Crypto Conclusion (maybe)

Message Authentication Codes

Key Management

Message Authentication

- message authentication is concerned with:
  - protecting the integrity of a message
  - Confirming identity of sender
  - non-repudiation of origin (dispute resolution)
  - Very important for e-commerce
- will consider the security requirements
- then three alternative functions used:
  - message encryption
  - message authentication code (MAC)
  - hash function

General Security Requirements

- disclosure
- traffic analysis

This is message confidentiality.
We've dealt with it already.

- Masquerade: insertion of message into network from fraudulent source
- content modification: modification to content of message
- sequence modification: modification to a sequence of messages, including insertion, deletion, reordering, etc.

All the rest are authentication or integrity issues (including next slide)

General Security Requirements

- Timing modification: Delay or replay of messages
  - E.g. in a connection-oriented application (say one that uses TCP) an entire session could be a replay of some previous valid session
- Source repudiation: denial of transmission of message by source
- Destination repudiation: Denial of receipt of message by destination

Message Encryption?

- Does message encryption by itself also provides a measure of integrity and/or authentication?
- After all, if symmetric encryption is used then it appears that:
  - receiver knows sender must have created it, since only sender and receiver know key used
  - know content cannot have been altered if message has suitable structure, redundancy, or a checksum to detect any changes
- Answer: **NO!** Message encryption does not by itself provide either integrity or authentication.

Encryption does not provide authentication or integrity

- It is often the case that a cryptographic key is shared by more than two parties, in which case authentication is out the window
- Regarding integrity, the amount of damage (i.e., changes to the original plaintext) an adversary can inflict depends on the type of encryption as well as the mode of operation.
- Ex. Stream ciphers allow an attacker to flip bits anywhere in the message
  - Person-in-the-middle can change ciphertext $C = P \oplus R$ to C', which decrypts to $C' \oplus R$
  - Can't control what P is because they don't know R, but can nevertheless change it!

Encryption does not provide authentication or integrity

- It's not just stream ciphers. Consider AES-CBC

Fix some key k (unknown to the attacker). Let E(k,-) and D(k,-) be the bare encryption and decryption functions of some block cipher. Let p be some single block of plaintext. I'll denote XOR by +. After fixing some IV, we encrypt as follows:

c = E(k,p + IV).

Then, we send IV and c over the wire. To decrypt, we compute

p = D(k, c) + IV.

(Note that this is equivalent to the statement D(k,c) = IV + p.)

Now, suppose that an attacker knows a single plaintext/ciphertext pair. Let's denote them as p and (IV, c), as above. Now, suppose that the attacker would like to create ciphertext that will decrypt to some other plaintext block of his choosing -- say p'. I claim that (IV + p + p', c) decrypts to p'. Why?

Well, we just follow the decryption procedure above, replacing IV with IV + p + p'. We have

D(k,c) + (IV + p + p') = (IV + p) + (IV + p + p') = p'.

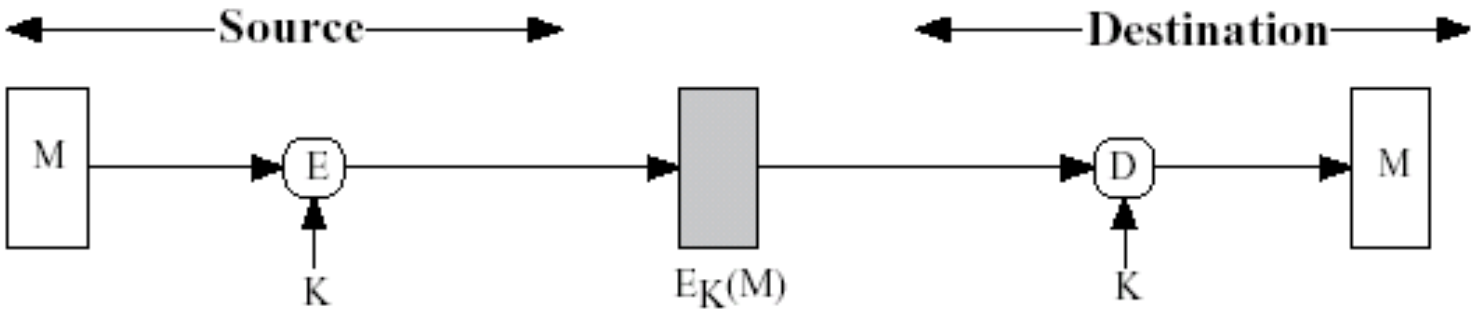
Example thanks to StackExchange

Encryption does not provide authentication or integrity

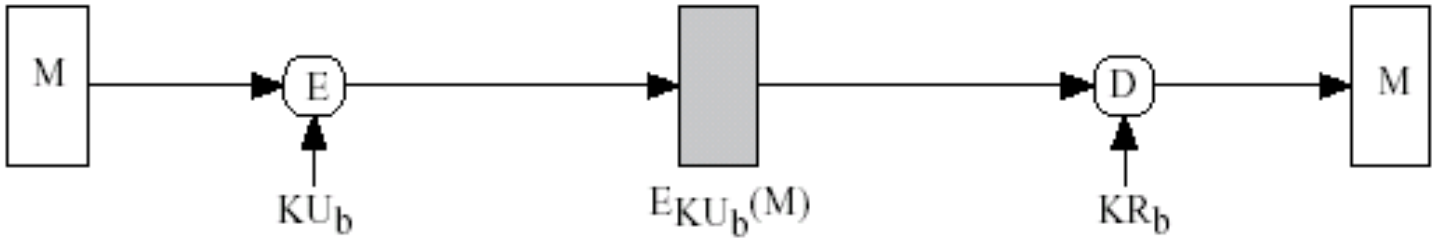
- Note that if plaintext is longer than one block, first block can still be changed (at the cost of creating garbage for the remaining plaintext blocks, which may or may not be detected).
- Authenticated encryption schemes solve this
  - AES-CBC is not one of them
- In general, encryption without authentication/integrity is one of the most common mistakes in the use of cryptography
  - Serious vulnerabilities have resulted, including ASP.NET, XML encryption, Amazon EC2, JavaServer Faces, Ruby on Rails, OWASP ESAPI, IPSEC, and WEP.
  - See e.g., https://security.stackexchange.com/questions/2202/lessons-learned-and-misconceptions-regarding-encryption-and-cryptology/2206#2206

