CSCI 4250/6250 – Fall 2013
Computer and Networks Security

INTRODUCTION TO CRYPTO
CHAPTER 8 (Goodrich)
CHAPTER 2-6 (Kaufman)
CHAPTER 8 (Kurose)

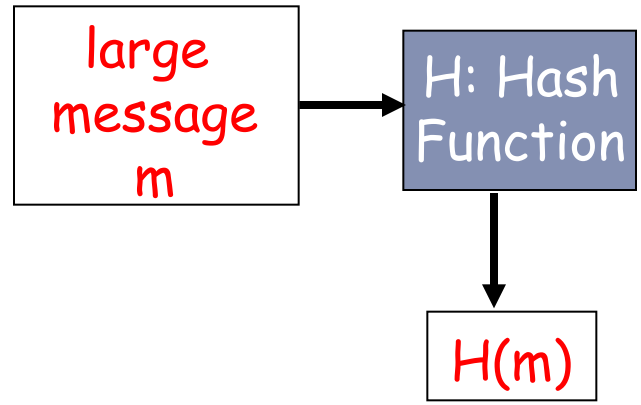
- Slides adapted from Kurose et al., Goodrich et al., and Kaufman et al.

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
- H( ) is often called a "hash function"



- 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***

Internet checksum: poor message digest

Internet checksum has some properties of hash function:

- produces fixed length digest (16-bit sum) of input
- is many-to-one

- But given message with given hash value, it is easy to find another message with same hash value.
- Example: Simplified checksum: add 4-byte chunks at a time:

message	ASCII format
I O U 1	49 4F 55 31
0 0 . 9	30 30 2E 39
9 B O B	39 42 D2 42
	B2 C1 D2 AC

message	ASCII format
I O U 9	49 4F 55 39
0 0 . 1	30 30 2E 31
9 B O B	39 42 D2 42
	B2 C1 D2 AC

different messages but identical checksums!

Hash Functions

- A hash function h maps a plaintext x to a fixed-length value x = h(P) called hash value or digest of P
  - A collision is a pair of plaintexts P and Q that map to the same hash value, h(P) = h(Q)
  - Collisions are unavoidable
  - For efficiency, the computation of the hash function should take time proportional to the length of the input plaintext
- Example of application: Hash table
  - Search data structure based on storing items in locations associated with their hash value
  - Chaining deals with collisions
  - Domain of hash values proportional to the expected number of items to be stored
  - The hash function should spread plaintexts uniformly over the possible hash values to achieve constant expected search time

Cryptographic Hash Functions

- A cryptographic hash function satisfies additional properties
  - Preimage resistance (aka one-way)
    - Given a hash value x, it is hard to find a plaintext P such that h(P) = x
  - Second preimage resistance (aka weak collision resistance)
    - Given a plaintext P, it is hard to find a plaintext Q such that h(Q) = h(P)
  - Collision resistance (aka strong collision resistance)
    - It is hard to find a pair of plaintexts P and Q such that h(Q) = h(P)
- Collision resistance implies second preimage resistance
- Hash values of at least 256 bits recommended to defend against brute-force attacks

How to build a Hash Function

- Can we use a block cipher + CBC?
- How?

How to build a Hash Function

- Can we use a block cipher + CBC?
- How?

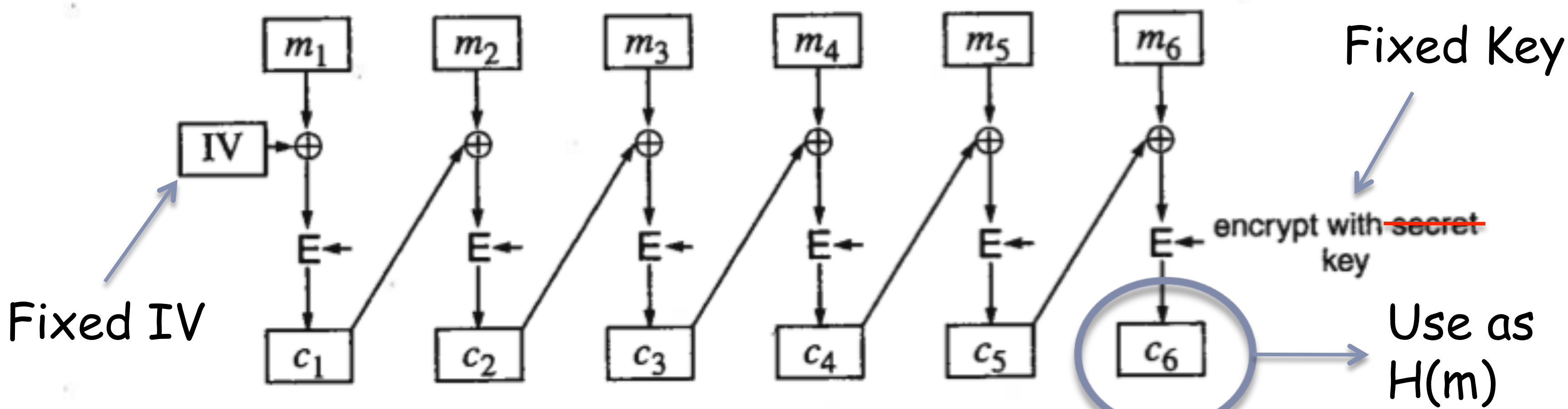


Figure 4-5. Cipher Block Chaining Encryption

- Problem
  - Not very efficient!

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

Often, no good justification for design choices in Hash functions.

Message-Digest Algorithm 5 (MD5)

- Developed by Ron Rivest in 1991
- Uses 128-bit hash values
- Still widely used in legacy applications although considered insecure
- Various severe vulnerabilities discovered
- Chosen-prefix collisions attacks found by Marc Stevens, Arjen Lenstra and Benne de Weger
  - Start with two arbitrary plaintexts P and Q
  - One can compute suffixes S1 and S2 such that P||S1 and Q||S2 collide under MD5 by making 250 hash evaluations
  - Using this approach, a pair of different executable files or PDF documents with the same MD5 hash can be computed

Problems with MD5

- Hash collisions created this way are usually not directly applicable to attack widespread document formats or protocols.
- Attacks are possible by abusing dynamic constructs present in many formats
  - E.g., a malicious document would contain two different messages in the same document, but conditionally displays one or the other
- Computer programs have conditional constructs (if-then-else) that allow testing whether a location in the file has one value or another.
- Some document formats like PostScript, or macros in Microsoft Word, also have conditional constructs.

