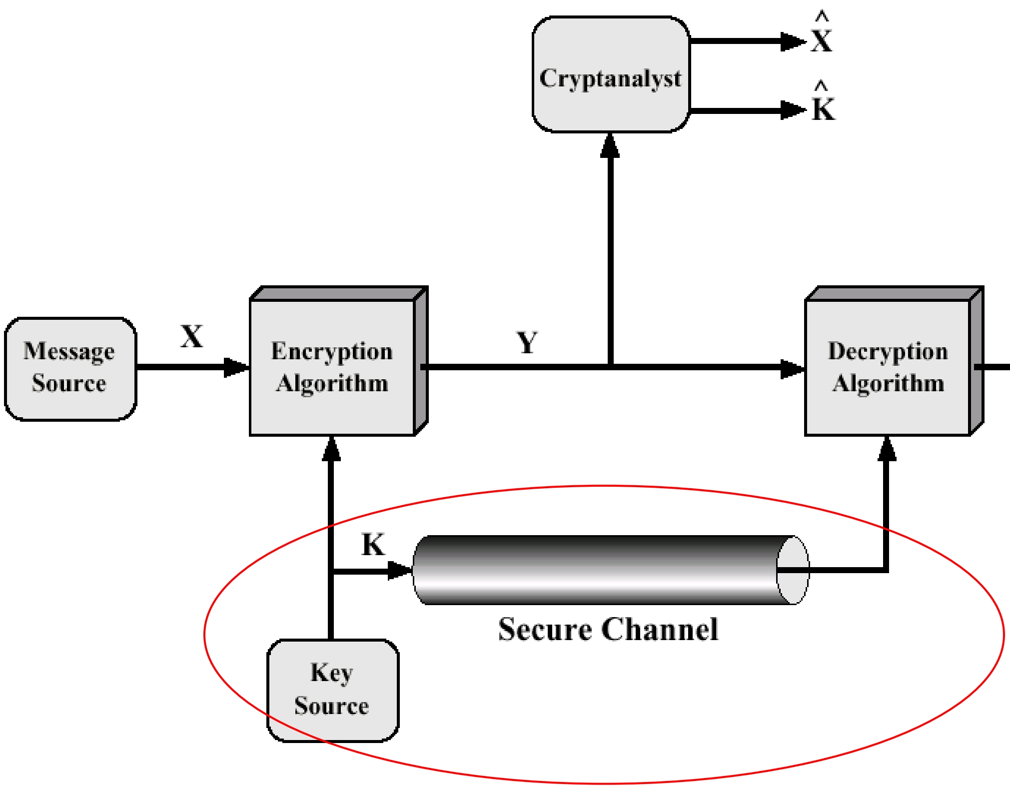
CT437
COMPUTER SECURITY AND FORENSIC COMPUTING

MANAGEMENT / DISTRIBUTION OF SYMMETRIC AND PUBLIC KEYS

Dr. Michael Schukat

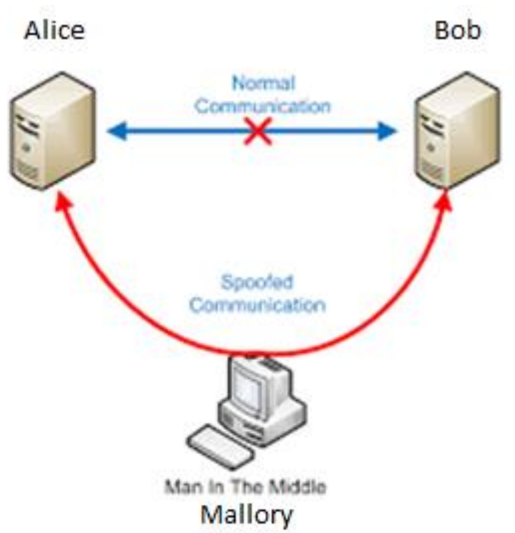
Ollscoil na Gaillimhe
University of Galway

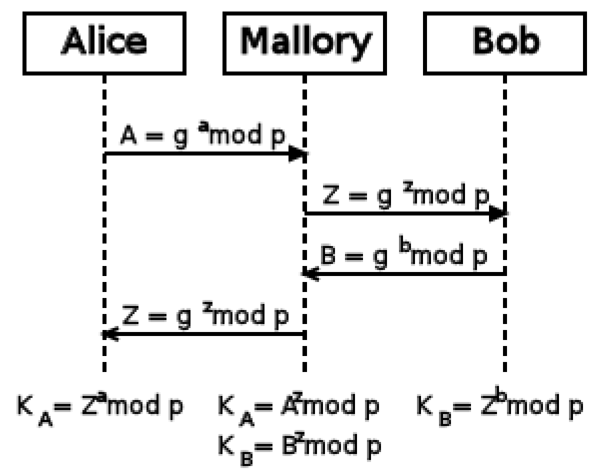
Recap: Model of a Conventional Cryptosystem



