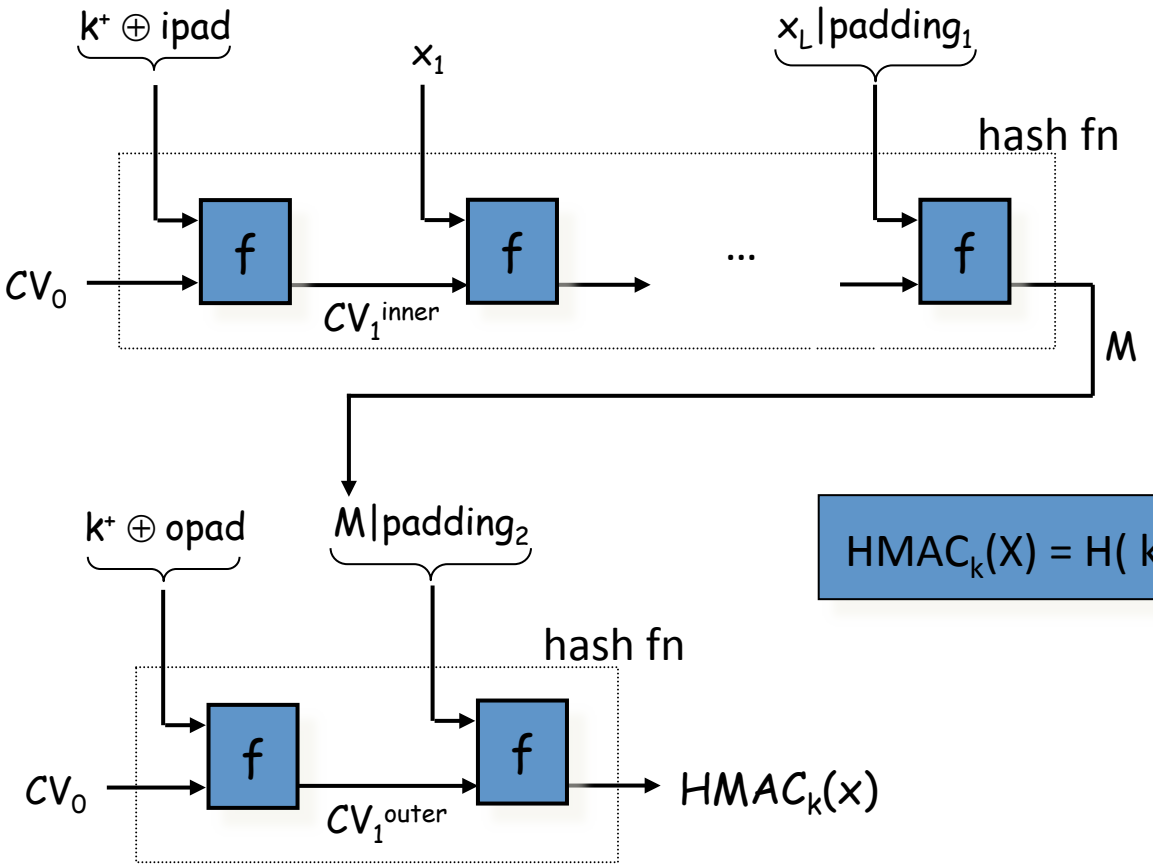
Desirable Properties of MAC Functions

- **Ease of computation**
  - given an input x and a secret key k, it is easy to compute $MAC_k(x)$
- **Key non-recovery**
  - it is computationally infeasible to recover the secret key k, given one or more text-MAC pairs $(x_i, MAC_k(x_i))$ for that k
- **Computation resistance**
  - given zero or more text-MAC pairs $(x_i, MAC_k(x_i))$, it is computationally infeasible to find a text-MAC pair $(x, MAC_k(x))$ for any new input $x \neq x_i$
  - computation resistance implies key non-recovery but the reverse is not true in general

HMAC

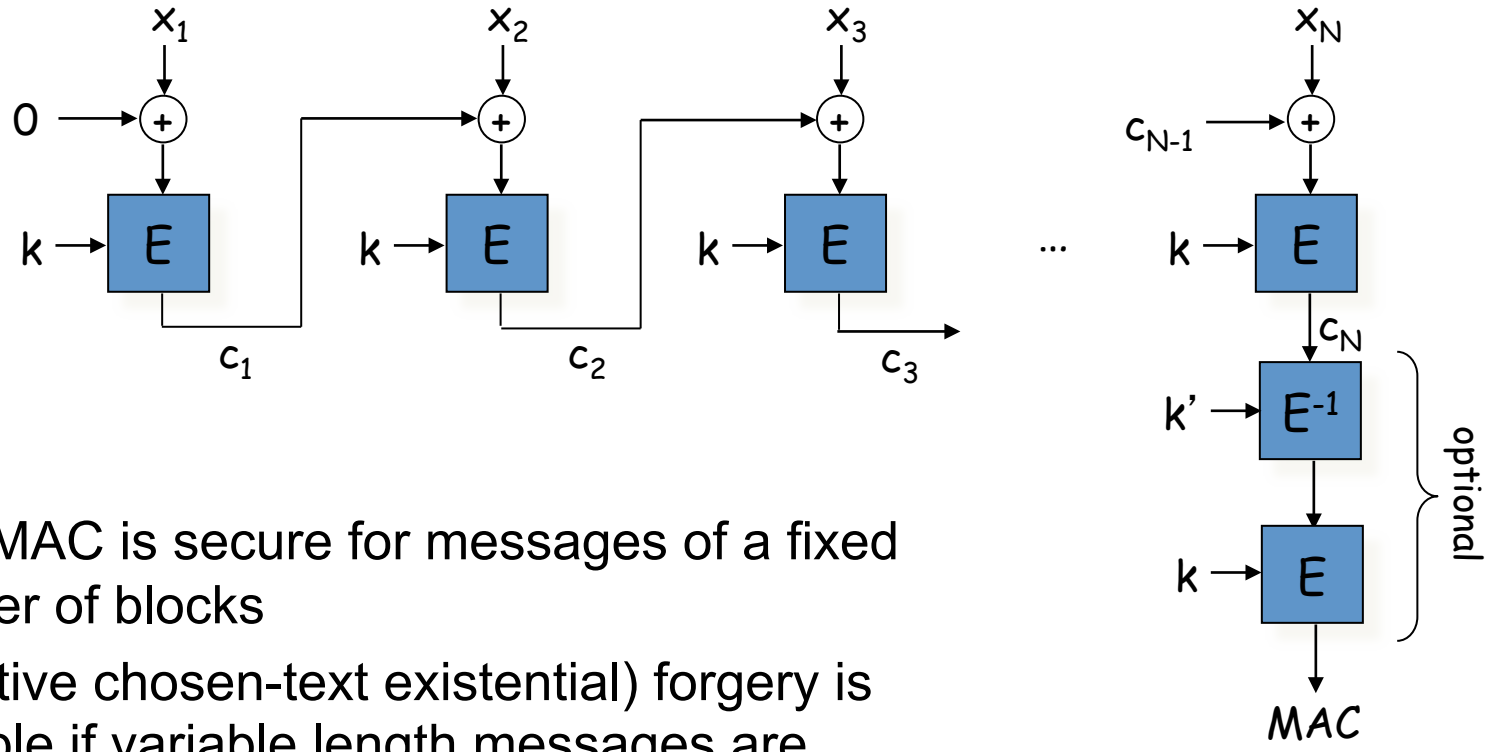
- Popular MAC standard
- Addresses some subtle security flaws
- How it works:
  1. Concatenates secret to front of message.
  2. Hashes concatenated message
  3. Concatenates the secret to front of digest
  4. Hashes the combination again.