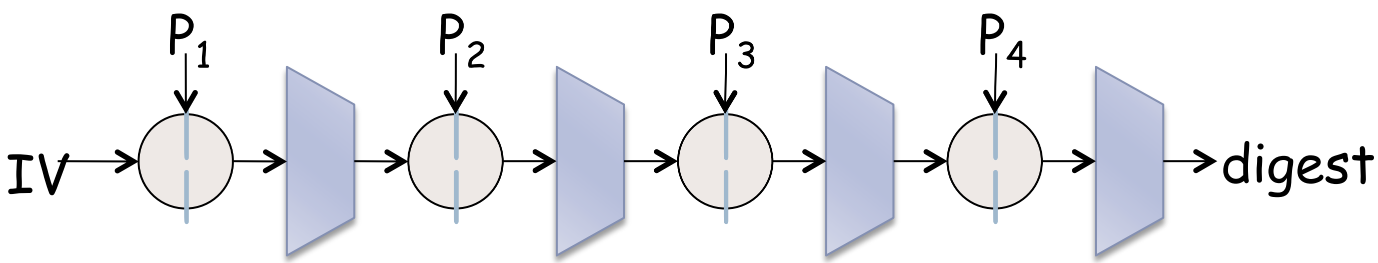
- Finding such colliding docs/programs may take just a few seconds on modern CPUs

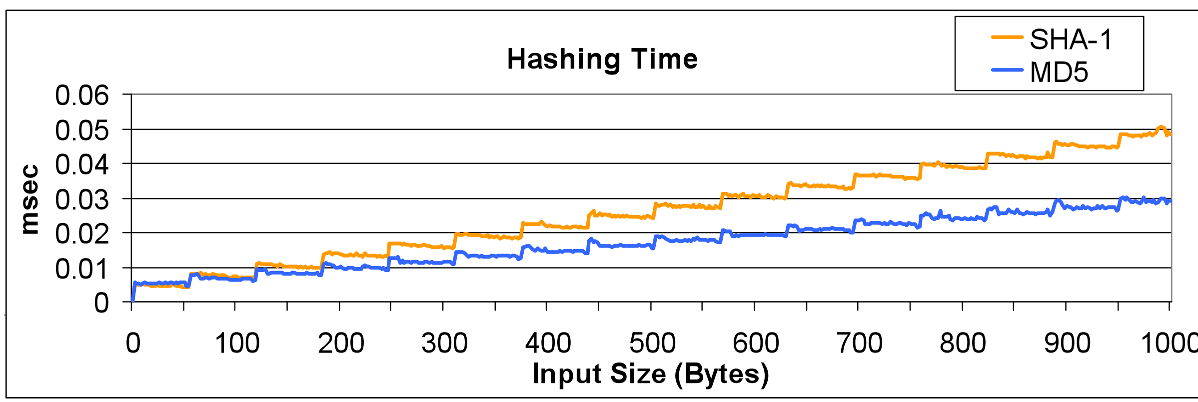
Secure Hash Algorithm (SHA)

- Developed by NSA and approved as a federal standard by NIST
- SHA-0 and SHA-1 (1993)
  - 160-bits
  - Considered insecure
  - Still found in legacy applications
  - Vulnerabilities less severe than those of MD5
- SHA-2 family (2002)
  - 256 bits (SHA-256) or 512 bits (SHA-512)
  - Still considered secure despite published attack techniques
- Public competition for SHA-3 announced in 2007

Iterated Hash Function

- A compression function works on input values of fixed length
  - Inputs: X,Y with len(X)=m, len(Y)=n; Output: Z with len(Z)=n
- An iterated hash function extends a compression function to inputs of arbitrary length
  - padding, initialization vector, and chain of compression functions
  - inherits collision resistance of compression function
- MD5 and SHA are iterated hash functions

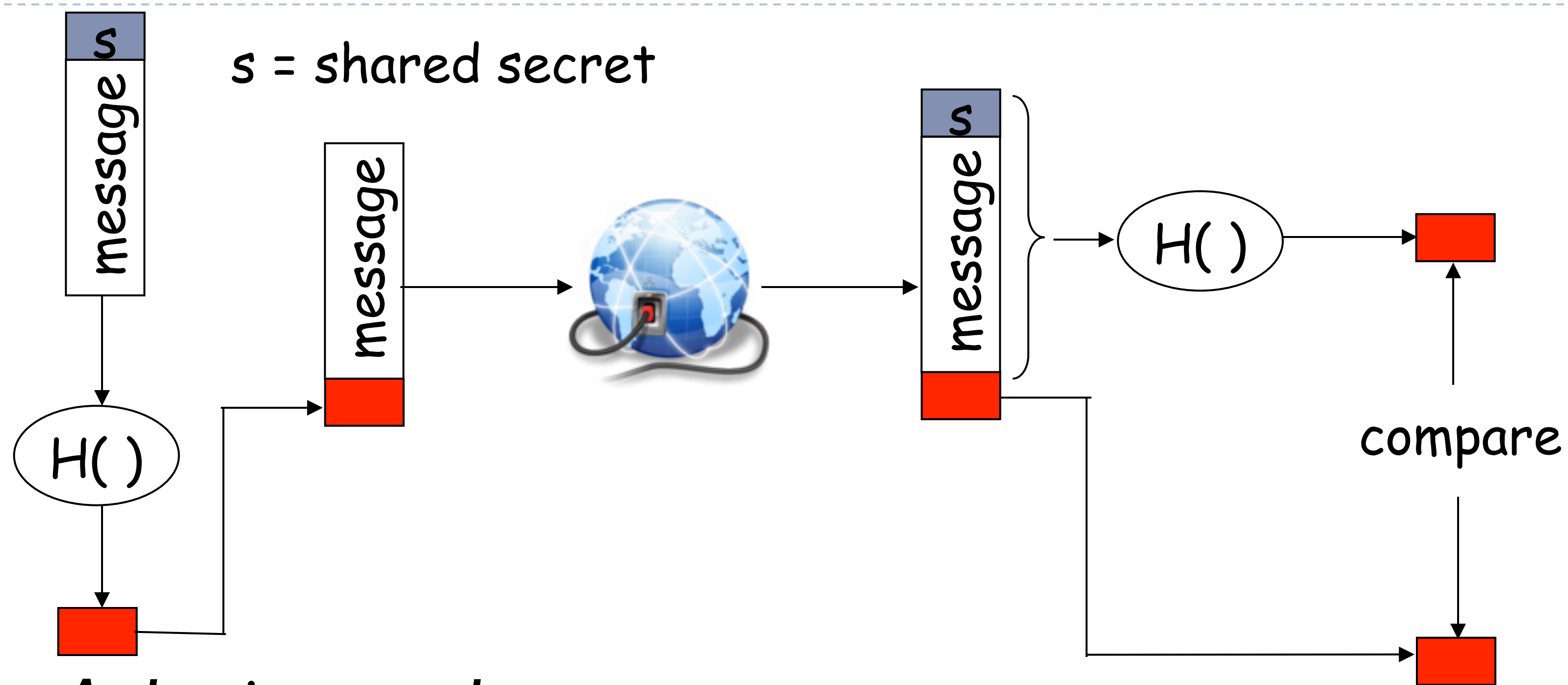




Question

- Assume we want to send a message
  - We are not concerned with confidentiality, only integrity
- What if we send
  - m' = m || MD5(m)
  - The receiver can extract m, compute MD5(m), and check if this matches the MD5 that was sent
- Does this guarantee integrity?

