Security

Thomas Schwarz, SJ

Basics

- A dependable system provides availability, reliability, safety, maintainability, confidentiality, and integrity.
- **Confidentiality**: refers to the property that information is disclosed only to authorized parties.
- **Integrity**: alterations to a system's assets can be made only in an authorized way, ensuring accuracy and completeness.

Basics

- We attempt to protect against security threats:
  1. Unauthorized information disclosure (confidentiality)
  2. Unauthorized information modification (integrity)
  3. Unauthorized denial of use (availability)

Basics

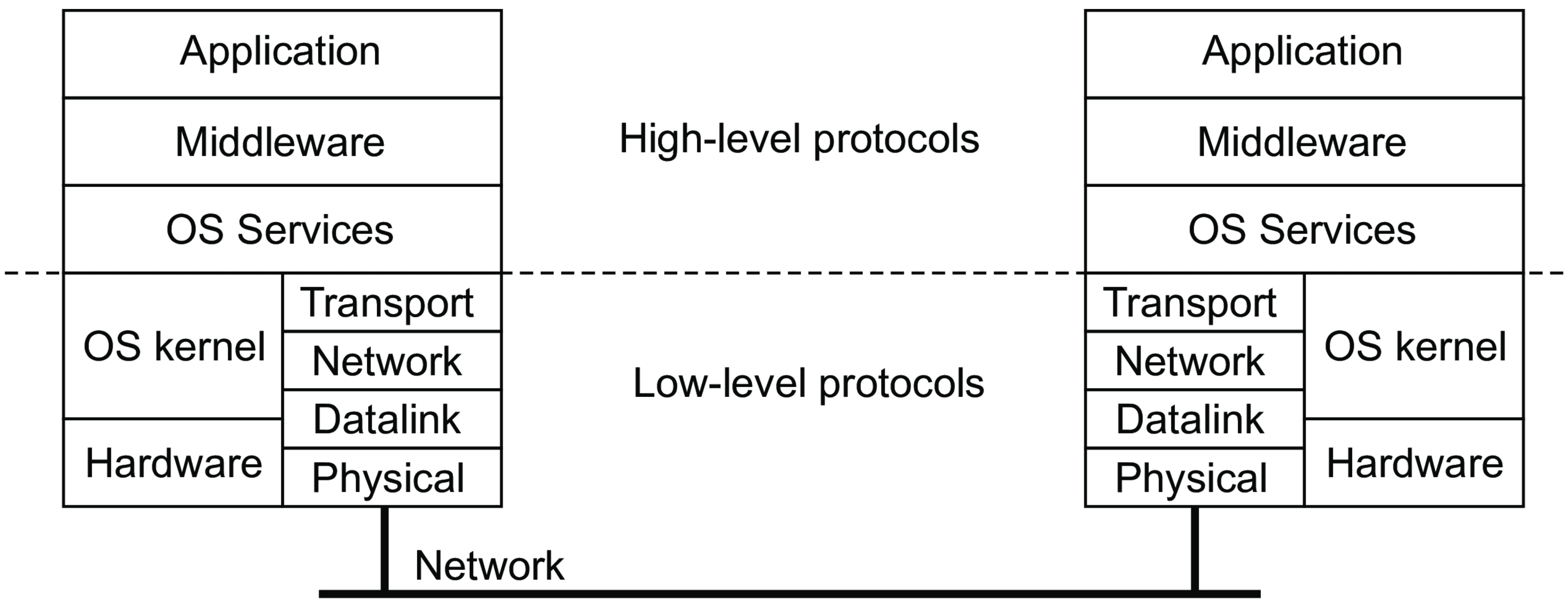
- Mechanisms:
  - **Encryption**: transform data to something an attacker cannot understand, or that can be checked for modifications.
  - **Authentication**: verify a claimed identity.
  - **Authorization**: check an authenticated entity whether it has the proper rights to access resources.
  - **Monitoring and auditing**: (continuously) trace access to resources

Basics

- Security principles:
  - Fail-safe defaults: defaults should already provide good protection. Infamous example: the default “admin.admin” for edge devices.
  - Open design: do not apply security by obscurity: every aspect of a distributed system is open for review.
  - Separation of privilege: ensure that critical aspects of a system can never be fully controlled by just a single entity.
  - Least privilege: a process should operate with the fewest possible privileges.
  - Least common mechanism: if multiple components require the same mechanism, then they should all be offered the same implementation of that mechanism.

Basics

- Where to implement?



Basics

- We are increasingly seeing end-to-end security, meaning that mechanisms are implemented at the level of applications.

- Trusted Computing Base: The set of all security mechanisms in a (distributed) computer system that are necessary and sufficient to enforce a security policy.

Privacy

- Privacy and confidentiality are closely related, yet are different. Privacy can be invaded, whereas confidentiality can be breached
    - ensuring confidentiality is not enough to guarantee privacy.

Privacy

- Right to privacy
- The right to privacy is about “a right to appropriate flow of personal information.”
    - Control who gets to see what, when, and how
        - a person should be able to stop and revoke a flow of personal information.

Privacy

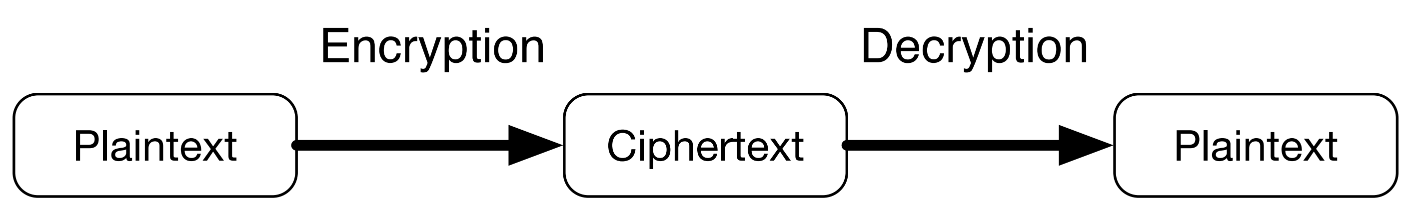
- **General Data Protection Regulation (GDPR)**
  - European Union regulation
  - The GDPR is a comprehensive set of regulations aiming to protect personal data.

Privacy

GDPR regulation	Impact on database systems	
	Attributes	Actions
Collect data for explicit purposes	Purpose	Metadata indexing
Do not store data indefinitely	TTL	Timely deletion
Inform customers about GDPR metadata associated with their data	Purpose, TTL, Origin, Sharing	Metadata indexing
Allow customers to access their data	Person id	Metadata indexing
Allow customers to erase their data	TTL	Timely deletion
Do not use data for objected reasons	Objections	Metadata indexing
Allow customers to withdraw from algorithmic decision-making	Automated decisions	Metadata indexing
Safeguard and restrict access to data		Access control
Do not grant unlimited access to data		Access control
Audit operations on personal data	Audit trail	Monitor and log
Implement appropriate data security		Encryption
Share audit trails from affected systems	Audit trail	Monitor and log

Cryptography

- Traditional use / symmetric encryption:
  - Confidentiality of information in transit or at rest



- Encryption uses a key (or another secret)
- Decryption (up till lately) uses the same key

Cryptography

- Rail Fence Scheme (used by Spartans)
  - A transposition scheme
    - Symbols stay but position is scrambled
    - Secret is the pattern

T						C						O					TCO
	H				A		K				E		F				HAKEF
		E		L				P		P				T		E	ELPPTE
			B						O						H		BOH

TCOHAKEFELPPTEBOH

Cryptography

- Caesar's Cipher
  - Substitution Code
    - Key: A shift amount x, often represented by a letter
    - Encryption:
      - Each letter is replaced by a letter moved by x positions down in the alphabet
    - Decryption:
      - Each letter is replaced by a letter moved by x positions upwards in the alphabet
- Example

```
GALLIAOMNIAESTDIVISAINPARTESTRES
PITTRIYVXRINCDMRERCIRXZIBDNCDBNC
```

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

Cryptography

- General Substitution Cipher
  - Key: Uses a fixed permutation of letters
    - Example: Generate a permutation of letters using a random number generator, the key is the seed of the generator
  - Encryption:
    - Replaces a letter by the image of the letter under the permutation
  - Decryption:
    - Replaces a letter by the image of the letter under the inverse permutation

Vigenère Cipher

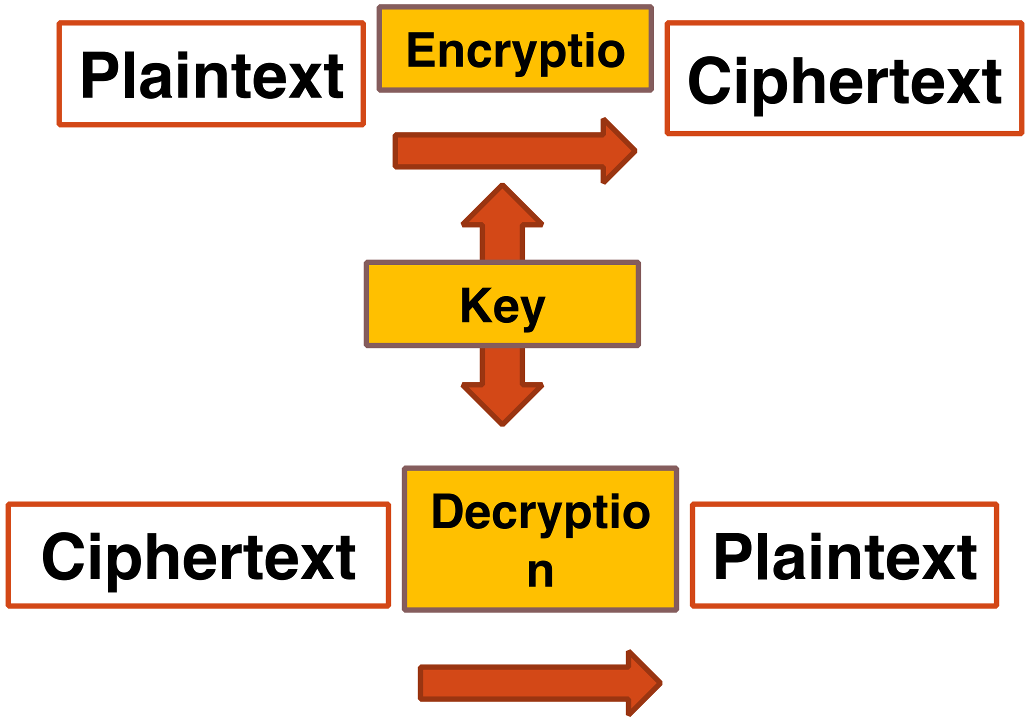
- A.k.a: The unbreakable code
  - Idea: Use a Caesar cipher, but vary the shift amount with each letter
  - Key is a long phrase
    - CSA used cipher disks and phrases
      - “Manchester Bluff”
      - “Complete Victory”
      - “Come Retribution”
  - To encode a letter:
    - Move the inner disk’s A under the current letter of the key phrase
    - Take the letter of the plain text
    - Read the inner disk’s letter under that letter in the outer disk.
    - This is your cipher letter

CSA cipher disk: Luckily, CSA SIGSEC was atrocious

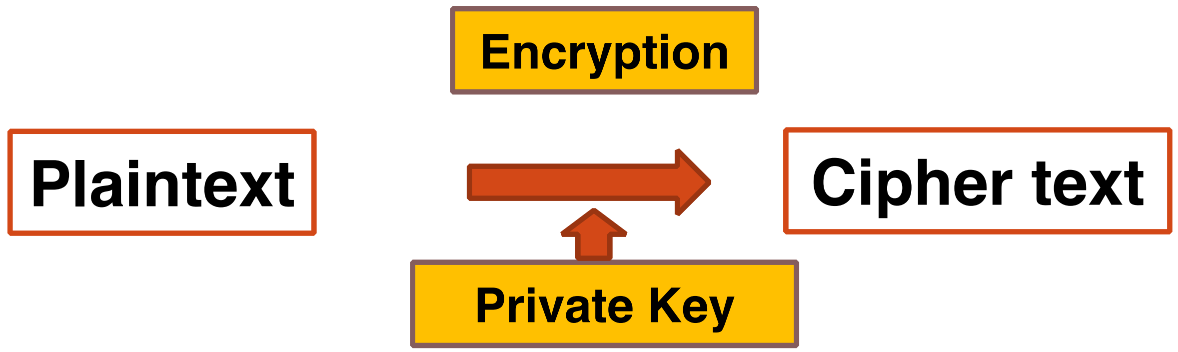
Vigenère Cipher

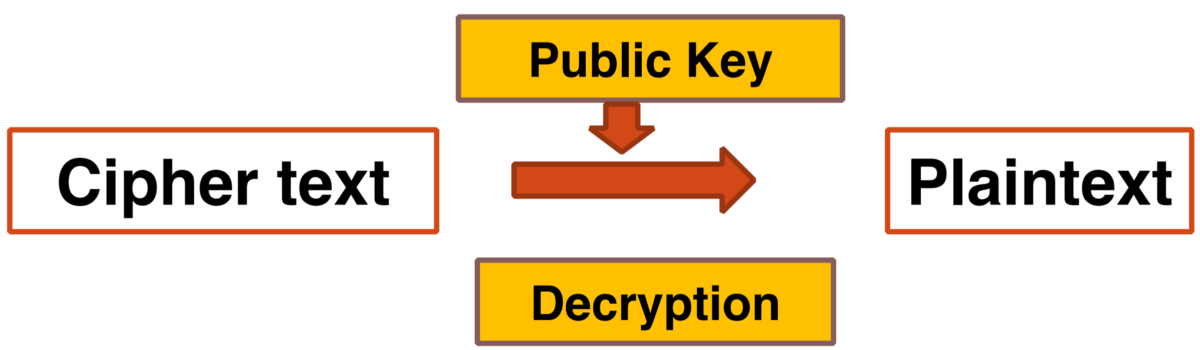
- Example
  - “Divisions (commanded by) Maj. Gen. Picket, Brig. General Pettigrew and Maj. Gen. Trimble will charge cemetery hill after a preliminary bombardment”
  - “DIVISIONSPICKETPETTIGREWTRIMBLEWILLCHAR”
  - key: “FINALVICTORY”
  - cipher: D^F, I^I,V^N,I^A,S^L, …
  - Result:
  - IQIIDDWPLDZAPMGPPOBKZFVUYZVMMGMYBZCAMIE