$$Y = E_K(X),\ X = E_K^{-1}(Y)$$

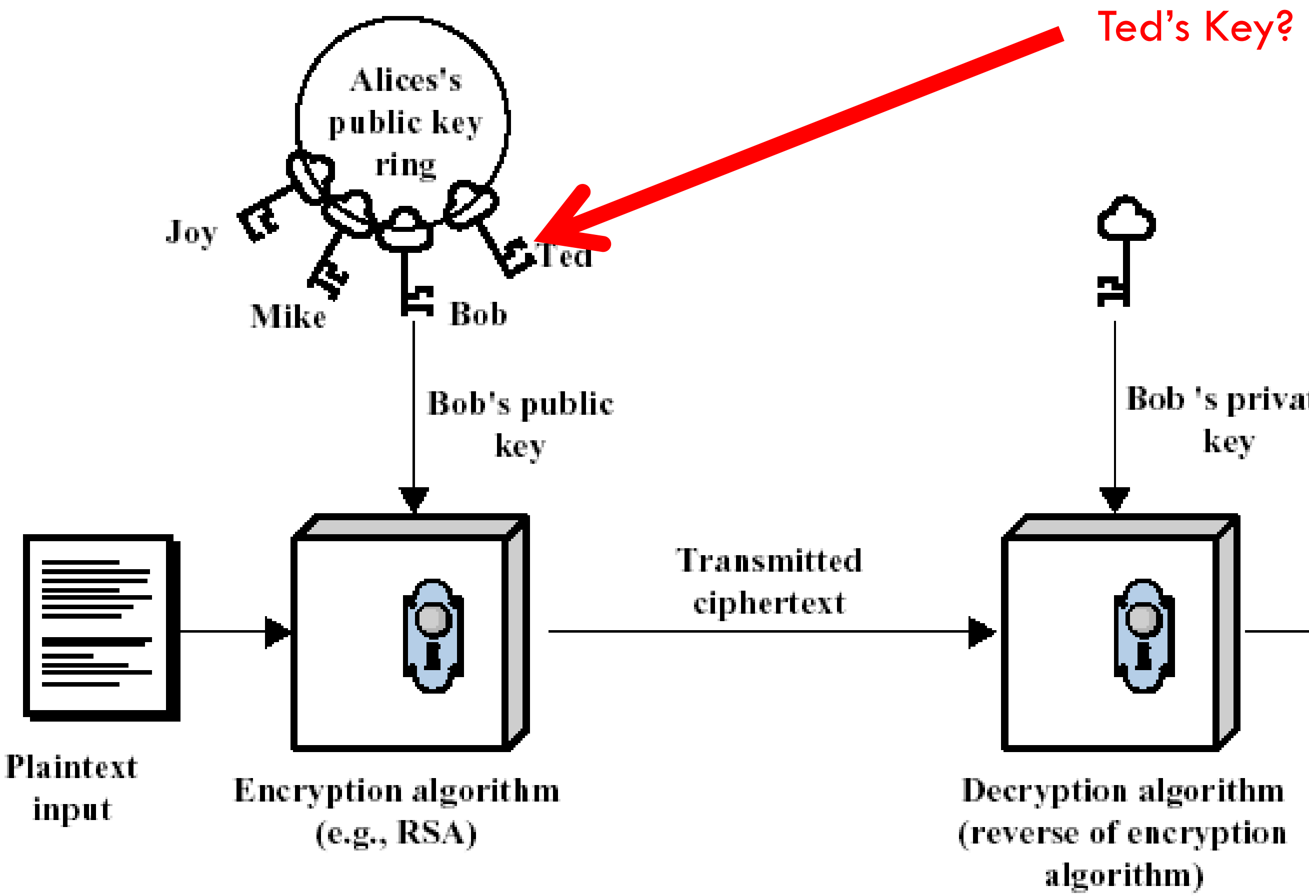
Recap: DH and Man-in-the-Middle (MitM) Attacks





- Mallory is a MitM attacker and performs message interception and message fabrication
- Mallory establishes two individual (secure) connections with Alice and Bob
- Both Alice and Bob are unaware of Mallory's existence (as there is no authentication)
- **DH alone is not sufficient for secure key distribution!**

Recap: Public-Key Encryption



Key Issues that need to be addressed

1. Symmetric key encryption
   - Key distribution mechanism?
   - Key management (key renewal / generation)?
   - Key authentication?
2. Public key encryption
   1. Public key distribution / management?
   2. Public key authentication, i.e. validation of owner?

Terminology

- **Key rotation** is the general term for creating a new key and starting to encrypt data with it, while retiring the old key, hence the rotation
  - Time-Based Key Rotation
    - E.g., every week
  - Usage-Based Key Rotation
    - E.g., after using it to process x Gigabyte of data
  - Incident-Triggered Key Rotation
    - Change key if it was compromised
- **Re-keying** involves changing cryptographic keys in an on-going communication channel (e.g., TLS → later)
- **Re-encryption** refers to the process of encrypting previously encrypted data using a new key

Key Management Lifecycle

- **Generation:** Generating strong cryptographic keys using a cryptographically secure random number generator (as seen before)
- **Distribution:** Safely transmit them using encrypted channels / protocols, to authorised parties without risking unintended exposure
  - **Key wrapping** is a common approach here, i.e. encrypt the new key using the old key before circulation
  - Possibly **DH** if hardened against MitM attacks
- **Storage:** Utilize key management systems (KMS) to encrypt, store and manage cryptographic keys to protect them from theft or unauthorised access
  - See next slide
- **Usage:** Utilise keys for encryption / decryption / authentication
- **Rotation:** Replace cryptographic keys regularly or according to a policy to limit their exposure and minimize any data exposure impact from potential key compromise
- **Destruction:** Safely delete keys once they are no longer needed to prevent their recovery or misuse