Message Encryption

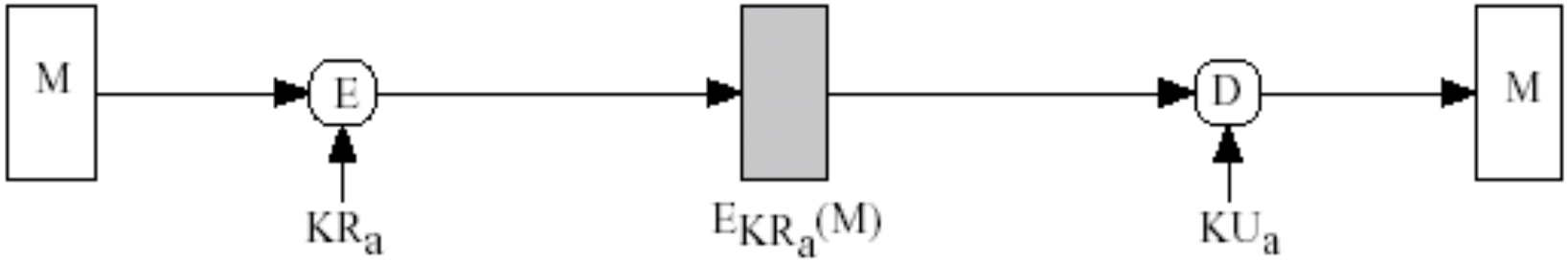
- if public-key encryption is used:
  - encryption provides no confidence of sender, since anyone potentially knows public-key
  - however if
    - sender **signs** message using their private-key
    - then encrypts with recipients public key
    - have both secrecy and authentication
  - again need to recognize corrupted messages
  - but at cost of two public-key uses on message



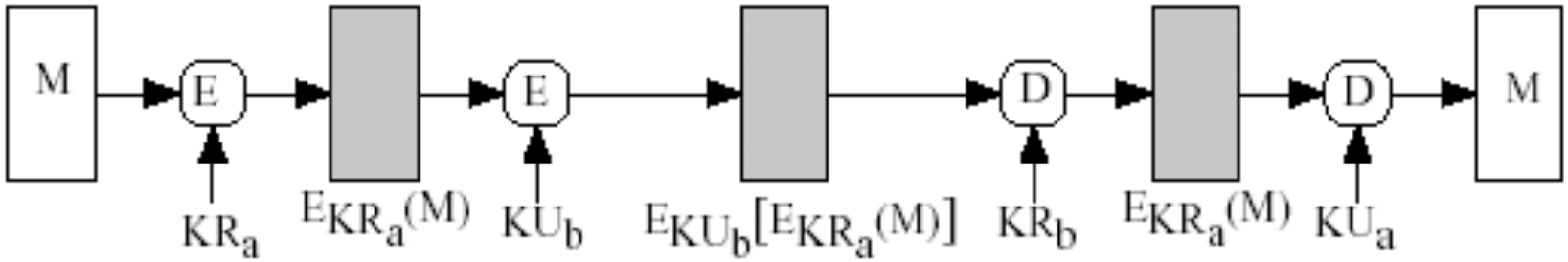
(a) Conventional encryption: confidentiality and authentication



(b) Public-key encryption: confidentiality



(c) Public-key encryption: authentication and signature

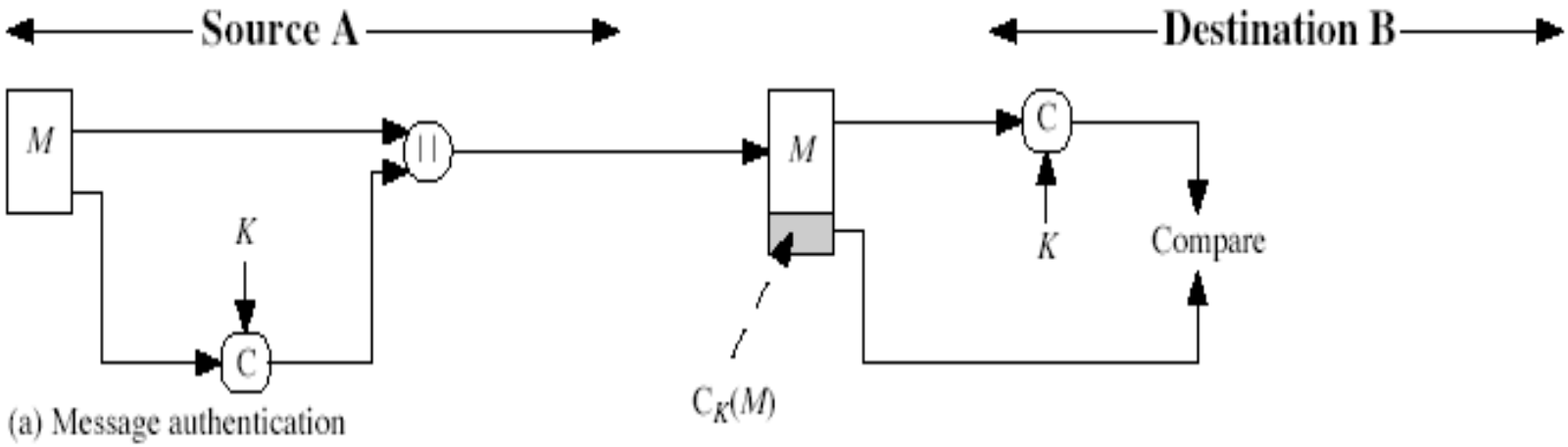


(d) Public-key encryption: confidentiality, authentication, and signature

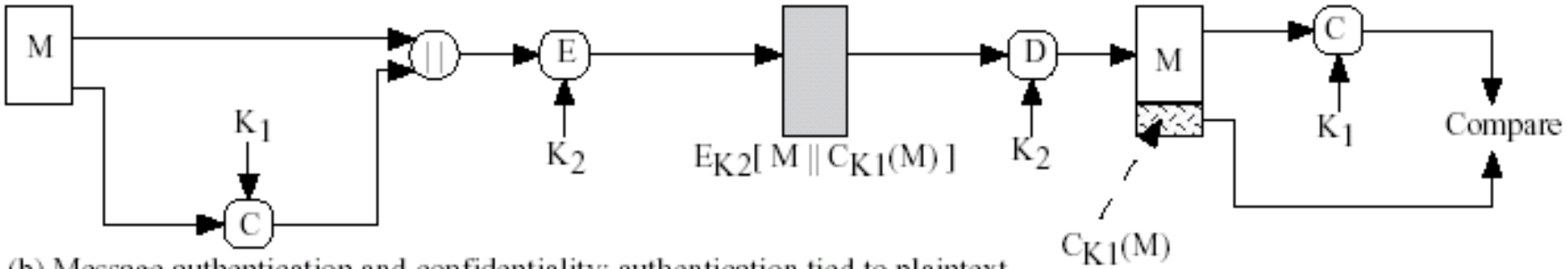
Message Authentication Code (MAC)