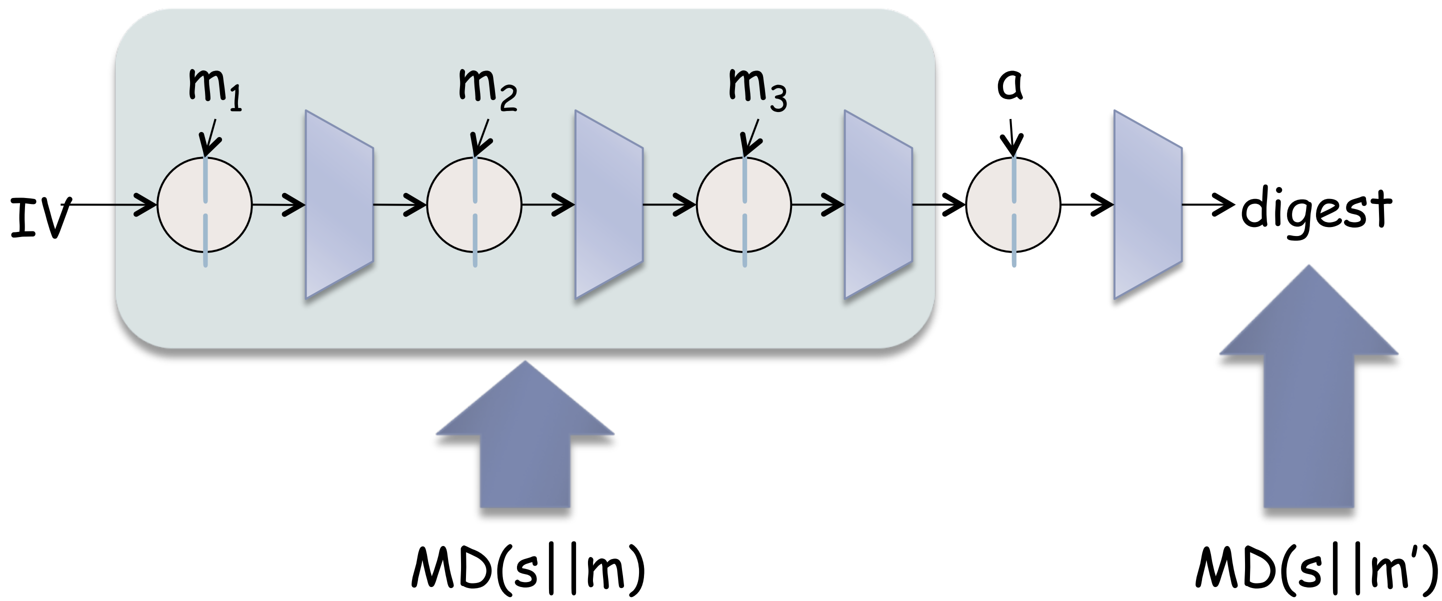
Message Authentication Code (MAC)



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

Not so fast!

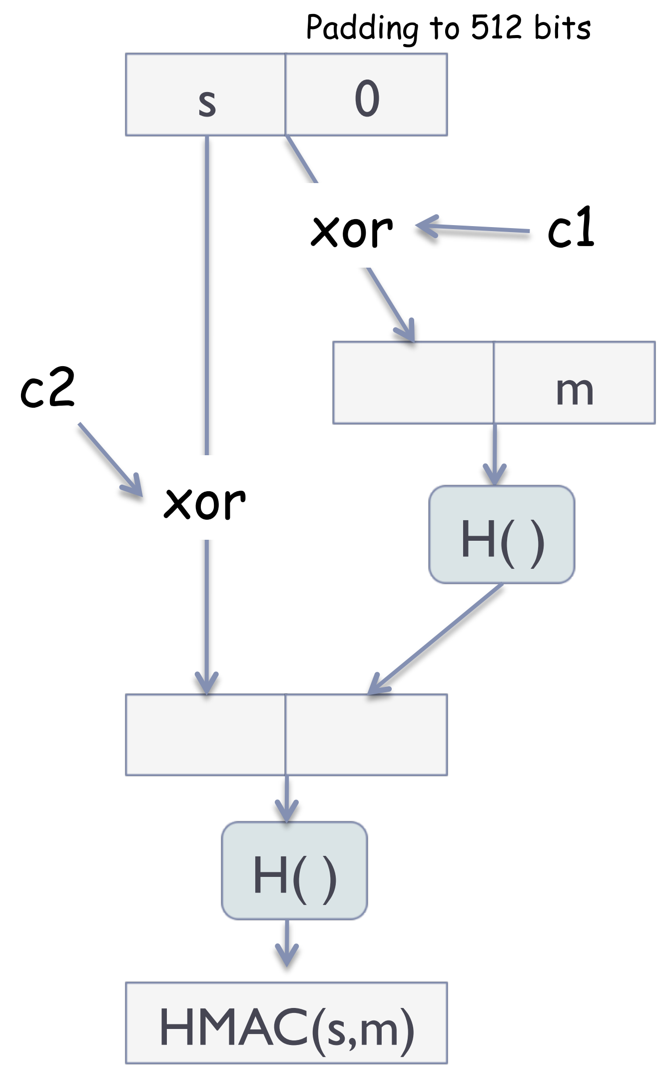
- Because most hash functions are iterated hash functions
  - Trudy knows the message m and MD(s||m)
  - She could append something to m to get m' = m||a, and use MD(s||m) to initialize the computation of MD(s||m')



HMAC***

- Popular MAC standard
- Addresses some subtle flaws
  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(s,m) = H(s||H(s||M))