HMAC



$HMAC_k(X) = H(k'' | H(k' | X))$

CBC MAC



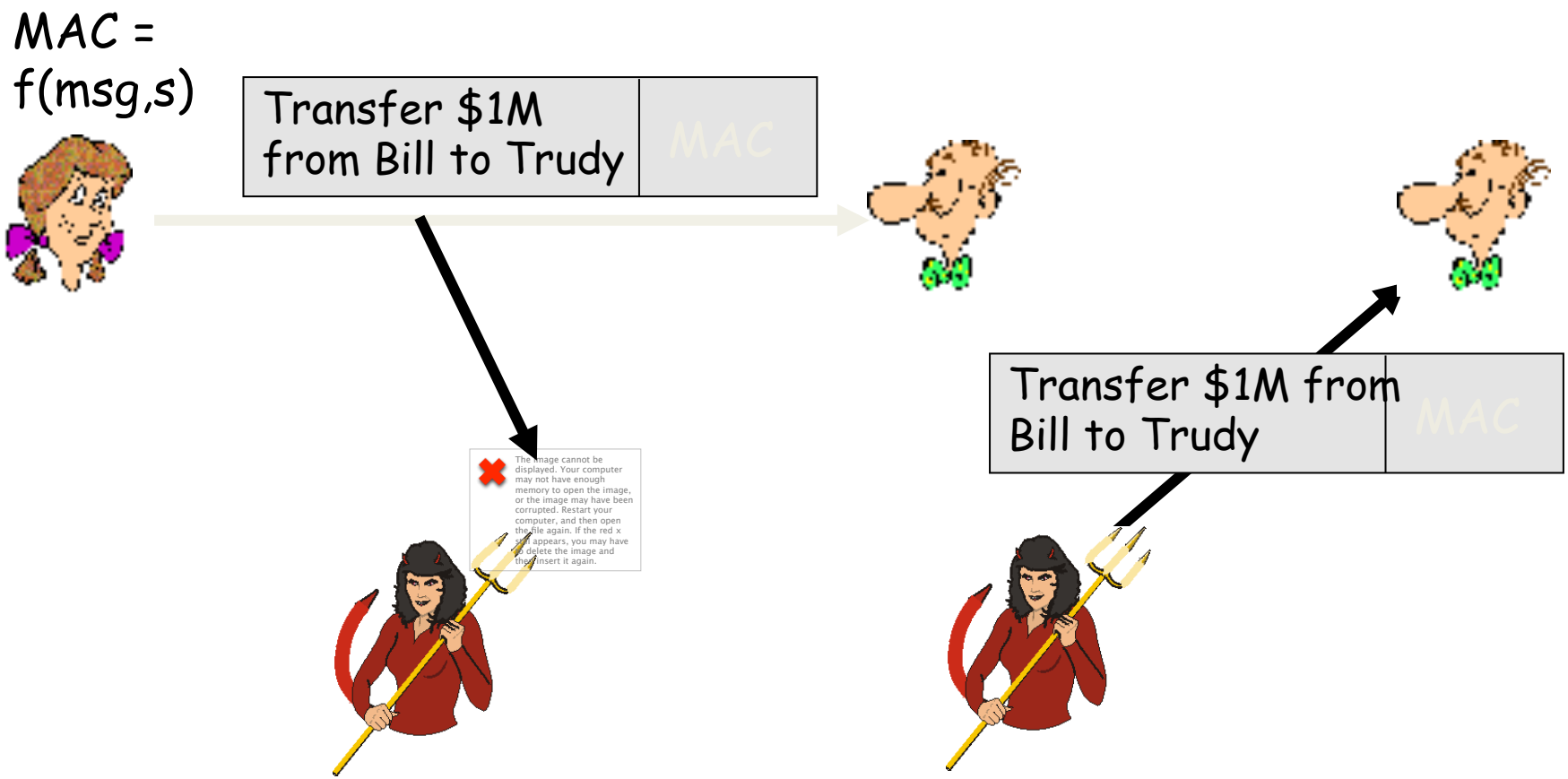
- CBC MAC is secure for messages of a fixed number of blocks
- (adaptive chosen-text existential) forgery is possible if variable length messages are allowed

→ it is recommended to involve the length of the message in the CBC MAC computation

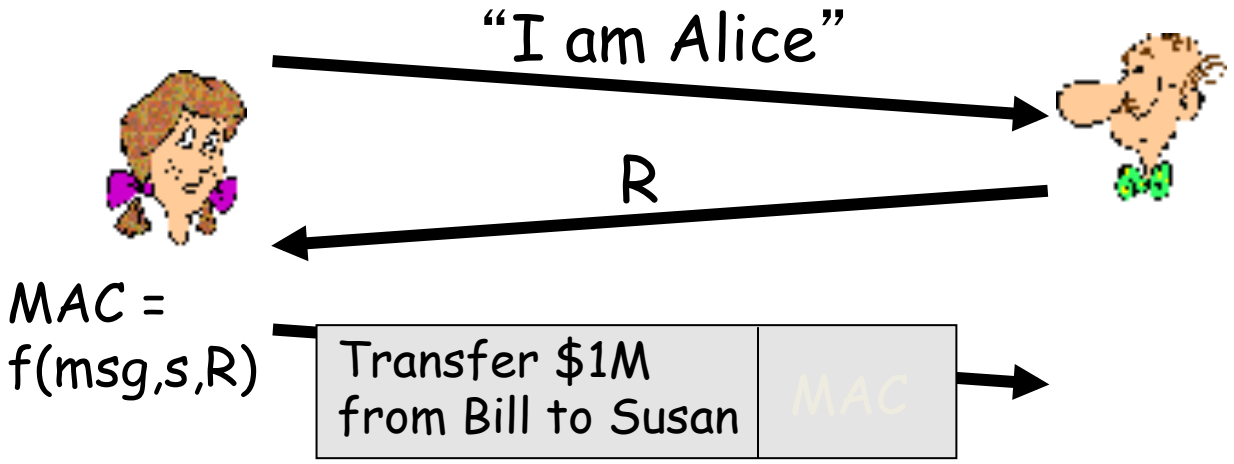
End-point Authentication

- Want to be sure of the originator of the message – *end-point authentication*.
- Assuming Alice and Bob have a shared secret, will MAC provide end-point authentication.
  - *We do know that Alice created the message.*
  - But did she send it?

Playback attack



Defending against playback attack: nonce

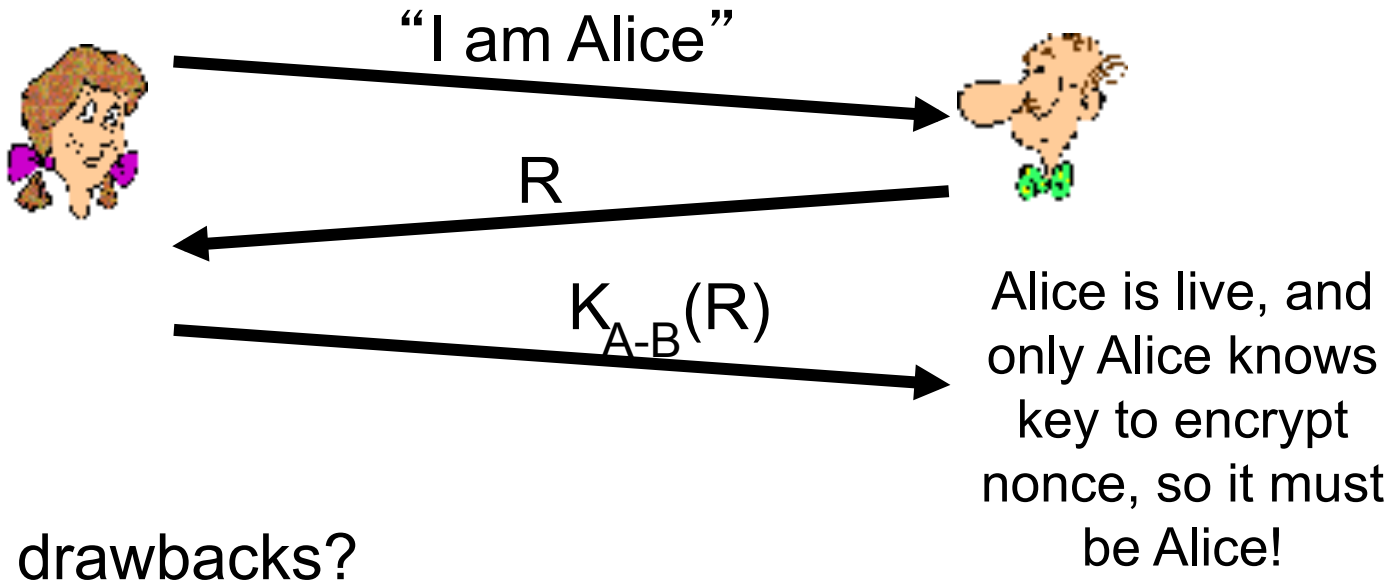


Authentication: yet another try

Goal: avoid playback attack

nonce: number (R) used only *once-in-a-lifetime*

ap4.0: to prove Alice "live", Bob sends Alice *nonce*, R. Alice must return R, encrypted with shared secret key