Secret Key Cryptography



Public Key Cryptography





Using Public Keys

- Alice has public - private key pair P_A and R_A
- Bob has public - private key pair P_B and R_A
- Alice wants to send Bob a private message
  - Invents a symmetric key K
  - Sends $m_K, P_B(K)$ to Bob
    - Only Bob can decrypt $P_B(K)$ as $K = R_B(P_B(K))$ and therefore decrypt m_K

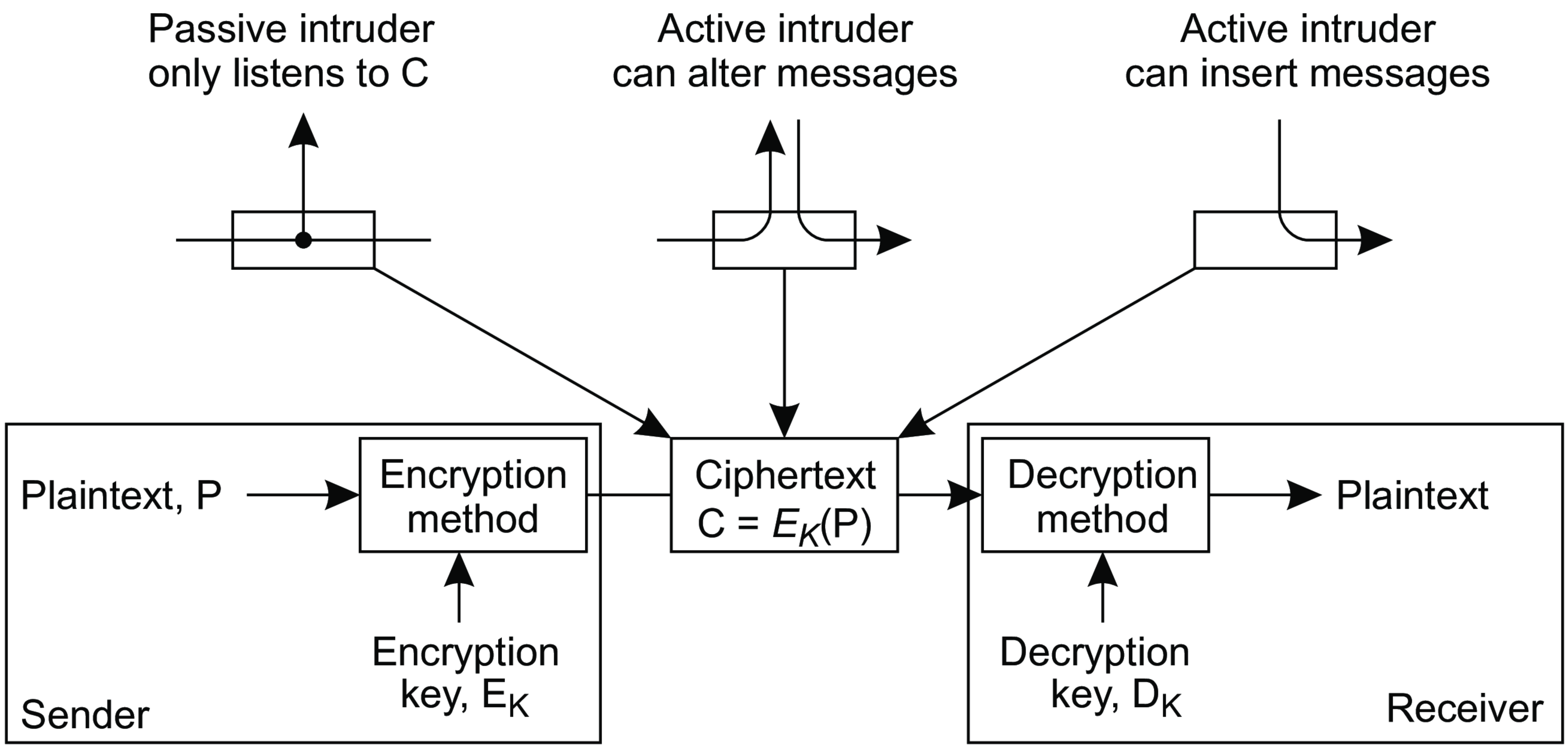
Using Public Keys

- Alice has public - private key pair P_A and R_A
- Bob has public - private key pair P_B and R_A
- Alice wants to sign a message
  - Sends $\left(R_A(m), m\right)$
  - Anyone knowing P_A can decrypt: $m = P_A(R_A(m))$

Homomorphic Encryption

- Can use operations directly on encrypted data
    - $m_K \times n_K = (m \oplus n)_K$

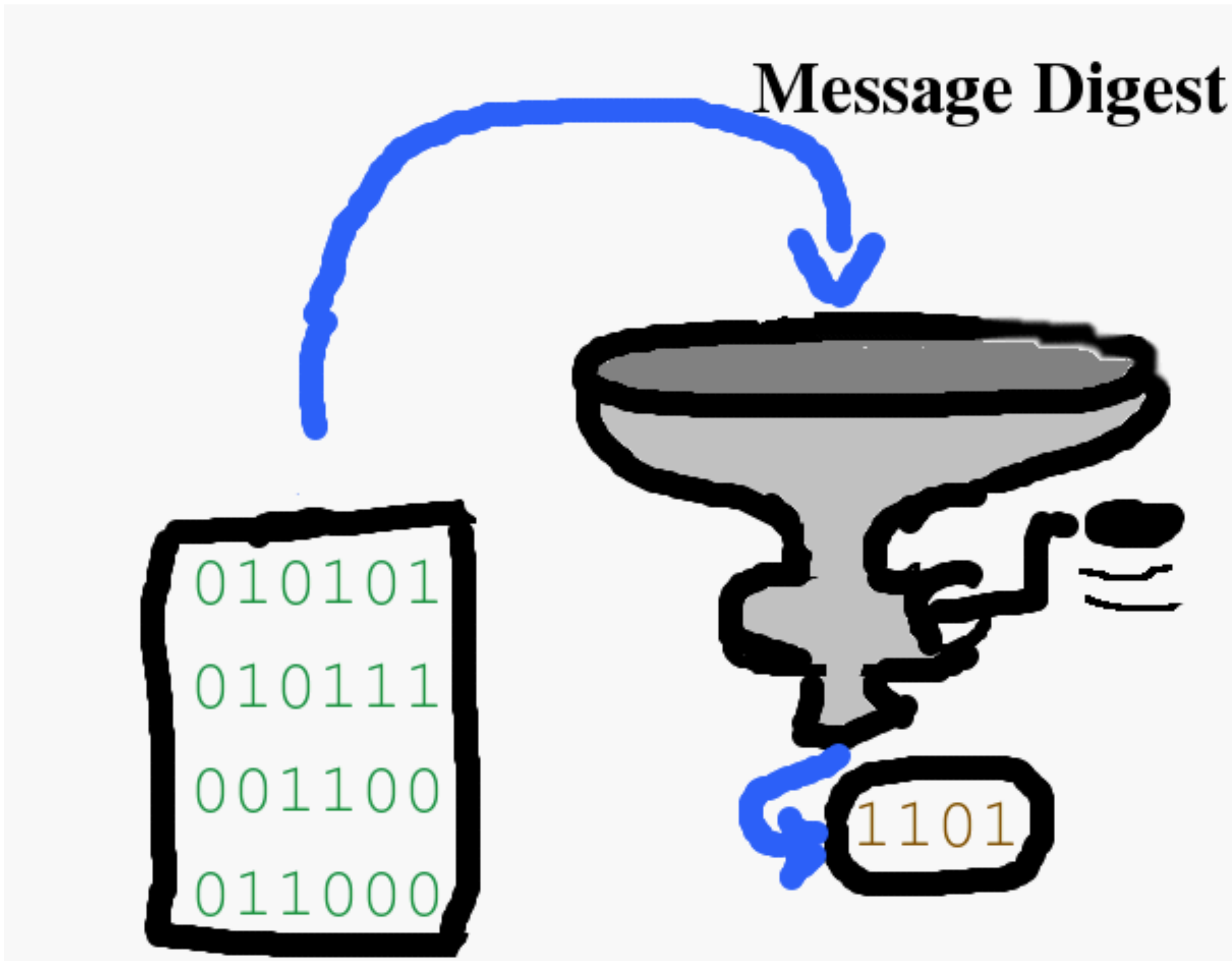
Attacks on Communication Channel



Examples

- Pedestrian behavior
  - Detect device information of people walking with a cell-phone or similar device
    - Cellphone signals
    - Wifi enabled cell-phones: MAC address
  - Traffic planning does not know about the identity
    - Find a way to anonymize identities

Hashes and message Digests



Hashes

- Hashes are one-way functions
  - Space of BLOB (Binary Large OBjects) to space of small fixed-length string
  - Blob changes a little bit induces big changes in hash

Hashes

- Various notions of security
  - Pre-image resistance
    - Given hash *h* it should be hard to find a blob *m* such that hash(*m*) = *h*
  - Second pre-image resistance
    - Given blob *m* it should be hard to find a blob *n* such that hash(*m*) = hash(*n*)
  - Collision resistance
    - It should be hard to find two different blobs *m* and *n* such that hash(*m*) = hash(*n*)

Hashes

- “Provably” secure hashes:
  - If there is an algorithm that finds a pre-image
  - Then we can solve an NP difficult problem / probabilistic NP difficult problem in polynomial time
- Tend to be too slow

Hashes

- Can be used to ID documents.
  - Computer Forensics:
    - Use hashes of known good functions (OS, standard software) to exclude artifacts from examination
  - Public signatures of documents
    - Instead of encrypting objects with private key
    - Encrypt secure hash of object with private key
      - Because public key encryption takes time
    - $E_p(O)$ vs $E_p(h(O))$

Hashes

- Large number of proposed hashes
- Selection process by NIST
  - Validation of implementation through CMVP

Hashes for signing

- Public encryption is expensive
  - Alice has public - private key pair U_A and R_A
  - Alice wants to sign a message
  - Sends $\left(R_A(h(m)), m\right)$
    - Only she knows R_A
  - Anyone knowing can decrypt: $U_A(R_A(h(m))) \stackrel{?}{=} h(m)$

Hashes

Name	Size	Speed	Status
MD5	128b	335MB/sec	completely broken
SHA 0	160b		broken
SHA-1	160b	192 MB/sec	no practical attacks yet but theoretically broken
SHA-2	224b / 256b / 384b / 512 b	139 - 154 MB/sec	secure but not recommended
SHA-3	224b - 512b		recommended

Key Management

Thomas Schwarz, SJ

Key-Management

- Keys are generated by programs
  - Hashes of user-provided passwords
  - Random strings
  - Public-private key programs
- Problem is key-distribution

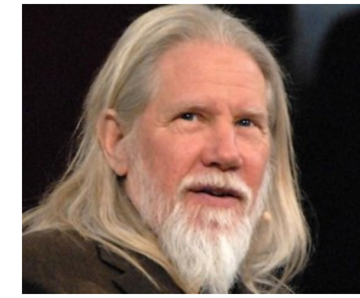
Session Keys

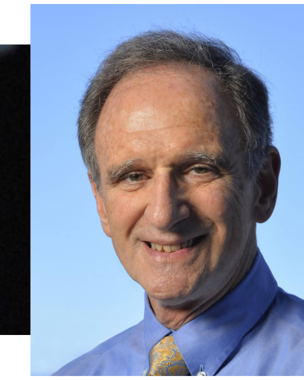
- Common practice to use a ephemeral key used during a single session
  - Can be generated by Diffie-Hellman
  - Invented by one of the parties and distributed with a master key
- Provides “forward security”:
  - If an adversary stores encrypted communication and later obtains the key information of one of sender and recipient, then the adversary can still not decrypt the communication

Key Management

- Session keys
  - Use Diffie-Hellman
    - Based on the hardness of the discrete logarithm problem
  - Share prime p and multiplicative generator g
    - $\{g^i \% p \mid i \in \{0,1,\ldots,p-1\}\}$
  - Alice invents a and sends g^a to Bob
  - Bob invents b and sends g^b to Alice
  - Both use $g^{ab} = \left(g^a\right)^b = \left(g^b\right)^a$
  - Only they can do this calculation