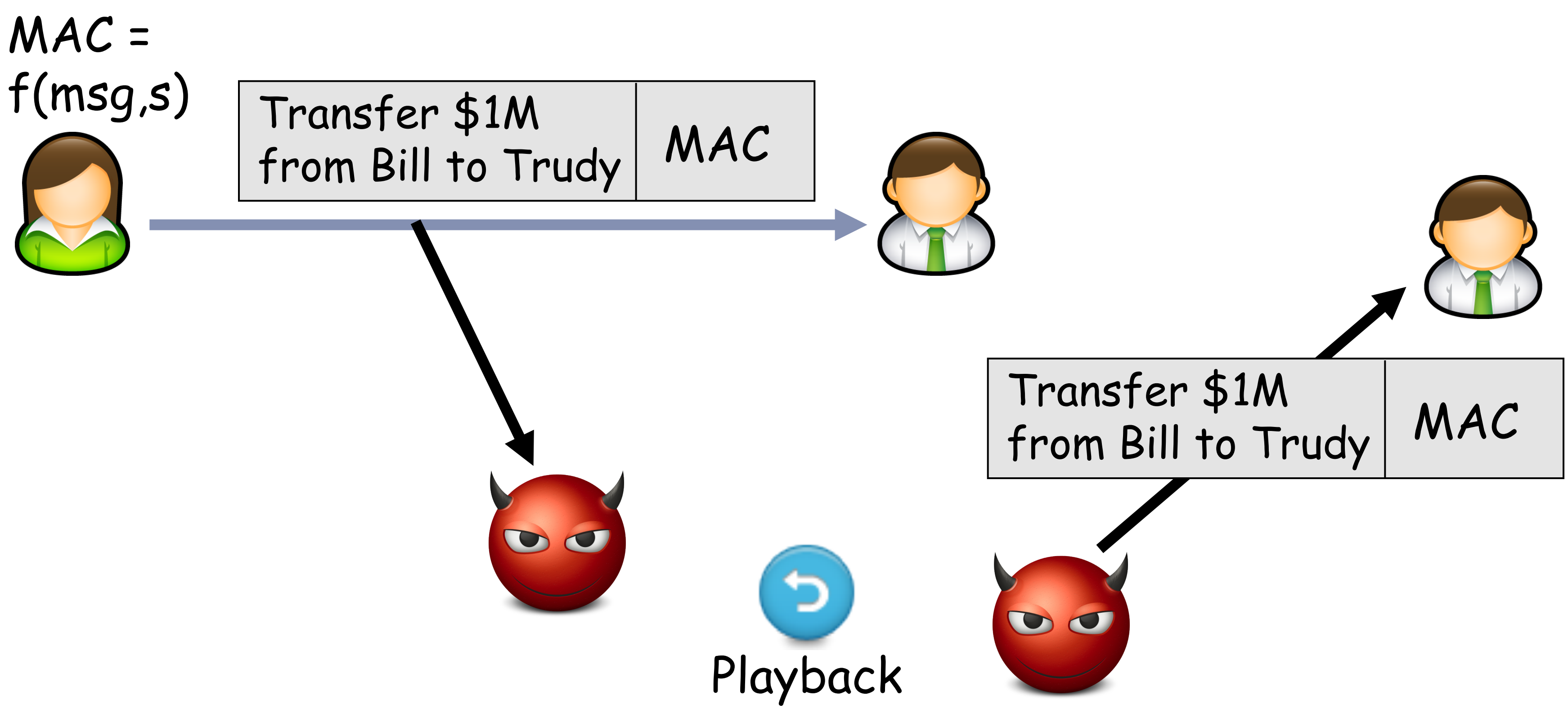
Other nifty things to do with a hash

- Hashing passwords
- Document/Program fingerprint
- Authentication

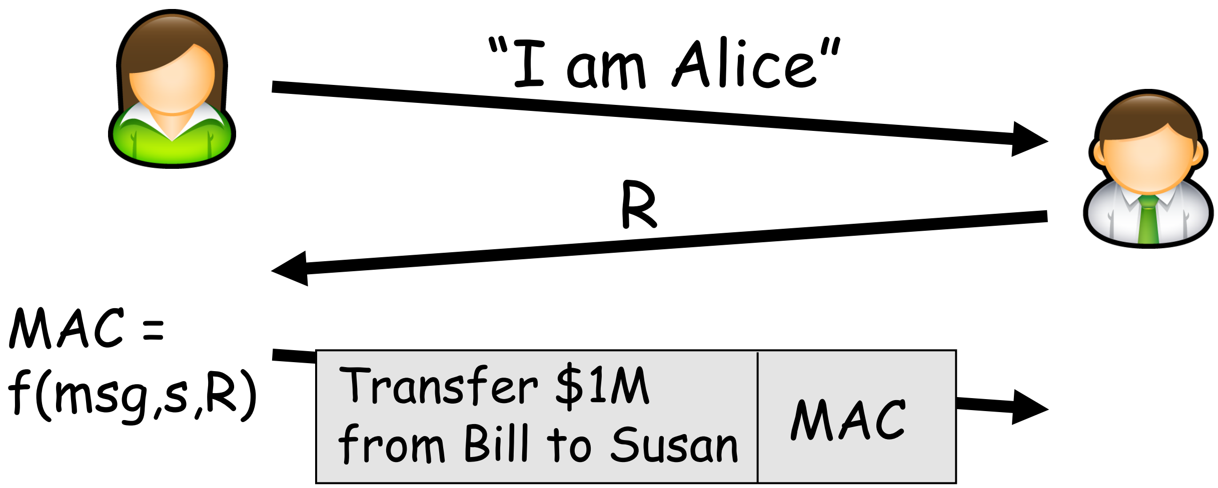
- Encryption

b1 = H(Kab|IV) c1 = p1 xor b1
b2 = H(Kab|c1) c2 = p2 xor b2
b3 = H(Kab|c2) c3 = p3 xor b3
...

Playback attack



Defending against playback attack: nonce



Digital Signatures

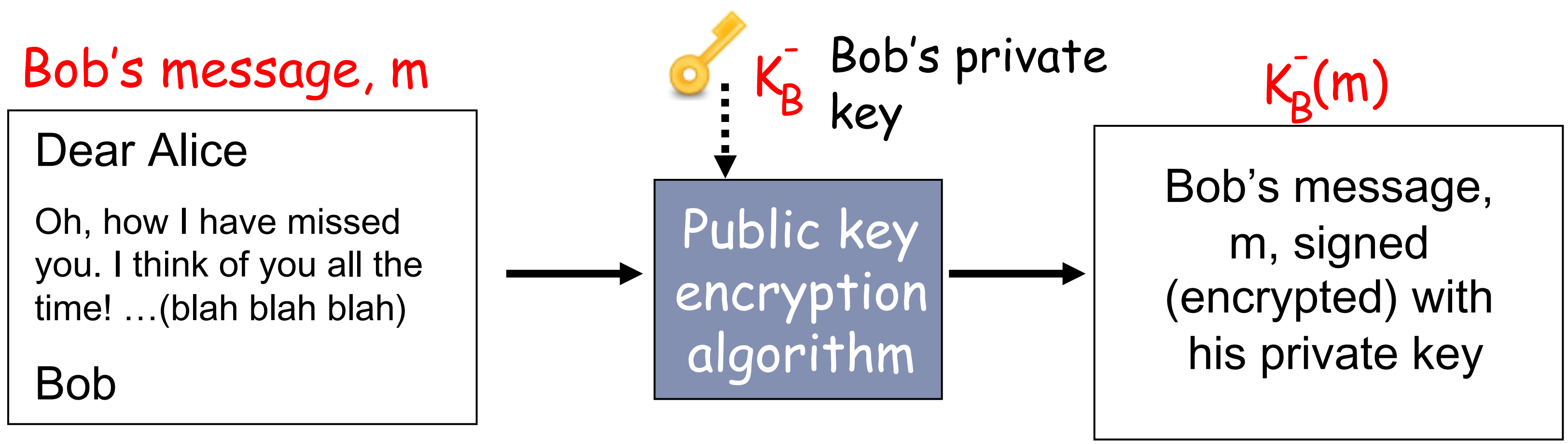
Cryptographic technique analogous to hand-written signatures.