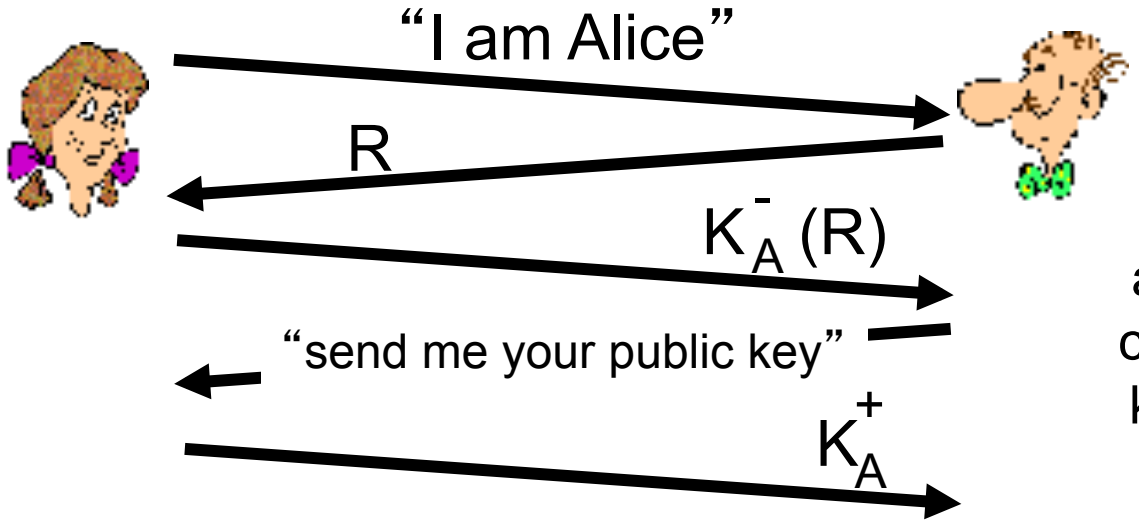
Failures, drawbacks?

Authentication: ap5.0

ap4.0 requires shared symmetric key

- can we authenticate using public key techniques?

ap5.0: use nonce, public key cryptography



Bob computes

$K_A^+(K_A^-(R)) = R$

and knows only Alice could have the private key, that encrypted R such that

$K_A^+(K_A^-(R)) = R$

Contents

- Asymmetric-key Encryption
- Message Authentication Codes and Hash Function
- Digital Signature
- Session Key Establishment Protocols
- Pseudo-Random Generator
- Advanced Authentication Techniques

Digital Signatures

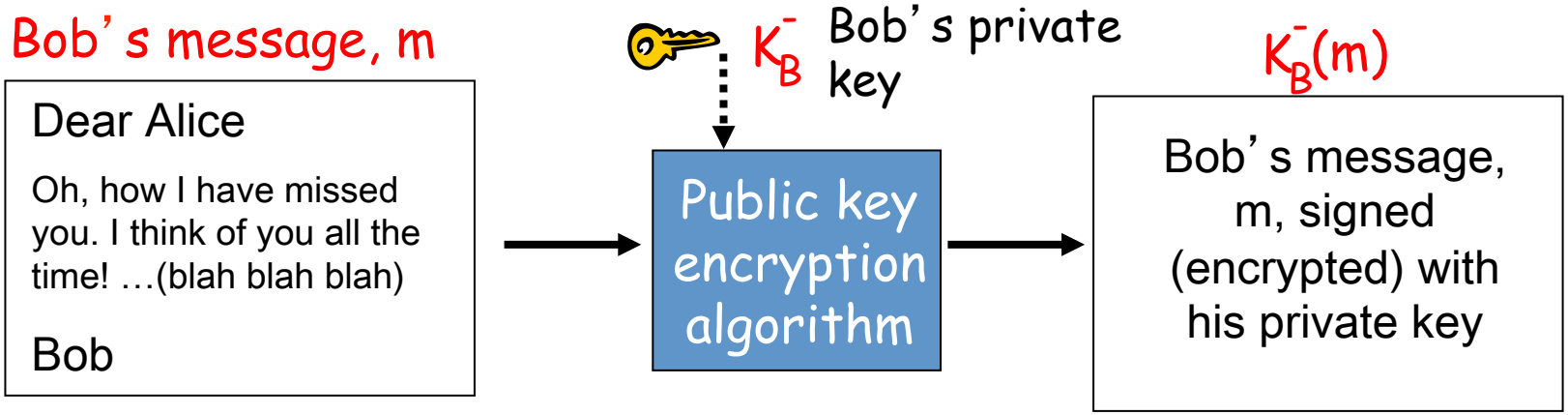
Cryptographic technique analogous to hand-written signatures.

- Sender (Bob) digitally signs document, establishing he is document owner/creator.
- Goal is **similar to that of a MAC**, except **now use public-key** cryptography
- Verifiable, nonforgeable: recipient (Alice) can prove to someone that Bob, and no one else (including Alice), must have signed document

Digital Signatures

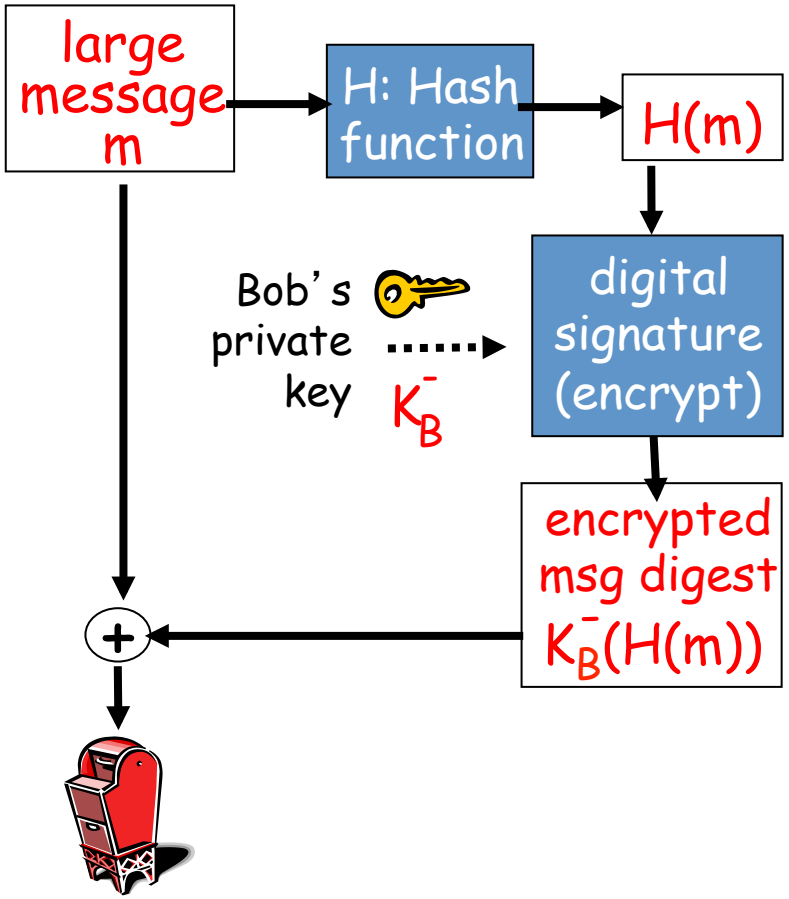
Simple digital signature for message m:

- Bob signs m by encrypting with his private key K_B^-, creating "signed" message, $K_B^-(m)$

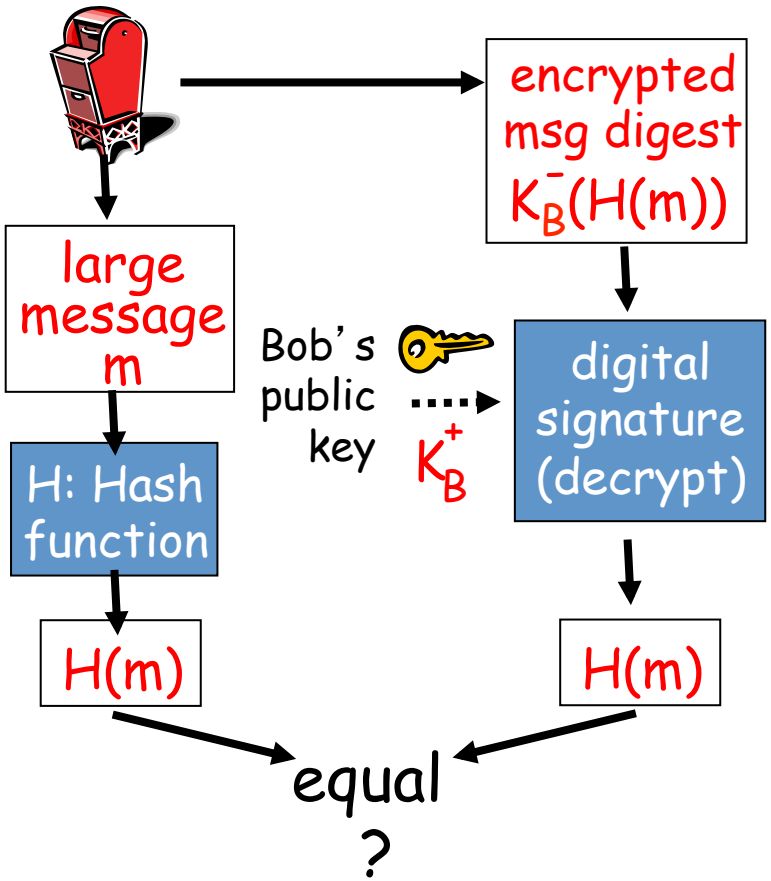


Digital Signature = Signed Message Digest

Bob sends digitally signed message:



Alice verifies signature and integrity of digitally signed message:



Digital Signatures (more)

- Suppose Alice receives msg m, digital signature $K_B^-(m)$
- Alice verifies m signed by Bob by applying Bob's public key K_B^+ to $K_B^-(m)$ then checks $K_B^+(K_B^-(m)) = m$.
- If $K_B^+(K_B^-(m)) = m$, whoever signed m must have used Bob's private key.

Alice thus verifies that:

- Bob signed m.
- No one else signed m.
- Bob signed m and not m'.

Non-repudiation:

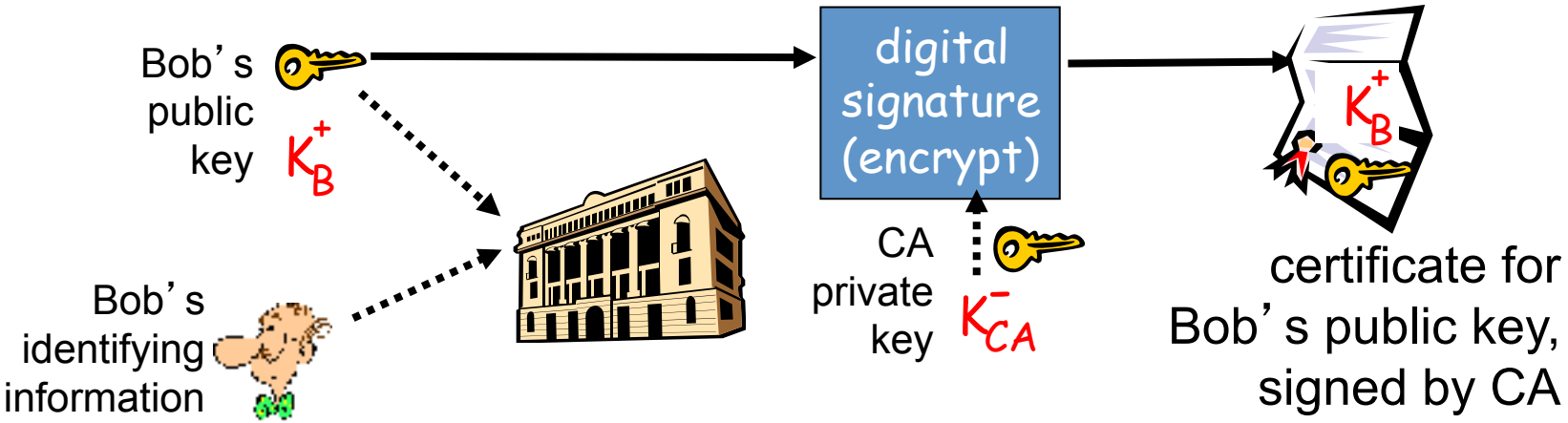
- ✓ Alice can take m, and signature $K_B^-(m)$ to court and prove that Bob signed m.

Public-key Certification

- Motivation: Trudy plays pizza prank on Bob
  - Trudy creates e-mail order:
    Dear Pizza Store, Please deliver to me four pepperoni pizzas. Thank you, Bob
  - Trudy signs order with her private key
  - Trudy sends order to Pizza Store
  - Trudy sends to Pizza Store her public key, but says it's Bob's public key.
  - Pizza Store verifies signature; then delivers four pizzas to Bob.
  - Bob doesn't even like Pepperoni

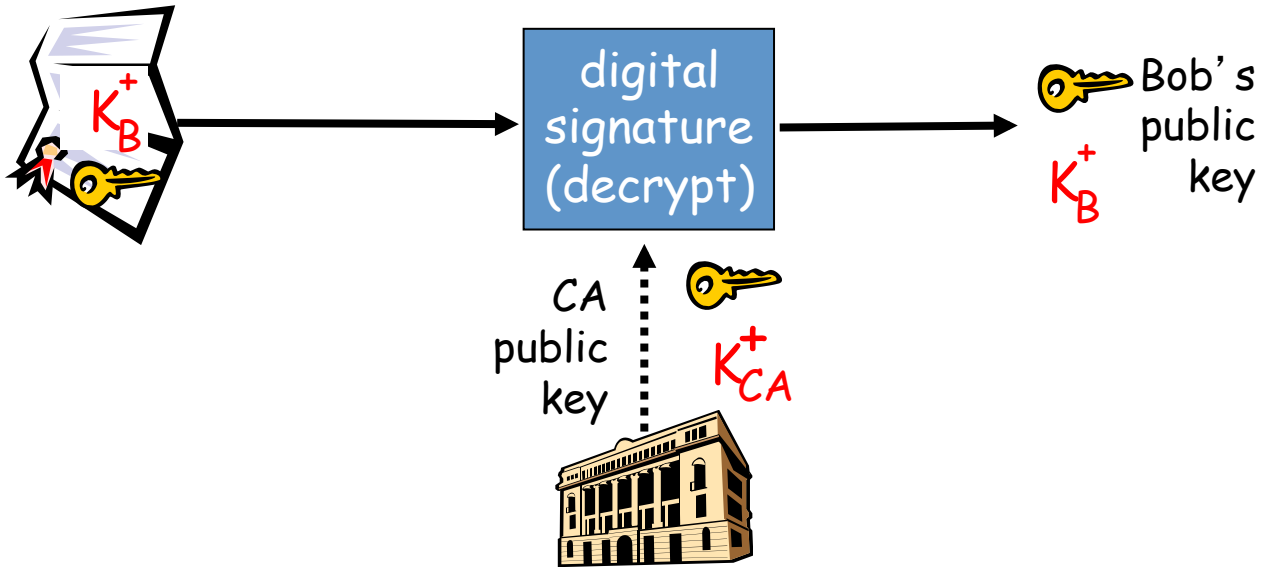
Certification Authorities

- Certification authority (CA): binds public key to particular entity, E.
- E (person) registers its public key with CA.
  - E provides "proof of identity" to CA.
  - CA creates certificate binding E to its public key.
  - Certificate containing E's public key digitally signed by CA – CA says "this is E's public key"