Key Management

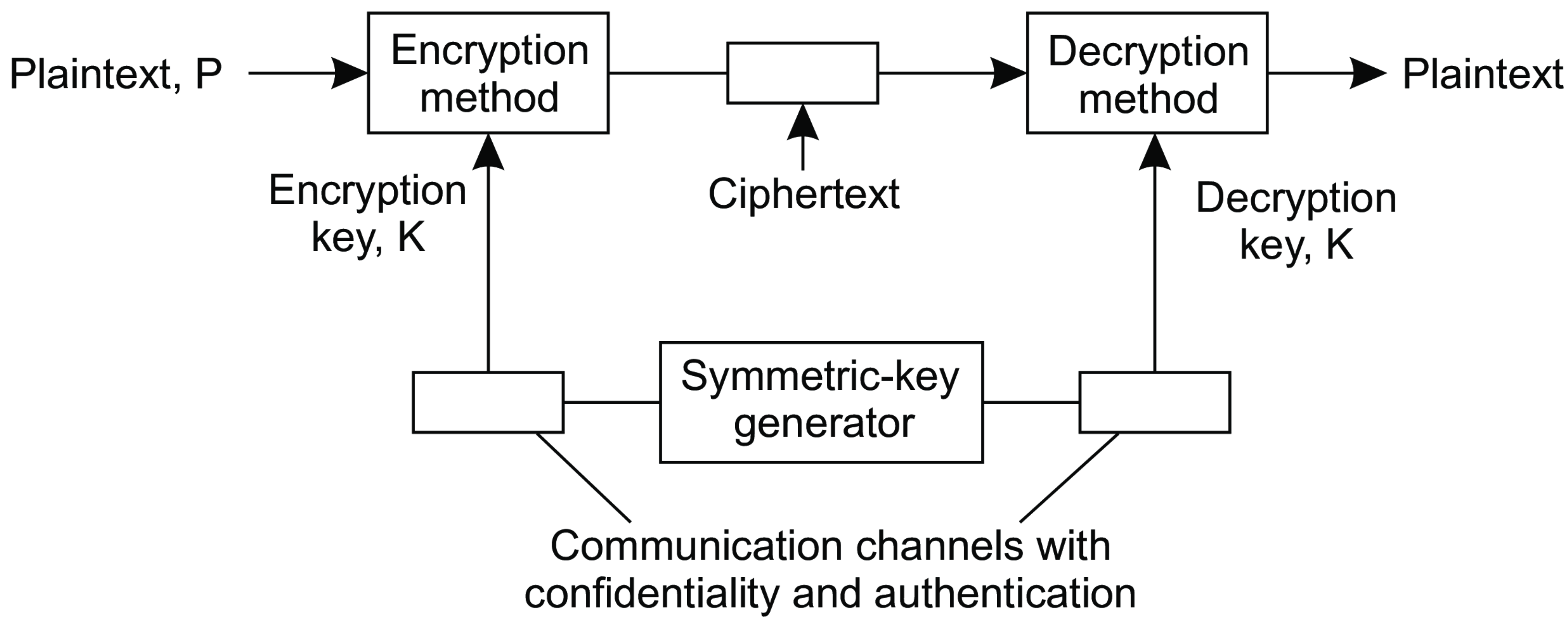




- Distributed Diffie-Hellman:
  - All servers agree on a generator g and p
  - Each server S_i invents a secret a_i
  - Servers S_i and S_j communicate via the secret key $g^{a_i a_j}$

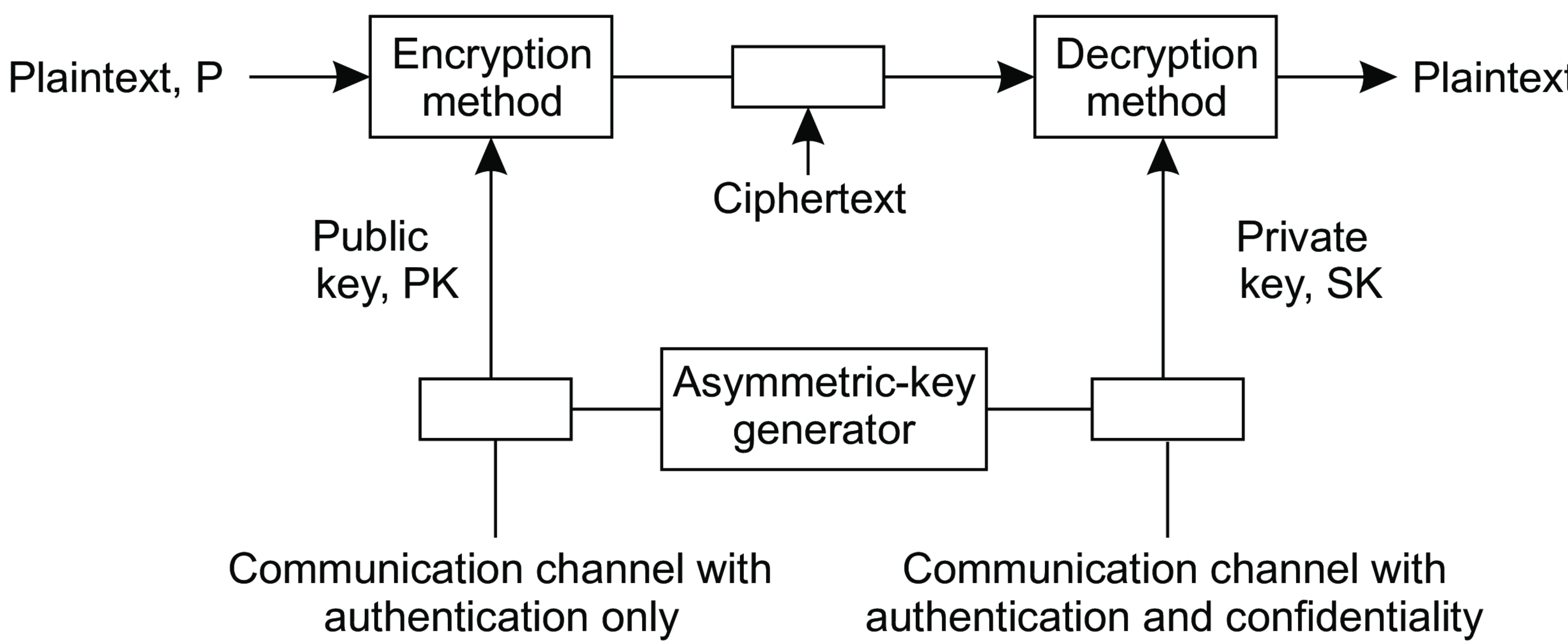
Key Management

- Key distribution
  - For secret keys:

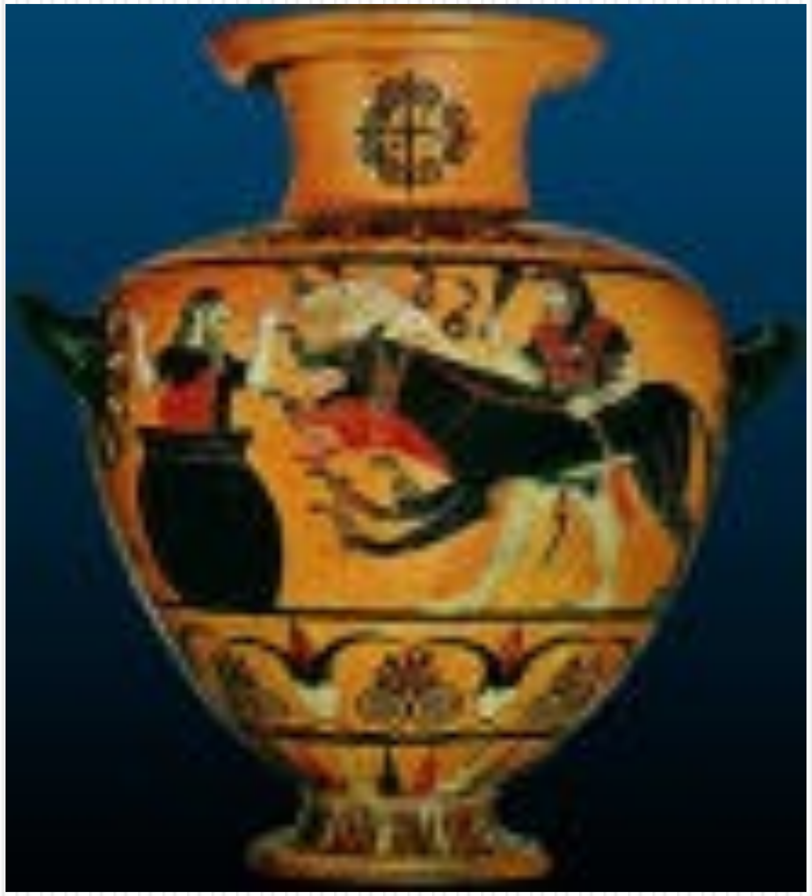


Key Management

- Key distribution
  - For public-private keys:



Kerberos



Kerberos

- Simplifies administration of authentication
  - User signs in on her/his workstation
    - Using password, …
  - User is automatically authenticated to all services
  - Network is not considered safe
- Uses the concepts of
  - KDC – Mediated authentication
  - Trusted intermediary
- Implements Single Sign-On SSO

Kerberos

- Kerberos: Developed at MIT based on the work by Needham Schroeder
  - Uses symmetric encryption (for patent reasons)
  - Available in versions 4 and 5
    - Version 4 has better performance, but needs TCP/IP
    - Version 5 has more functionality
      - Part of Windows OS
      - Available in Linux as Heimdal
      - Available in variants for commercial UNIX and Apple OS X
  - PKINIT is a version using public keys

Kerberos

- Tickets and Ticket Granting Tickets (TGT)
  - All principals have a shared, secret key with the KDC: master key / principal key
  - If Alice needs a service from Bob, she goes to the KDC to obtain a session key K_{AB} and a ticket for B
    - Alice's *Credential:* K_{AB} and the ticket for Bob
    - Alice cannot read what is in the ticket

Kerberos

- Original Authentication
  - Alice authenticates with her workstation
  - Her master key is derived from her password
  - Her workstation asks the KDC for a session key: S_{Alice}
  - Session key is valid for some hours
    - Protects Alice's master key
  - She also obtains a TGT (Ticket Granting Ticket)
    - [S_{Alice}, Information on Alice's identity, expiration time] encrypted by the master key of the KDC
  - Her workstation only maintains the TGT and S_{Alice}

Kerberos

Alice password →

Machine

KRB_AS_REQ
"Alice needs TGT" →

KDC

KRB_AS_REP
$K_{Alice}\{S_{Alice}, TGT\}$ ←

TGT = K_{KDC}\{"Alice", S_{Alice}\}

Kerberos

- Alice's credentials are encrypted with K_{Alice} but the TGT is already encrypted
- Why do we need a TGT?
  - If Alice needs a service, her machine sends the TGT with the request to the KDC
  - The KDC decrypts the request and has all the information needed on Alice
  - KDC does not need to maintain state such as information send previously to Alice.

Kerberos

- In order to obtain service from Bob, Alice uses a program
  - The program needs to interact with Kerberos
- Her workstation generates a request to the KDC for a ticket
  - The request contains an authenticator
- KDC uses the information in the TGT to authenticate Alice and returns a ticket.
- Her workstation sends ticket and authenticator to Bob
- Bob deciphers the ticket to obtain the session key and uses the authenticator in order to authenticate Alice
- Bob then sends his authenticator to Alice who now can authenticate Bob

Kerberos

- Wants service from Bob.

Alice
rlogin
Bob

M a c h i n e

KRB_TGS_REQ
"Alice needs Bob"
TGT = K_{KDC} {"Alice", S_{Alice} }
authenticator = S_{Alice} {time}

KRB_TGS_REP
S_{Alice} {"Bob", K_{AB}, Ticket}
Ticket = K_{Bob} {"Alice", K_{AB}}

K D C

Kerberos

Machine

KRB_AP_REQ

Ticket = K_{Bob} {"Alice", K_{AB}}

authenticator = K_{AB} {time}

→

Bob

KRB_AP_REP

K_{AB} {time +1}

←

Kerberos

- Login implementation:
  - V4: Kerberos asks for login only after credentials are obtained from the KDC
    - Minimizes the time that the password is stored in machine memory
    - Makes it easier for an adversary who can obtain information from the KDC for a dictionary attack
  - V5: Kerberos asks for the password before getting credentials from the KDC

Kerberos

- Replicated KDC
  - KDC is a single failure point
  - Replicas need access to the master database of users
  - Usually use Master-Slave protocol
    - Only the master can change the database
    - Master keys in database are encrypted by the master key of the KDC and protected in transit
    - The rest of the contents are stored as hashes.