Types of KMS and cryptographic Key Stores

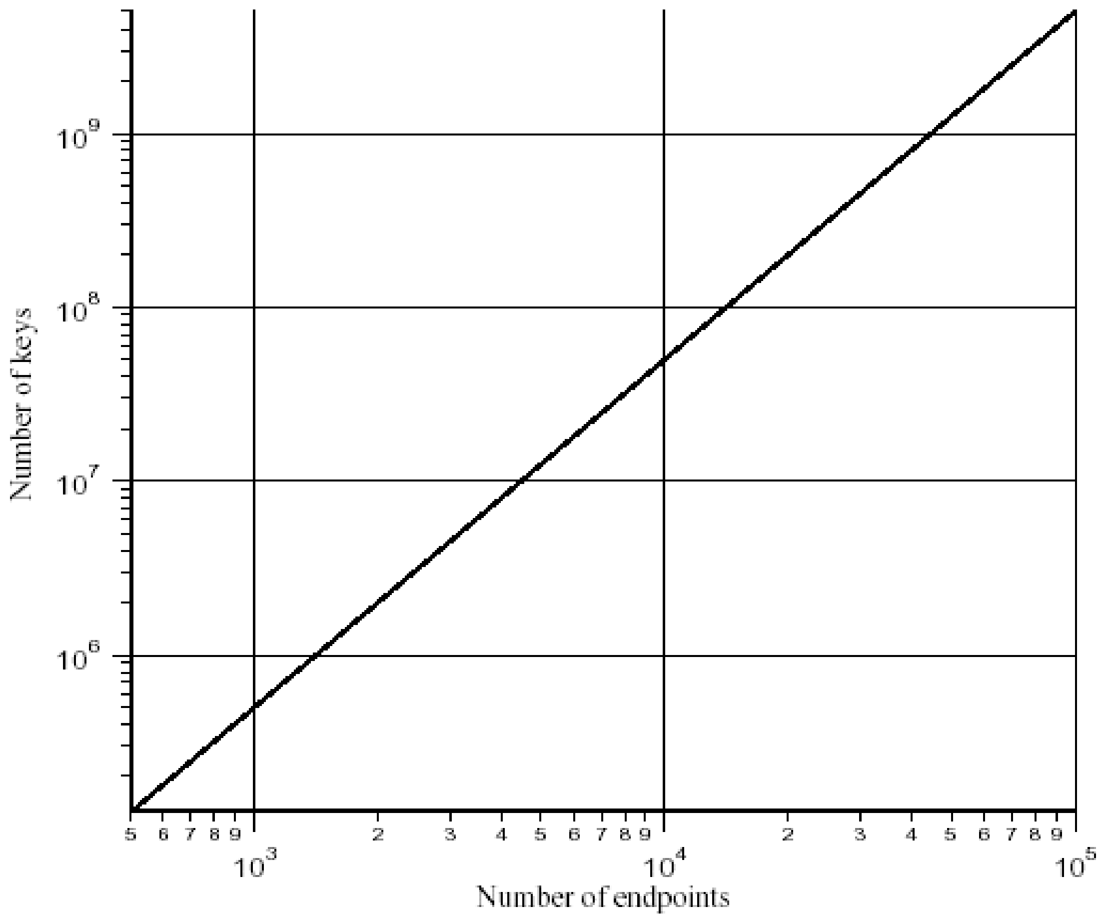
- Hardware Security Modules (HSMs):
  - These are physical devices that provide secure key storage and cryptographic operations, e.g. USB HSM
- Cloud-based Key Management Services:
  - E.g., AWS Key Management Service, Azure Key Vault, and Google Cloud KMS
- Software-based Key Stores:
  - E.g., OpenSSL, Java KeyStore (JKS), and Microsoft's Cryptographic API
  - They are used for storing keys in software applications
- Hardware-based Key Stores:
  - Devices like TPM (Trusted Platform Module) and smart cards can securely store cryptographic keys and perform cryptographic operations

Symmetric Key Cryptosystems

Key Distribution Case Study

- Problem
  - Two parties PA and PB want to securely communicate over a public network using symmetric key encryption
  - How can the key distribution be achieved?
- Simple solutions
  1. A key is selected by PA and physically delivered to PB
  2. Some independent authority PC selects a key and physically delivers it to PA and PB
- Drawbacks of both solutions:
  - Manual delivery of keys → this is tedious and is cumbersome
  - The solution does not scale, as for N parties (e.g. endpoints in a computer network)
    N * **(N – 1)** / **2** unique keys are required

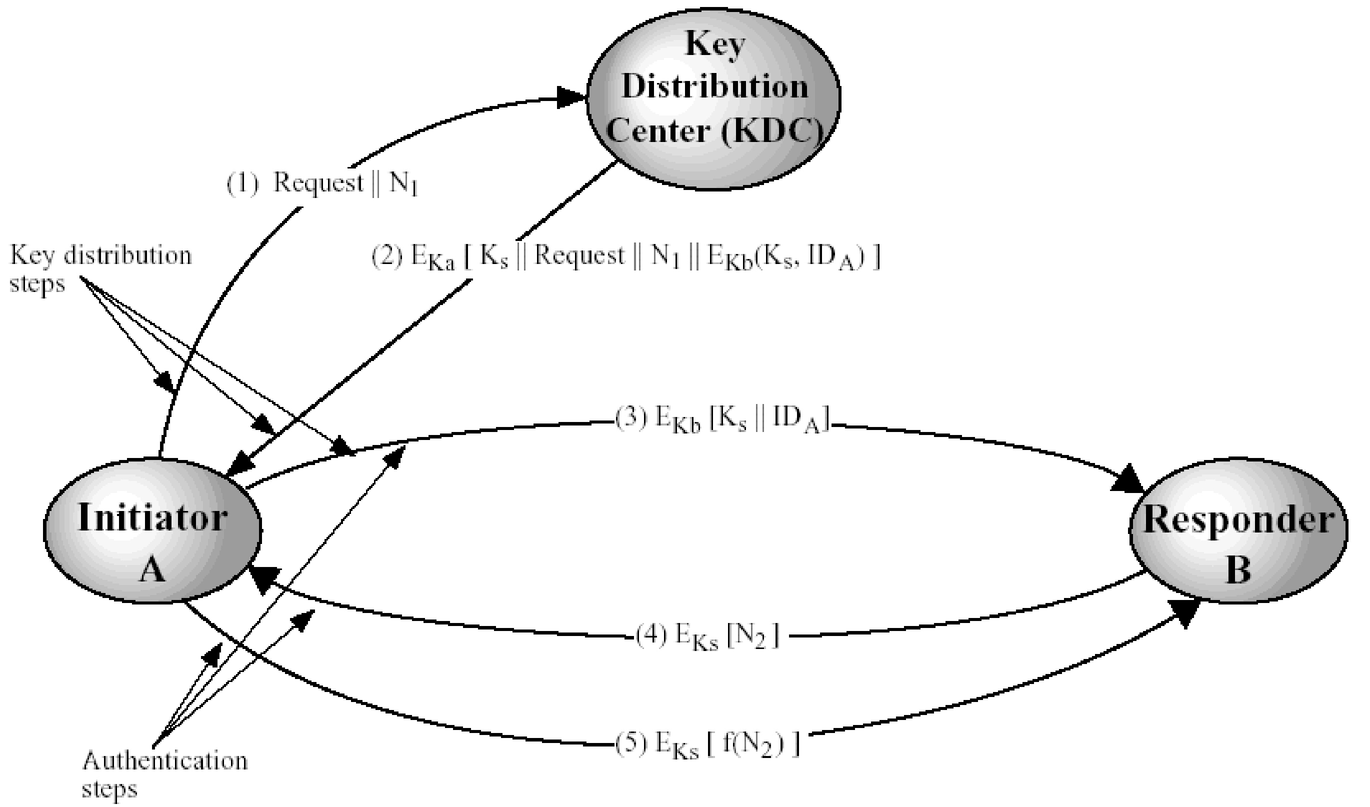
Number of (unique) Keys versus Number of Endpoints