- 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

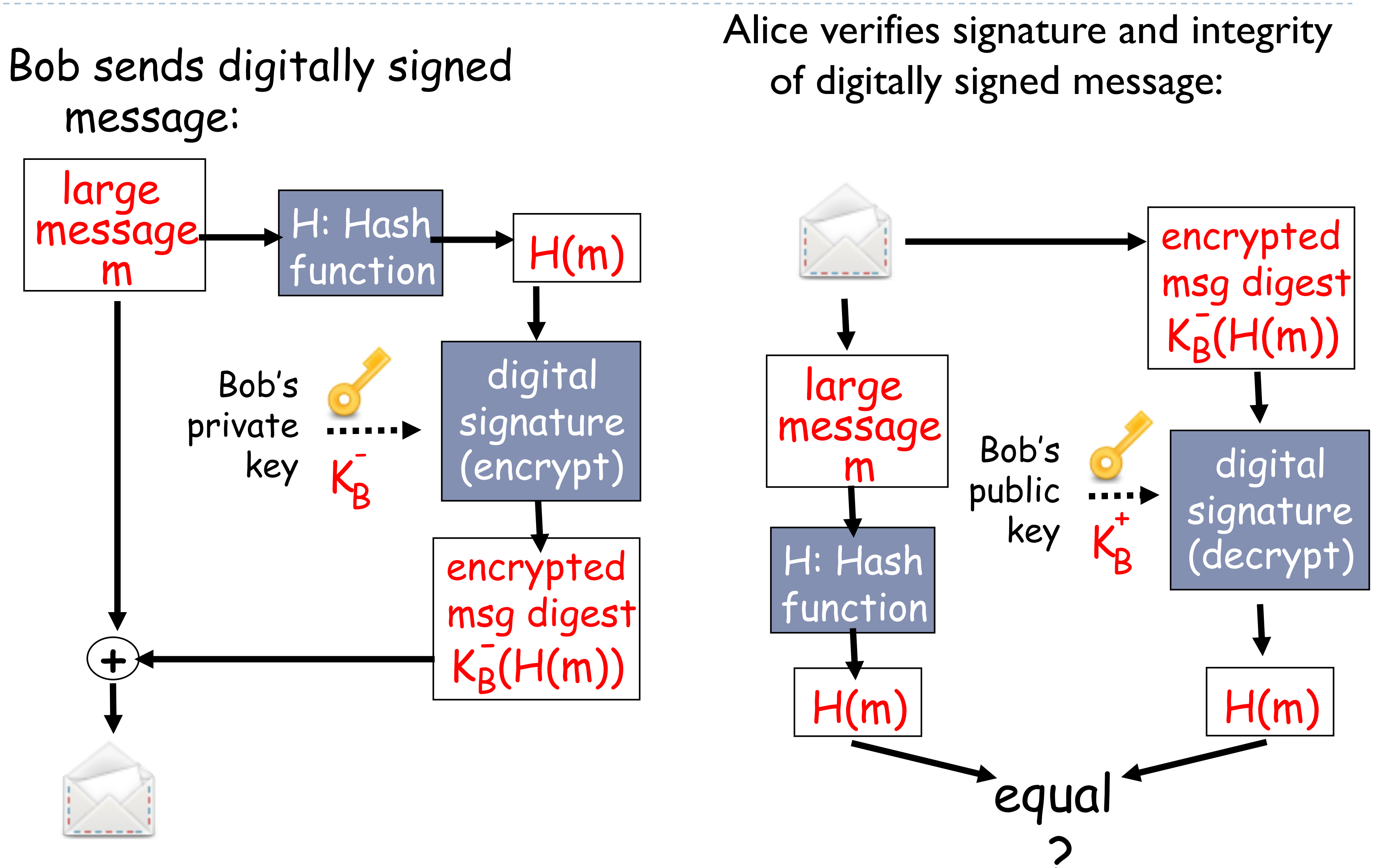
Digital Signatures

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



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:

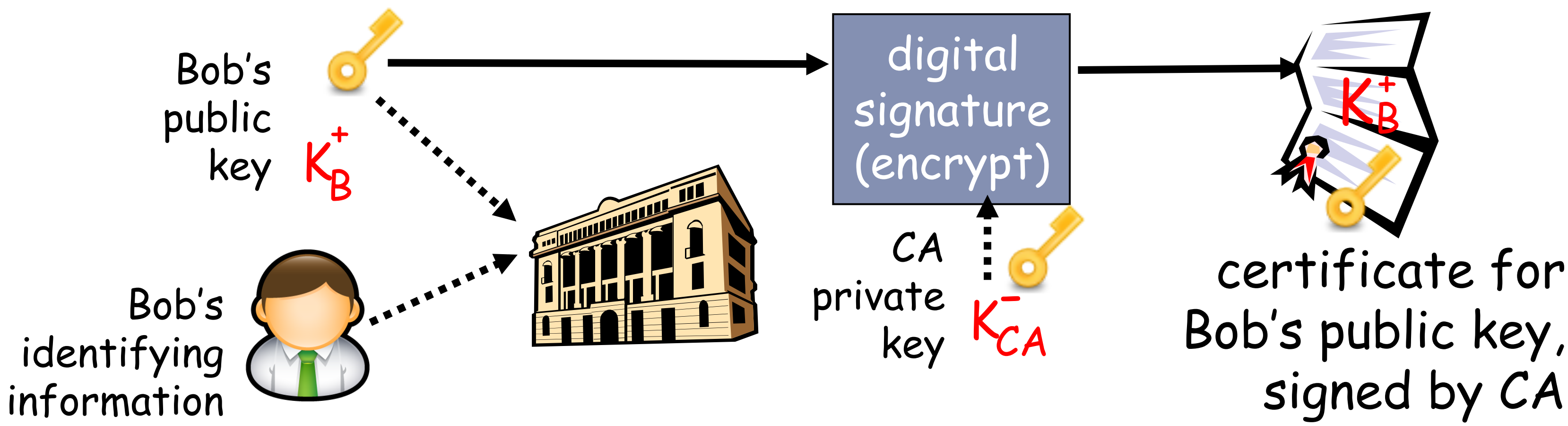
- ✓ Alice can take m, and signature $K_B^-(m)$ to court and prove that Bob signed m.