Kerberos

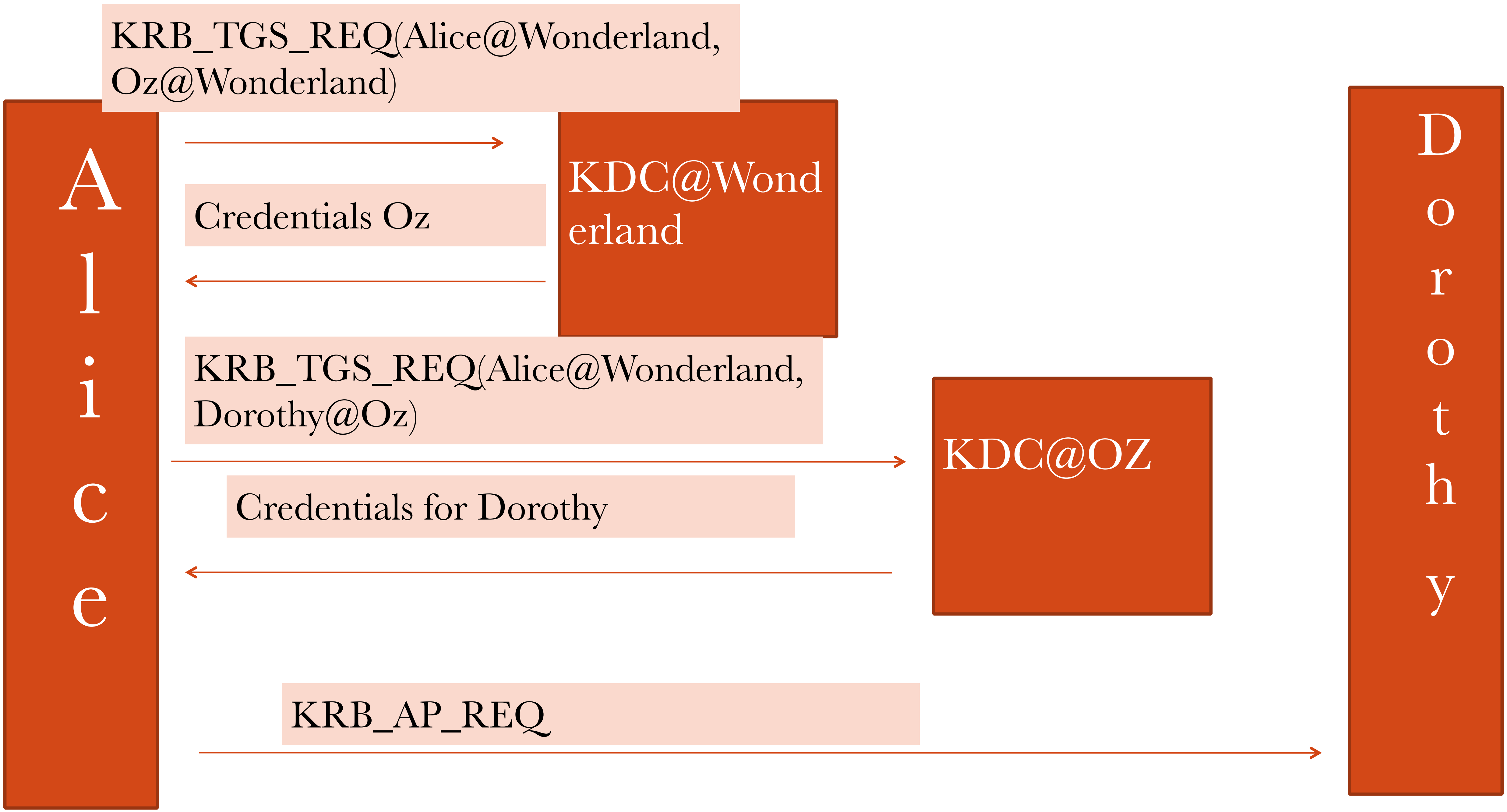
- Realms
  - Principals of Kerberos are divided into reals with their own database
  - The KDC in each realm have their own master keys.
- V4: Every principal has a name consisting of three components
  - Name
  - Instance
  - Realm
  - E.g: Alice.Systemmanager.DMSCS

Kerberos

- To authenticate for principals in another realm
  - Need to find a chain of KDCs that know each other
  - One obtains the TGT of the previous KDC for the subsequent KDC
  - Until one reaches the KDC for the realm of the target

Kerberos

ΕΛΛΑΣ - HELLAS Λ.30



KRB_TGS_REQ(Alice@Wonderland, Oz@Wonderland)
Alice
KDC@Wonderland
Credentials Oz
KRB_TGS_REQ(Alice@Wonderland, Dorothy@Oz)
KDC@OZ
Credentials for Dorothy
KRB_AP_REQ
Dorothy

Kerberos

- Change of keys:
  - Naive solution
    - Bob changes key
    - Alice’s credential no longer work
    - Her machine needs to find new credentials based on Bob’s correct key
    - Not always possible in a batch job
  - Implemented solution
    - All keys have a version number
    - Ticket expire in 21 hours, so there is no problem having more than one version of the key
    - Users might still have problems:
      - A change of password is effective immediately in a master KDC
      - A slave KDC can use the previous key until updating the database
        - Which results in failure to authenticate Alice

Kerberos

- V4: IP-directions in tickets
  - As an additional authentication mechanism:
    - All tickets have the IPv4 address of the user
      - Prevents Alice from delegating her rights to someone else
      - Prevents interception of credentials
        - The spoofer needs to obtain ticket and authenticator and prevent that the true authenticator gets to the server
        - Now needs to falsify also his IP address

Kerberos

- Changes in V5
  - Uses ASN.1 (data representation language)
  - Example:
    - Address in V4: 4 bytes
      - Assumes IPv4
    - V5:
      - addr-type needs a byte to specify if it is type 0 and another one to specify the length
      - INTEGER needs a byte to specify that it is an integer, one for length, and at least one for the value itself
      - address[1] needs four bytes plus the value
      - 11 additional bytes

```
HostAddress := SEQUENCE           {
          addr-type[0]        INTEGER
          address[1]          BYTE STRING }
```

Kerberos

- V5: Names
  - simpler and without prohibited characters like “.”
  - contain a type and a variable number of fields.

Kerberos

- Delegation of rights
  - Capacity to give access to someone else over something over which one has a right
    - Necessary if someone (Bob) acts instead of another entity
    - Cannot be implemented by sharing master key
  - V5: Alice can request a TGT with Bob's address
    - Can only be used by Bob
  - V5: Alice can request a ticket with Bob's IP-address
    - This limits what Bob can do in lieu of Alice
  - Additional interaction for delegation allows KDC to establish an audit trail
  - Possible to put in a rule in the master database to specify
    - If TGT can be used to obtain a TGT or ticket with different network address
    - If the TGT can be forwarded.

Kerberos

- Implementation of Delegation (continued)
  - Flag Forwardable:
    - TGT with this flag can be exchanged for a TGT with different network address
      - Allows Alice to give a TGT to Bob with which Bob can obtain tickets in lieu of Alice
  - Flag Proxiable:
    - TGT with this flag can be utilized to obtain tickets with different network address
    - Allows Alice to obtain a proxy ticket to give to Bob,
    - Does not allow Alice to get a TGT for Bob

Kerberos

- Implementation of Delegation (continued)
  - TGT can have flag FORWARDED
  - Tickets can have flag FORWARDED and PROXY
    - An implementation can distinguish between tickets obtained by forwarding and proxying
  - KDC and application decide on permissibility of forwarding and proxying

Kerberos

- Ticket Life Time
  - V4: Limited to about 21 hours
    - Creates problem for long running batch job
  - V5: Ticket life time is extended, implemented with
    - STARTTIME
    - ENDTIME
    - AUTHTIME
      - when Alice logged in
    - RENEWTILL

Kerberos

- Renewable tickets
  - Why? Tickets with long life are difficult to revoke
  - If Alice needs a long-life ticket, KDC sets flag RENEWABLE
  - Before ticket expiration:
    - Alice needs to renew the ticket
      - The KDC just gives a ticket with changed expiration time
  - Alice needs to run a daemon in order to renew tickets about to expire

Kerberos

- Post-dating tickets
  - Used to allow tickets to valid in the future
  - KDC emits a ticket with a flag INVALID and STARTTIME set to when the ticket will be needed
  - When the start time comes, Alice presents the ticket to get a new ticket
    - This allows easy ticket revocation
  - A post-dated ticket has a flag POSTDATED to allow an application to refuse post-dated tickets.

Kerberos

- Key versions
  - Problem with key change:
    - Tickets become suddenly invalid
  - Key versions allow server Bob to change his principal key
    - All keys carry version number
    - The database entry for Bob contains *key*, *p_kvno*, *k_kvno*
      - *key* – key of Bob encrypted by the principal key of KDC
      - *p_kvno* – version number of Bob's key
      - *k_kvno* – version number of the KDC key used
      - Allows the KDC to change key

Kerberos

- If human user Alice is registered in various realms, she probably wants to reuse the same password
- To avoid generating the same principal key, V5 generates it as hash of the password and the realm name
- Does not avoid dictionary attacks but protects good passwords

Kerberos

- Hierarchy of realms
  - V4: To obtain a ticket of another realm, both realms need to be mutually registered
  - V5: Allows a request to pass through a chain of realms
  - An intermediate KDC can pretend to be any user in the whole world
    - Counter-measure:
      - Tickets with field TRANSITED with a list of realms through which the request went
      - A rogue KDC cannot change its name in this list, only the ones that came before

Kerberos

- Measures against password guessing
  - Login attacks
    - V4 sends a request without authentication to the KDC
      - Adversary can use the resulting ticket for an offline dictionary attack
    - V5 can use PREAUTHENTICATION-DATA to prove that the user known his principal key
  - Ticket attacks
      - Alice asks KDC for a ticket for a human user, Bob
      - Receives a ticket encrypted with Bob's principal key
      - She can now verify a guess of Bob's password
    - V5 has a field in the user list that disallows the creation of a ticket.

Kerberos

- PKINIT
  - Version that avoids dictionary attacks on passwords
  - Two modes:
    - Certificates
    - Diffie-Hellman

Kerberos

- PKINIT
  - Uses Public Key Infrastructure
    - All principals have public / private key pairs
      - For Alice:
        - Public key U_{Alice}
        - Private key R_{Alice}
    - All principals have certificates
      - (Alice - Alice’s public key)$_{\text{signed by certifying authority}}$

Kerberos

Machine

KDC

KRB_AS_REQ:
"Alice needs TGT from KDC"
n_1
Certificate(Alice, U_{Alice})
R_{Alice}\{timestamp, n_2\}

KDC invents symmetric key k
and session key S_{Alice}
checksum is a keyed hash of Alice's request

KRB_AS_REP:
Alice, TGT,
U_{Alice}\{Certificate(KDC, U_{KDC}) , $R_{KDC}\{k, checksum\}$\}
$k\{S_{Alice}, n_1, TGT\}$

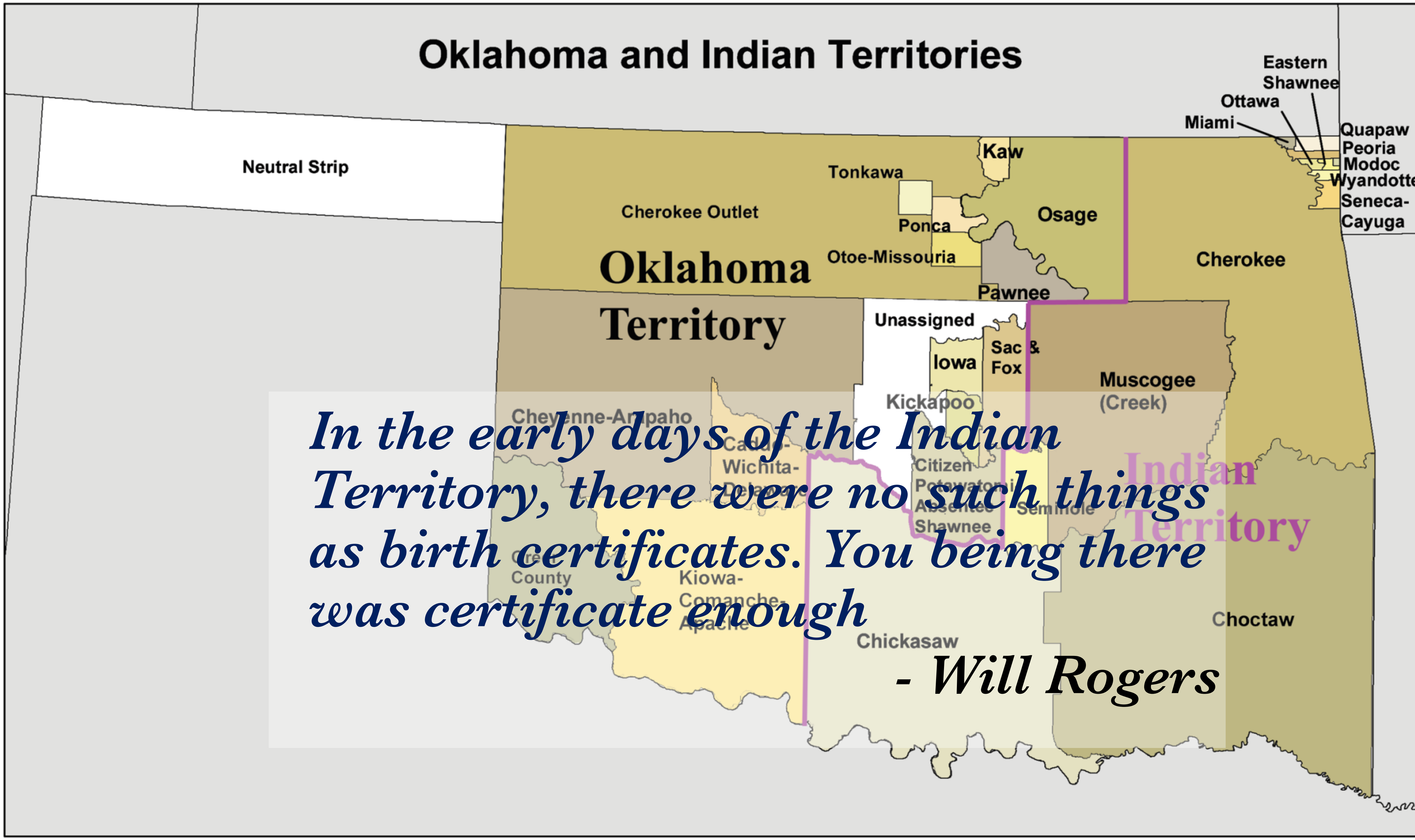
Kerberos

- Diffie-Hellman modality
  - Secret key *k* is not invented by KDC
  - Rather: it is generated by a Diffie-Hellman exchange between Alice and KDC