Key Distribution using a KDC

- Solution 3 overview
  - PA and PB can rely on a secure (encrypted) connection to a **key distribution centre (KDC)**
  - The KDC delivers a key via the encrypted links to A and B on demand
- Details:
  - Each endpoint and the KDC already share a unique **master key**
  - This key is used to securely exchange messages between both (E_{Kx} in the next slide)
  - For N hosts, N master keys are required
  - Two hosts communicate securely with each other, by using a secure session key K_S, which is provided by the KDC

KDCs and the Needham–Schroeder Protocol



1. A → KDC: $ID_A \| ID_B \| N_1$
2. KDC → A: $E(K_a, [K_s \| ID_B \| N_1 \| E(K_b, [K_s \| ID_A])])$
3. A → B: $E(K_b, [K_s \| ID_A])$
4. B → A: $E(K_s, N_2)$
5. A → B: $E(K_s, f(N_2))$

The Needham–Schroeder Protocol explained

- Initiator A (IA) and responder B (RB) share a unique master key each with the KDC (K_{KA} and K_{KB})

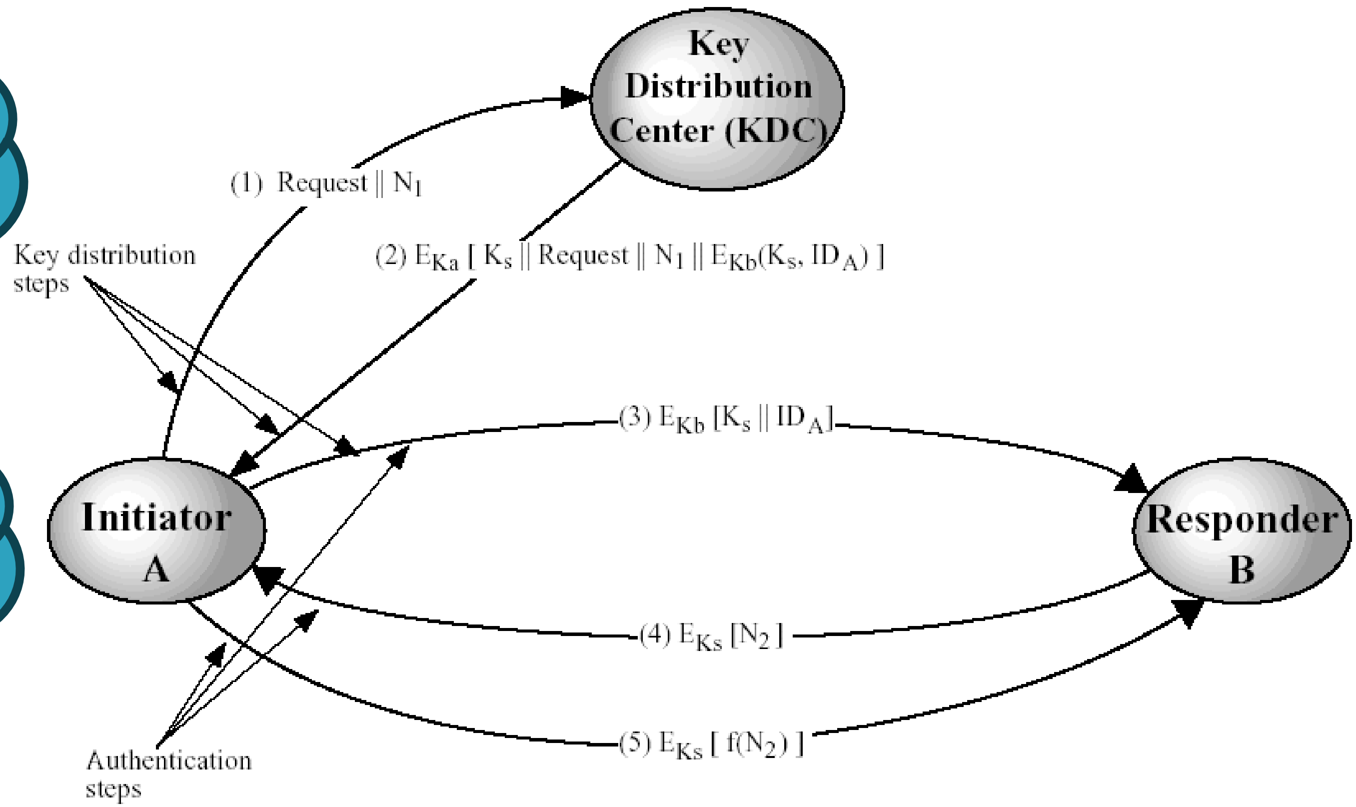
1. IA issues a message to the KDC for a session key to be shared with RB; it includes:
   - The Request, containing the identity of IA and RB (e.g., their network addresses)
   - A unique nonce N_1 for this transaction
2. The KDC responds with a message encrypted using K_{KA} that contains:
   - The session key K_S
   - The original Request and N_1
   - A message for the responder RB that is encrypted using K_{KB} and that includes:
     - The session key K_S
     - The identity of A, ID_A (e.g., its network address)
3. IA decrypts / validates the response and sends only the above message to RB
4. RB:
   - Decodes the message using K_{KB} and validates IA to be the message sender
   - Sends a new nonce N_2 to IA, encrypted using KS
5. IA:
   - Decrypts the message using its copy of K_S
   - Processes the nonce in an agreed fashion (e.g., $N_2 = N_2 + 1$)
   - Encrypts N_2 and sends it to RB

- RB validates the content of the message and by doing so authenticates IA

KDCs and the Needham–Schroeder Protocol

How can the KDC be compromised?