Certification Authorities

- When Alice wants Bob's public key:
  - Gets Bob's certificate (Bob or elsewhere).
  - Apply CA's public key to Bob's certificate, get Bob's public key

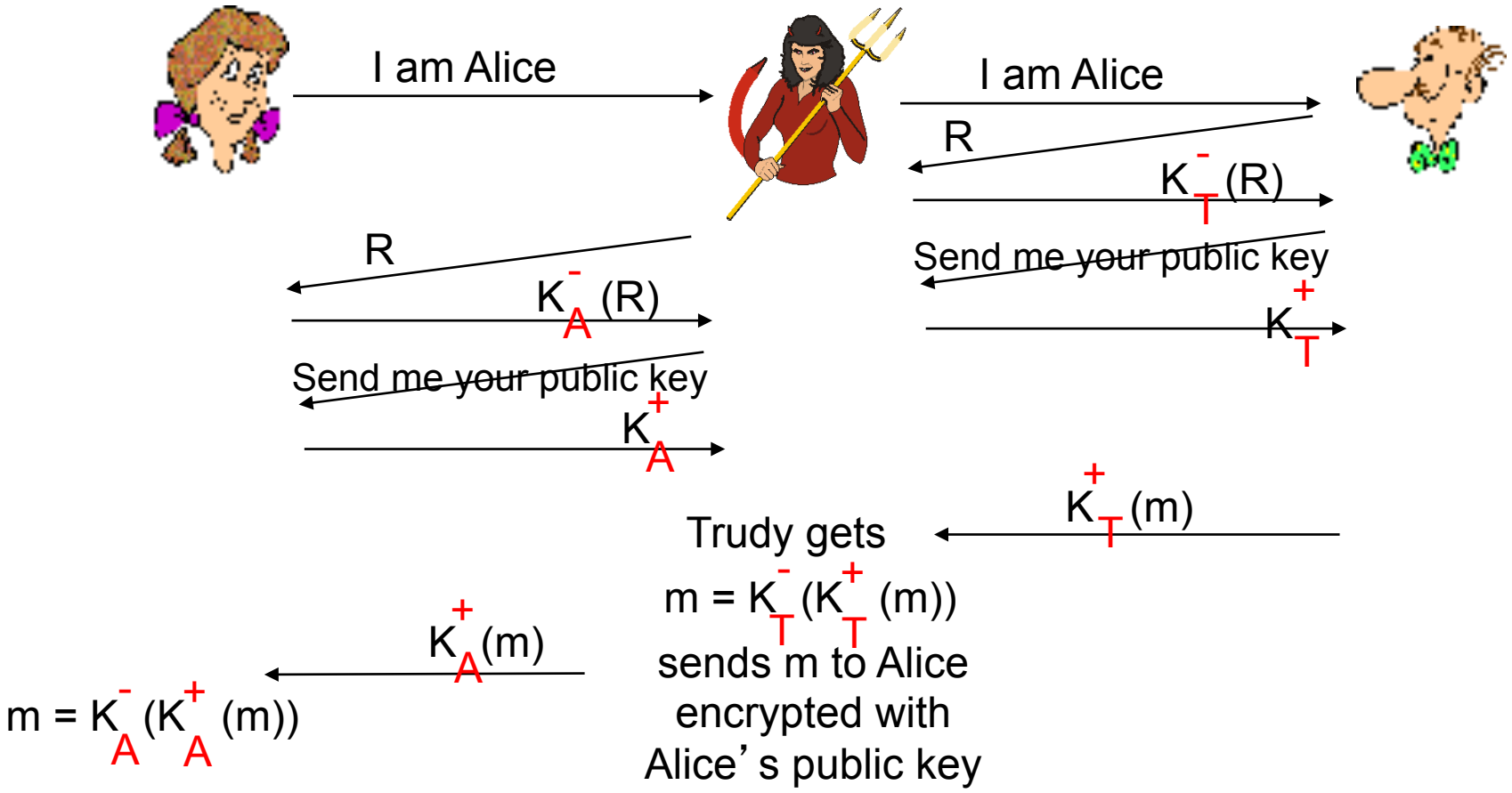


Certificates: Summary

- Primary standard X.509 (RFC 2459)
- Certificate contains:
  - Issuer name
  - Entity name, address, domain name, etc.
  - Entity's public key
  - Digital signature (signed with issuer's private key)
- Public-Key Infrastructure (PKI)
  - Certificates and certification authorities
  - Often considered "heavy"

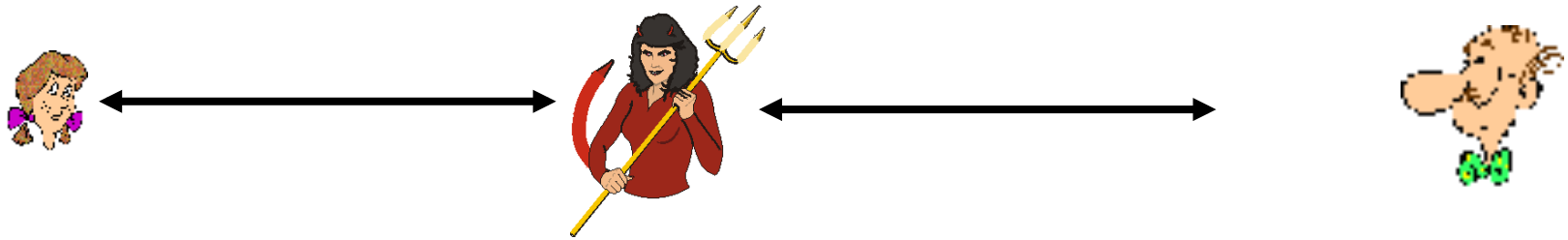
Security Hole

man (or woman) in the middle attack: Trudy poses as Alice (to Bob) and as Bob (to Alice)



Security Hole

man (or woman) in the middle attack: Trudy poses as Alice (to Bob) and as Bob (to Alice)



difficult to detect:

- Bob receives everything that Alice sends, and vice versa. (e.g., so Bob, Alice can meet one week later and recall conversation!)
- problem is that Trudy receives all messages as well!

Attacks on Digital Signature Schemes

- **Key-only attack**
  - only the public key is available to the adversary
- **Known-message attack**
  - the adversary has signatures for a set of messages known to her but not chosen by her
- **Chosen-message attack**
  - the adversary obtains signatures for messages chosen by her before attempting to break the signature scheme
- **Adaptive chosen-message attack**
  - the adversary is allowed to use the signer as an oracle
  - she may request signatures for messages which depend on previously obtained signatures

Examples of Digital Signature Scheme

- **RSA**
  - essentially identical to the RSA encryption scheme
  - signature = decryption with private key
  - typical signature length is 1024 bits
- **DSA (Digital Signature Algorithm)**
  - based on the ElGamal signature scheme
  - typical signature length is 1024 bits
- **ECDSA (Elliptic Curve DSA)**
  - same as DSA but works over elliptic curves
  - reduced signature length (typically 320 bits)

Contents

- Asymmetric-key Encryption
- Message Authentication Codes and Hash Function
- Digital Signature
- Session Key Establishment Protocols
- Pseudo-Random Generator
- Advanced Authentication Techniques

Session Key Establishment Protocols

- **Goal of session key establishment protocols**
  - to **setup a shared secret** between two (or more) parties
  - it is desired that the secret established by a fixed pair of parties **varies on subsequent** executions of the protocol (dynamicity)
  - established shared secret is used as a ***session key*** to protect communication between the parties

- **Motivation for use of session keys**
  - to **limit available ciphertext** for cryptanalysis
  - to **limit exposure caused by the compromise** of a session key
  - to **avoid long-term storage** of a large number of secret keys (keys are created on-demand when actually required)
  - to **create independence across communication** sessions or applications