- The answer to recognition of bad messages lies in creating a known structure somewhere in the message. This is part of the idea behind MACs
- generated by an algorithm that creates a small fixed-sized block
  - depending on both message and some key
  - like encryption, **BUT** need not be reversible
- appended to message as a **signature**
- receiver performs same computation on message and checks it matches the MAC
- provides assurance that message is unaltered and comes from sender

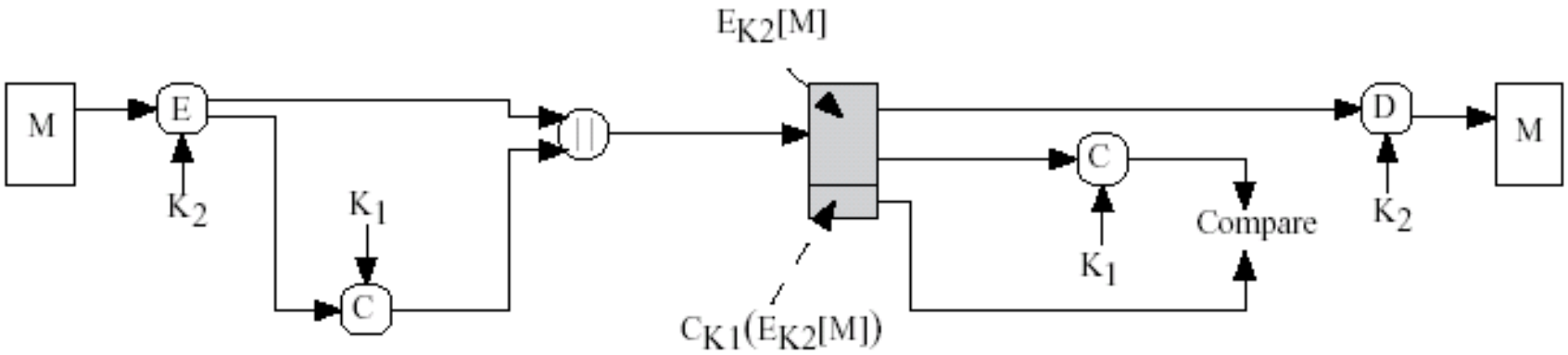
Message Authentication Code



(a) Message authentication



(b) Message authentication and confidentiality; authentication tied to plaintext



(c) Message authentication and confidentiality; authentication tied to ciphertext

Message Authentication Codes

- MAC does not provide secrecy
- If using MAC with symmetric cipher:
  - generally use separate keys for each
  - can compute MAC either before or after encryption
  - is generally regarded as better done before (maybe)
    - Consider, e.g., malleability
- why use a MAC?
  - sometimes only authentication is needed
  - sometimes need authentication to persist longer than the encryption (eg. archival use)
- note that a MAC is not a digital signature
  - That is, the sender can still deny having sent the message

MAC Properties

- a MAC is a cryptographic checksum

  $MAC = C_K(M)$

  – condenses a variable-length message M
  – using a secret key K
  – to a fixed-sized authenticator
- is a many-to-one function
  – potentially many messages have same MAC
  – but (obviously) finding these needs to be very difficult