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

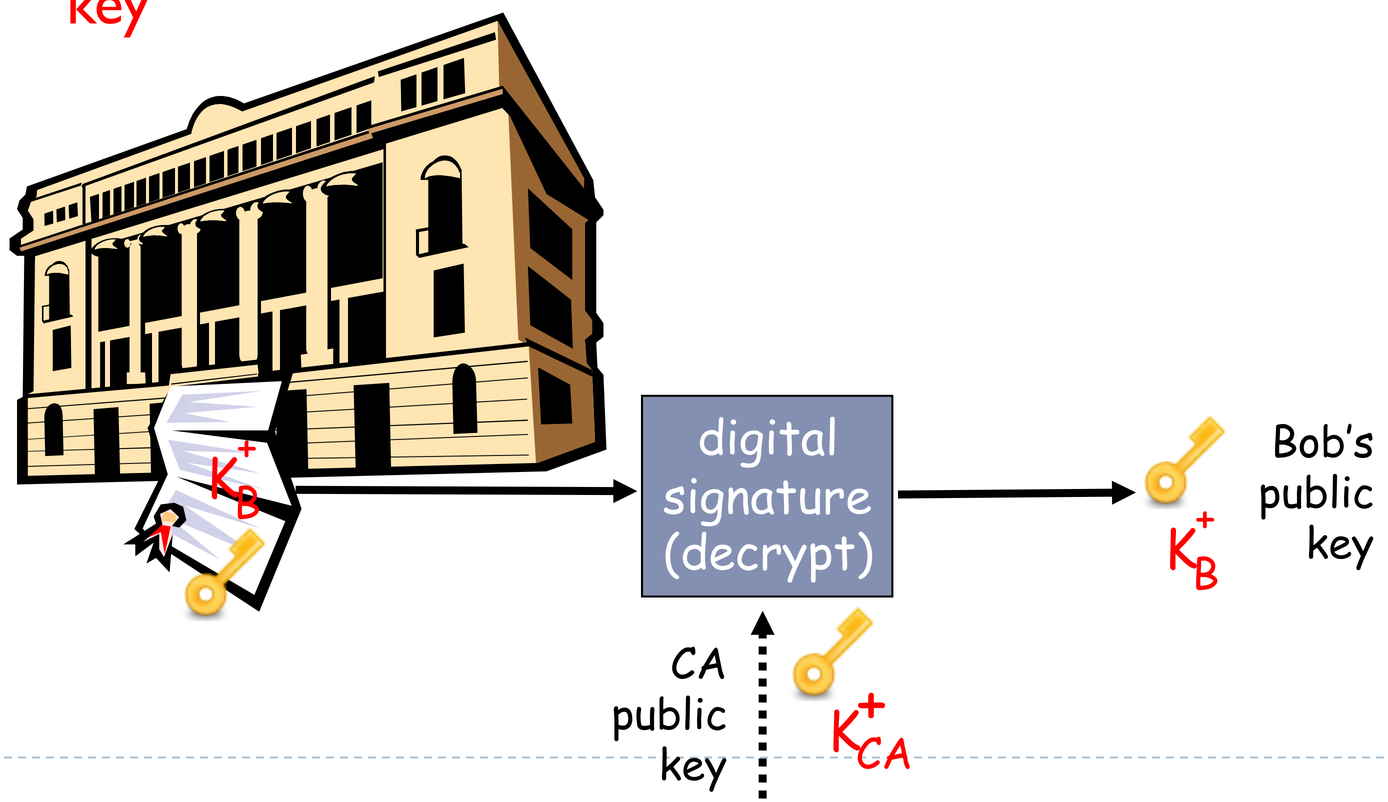
Certification Authorities

- Certification authority (CA): binds public key to particular entity, E.
- E (person, router) 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"



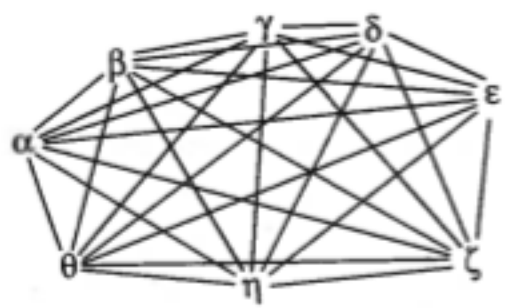
Certification Authorities

- 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

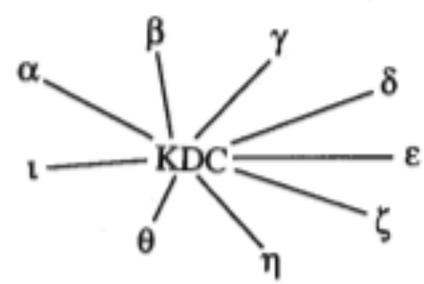


Alternative: symmetric crypto + KDC

- KDC = Key Distribution Center
  - Trusted Node
  - When Alice and Bob want to talk
    - Alice asks KDC for a symmetric session key to be shared with Bob
  - Reduces the number of keys that need to be distributed
    - If a new node joins the network, we need to generate *n* new keys
    - With KDC, only the new node and the KDC need to agree on a key



without KDC



with KDC

Key Exchange via KDC

- Needham-Schroeder protocol
  1. Alice >> KDC : "Alice" | "Bob" | Rand1
  2. KDC >> Alice : Ka("Alice" | "Bob" | Rand1 | Ks | Kb("Alice" | Ks))
  3. Alice >> Bob : Kb("Alice" | Ks)
  4. Bob >> Alice : Ks(Rand2)
  5. Alice >> Bob : Ks(Rand2-1)

See Bishop "Introduction to Computer Security"

KDC vs. CA

- KDC = Key Distribution Center
  - KDC can eavesdrop conversations
  - Single point of failure
- CA = Certification Authority
  - CA signs Alice's and Bob's pub keys
  - CA cannot decrypt communications between Alice and Bob
    - It does not have a copy of their private keys
    - If CA is compromised, attacker cannot gain access to the plaintext
  - Even if CA stops functioning, Alice and Bob can still communicate