Basic classification

- **key transport protocols**
  - One party creates or otherwise obtains a secret value, and securely transfers it to the other party

- **key agreement protocols**
  - A shared secret is derived by the parties as a function of information contributed by each, such that no party can predetermine the resulting value

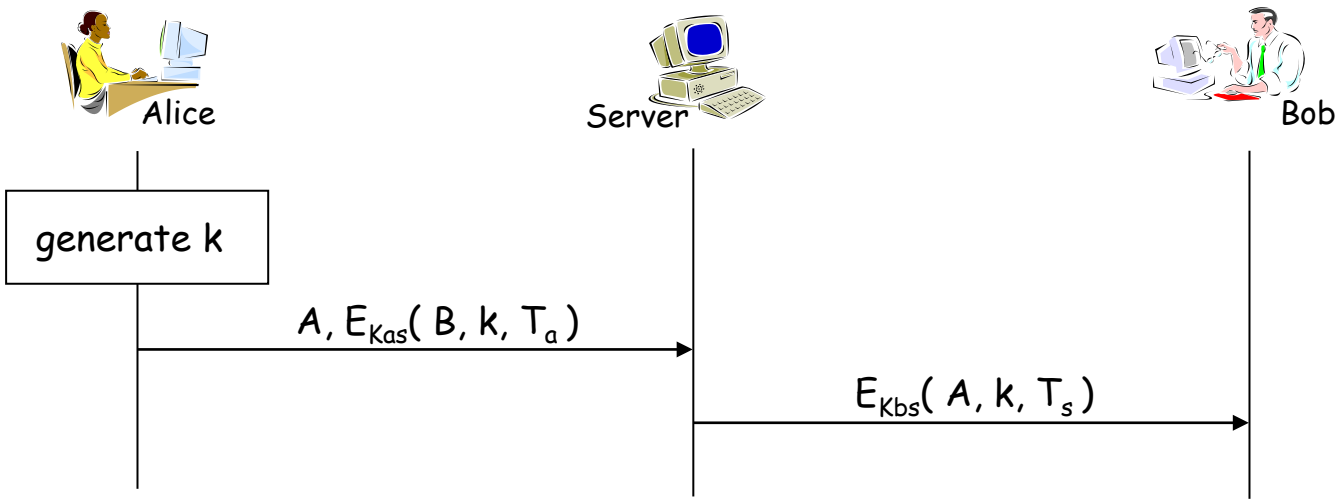
Further services

- **Entity authentication**
- **Implicit key authentication**
  - one party is assured that no other party aside from a specifically identified second party (and possibly some trusted third parties) may gain access to the established session key
- **Key confirmation**
  - one party is assured that a second (possibly unidentified) party actually possesses the session key
  - possession of a key can be demonstrated by
    - producing a one-way hash value of the key or
    - encryption of known data with the key
- **Explicit key authentication**
  - implicit key authentication + key confirmation
- **Key freshness**
  - one party is assured that the key is new (never used before)

Further Protocol Characteristics

- **Reciprocity**
  - guarantees are provided unilaterally
  - guarantees are provided mutually
- **Efficiency**
  - number of message exchanges (passes) required
  - total number of bits transmitted (i.e., bandwidth used)
  - complexity of computations by each party
  - possibility of precomputations to reduce on-line computational complexity
- **Third party requirements**
  - on-line, off-line, or no third party at all
  - degree and type of trust required in the third party
- **System setup**
  - distribution of initial keying material

The Wide-Mouth-Frog Protocol



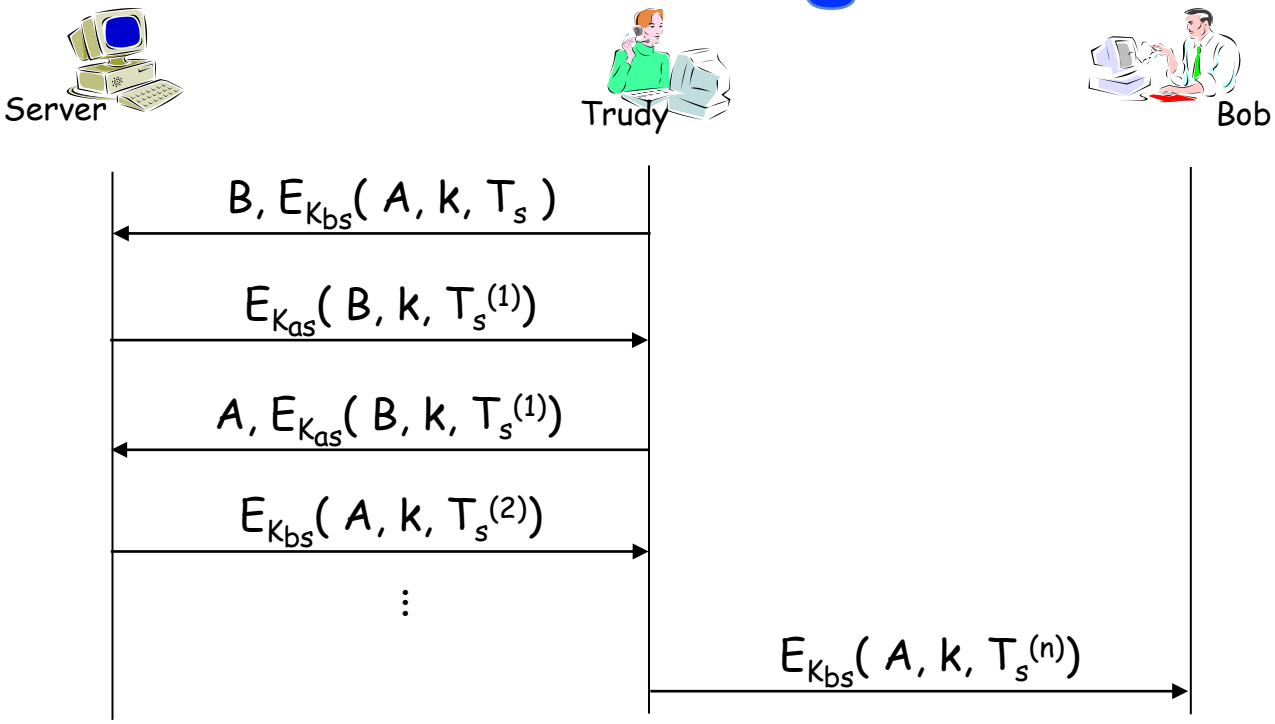
Summary: a simple key transport protocol that uses **a trusted third party**

Alice generates the session key and sends it to Bob via the trusted third party

Characteristics:

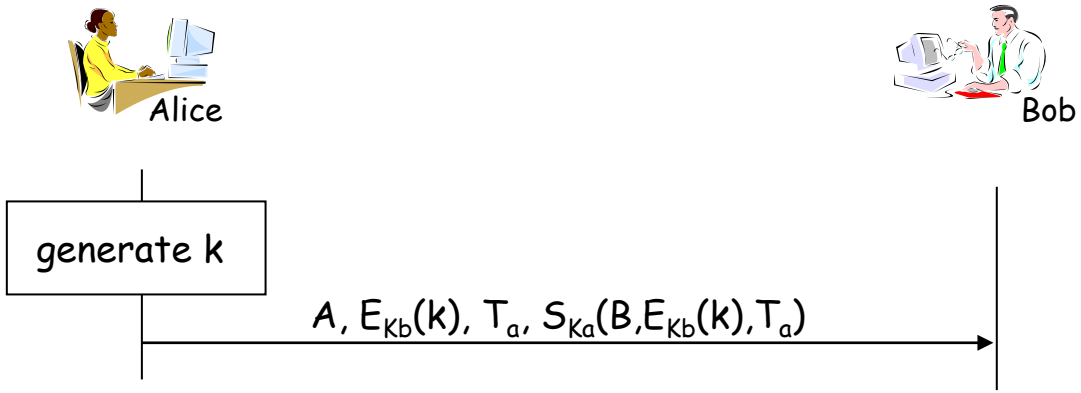
- ✧ Implicit key authentication for Alice
- ✧ Explicit key authentication for Bob
- ✧ Key freshness for Bob with timestamps (flawed)
- ✧ Unilateral entity authentication of Alice
- ✧ On-line third party (Server) trusted for secure relaying of keys and verification of freshness
- ✧ In addition A is trusted for generating good keys
- ✧ Initial long-term keys between the parties and the server are required