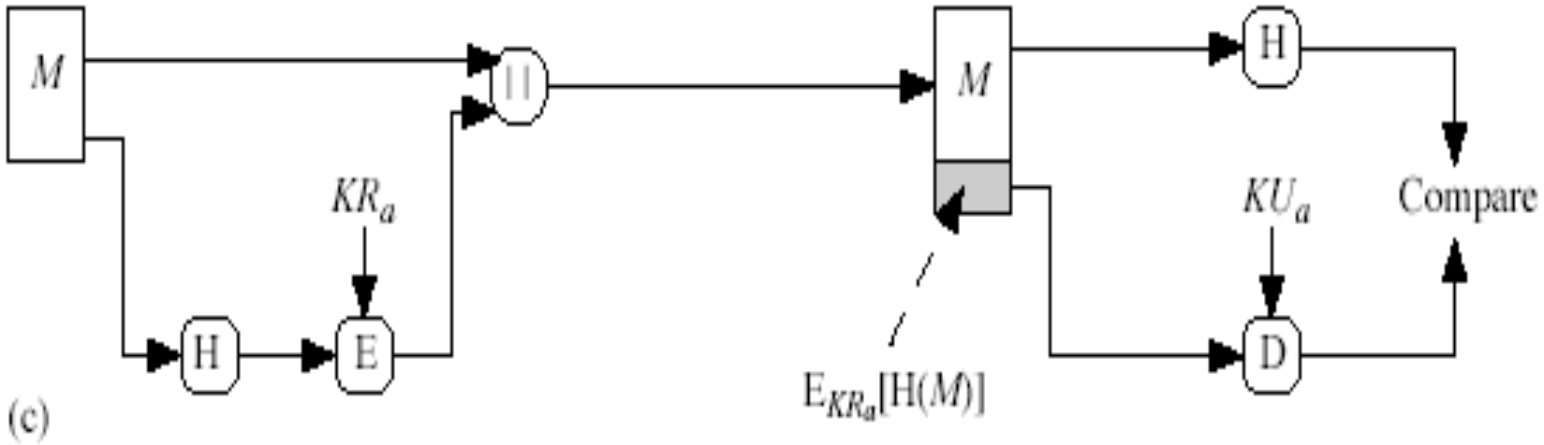
Requirements for MACs

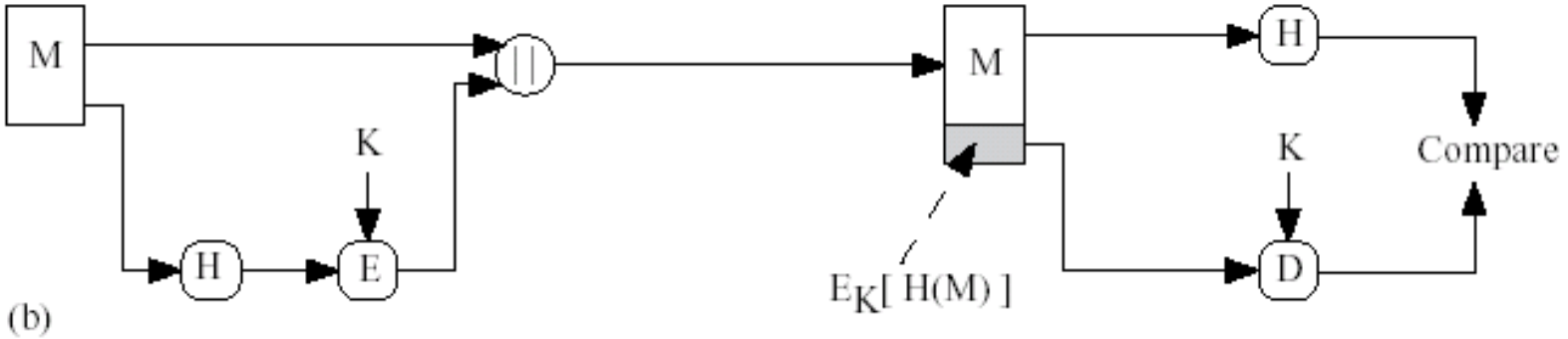
- Knowing a message and MAC, it is infeasible to find another message with the same MAC
- MACs should be uniformly distributed (among the space of possible MACs)
- MAC should depend equally on all bits of the message

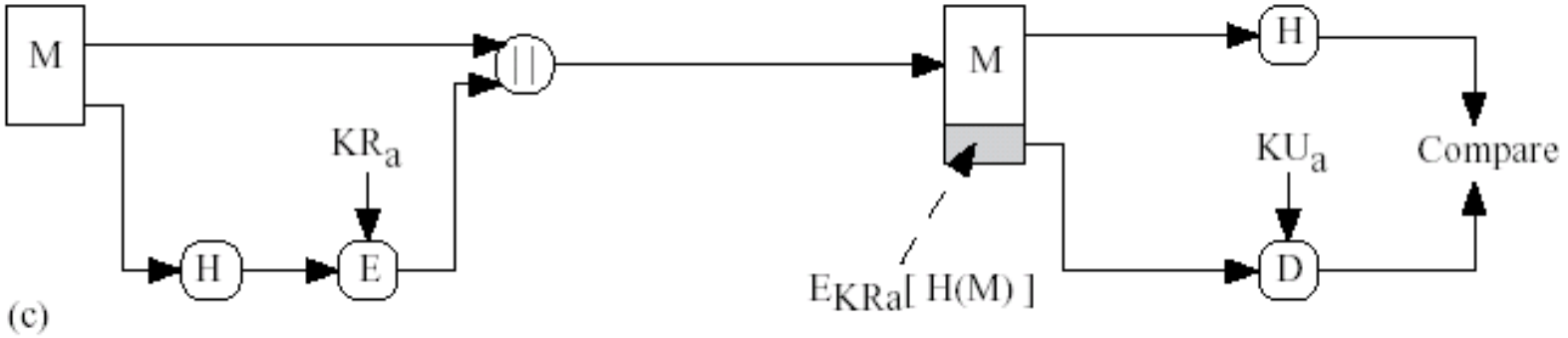
Hash Functions

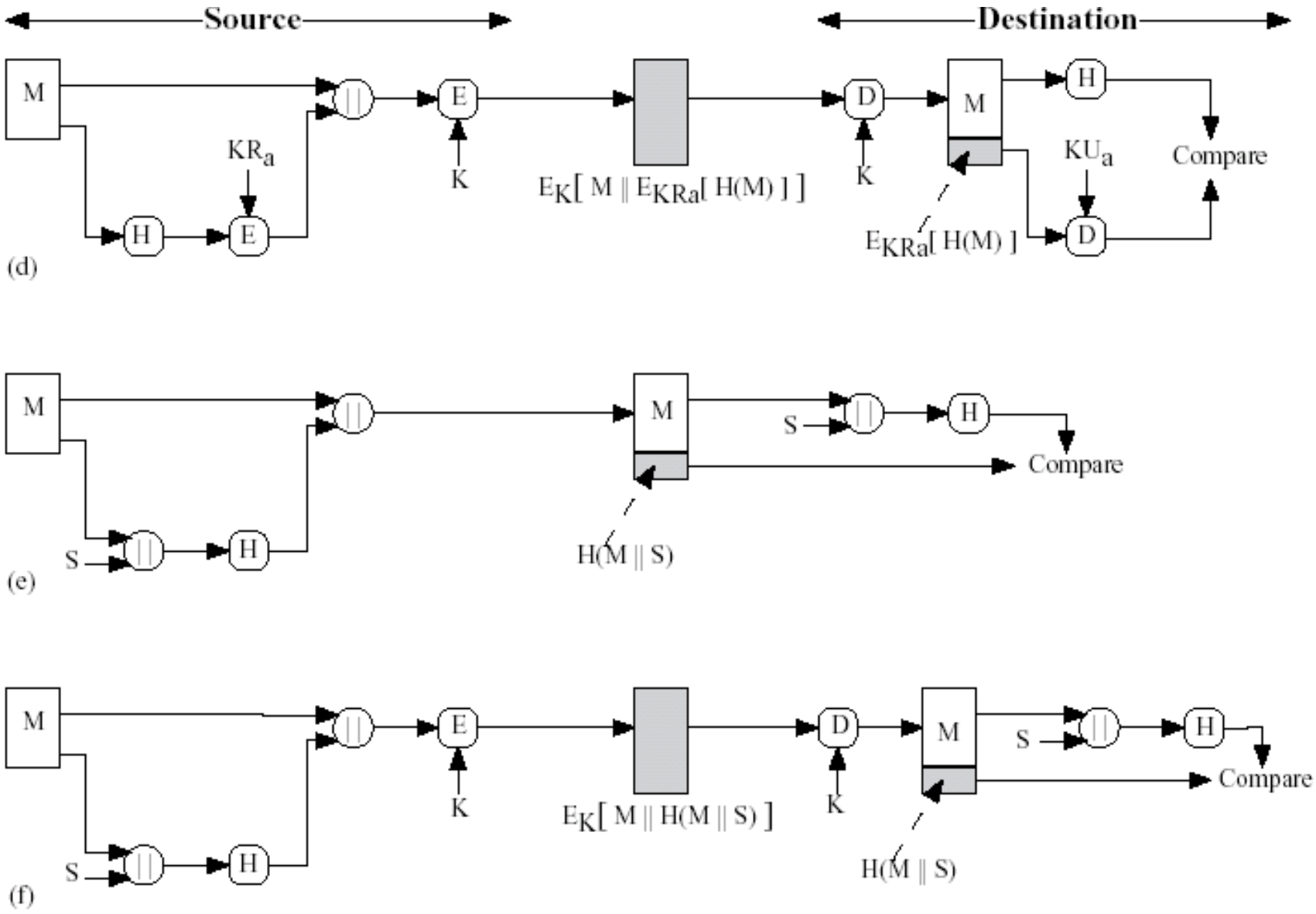
- condenses arbitrary message to fixed size
- usually assume that the hash function is public and ***not keyed***—this is the difference between a hash function and a MAC (the lack of key)
- hash used to detect changes to message
- can use in various ways with message
- most often to create a digital signature