Key Management

- Distribute public keys using certificates


PKI
PUBLIC KEY INFRASTRUCTURE



In the early days of the Indian Territory, there were no such things as birth certificates. You being there was certificate enough

- Will Rogers

Certificates

- Certificates link entity's name to its public key
- Are signed by a certifying authority

`Alice has public key 1450293485797`$_{\texttt{signed Carol}}$

Alice is the *subject*
Carol is the *issuer*
If Bob uses the certificate,
he becomes the *verifier*

Certificates

- Need to specify algorithms used
- Add validity dates

X.509

X.509 Version Number

Serial Number

Signature Algorithm Identifier

Issuer (X.500 Name)

Validity Period (Start – Expiration dates / times)

Subject (X.500 Name)

Subject Public Key Information: Algorithm Identifier, Public Key Value

Issuer Unique Identifier

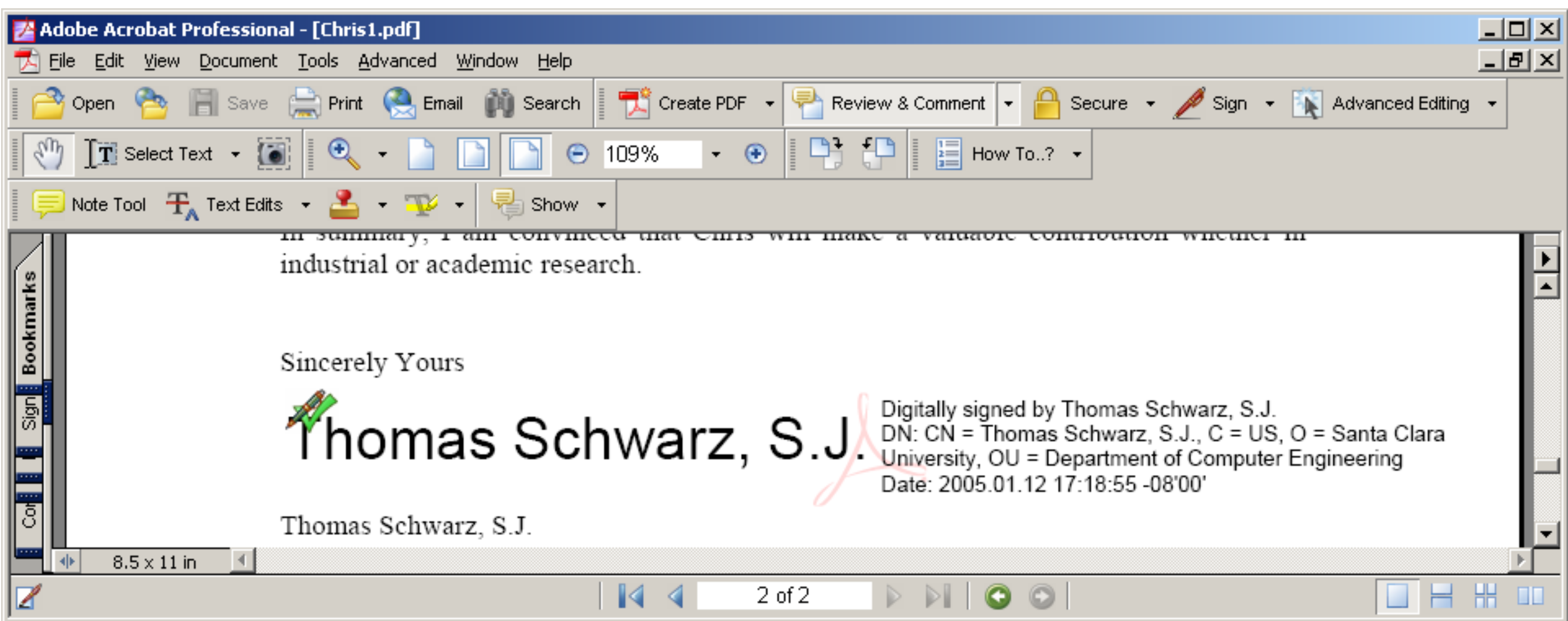
CA Digital Signature

Certificates

- Naming is a security concern
  - Example: “Microsoft Corporation” vs “Microsoft Corporation”
    - Look the same, but are different unicode strings (the second one uses the Hungarian letter o)
  - How many “Bill Smith” are there?

Names

- X 500 names
  - Common name, country, organization, organizational unit



X.509 Certificates

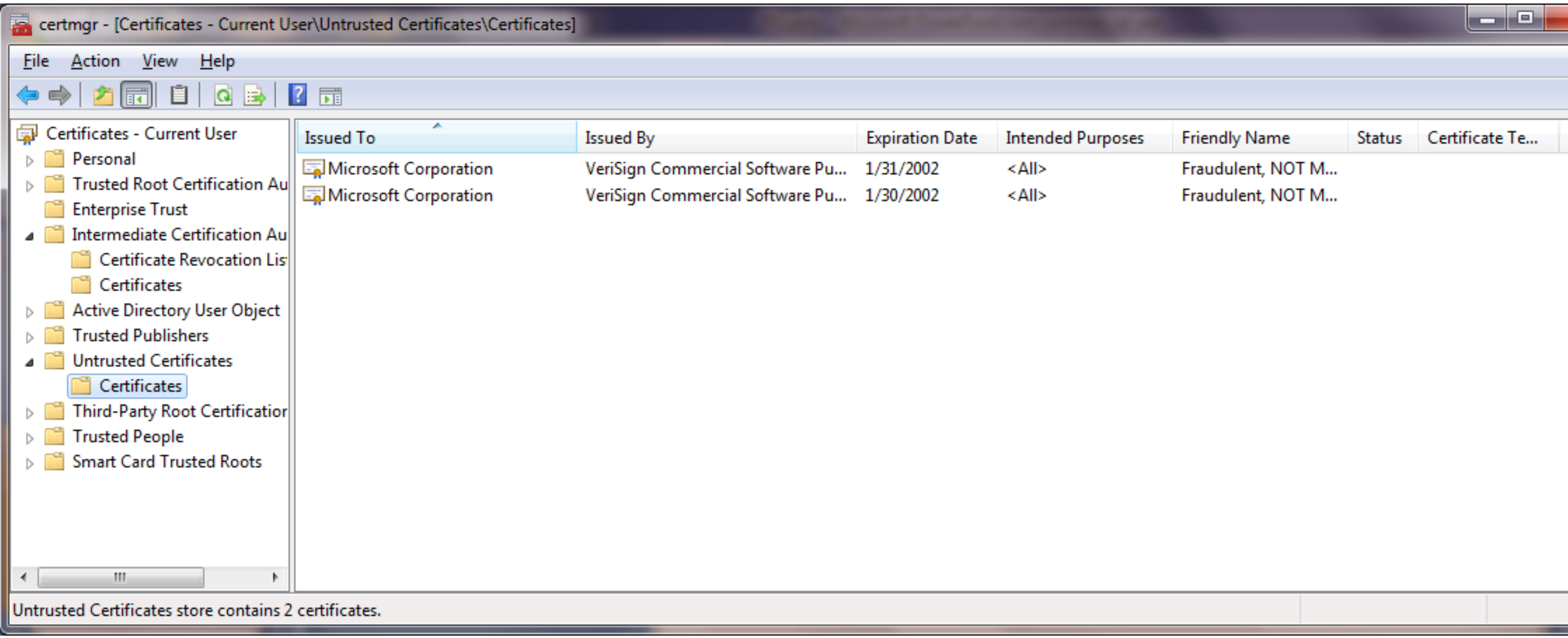
- Uses X.500 identifiers for used algorithms
  - Issuer signature
  - Public key certified
- Algorithms are registered and numbered in the Abstract Syntax Notation 1

X.509 Certificate

- VERSION — 2 or 3
- SERIALNUMBER — identifies uniquely certificates of issuer
- SIGNATURE — Method used for signing
- VALIDITY — start and end time of validity
- SUBJECT
- SUBJECTPUBLICKEYINFO
- ISSUERUNIQUEIDENTIFIER
- ALGORITHMIDENTIFIER
- ENCRYPTED

Value of Certificates

- Value of certificate depends on the diligence of the issuer in verification of the identity of the subject
  - Verisign once issued a certificate for Microsoft on a stolen credit card
    - Claimed to have changed procedures so that it could never happen again
    - But would not say how for security reasons


certmgr - [Certificates - Current User\Untrusted Certificates\Certificates]
File Action View Help
Certificates - Current User
Personal
Trusted Root Certification Au
Enterprise Trust
Intermediate Certification Au
Certificate Revocation Lis
Certificates
Active Directory User Object
Trusted Publishers
Untrusted Certificates
Certificates
Third-Party Root Certificatior
Trusted People
Smart Card Trusted Roots
Issued To
Issued By
Expiration Date
Intended Purposes
Friendly Name
Status
Certificate Te...
Microsoft Corporation
VeriSign Commercial Software Pu...
1/31/2002
<All>
Fraudulent, NOT M...
Microsoft Corporation
VeriSign Commercial Software Pu...
1/30/2002
<All>
Fraudulent, NOT M...
Untrusted Certificates store contains 2 certificates.

Value of Certificate

- Need public / private key pairs and hence certificates for each use of asymmetric cryptography
  - One for signing
  - One for each authentication protocol
  - One for confidentiality

X.509 Extensions

- Version 3 allows extensions
  - Standard extensions
    - Key information
    - Policy information
    - Subject and issuer attributes
    - Restrictions on certificate path
    - Extensions for certificate revocation

PKIX

- Another standard from IETF 1994
  - Has extensions
    - AuthorityKeyIdentifier
    - SubjectKeyIdentifier
    - KeyUsage
    - PrivateKeyUsagePeriod
    - CertificatePolicies
    - PolicyMappings
    - SubjectAltName

Other standards

- PBP
- WAP WTL — replaces ASN.1 names with simpler ones
- DNSSEC — certificates for DNS
- SPKI (Simple PKI) RFC 2693

Revocation

- Certificates need to be revoked because of
  - Issuer mistakes
  - Loss of private keys

Revocation

- Certificate Revocation Lists (CRL)
  - Basic Idea: publish a list of revoked certificates periodically
  - Certificates are identified by serial number and issuer
  - Security problems:
    - Confidentiality:
      - Publish only serial number and hash of contents
    - Adversary can publish old CRL
      - Always publish a CRL at a given time
      - Sign CRL

Revocation

- Need to save space:
  - Delta CRL
    - Publish difference between previous and current CRL
  - Include first valid serial number
    - Allows to revoke old certificates en masse

Implementation of Revocation

- On-Line Revocation Service (OLRS)
  - Server that responds whether a certificate is valid
- Black List versus White List
  - Black List of revoked certificates
  - White List of valid certificates
  - Both prevent falsifying a certificate with a used serial number

PKI modes

- Certificate chains
  - Verifier needs to know public key of issuer
  - Might need a certificate for issuer
  - Gives raise to chains of certificates

PKI models

- Who can become an issuer?
  - Anarchy (PGP)
    - Everybody can sign certificates
    - Verifiers have a database of certificates
    - Verifier assigns trust to certificates
      - Usually multiple chains for the same subject — public key pair

PKI models

- Monopoly
  - Only one entity (certifying authority) in the universe
  - Its public key is embedded in all software and hardware products
- Who should this entity be?