How can the protocol be compromised?



1.	A → KDC:	$ID_A \| ID_B \| N_1$
2.	KDC → A:	$E(K_a, [K_s \| ID_B \| N_1 \| E(K_b, [K_s \| ID_A])])$
3.	A → B:	$E(K_b, [K_s \| ID_A])$
4.	B → A:	$E(K_s, N_2)$
5.	A → B:	$E(K_s, f(N_2))$

Possible Attacks on the Needham–Schroeder Protocol

- Assume an attacker is positioned between IA and KDC
- The MitM intercepts (1), identifies IA and RB, and intercepts (2)
- The protocol is completed as before, and K_S is used by IA and RB
- At some stage in the future K_{KA} is compromised
- **The MitM can now**
  - **decode (2)**
  - **impersonate IA (by using ID_A)**
  - **resend X = EKB(K_S, ID_A) to RB (3),**
  - **complete the protocol**
- **RB believes it is talking to IA**
- Solution:
  - X must be complemented with a timestamp (when K_S was created) and / or K_S validity period, so RB can validate that KS is not stale (and must not be used any more)
  - all entities must be time-synchronised (→ NTP / PTP)

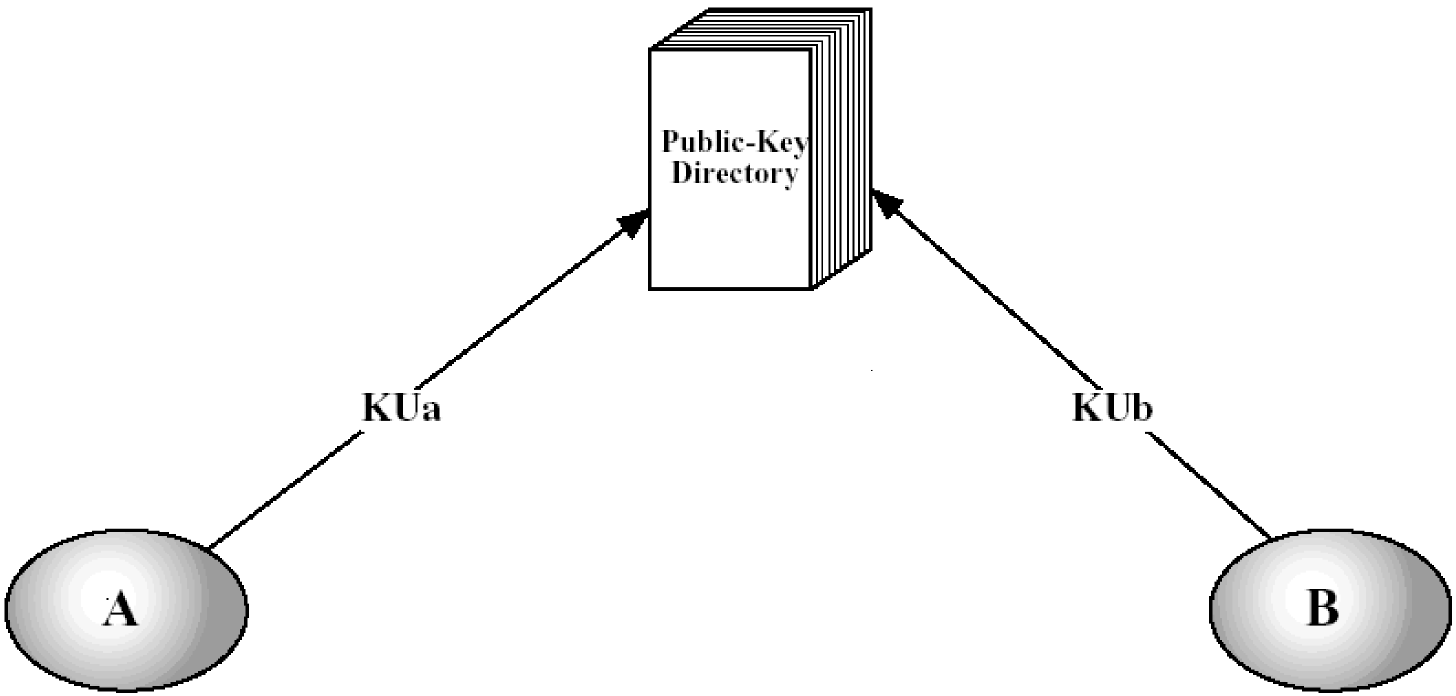
Public Key Cryptosystems

Key Management via uncontrolled Public-Key Distribution

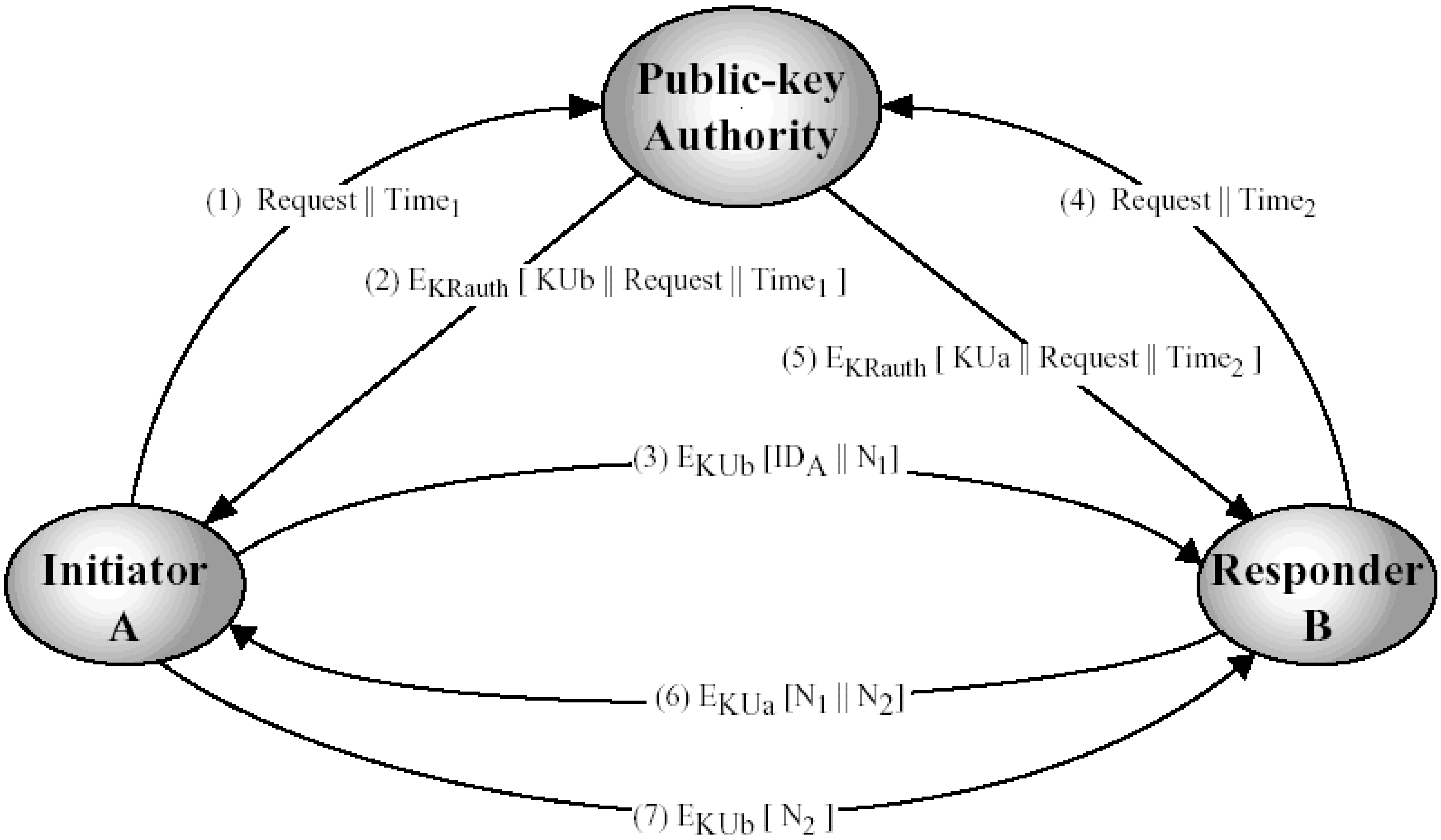
- Simplistic approach, but easy to forge, e.g., anyone could pretend to be user A

Key Management via Public-Key Directory

- The directory is just a public platform where everybody can upload their public key
- Similar issues as before



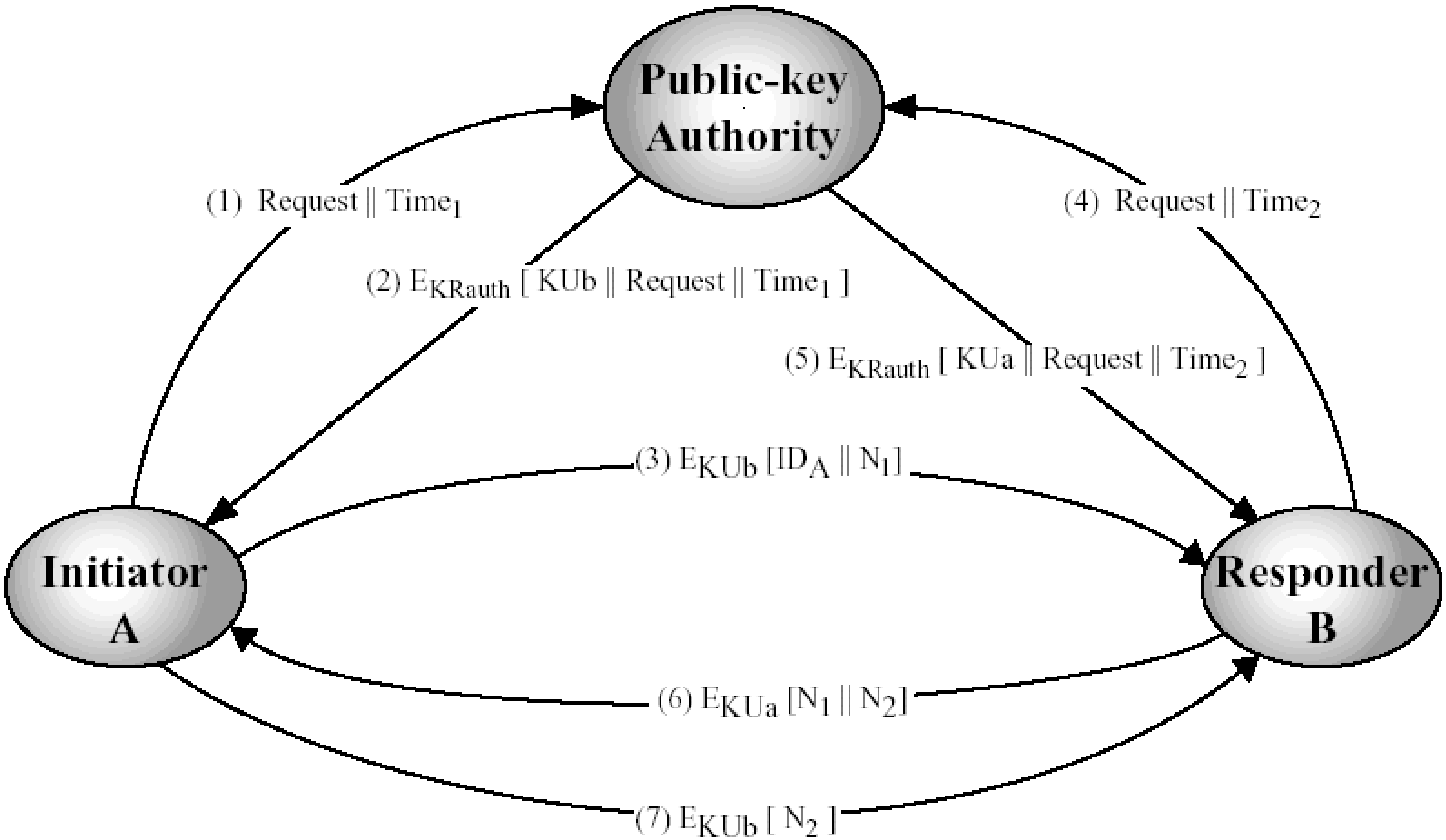
Key Management using a Public-Key Authority