Hash Functions & Digital Signatures




M
||
K
H
E
M
H
K
Compare
D
$E_K[H(M)]$
(b)


M
||
KR_a
H
E
M
H
KU_a
Compare
D
$E_{KRa}[H(M)]$
(c)


Source
Destination
M
KRa
H
E
K
E
K[M || EKRa[H(M)]]
D
K
M
H
KUa
Compare
EKRa[H(M)]
D
(d)
M
S
H
H(M || S)
M
S
H
Compare
(e)
M
S
H
K
E
K[M || H(M || S)]
D
K
M
S
H
Compare
H(M || S)
(f)

Hash Function Properties

- a Hash Function produces a fingerprint of some file/ message/data

  `h = H(M)`

  – condenses a variable-length message M
  – to a fixed-sized fingerprint
- assumed to be public

Requirements for Hash Functions

1. can be applied to any sized message `M`
2. produces fixed-length output `h`
3. is easy to compute `h=H(M)` for any message `M`
4. given `h` is infeasible to find `x` s.t. `H(x)=h`
   - one-way property
5. given `x` is infeasible to find `y` s.t. `H(y)=H(x)`
   - weak collision resistance
6. is infeasible to find any `x,y` s.t. `H(y)=H(x)`
   - strong collision resistance

Birthday Attacks

- might think a 64-bit hash is secure
- but by **Birthday Paradox** is not
- **birthday attack** works thus:
  - opponent generates $2^{m/2}$ variations of a valid message all with essentially the same meaning (m is length of hash)
  - opponent also generates $2^{m/2}$ variations of a desired fraudulent message
  - two sets of messages are compared to find pair with same hash (probability > 0.5 by birthday paradox)
  - have user sign the valid message, then substitute the forgery which will have a valid signature
- conclusion is that need to use larger hashes

Birthday Paradox

- Classic probability problem that demonstrates that probability results often nonintuitive
- The problem: Given a room with k people, what is the probability that two of them have the same birthday (same month and day, assume no twins, etc)
- We seek

$$P(n,k) = \Pr[\text{at least one duplicate in } k \text{ items, with each item able to take on one of } n \text{ equally likely values between 1 and } n]$$

We want P(365,k)

We start by computing Q = Pr[no matches], so P = 1-Q.

First, number of ways of choosing k objects from group of 365 with no repeats:

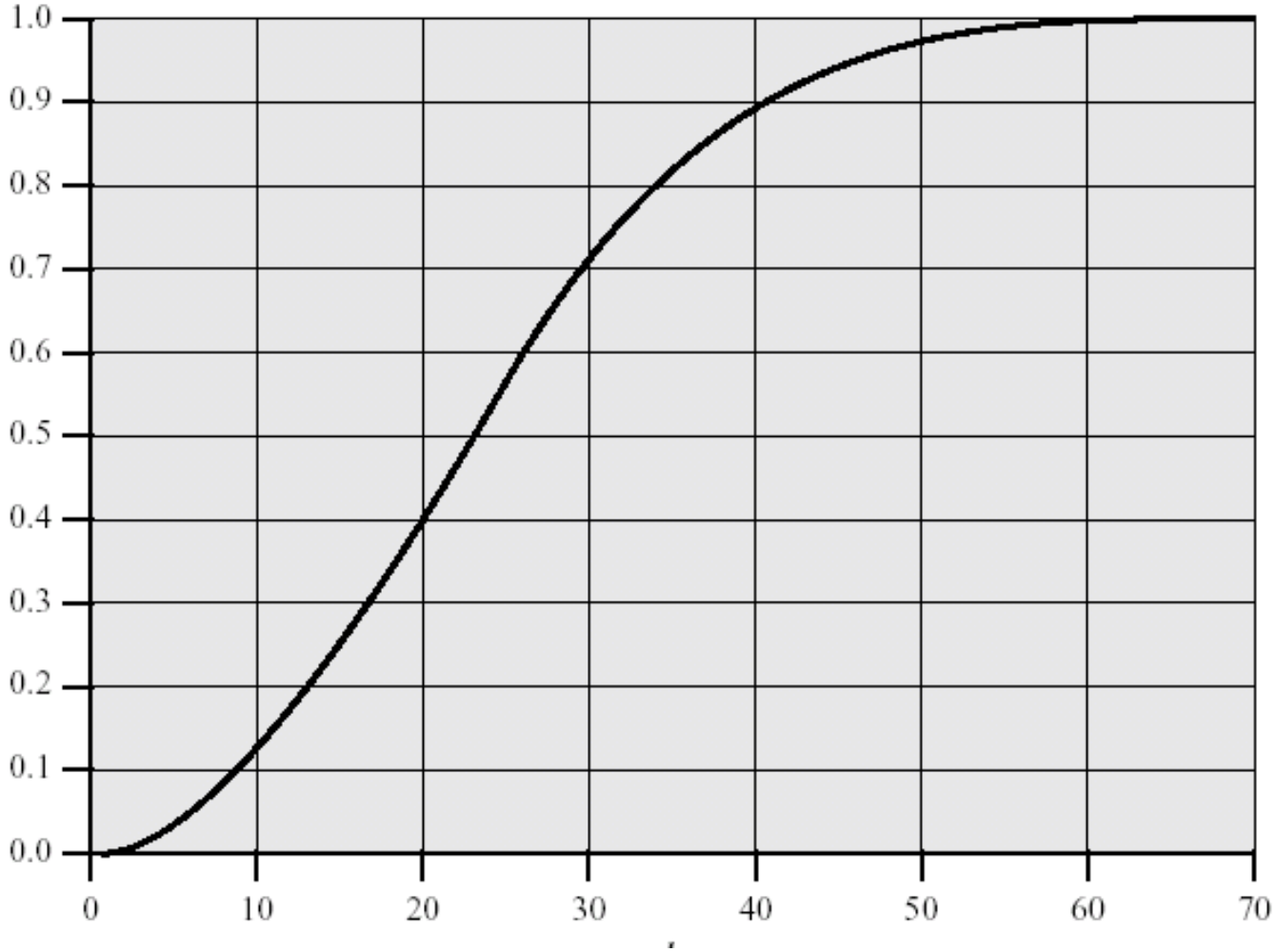
$$N = 365 \times 364 \times 363 \times \ldots \times (365 - k + 1) = \frac{365!}{(365 - k)!}$$