PKI models

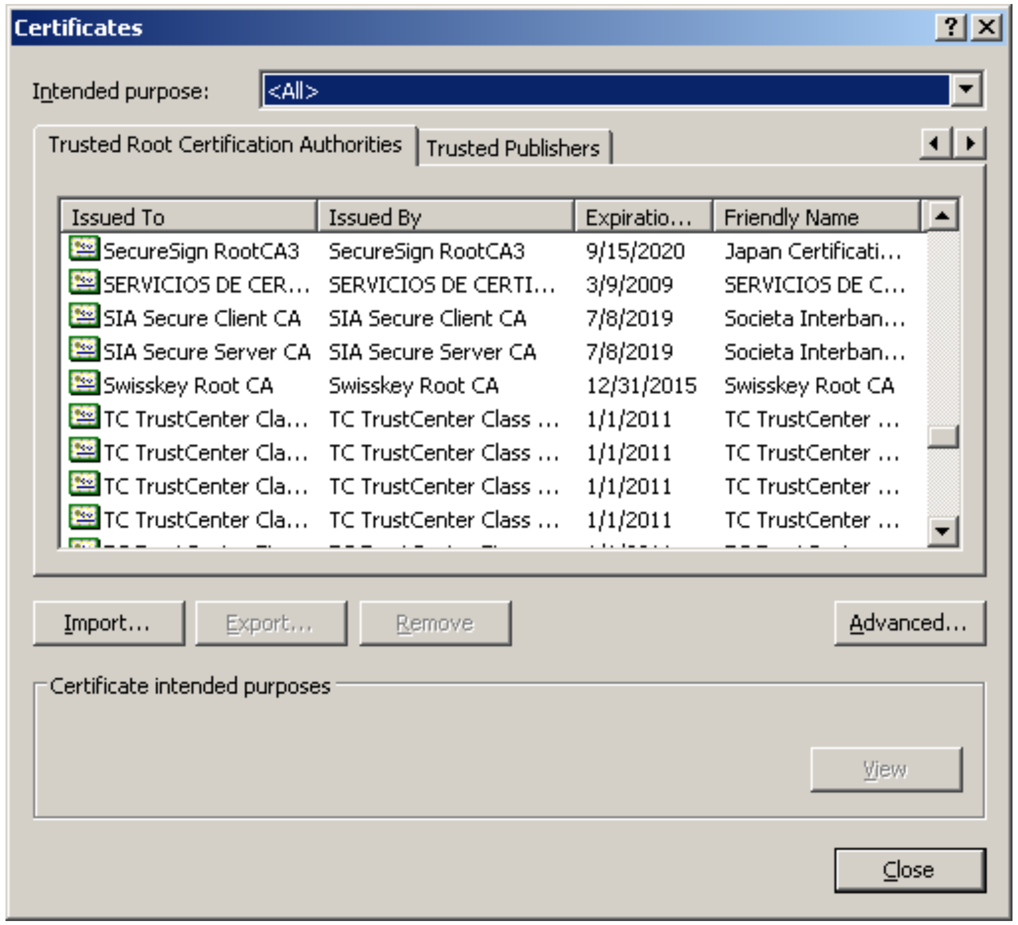
- Monopoly with Registration Authorities (RA)
  - Sole CA allows RA
    - RA authenticate the identity of a subject, create keys, and guarantee the link between subject and key
  - All certificates are from the same CA

PKI models

- Monopoly with delegated CA
  - Root CA authenticates other CAs
  - CAs sign their own certificates

PKI models

- Oligopoly
  - There are several trusted CAs
    - Software / OS manufacturer decide which CAs are trusted



PKI models

- Domain CAs
  - Certificates for users in a certain domain
    - Each CA administers its own domain
    - Possible to allow cross-site certification
    - PKIX allows to restrict certificates to certain domains

Handshake Protocols

Handshake Protocols

- Threat model
  - Passive sniffing
    - Malicious Mallory can read messages between Alice and Bob
      - but does not change / suppress them
  - Spoofing
    - Mallory pretends to be Alice or Bob
  - Breaking Crypto
  - Man-in-the-Middle
  - Replay attacks
  - Reflection attacks (open several sessions)

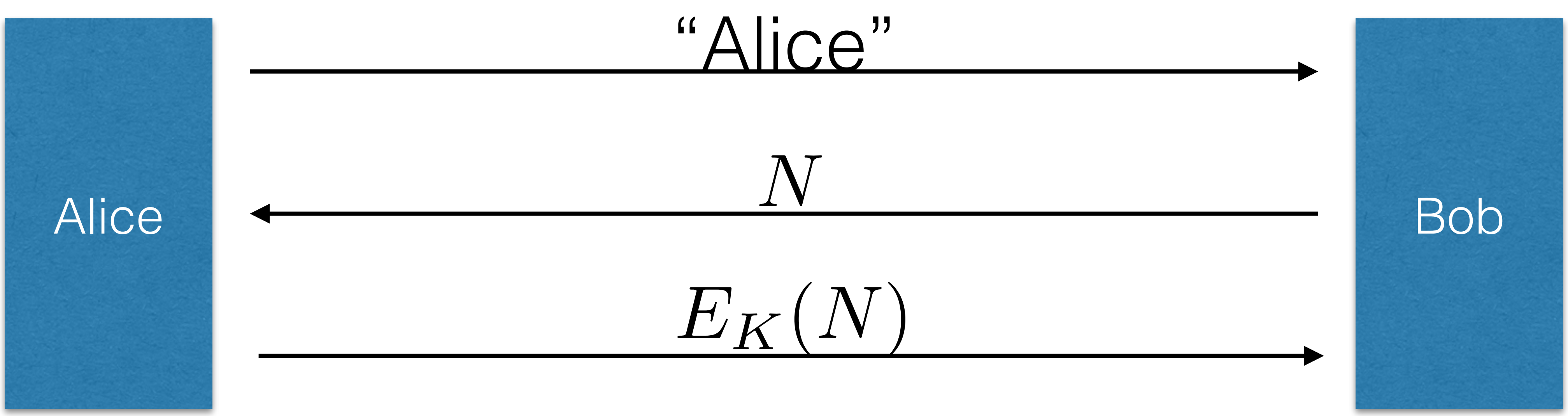
Handshake Protocols

- Simple password protocol

- Vulnerable to:
  - Sniffing
  - Spoofing (Mallory pretends to be Bob)
  - Replay attacks

Handshake Protocols

- One sided authentication (Alice to Bob)
  - Alice and Bob share secret K and use symmetric encryption



- Bob challenges Alice with a random number N.
  - A nonce
- Alice encrypts the random number
- Bob verifies the encryption

Handshake Protocols

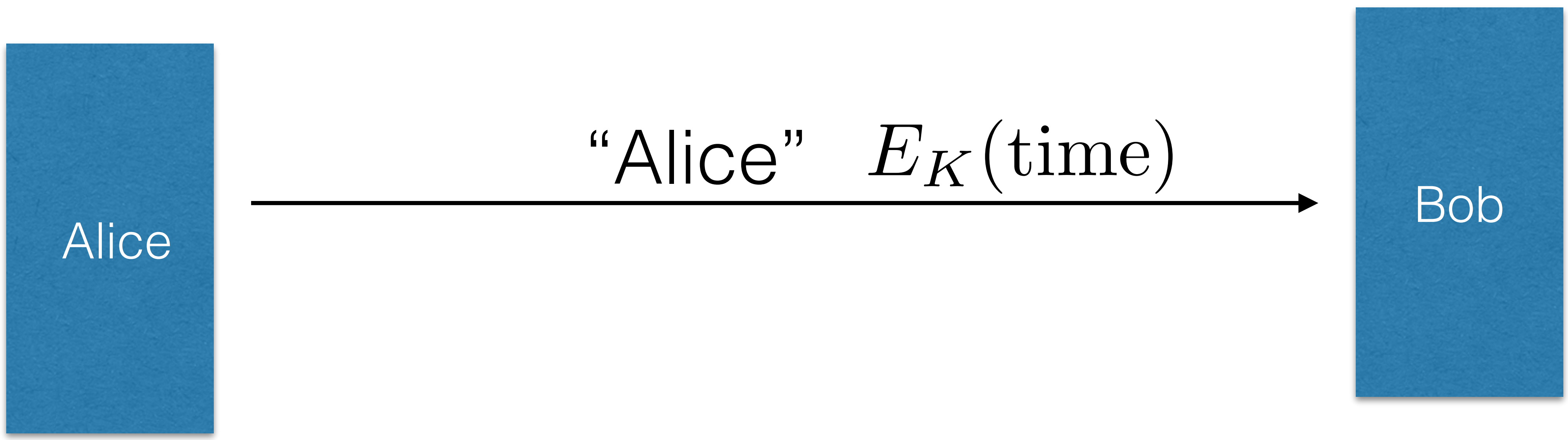
- Variations
  - Bob asks Alice to decrypt a random value that Bob has encrypted
  - Same vulnerabilities and same work

Handshake Protocols

- Vulnerable to
  - Denial of Service Attack
    - Mallory makes lots of login attempts
    - Each time, Bob encrypts something
  - Can be vulnerable to sniffing / replay if the random number is not random or repeated

Handshake Protocols

- Clock-based scheme



- Instead of a challenge, Alice just encrypts the time

Handshake Protocols

- Clock-based schemes
  - Problem is clock drift
    - If Bob demands too much accuracy, then clocks need to be closely synchronized
      - Synchronization can be a point of attack
    - Otherwise, vulnerable to replay attack

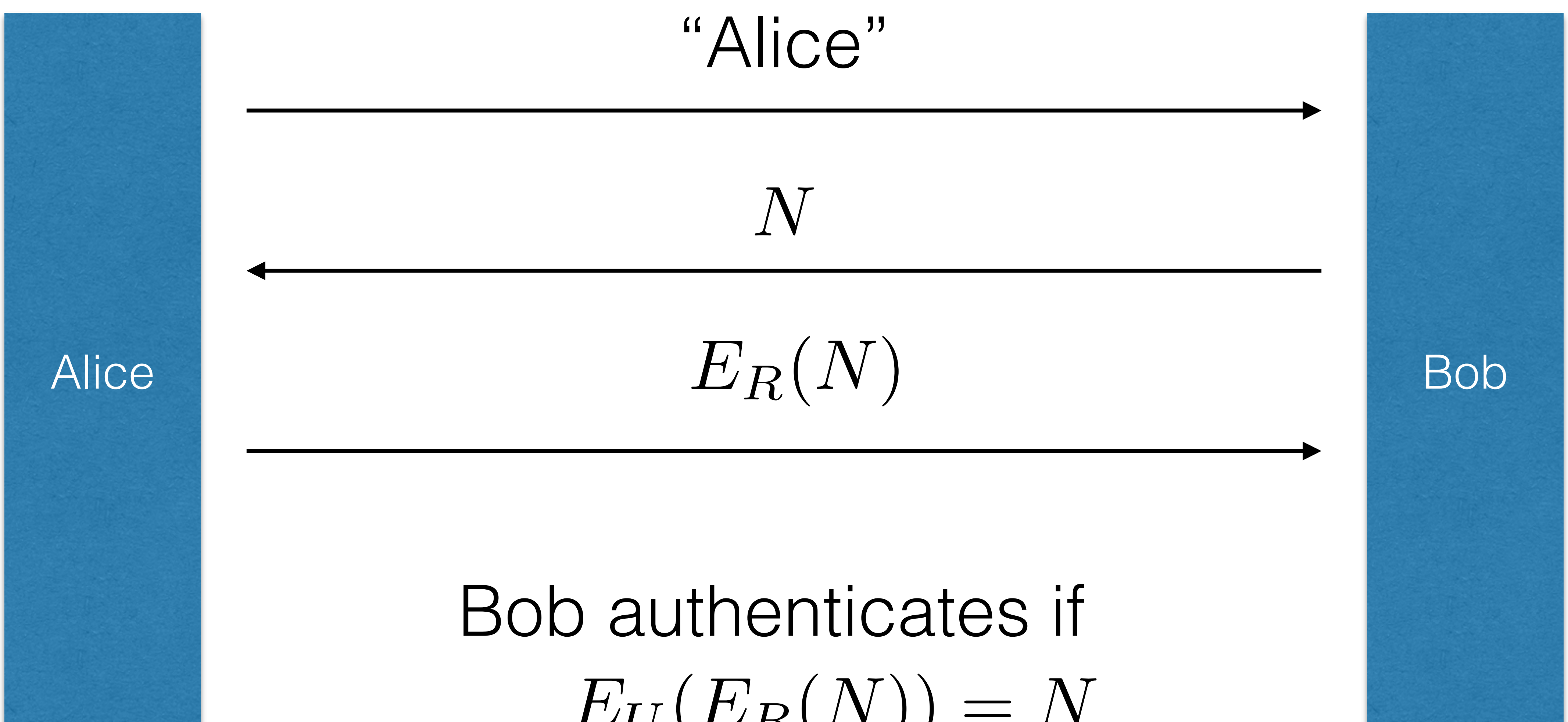
Handshake Protocols

- How to organize a replay attack
  - Mallory fills up Bob's message queue with pseudo-messages
  - Alice sends login message to Bob, which is intercepted

    "Alice, $E_K(\text{time})$"
  - Mallory stops attacking Bob's message buffer and starts attacking Alice's
  - Mallory resends the intercepted login message

Handshake Protocols

- Public key based one-sided authentication
  - Alice has private key R and public key U