A Flaw in the Wide-Mouth-Frog Protocol



Summary: After observing one run of the protocol, Trudy can continuously use the Server as an oracle until she wants to bring about re-authentication between Alice and Bob

Signed Encrypted Keys

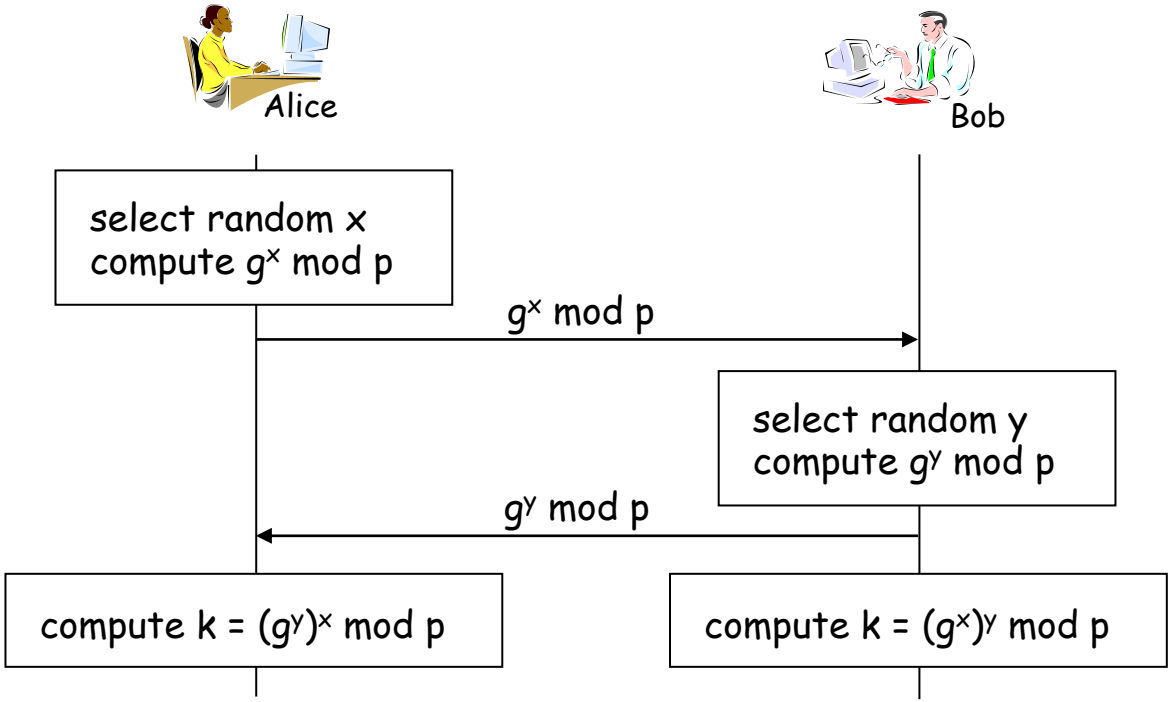


Summary: Alice generates a session key, encrypts it with Bob's public key, then sings it, and sends it to Bob

Characteristics:

- ✧ Unilateral entity authentication (of Alice)
- ✧ Mutual implicit key authentication
- ✧ No key confirmation, key freshness with timestamp, clock synchronization
- ✧ Off-line third party for issuing public key certificates may be required
- ✧ Initial exchange of public keys between the parties may be required
- ✧ Alice is trusted to generate keys
- ✧ Non-repudiation guarantee for Bob

The Diffie-Hellman protocol



Summary: a key agreement protocol based on one-way functions; in particular, security of the protocol is based on the hardness of the discrete logarithm problem and that of the Diffie-Hellman problem

Assumptions: p is a large prime, g is a generator of Z_p^*, both are publicly known system parameters

Characteristics: NO AUTHENTICATION, key freshness with randomly selected exponents, no party can control the key, no need for a trusted third party

Contents

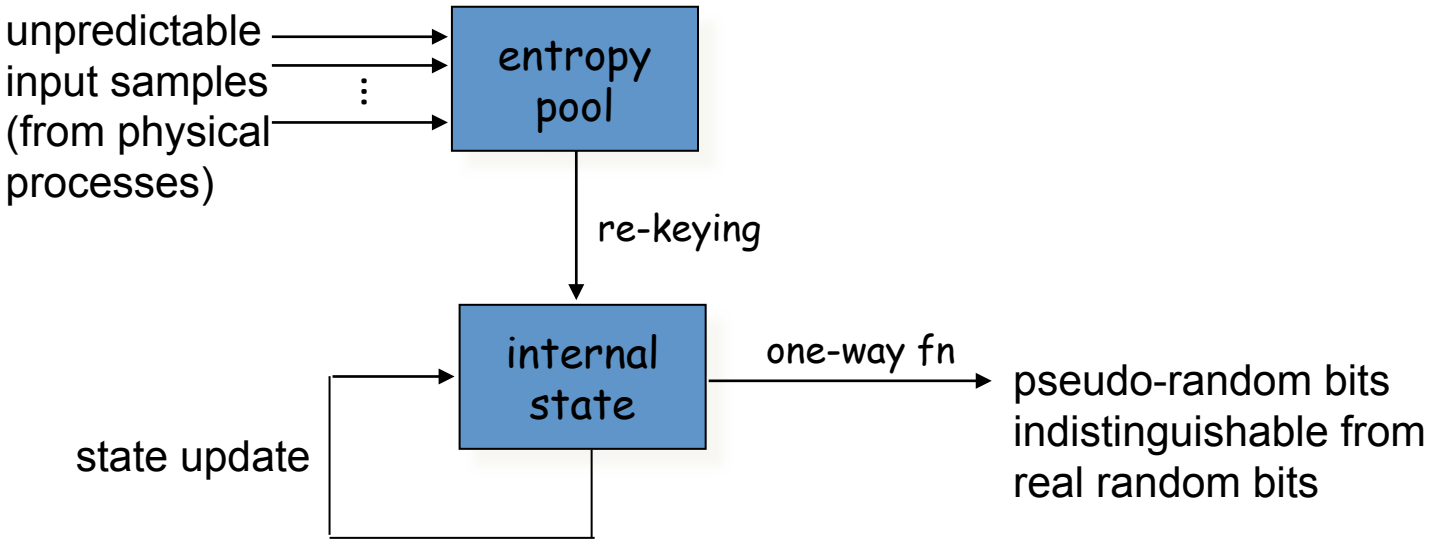
- Asymmetric-key Encryption
- Message Authentication Codes and Hash Function
- Digital Signature
- Session Key Establishment Protocols
- Pseudo-Random Generator
- Advanced Authentication Techniques

Pseudo-random Number Generators (PRNGs)

- A random number is a number that **cannot be predicted** by an observer before it is generated
  - If the number is generated within the range [0, N-1], then its value cannot be predicted with any better probability than 1/N
  - The above is true even if the observer is given all previously generated numbers

- A cryptographic pseudo-random number generator (PRNG) is a mechanism that **processes somewhat unpredictable inputs** and generates pseudo-random outputs
  - If designed, implemented, and used properly, then even an adversary with enormous computational power should not be able to distinguish the PRNG output from a real random sequence

General Operation of PRNGs



Desirable Properties of PRNGs

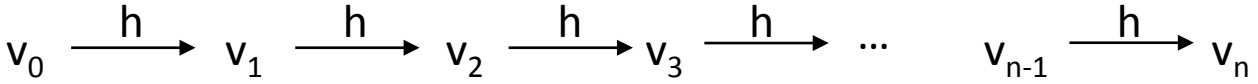
- The adversary cannot compute the internal state of the PRNG, even if she has observed many outputs of the PRNG
- The adversary cannot compute the next output of the PRNG, even if she has observed many previous outputs of the PRNG
- If the adversary can observe or even manipulate the input samples that are fed in the PRNG, but she does not know the internal state of the PRNG, then the adversary cannot compute the next output and the next internal state of the PRNG
- If the adversary has somehow learned the internal state of the PRNG, but she cannot observe the input samples that are fed in the PRNG, then the adversary cannot figure out the internal state of the PRNG after the re-keying operation