If we allow repeats, then there are 365^k possibilities. So, probability of no repeats is

$$Q(365, k) = \frac{\frac{365!}{(365 - k)!}}{365^k} = \frac{365!}{(365 - k)!365^k}$$

$$\text{Thus, } P(365, k) = 1 - Q(365, k) = 1 - \frac{365!}{(365 - k)!365^k}$$

Graph of P(365,k)



Hash Functions & MAC Security

- **brute-force** attacks exploiting
  - strong collision resistance hash have cost $2^{m/2}$
    - have proposal for h/w MD5 cracker
      - UPDATE: As of 2010, MD5 is no longer suitable for cryptographic use (trashed)
    - Use SHA-2 instead (has digest sizes of 224, 256, 384, 512)
      - Similar to SHA-1 (Which has mathematical weaknesses), though SHA-2 not broken
  - UPDATE UPDATE: On October 12, 2012, Keccak named winner of the NIST Hash Function Competition (and is thus SHA-3)
    - NIST wanted a hash that was not similar in design to SHA-1 (or SHA-2 in case that was broken)
    - Joan Daemen (of AES fame) one of designers

Hash Functions & MAC Security

- **cryptanalytic attacks** exploit structure
  - like block ciphers, want brute-force attacks to be the best alternative
- have a number of analytic attacks on iterated hash functions
  - $CV_i = f[CV_{i-1}, M_i]$; $H(M)=CV_N$
  - typically focus on collisions in function f
  - like block ciphers is often composed of rounds
  - attacks exploit properties of round functions

Summary

- have considered message authentication using:
  - message encryption
  - MACs
  - hash functions
  - general approach & security

Key Management

Before We Start

- Might be surprised: key management is one of the most complex aspects of deploying cryptographic systems
- Usual assumption: ciphertext available to adversary, but key NOT
- Reality: Keys are stolen all the time
  - Sometimes by NSA
  - But also by lots others
- Bottom line: we do a lousy job of keeping secrets

Key Distribution Issues

- hierarchies of KDC's required for large networks, but must trust each other
- session key lifetimes should be limited for greater security
- use of automatic key distribution on behalf of users, but must trust system
- use of decentralized key distribution
- controlling purposes keys are used for

Symmetric Key Distribution

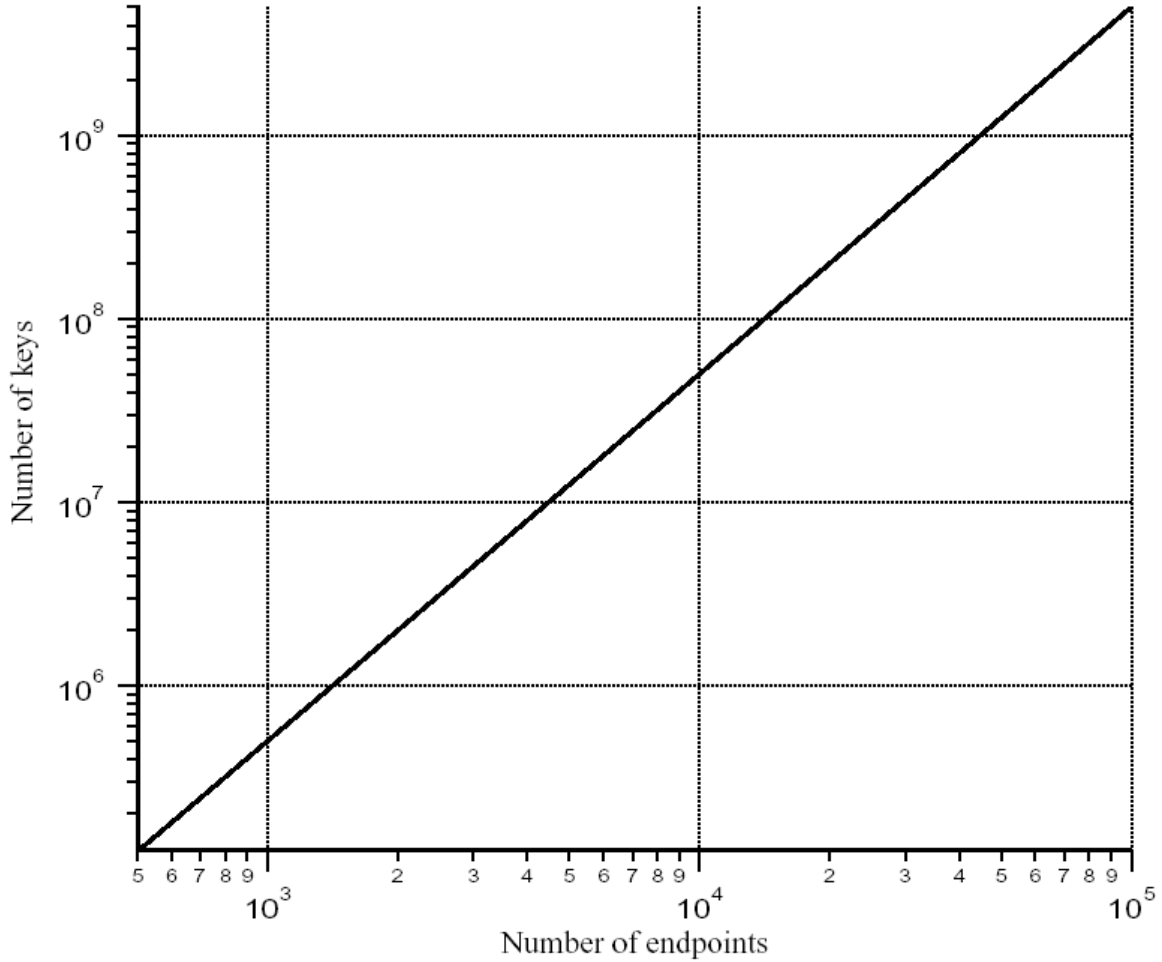
- symmetric schemes require both parties to share a common secret key
- issue is how to securely distribute this key
- often secure system failure due to a break in the key distribution scheme