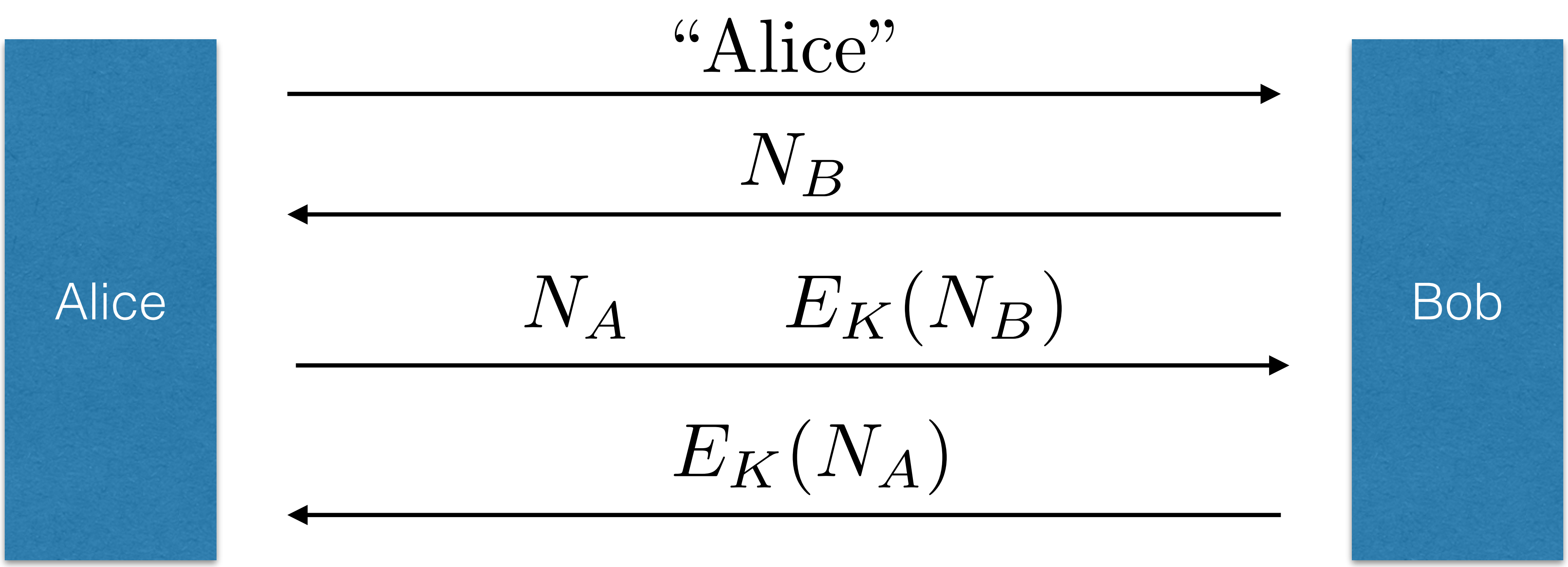


Handshake Protocols

- Variation
  - Bob can challenge with a key encrypted with Alice's public key
- Vulnerable to DoS attack
  - Bob spends time on public-key cryptography for each login attempt

Mutual Authentication

- With symmetric cryptography: Both Alice and Bob challenge each other with nonces

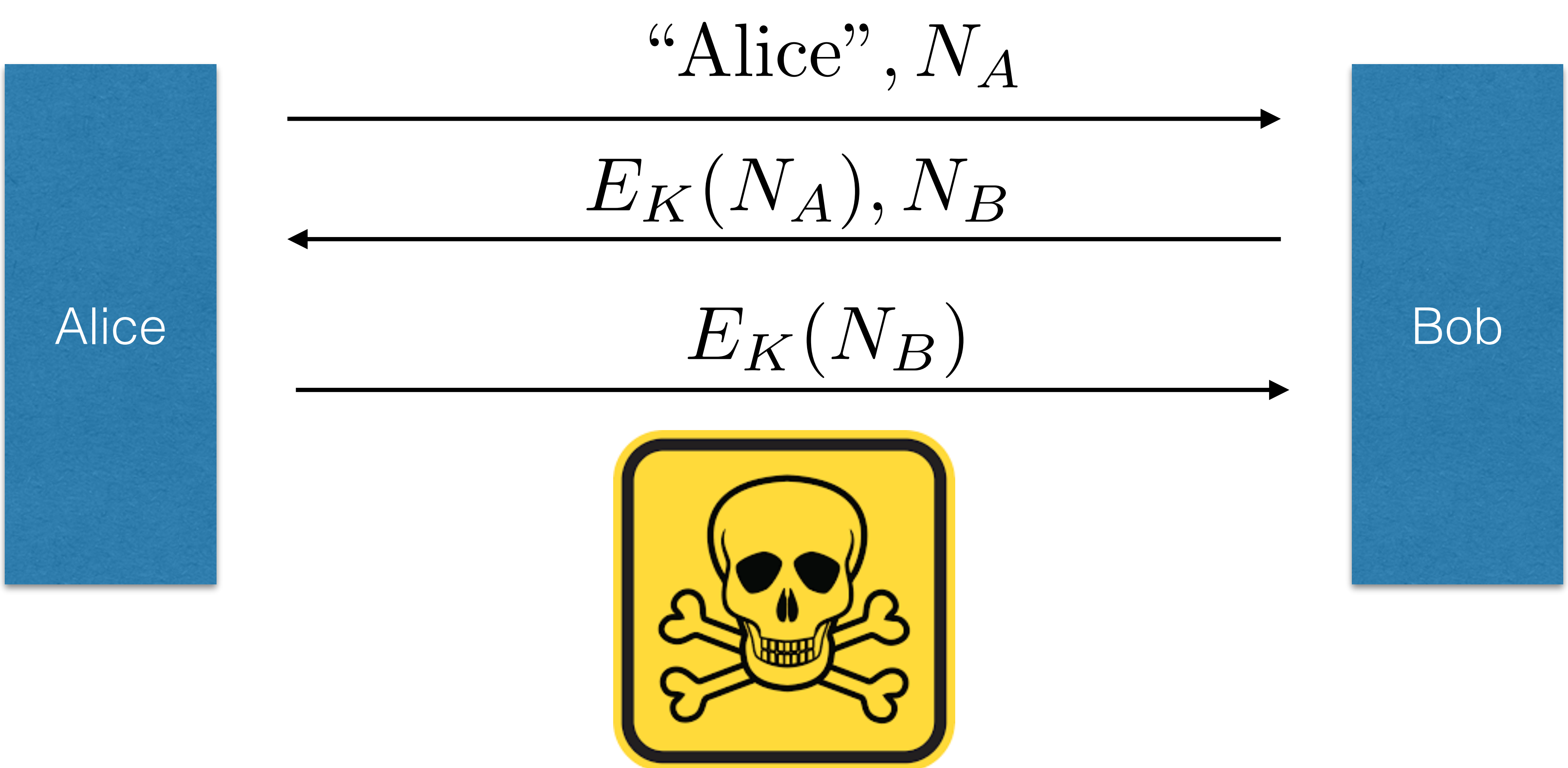


Mutual Authentication

- Less vulnerable to Denial-of-Service attacks
- Uses four rounds

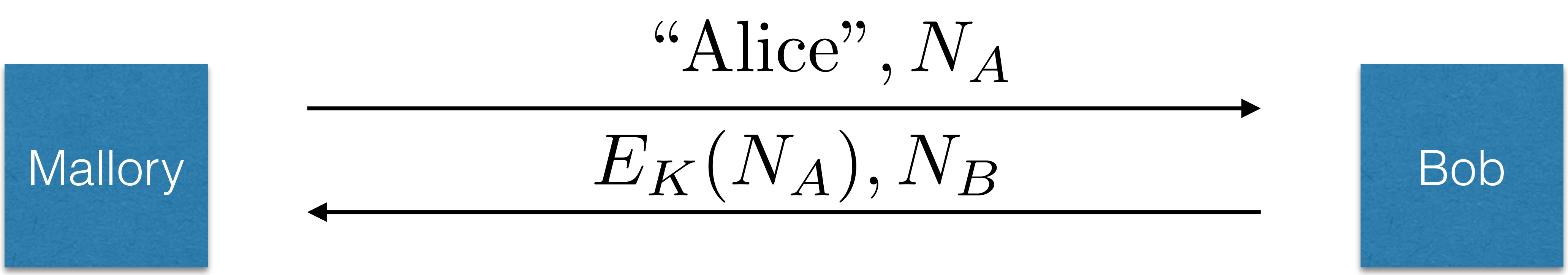
Mutual Authentication

- To save one round, one can have Alice challenge Bob first



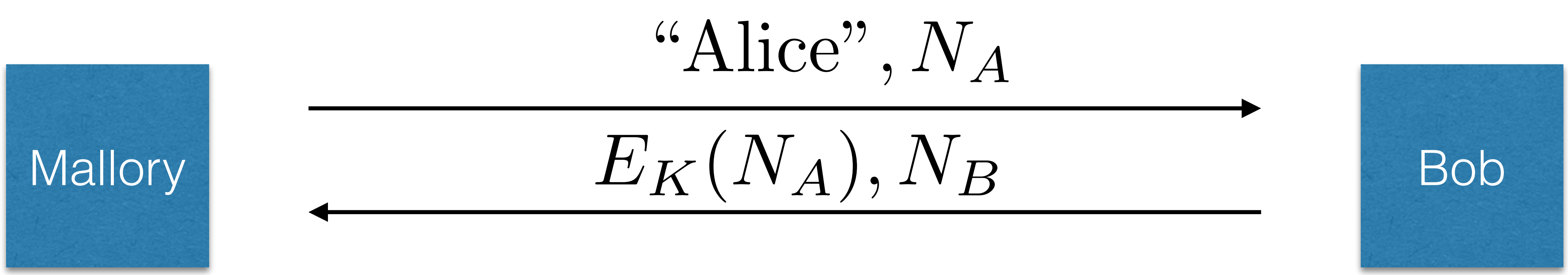
Mutual Authentication

- The three-round protocol is vulnerable to a replay attack
  - Mallory pretends to be Alice.



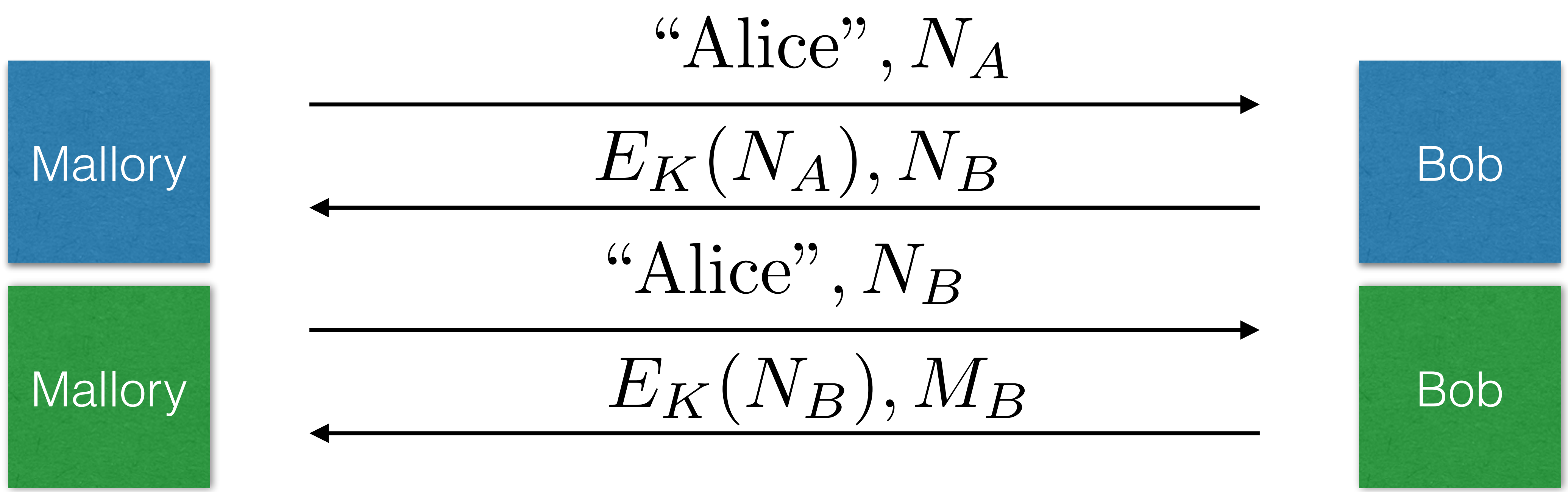
Mutual Authentication

- The three-round protocol is vulnerable to a replay attack
  - Mallory pretends to be Alice.



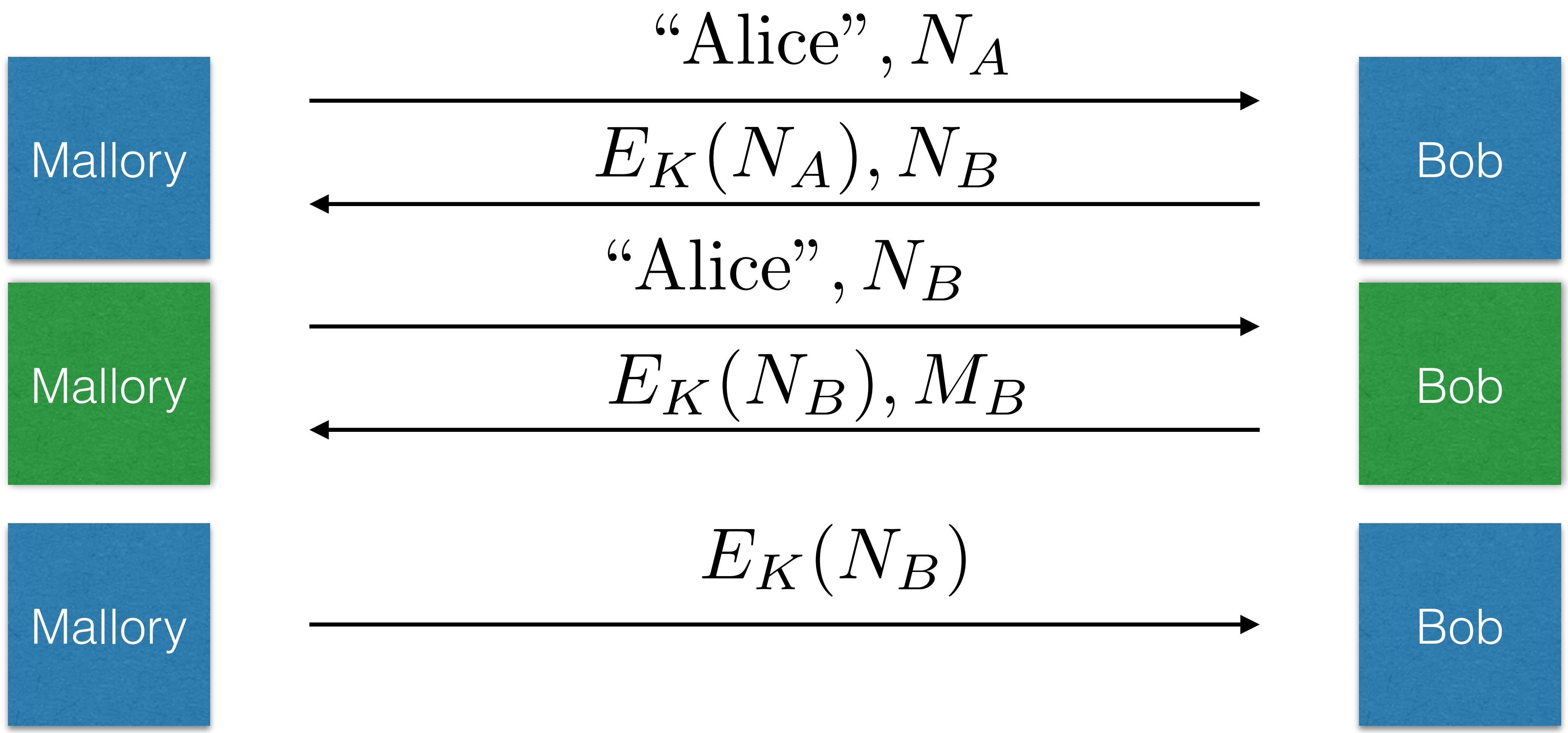
- Mallory is now stuck. (S)he opens a second connection to Bob.