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
  - Certificate Revocation List
  - Often considered "heavy"

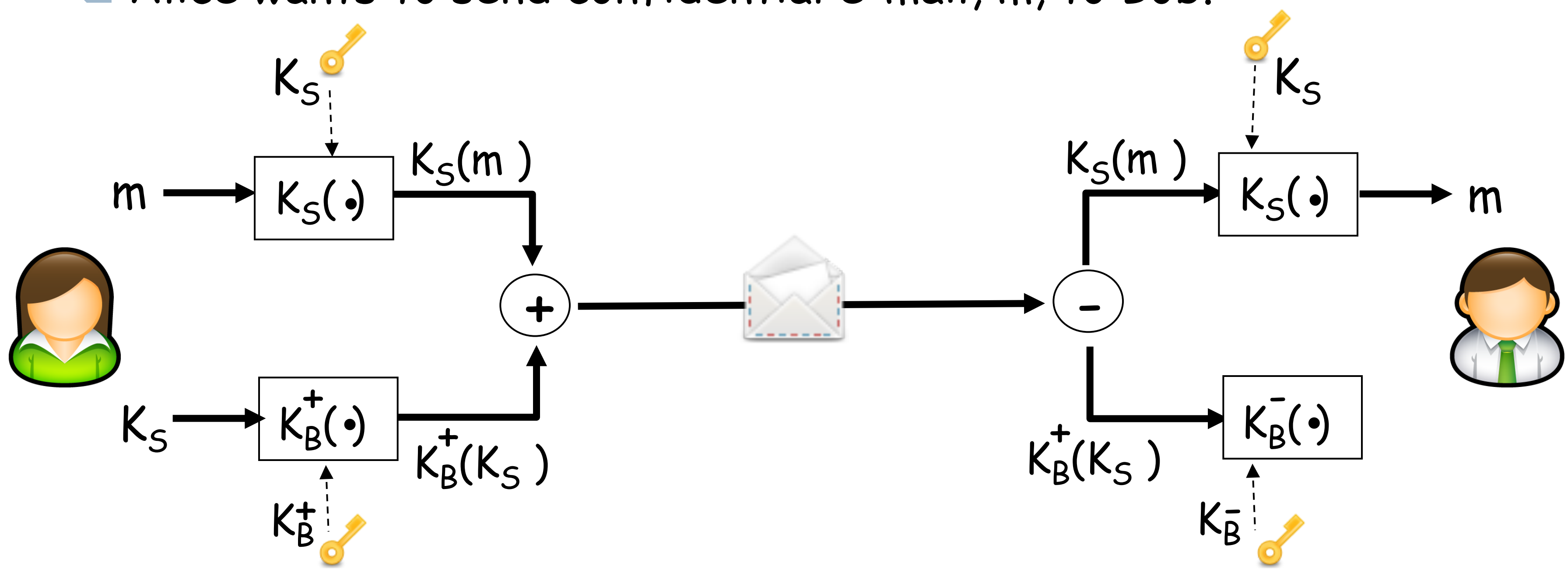
Components of a PKI

- Certificates
- Repository from which certificates can be retrieved
- A method for revoking certificates
  - E.g., see https://wiki.mozilla.org/CA:ImprovingRevocation
- An "anchor of trust" (root certificate)
- A method for verifying a chain of certificates up to the anchor of trust

- Browser example:
  - Browsers ship with many trust anchors (i.e., public key of trusted CAs)
- Can we really trust the CAs?
  - http://www.comodo.com/Comodo-Fraud-Incident-2011-03-23.html
  - http://googleonlinesecurity.blogspot.com/2011/08/update-on-attempted-man-in-middle.html
  - It may be possible to trick users to add a trust anchor into the default set
  - The browser itself may be compromised an forced to add a malicious trust anchor

Secure e-mail

- Alice wants to send confidential e-mail, m, to Bob.

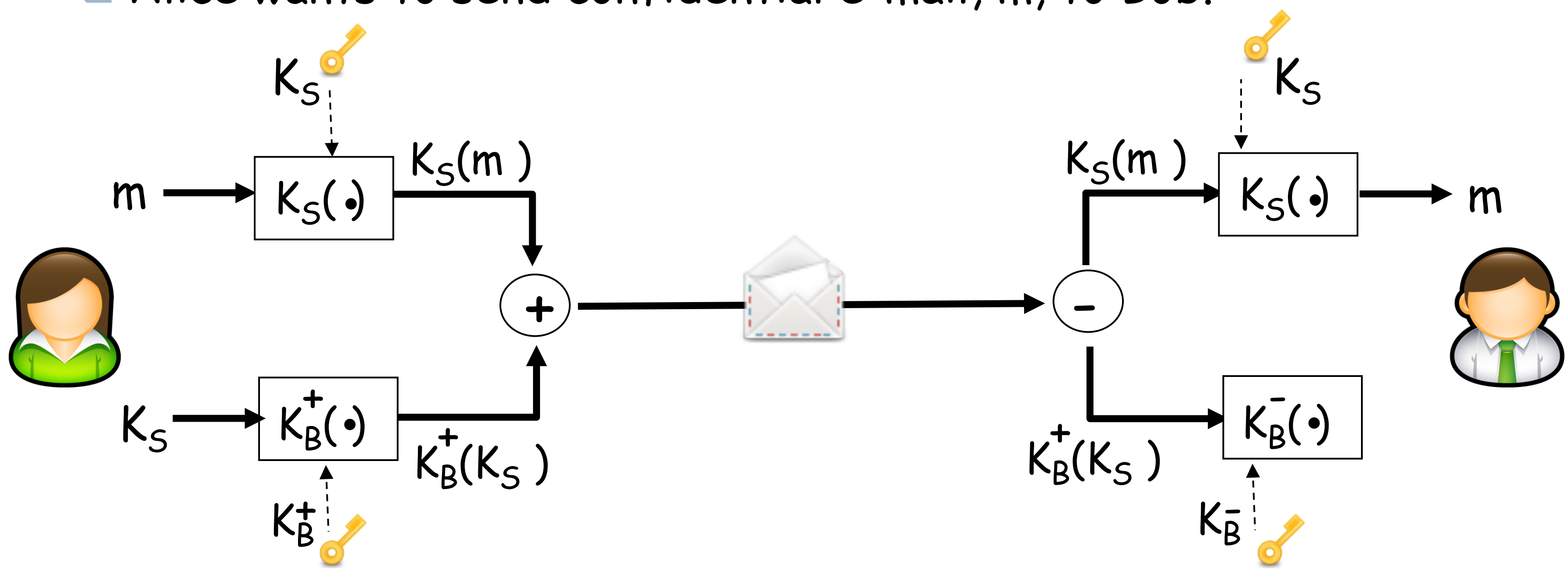


Alice:

- generates random *symmetric* private key, K_S.
- encrypts message with K_S (for efficiency)
- also encrypts K_S with Bob's public key.
- sends both $K_S(m)$ and $K_B(K_S)$ to Bob.