Key Distribution (Symmetric)

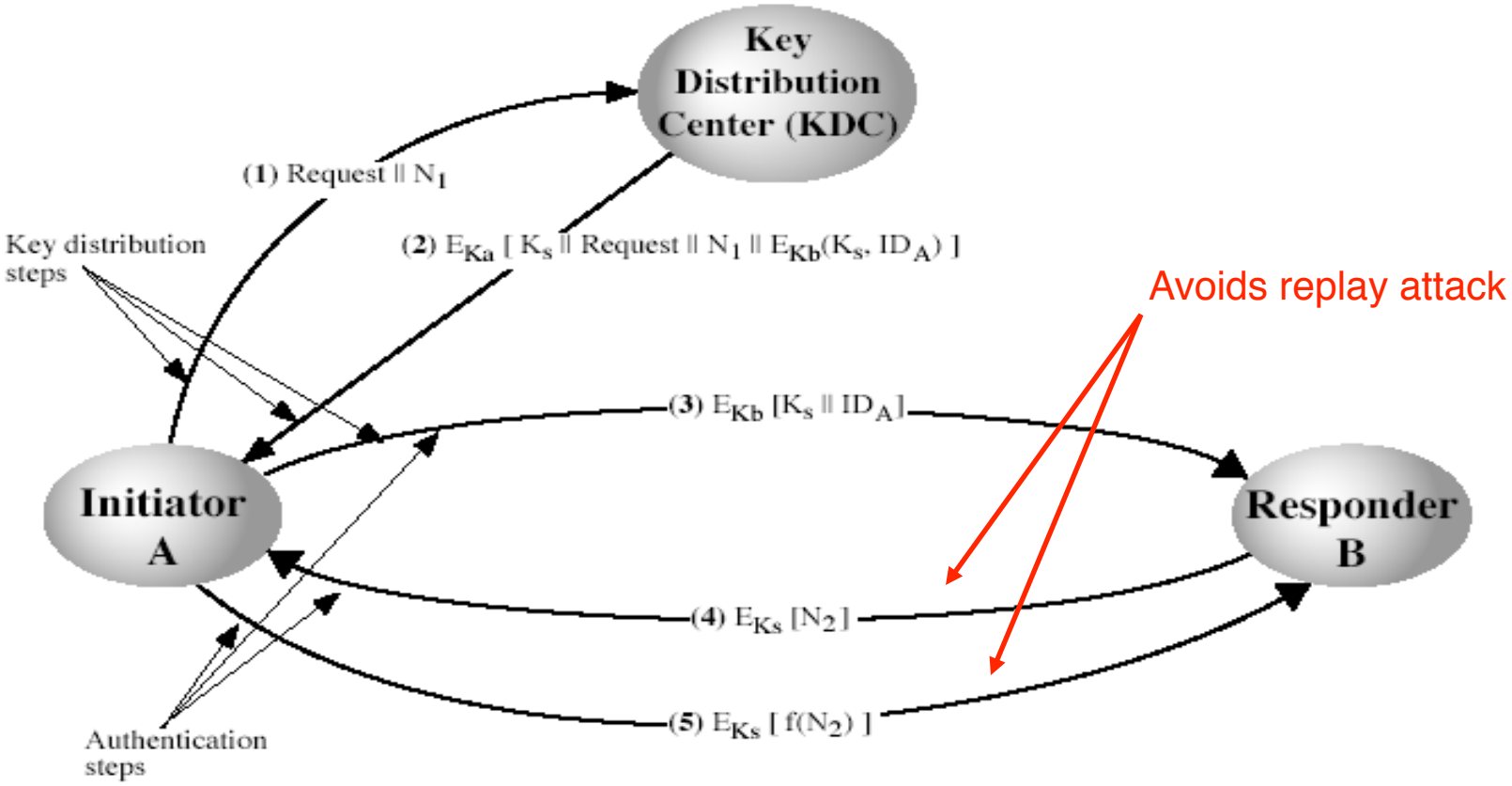
- given parties A and B have various **key distribution** alternatives:
  1. A can select key and physically deliver to B
  2. third party (trusted intermediary) can select & deliver key to A & B
  3. if A & B have communicated previously can use previous key to encrypt a new key
  4. if A & B have secure communications with a third party C, C can relay key between A & B

A Problem of Scale

- Number of keys needed depends on the number of communicating pairs that must be supported



Key Distribution Scenario (Symmetric Case)



The Logic

- In diagrams like the previous, be sure to understand *why* each step is needed, and *why* each piece of information is needed in each step.
- Ex. Steps 4 and 5 prevent replay attack.