Contents

- Asymmetric-key Encryption
- Message Authentication Codes and Hash Function
- Digital Signature
- Session Key Establishment Protocols
- Pseudo-Random Generator
- Advanced Authentication Techniques

Hash chains

- A hash chain is a **sequence of hash values** that are computed by iteratively calling a one-way hash function on an initial value v_0



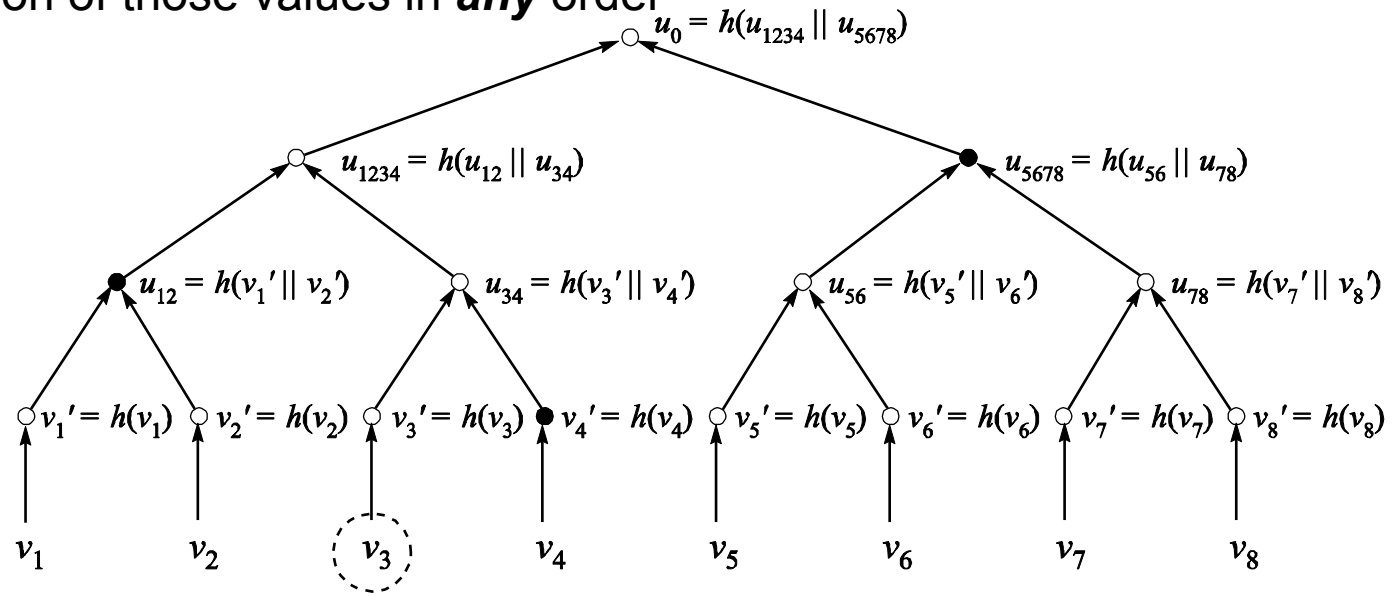
- **Properties:**
  - given v_i, it is easy to compute any v_j for $j > i$ ($v_j = h^{(j-i)}(v_i)$)
  - but it is difficult to compute v_k for $k < i$ (one-way property of h)

- Hash chains can be used for **repeated authentications** at the cost of a single digital signature
  - Alice computes a hash chain and commits to it by signing v_n and distributing it to potential verifiers
  - later on, Alice can authenticate herself repeatedly (at most n times) by revealing the elements of the hash chain in reverse order
  - when v_{n-i} is revealed, verifiers can check if $h^{(i)}(v_{n-i}) = v_n$ (or $h(v_{n-i}) = v_{n-i+1}$ if they remember the last revealed element)
  - each hash chain element can be used only ***once*** for authenticating Alice
  - verifiers are assured that only Alice could have released the next hash chain element

- Hash chains can be **stored efficiently** with a storage complexity that is logarithmic in the length of the hash chain

Merkle-Trees

- The limitation of hash chains is that elements can only be revealed sequentially
- Merkle-trees overcome this problem by allowing for the pre-authentication of a set of values with a single digital signature (on the root u_0 of the tree) and for the revelation of those values in ***any*** order



- When revealing a value v_i, Alice must also reveal all the values assigned to the sibling vertices on the path from v_i' to the root (e.g., v_3 is revealed together with v_4', u_{12}, u_{5678})
- Verifiers hash the revealed values appropriately and check if the result is u_0

- $h(h(u_{12} \| h(h(v_3) \| v'_4)) \| u_{5678})$

TESLA

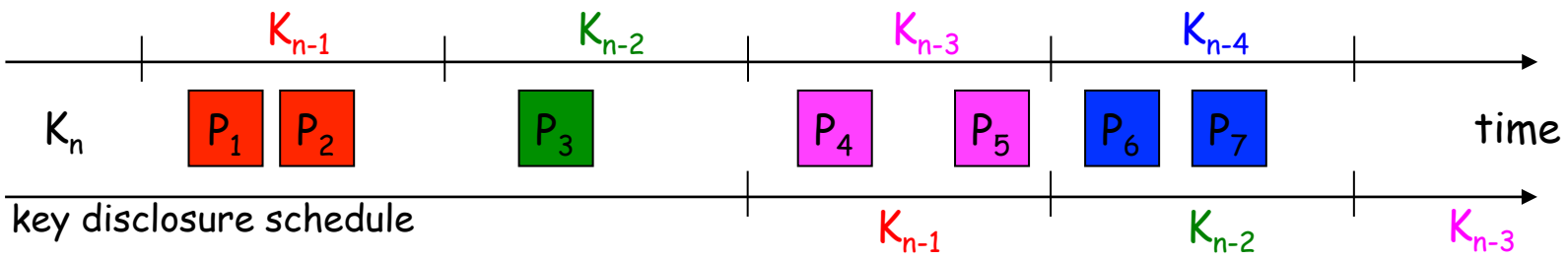
- A **broadcast authentication** mechanism based on symmetric key cryptographic primitives

- **Main Idea:** asymmetry through delayed disclosure of authentication keys
  - Alice wants to broadcast a message m
  - Alice computes a MAC on m with a key unknown to the verifiers
  - Verifiers receive message m with the MAC, but they cannot immediately verify authenticity
  - Later, Alice discloses the key used to compute the MAC
  - Verifiers can now verify the MAC; if it is correct, they know that the message was sent by Alice, because at the time of reception nobody else knew the key

- **Assumptions:**
  - Loose time synchronization between the participants
  - Each party knows an upper bound on the maximum synchronization error
  - Initial secret between the parties to bootstrap the whole mechanism

TESLA (cont'd)

- MAC keys are consecutive elements in a one-way key chain:
  - $K_0 \rightarrow K_1 \rightarrow \ldots \rightarrow K_n$
  - $K_i = h(K_{i-1})$
- protocol operation:
  - setup: Alice sends K_n to each verifier in an authentic manner
  - time is divided into epochs
  - each message sent in epoch i is authenticated with key K_{n-i}
  - K_{n-i} is disclosed in epoch i+d, where d is a system parameter
  - K_{n-i} is verified by checking $h(K_{n-i}) = K_{n-i+1}$
- example:



Conclusions

- Security services are implemented by using security mechanisms
- Many security mechanisms are based on cryptography (e.g., encryption, digital signature, message authentication codes, …)
- Other important aspects are
  - physical protection
  - procedural rules
  - education

But be Cautious!

"If you think cryptography is going to solve your problem, you don't understand cryptography and you don't understand your problem."

-- Roger Needham