Secure e-mail

- Alice wants to send confidential e-mail, m, to Bob.

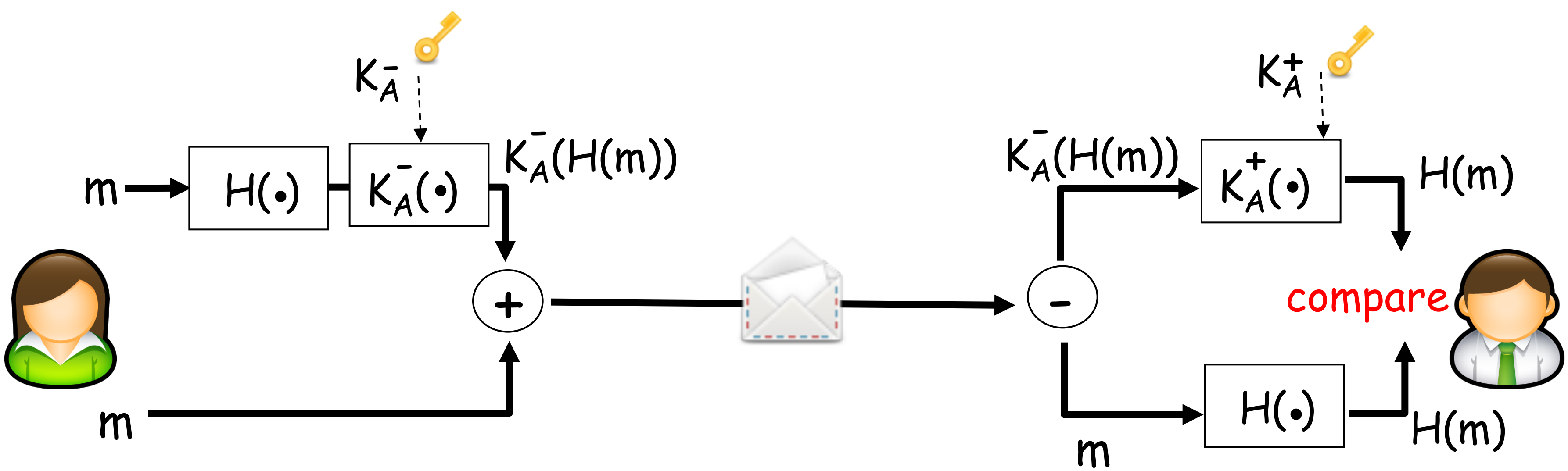


Bob:

- uses his private key to decrypt and recover K_S
- uses K_S to decrypt $K_S(m)$ to recover m

Secure e-mail (continued)

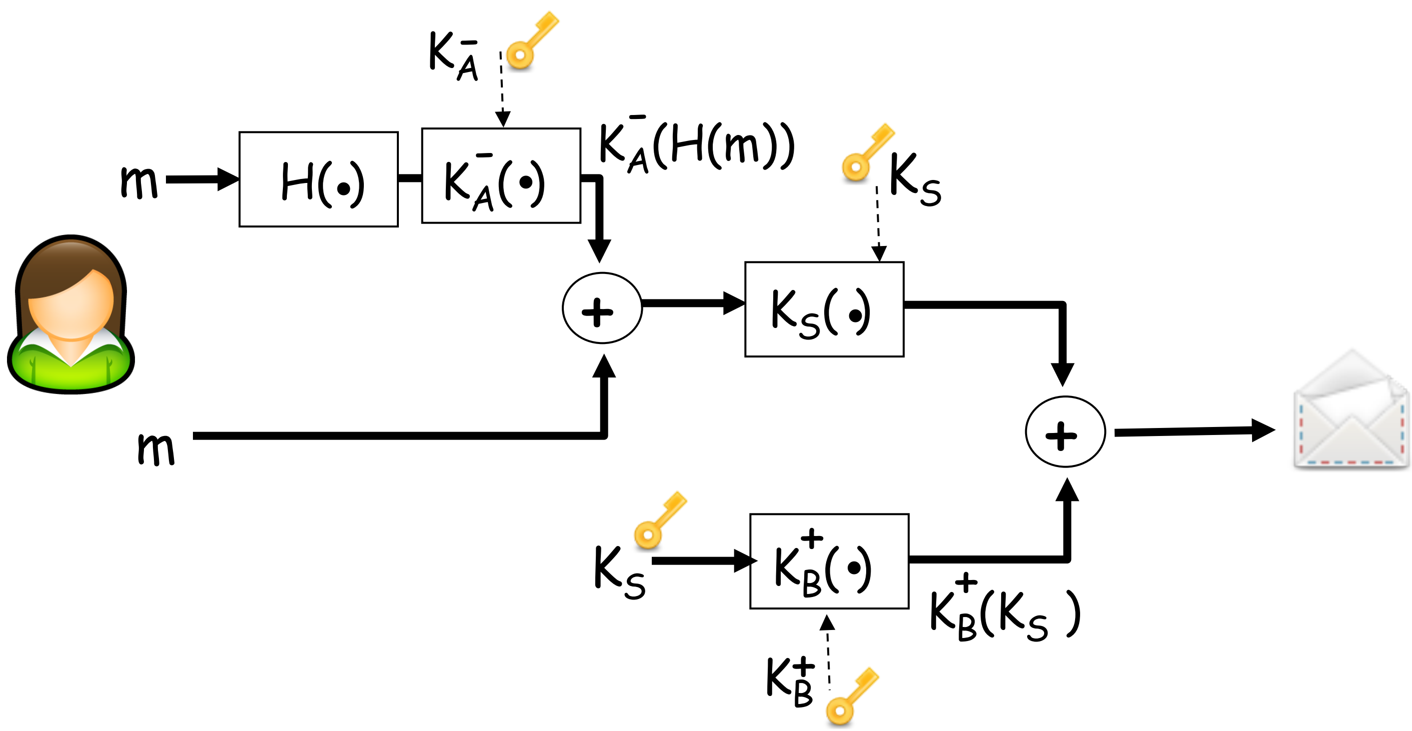
- Alice wants to provide sender authentication message integrity.



- Alice digitally signs message.
- sends both message (in the clear) and digital signature.

Secure e-mail (continued)

- Alice wants to provide secrecy, sender authentication, message integrity.



Alice uses three keys: her private key, Bob's public key, newly created symmetric key