Key Management using a Public-Key Authority

- Based on the Needham–Schroeder Protocol, but with some improvements
- The public-key authority (PKA) has a public / private key pair with:
  - Private key K_{RAuth}
  - Public key K_{UAuth} being shared with all clients
- Initiator A (IA) and responder B (RB) have a public / private key pair each
  - K_{UA} and K_{RA}
  - K_{UB} and K_{RB}
- K_{UA} and K_{UB} are managed by the PKA
- IA requests for K_{UB} in order to setup a secure connection with RB

Key Management using a Public-Key Authority



The Protocol explained

1. IA issues a message to the PKA to get K_{UB}; it includes:
   - The Request, containing the identity of IA and RB (e.g., their network addresses)
   - The timestamp $Time_1$ of this transaction
2. The PKA responds with a message authenticated using K_{Rauth} that contains
   - RB's public key K_{UB}
   - The original Request and $Time_1$
3. IA:
   - Validates the authenticity of the response by decoding the message using K_{rauth} and validating Request and $Time_1$; IA extracts K_{UB}
   - Use this key to encrypt a message containing its (network) id ID_A and a nonce N_1
   - IA sends the message to RB

The Protocol explained

- RB:
  - Decodes the message using K_{RB}
  - Validates the message sender's id to be ID_A
  - Extract N_1
  - Requests IAs public key in steps (4) and (5)

6. RB sends a new nonce N_2 together with N_1 to IA, encrypted using K_{UA}
7. IA:
   - Decrypts the message using K_{RA}
   - Validates the message origin (RB) by checking N_1
   - Encrypts N_2 using K_{UB} and sends it to RB

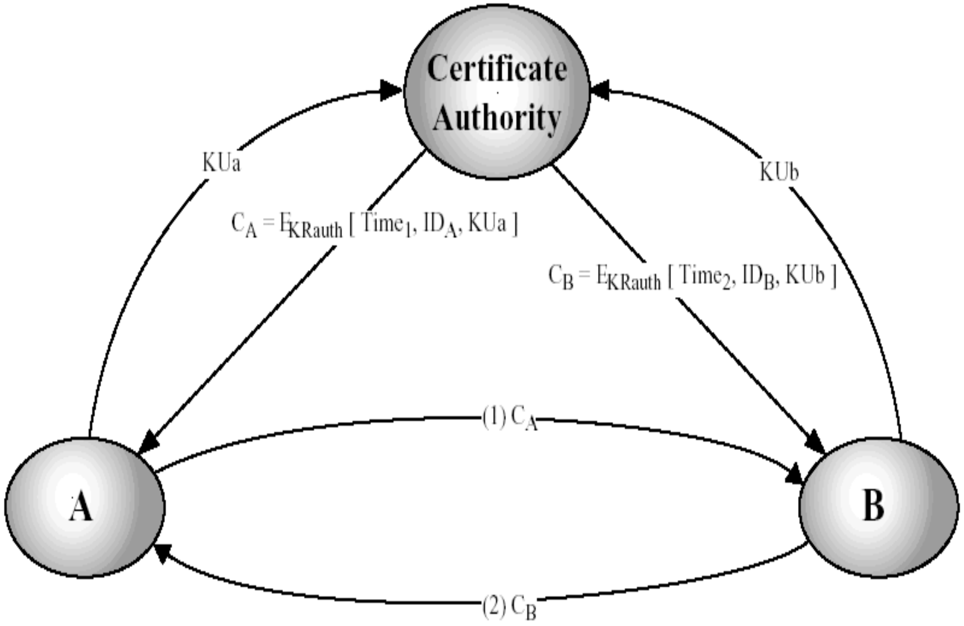
- RB:
  - Decrypts the message using K_{RB}
  - Validates the message authenticity by checking N_2

Key Management via Public-Key Authority

- Main problem:
  - The public-key authority is a single point of failure! If it is compromised (e.g., via a DoS attack), keys cannot be distributed
- Therefore:
  - Introduction of digital certificates, that can be used by nodes to exchange public keys without contacting a public-key authority
- Requirements:
  - Any participant can read a certificate to determine the name and public key of the certificate's owner
  - Any participant can verify that the certificate originated from the certificate authority and is not counterfeit
  - Only the certificate authority can create and update certificates
  - Any participant can verify the currency of the certificate

Key Management via Certificate Authority (CA)

- The CA is the root of trust
- Participants (A and B in the diagram) acquire a digital certificate each **that binds their public key KU_X to their identity ID_X**
- These certificates are subsequently exchanged to
  - setup a secure connection
  - authenticate both endpoints



Key Management via a Certificate Authority: Aquiring a Certificate

- The CA receives a request from A (or B) to certify their public key
- The CA creates a document that contains A's (or B's) identity ID_X, public key KU_X and the document's validity period $Time_1$
- The CA signs, i.e. encrypts, this document using its private key K_{Rauth}, and returns it to A (or B)
- Every entity that possesses CA's public key can validate the authenticity of a (signed) document by decoding it

Key Management via a Certificate Authority

- A and B have acquired their certificates from the CA at some stage in the past, and have a copy of CA's public key
- Now A wants to securely communicate with B, resulting in the following steps:
  - A sends C_A to B, and B in return sends C_B to A
  - Both mutually validate
    - the other party's certificate by decoding it using the CA's public key
    - the certificate's sender by comparing ID_X in the received certificate with the network address of the sender
- However, certificates are public documents and either side's network address could have been spoofed by an attacker, that impersonates A or B
- Therefore, additional steps as shown shortly are required