Public Key Management

- public-key encryption helps address key distribution problems
- have two aspects of this:
  - distribution of public keys
  - use of public-key encryption to distribute secret keys

Distribution of Public Keys

- can be considered as using one of:
  - Public announcement
  - Publicly available directory
  - Public-key authority
  - Public-key certificates

Public Announcement

- users distribute public keys to recipients or broadcast to community at large
  - eg. append PGP keys to email messages or post to news groups or email list
- major weakness is forgery
  - anyone can create a key claiming to be someone else and broadcast it
  - until forgery is discovered can masquerade as claimed user

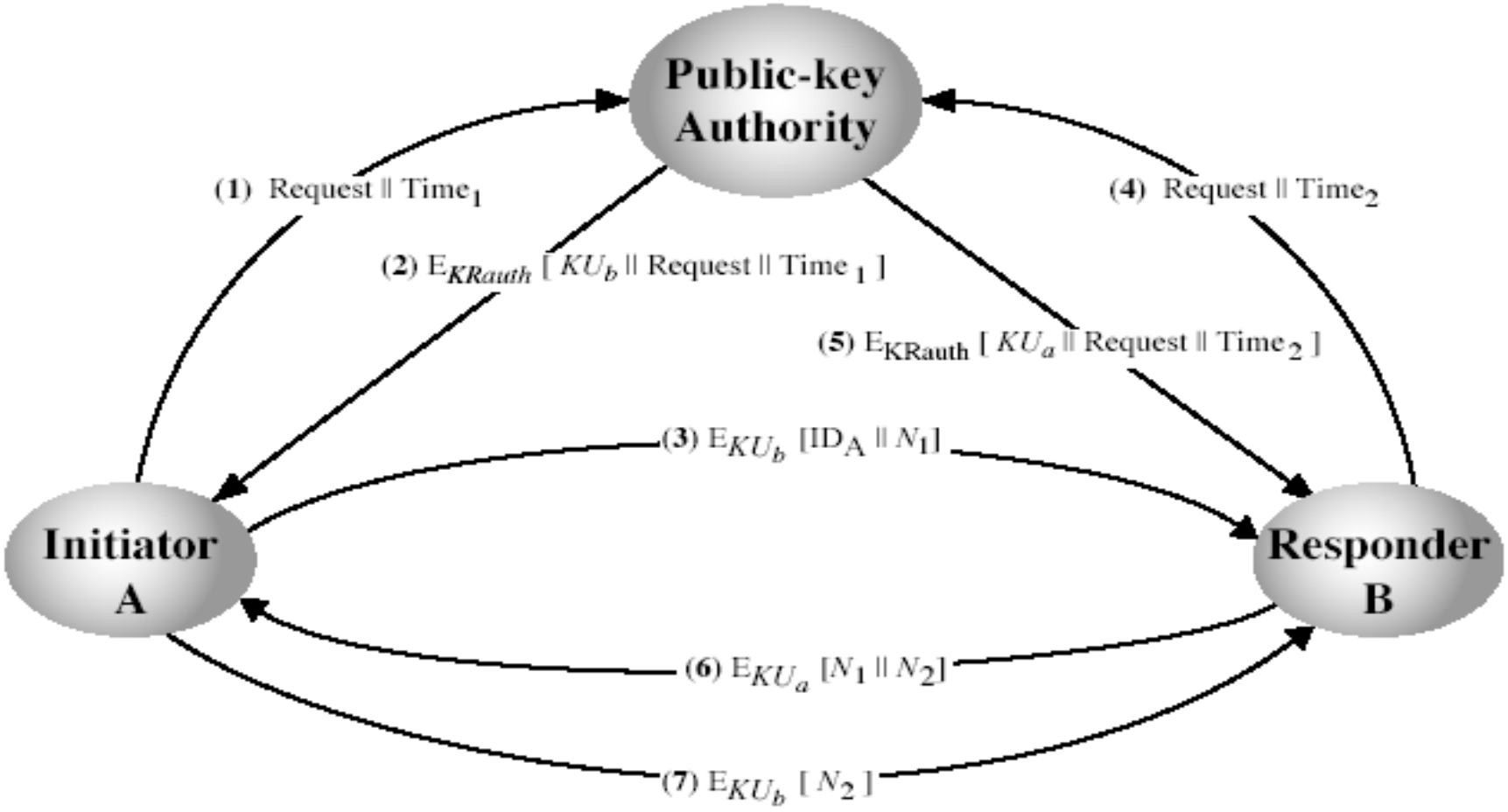
Publicly Available Directory

- can obtain greater security by registering keys with a public directory
- directory must be trusted with properties:
  - contains {name,public-key} entries
  - participants register securely with directory
  - participants can replace key at any time
  - directory is periodically published
  - directory can be accessed electronically
- still vulnerable to tampering or forgery
  - I.e., if someone gets the secret key of authority, then can pass out fake keys to everyone.

Public-Key Authority

- improve security by tightening control over distribution of keys from directory
- has properties of directory mechanism, but adds a bit more structure and the benefit of knowing data is current
- and requires users to know public key for the directory
- then users interact with directory to obtain any desired public key securely
  - does require real-time access to directory when keys are needed, which means authority can be a bottleneck

Public-Key Authority



The Logic

- So, why is each step needed, and why is each piece of information needed in each step.
- Ex. In step 2, authority returns copy of request so that A is guaranteed it was not altered in transit from A to authority
- In step 3, nonce is needed so that when step 6 occurs, A knows that only B could be the originator of the message (no one else knows the nonce), etc.

Public-Key Certificates

- certificates allow key exchange without real-time access to public-key authority
- a certificate binds **identity** to **public key**
  - usually with other info such as period of validity, rights of use etc
- with all contents **signed** by a trusted Public-Key or Certificate Authority (CA)
- can be verified by anyone who knows the public-key authority's public-key

Public-Key Certificate Properties

1. Any participant can read the certificate to determine name and public key of owner
2. Any participant can verify that certificate originated from the certification authority and is not counterfeit
3. Only certificate authority can create and update certificates
4. Any participant can verify the currency of the certificate
   - Certificates are akin to credit cards, so having an expiration date is a good thing. (Otherwise, someone who has stolen a private key can steal info in perpetuity)

Public-Key Certificates