Mutual Authentication



- Mallory reflects Bob’s challenge back to Bob
- Bob solves the challenge and poses a new one

Mutual Authentication



- Mallory then returns to the first session
- Reflects Bob's own answer to her challenge in the second session

Mutual Authentication

- Warning signs for the possibility of a reflection attack
    1. Requestor authenticates last
    2. Both requestor and authenticator use the same protocol for dealing with challenges

Mutual Authentication

- Can break the scheme by:
  - Have Alice authenticate first
    - (the previous scheme)
  - Destroy symmetry
    - E.g.: Alice’s nonces have to be even and Bob’s have to be odd

Mutual Authentication

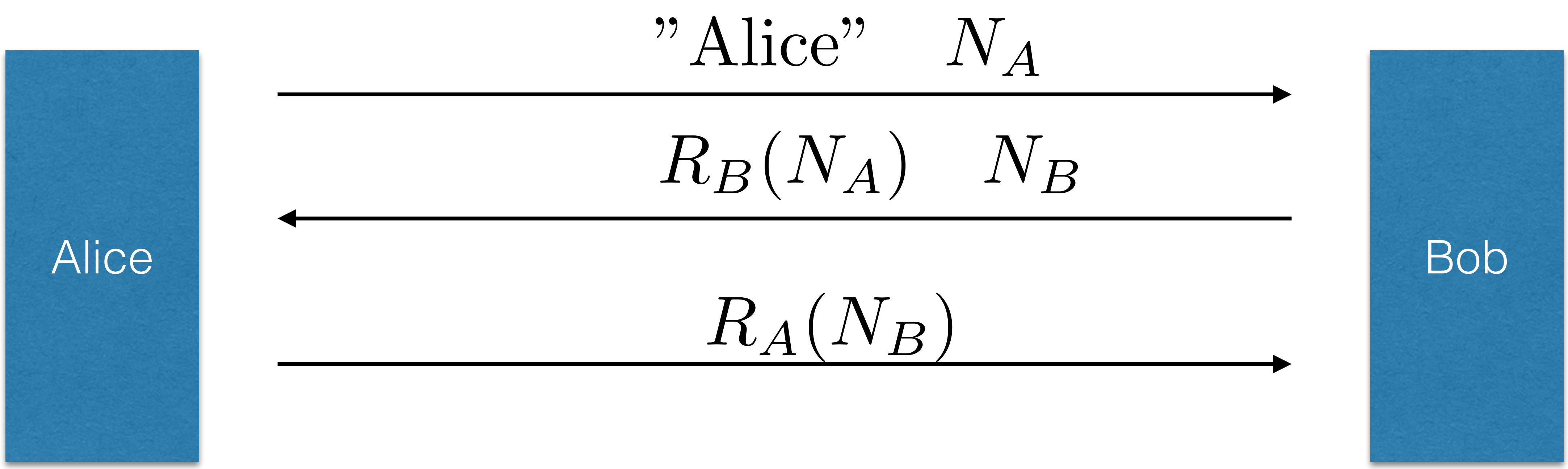
- Other attack: spoofing for offline password attack
  - Mallory spoofs Alice
  - Obtains $E_K(N_A), N_B$
    - First part controlled by her
    - Can now try to brute-force key

Mutual Authentication

- Mutual authentication based on asymmetric cryptography is not symmetric between requestor and authenticator

Mutual Authentication

- Assume that Alice has public key U_A and private key R_A



Alice checks:

$$U_B(R_B(N_A)) == N_A$$

Bob checks:

$$U_A(R_A(N_B)) == N_B$$

Mutual Authentication

- Vulnerabilities:
    - Possibility of Denial of Service Attack — yes
    - Spoofing / Sniffing:
        - No point, can directly brute-force the public keys
    - Replay:
        - Only if nonces get reused
    - Reflection attack:
        - No: protocol is not symmetric
    - Man-in-the-middle:
        - Vulnerable: MitM just passes on the messages

Mutual Authentication

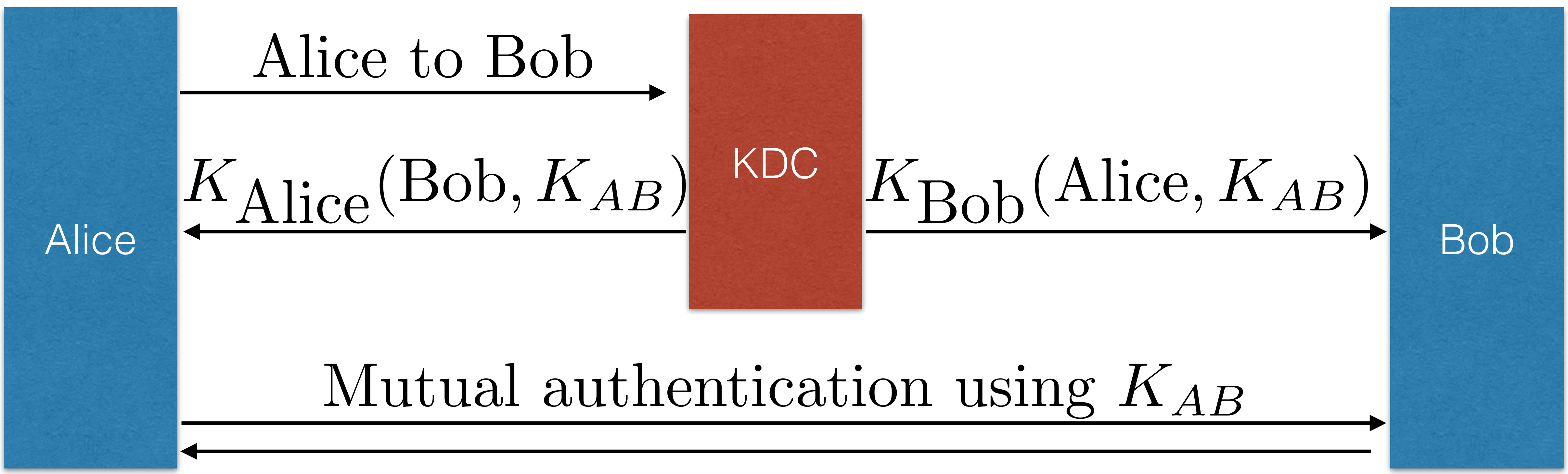
- Reuse of key-pairs
  - To sign, we encrypt a hash with the private key
  - Spoofing:
    - Pretend-Alice sends the hash of a message as nonce
    - Bob signs it
  - Prevention:
    - ALWAYS let requestor authenticate first
    - NEVER use private-public key-pairs for more than one purpose

Key Distribution Centers

- Users / services have an account with a KDC
  - Based on a shared secret
- In case of need to access someone else, KDC provides credential

Key Distribution Centers

- Attempt 1:



KDC invents a common session key
Distributes it to both

Key Distribution Centers

- Attempt 1:
  - KDC invents a common session key
  - Sends it to both Alice and Bob
  - Problem at Bob's side:
    - The sending by KDC and initialization of mutual authentication are not synchronized
    - Forces Bob to remember previous messages

Key Distribution Centers

- Needham Schroeder Protocol
  - Alice asks KDC for a session key
  - KDC sends Alice the session key (only visible to her) and a ticket
    - Ticket contains the KDC message to Bob
  - Alice presents both her authentication request and the ticket to Bob at the same time
  - Bob no longer has to remember credentials

Key Distribution Centers

- Needham Schroeder
  1. Alice sends request to KDC, stating her ID and the service she wants to contact

$$N_1, \text{Alice}, \text{Bob}$$

  - Nonce in order to label sessions

Key Distribution Centers

- Needham Schroeder
  2. KDC sends Alice a message encrypted with the key shared by her and the KDC
     - The nonce as a session identifier
     - The service name
     - A ticket for the service
  - The ticket can only be read by Bob

$$K_A(N_1, \text{Bob}, K_{AB}, K_B(K_{AB}, \text{Alice}))$$

Key Distribution Centers

- Since only Bob can read the ticket

$$K_B(K_{AB}, \mathrm{Alice})$$

  - No need to encrypt it with Alice's key

Key Distribution Centers

- Needham Schroeder
  - The KDC's job is now done
  - Alice and Bob mutually authenticate
    - Alice to Bob:

$$K_B(K_{AB}, \text{Alice}), K_{AB}(N_2)$$

    - with an additional nonce

Key Distribution Centers

- Needham Schroeder
  - Bob unpacks the ticket and then obtains the nonce
  - Proves that he could unpack the ticket and is therefore Bob by performing an arithmetic operation on Alice's nonce.
  - Adds a nonce of his own.
  - Sends to Alice

$$K_{AB}(N_2 - 1, N_3)$$

Key Distribution Centers

- Needham Schroeder
  - Alice responds by deciphering (proving that she can read the information send by the KDC to her),
  - performing an arithmetic operation on the result
  - and sending it back to Bob

$$K_{AB}(N_3 - 1)$$

Key Distribution Centers

- Alice to KDC: $N_1, \text{Alice}, \text{Bob}$
- KDC to Alice: $K_A(N_1, \text{Bob}, K_{AB}, K_B(K_{AB}, \text{Alice}))$
- Alice to Bob: $K_B(K_{AB}, \text{Alice}), K_{AB}(N_2)$
- Bob to Alice: $K_{AB}(N_2 - 1, N_3)$
- Alice to Bob: $K_{AB}(N_3 - 1)$

Key Distribution Centers

- What happens if the message from Bob to Alice is

$$K_{AB}(N_2 - 1), K_{AB}(N_3)$$

Key Distribution Centers

- There is a small vulnerability in Needham Schroeder
  - Trudy manages to determine the common key K_{AB} long after it has been used
  - Trudy has sniffed messages (3)-(5)
  - Trudy sends $K_B(K_{AB}, \text{Alice}), K_{AB}(N_2)$
  - Bob responds with $K_{AB}(N_2 - 1, N_3)$
    - where the last nonce is new
  - But Trudy can respond with $K_{AB}(N_3 - 1)$

Key Distribution Centers

- Solution 1: Timestamps
  - Add a time stamp T to the protocol

$$\text{Alice to KDS: Alice, Bob}, N_1$$

$$\text{KDS to Alice: } K_A(N_1, \text{Bob}, K_{AB}, T, K_B(\text{Alice}, K_{AB}, T))$$

$$\text{Alice to Bob: } K_B(\text{Alice}, K_{AB}, T), K_{AB}(N_2))$$

$$\text{Bob to Alice: } K_{AB}(N_2 - 1, N_3)$$

$$\text{Alice to Bob: } K_{AB}(N_3 - 1)$$

Key Distribution Centers

- Solution 2: Strong Needham Schroeder
  - Alice goes to Bob who hands her a nonce that only he can verify
  - Alice asks the KDC to put this verifier into the ticket for Bob

Key Distribution Centers

- Strong Needham Schroeder

$$
\begin{aligned}
\text{Alice to Bob: } & \text{I want to talk to you} \\
\text{Bob to Alice: } & K_B(N_B) \\
\text{Alice to KDS: } & \text{Alice, Bob}, N_1, K_B(N_B) \\
\text{KDS to Alice: } & K_A(N_1, \text{Bob}, K_{AB}, K_B(\text{Alice}, K_{AB}, N_B)) \\
\text{Alice to Bob: } & K_B(\text{Alice}, K_{AB}, N_B), K_{AB}(N_2)) \\
\text{Bob to Alice: } & K_{AB}(N_2 - 1, N_3) \\
\text{Alice to Bob: } & K_{AB}(N_3 - 1)
\end{aligned}
$$

Key Management

Key Management

Key Management