Example for a simple unsigned XML-based Certificate

```xml
<SimpleCertificate>
    <Authority> NUI-Galway </Authority>
    <SignatureType> SimpleSignature </SignatureType>
    <Created> 15-NOV-2019 </Created>
    <Expires> 14-NOV-2024</Expires>
    <OwnerName> William Simpson </OwnerName>
    <KeyType> RSA </KeyType>
    <KeyLength> 256 </KeyLength>
        <PublicKey>
        gHJgjh57JKf#j'\;gkwg@45tRET46$Ed
    </PublicKey>
</SimpleCertificate>
```

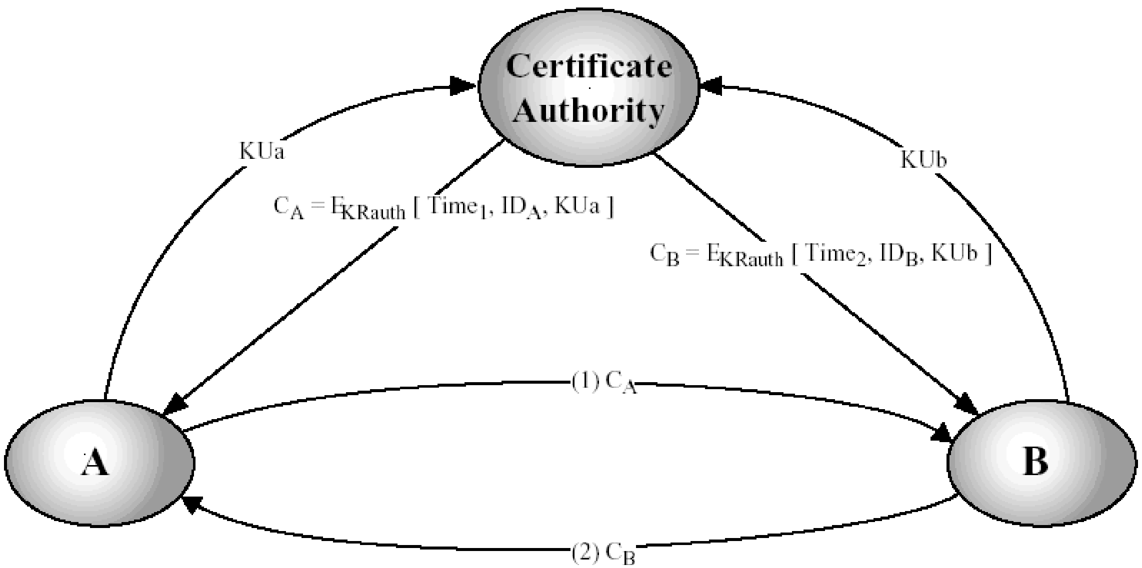
Example for a signed simple XML-based Certificate

hi6IGHJ^gu#":HGLFdyUf56EEdx3X5XxXuAzyl;*6/.,:g
wqui^09udfsqfhaspfaj#w994HK51'fjg095u321\er3f2875
gyor23ro32rj6yhgglGUoowqru07t99Y)*-36wrqwUluiill
No891 u[`[c0 t6Rt*(v858e3w70-v794x3xz7c8c9799999s
9udfsqfhaspfaj7t99 -v794x3xz7c8c9799 09udfsqfhaspfaj#
w994HK51'fjg095u32nfjewYU87Deffe7s%Rk936-J0D9d

- The signed certificate is just undecipherable text
- Its validation requires the decoding of the entire document
- Later ➔ X.509 digital certificates provide a much neater solution

In-Class Activity

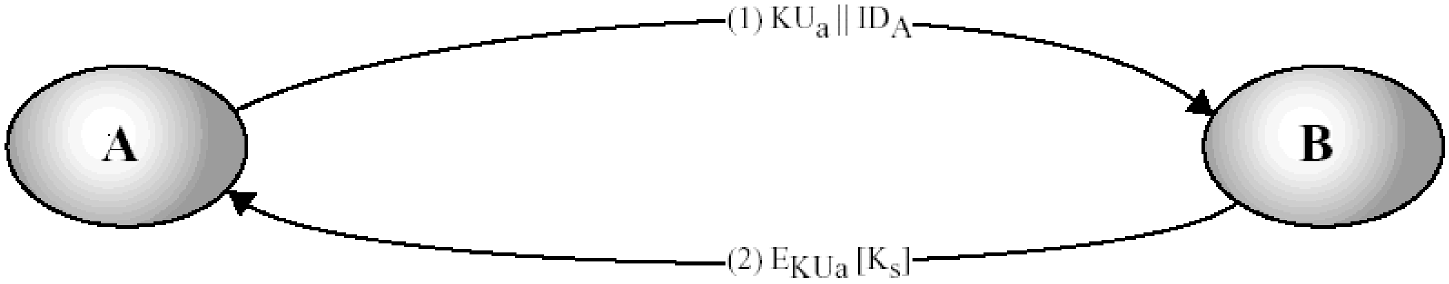
- Can you identify any "weak spots" in the CA system below?



Symmetric Key Distribution with Public Key Cryptosystems

Symmetric-Key Distribution Using a Public Key Encryption

- Public-key encryption is slow
- Therefore, it is often used for the distribution of a secret (session) key to be used for conventional symmetric encryption
- This is an example for a simple secret-key distribution, where A shares its public key KU_a with B:



- Problem: B cannot authenticate A or their public key (and vice versa), therefore
  - A or B could be impersonated via network address spoofing
  - A MitM attacker could place itself between A and B

Secret-Key Distribution with Confidentiality and Authentication

- In this protocol both sides have already acquired and validated the other side's certificate (that contains the owner's identity ID_X) and public key
- The 4-step authentication process guarantees that
  - mutual authentication is provided (no network address spoofing possible)
  - a MitM attacker cannot place itself between A and B
- It is the logical continuation of the protocol "Key Management via Certificate Authority (CA)"