CS306: Introduction to IT Security

Fall 2018

Lecture 6: Public-key cryptography

Instructor: **Nikos Triandopoulos**

October 2, 2018

STEVENS INSTITUTE *of* TECHNOLOGY
THE INNOVATION UNIVERSITY®

Last week

- Message authentication: Special topics
  - replay attacks
  - authenticated encryption
- Hash functions
  - design framework
  - generic attacks
  - applications: cryptography & security

Today

- Public-key cryptography
  - motivation
  - public-key encryption
    - hybrid encryption
    - ElGamal encryption scheme
  - digital signatures
  - key agreement
  - public-key certificates

6.0 Announcements

CS306: Announcements

- **HW 1**
  - discussion on solutions
- **HW 2**
  - going out tomorrow, due in a week
  - message authentication, hash functions & PK crypto as applied to cloud security
  - solutions to be discussed in help sessions next week

CS306: Announcements

- **upcoming labs**
  - **Lab 5** (October 4th)
    - covers public-key cryptography
    - ~~**2nd practical assignment**~~ regular practice quiz
  - **next week**: special optional **help-session** lab (October 11th)
    - run by TAs (impossible to find any time outside lab sections)
    - revision over materials relevant to midterm (October 16th)
    - discuss HW 1, HW 2
    - any questions
  - **no lab** the week of the midterm (October 18th)

CS306: Tentative Syllabus

Week	Date	Topics	Reading	Assignment
1	Aug 28	Introduction	Ch. 1	-
2	Sep 4	Symmetric encryption	Ch. 2 & 12	Lab 1
3	Sep 11	Symmetric encryption II	Ch. 2 & 12	Lab 2, HW 1
4	Sep 18	Message authentication	Ch. 2 & 12	Lab 3, HW 1
5	Sep 25	Hash functions	Ch. 2 & 12	Lab 4
6	**Oct 2**	**Public-key cryptography**	**Ch. 2 & 12**	**HW 2**
–	Oct 9	**No class** (Monday schedule)		**Help session**
7	Oct 16	**Midterm** (closed books)	All materials covered	**No labs**

CS306: Tentative Syllabus (continued)

Week	Date	Topics	Reading	Assignment
8	Oct 23	Access control & authentication		
9	Oct 30	Software & Web security		
10	Nov 6	Network security		
11	Nov 13	Database & cloud security		
12	Nov 20	Privacy		
13	Nov 27	Economics		
14	Dec 4	Legal & ethical issues		
15	Dec 11 (or later)	**Final** (closed books)	All materials covered*	

* w/ focus on what covered after midterm

CS306: Course outcomes

- **Terms**
  - describe common security terms and concepts
- **Cryptography**
  - state basics/fundamentals about secret and public key cryptography concepts
- **Attack & Defense**
  - acquire basic understanding for attack techniques and defense mechanisms
- Impact
  - acquire an understanding for the broader impact of security and its integral connection to other fields in computer science (such as software engineering, databases, operating systems) as well as other disciplines including STEM, economics, and law
- Ethics
  - acquire an understanding for ethical issues in cyber-security

Questions?

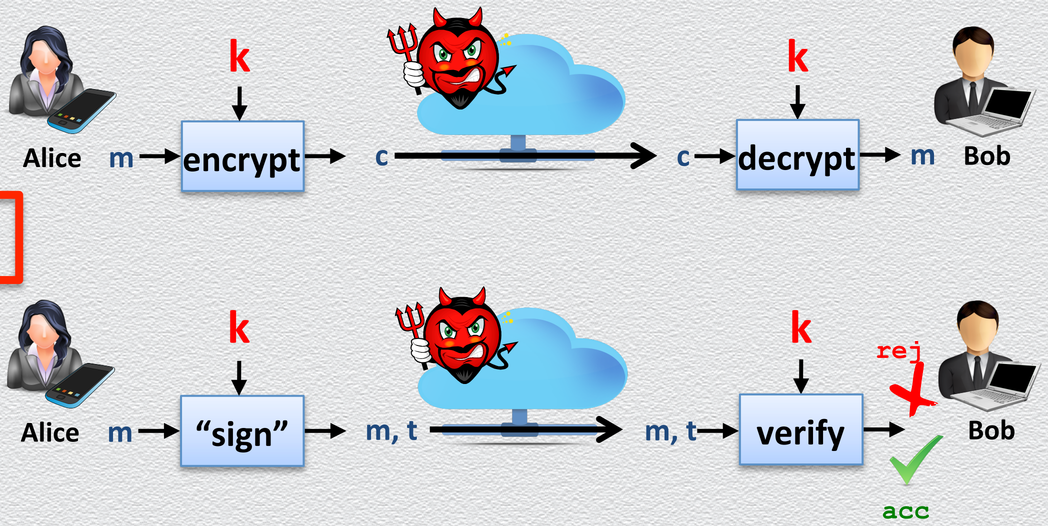
6.1 Public-key cryptography: Motivation

Recall: Principles of modern cryptography

(A) security definitions, **(B) precise assumptions**, (C) formal proofs

For **symmetric-key** message encryption/authentication

- adversary
  - types of attacks
- trusted set-up
  - secret key is distributed securely
  - secret key remains secret
- trust basis
  - underlying primitives are secure
  - **PRG, PRF, CR hashing, ...**
    - **e.g., block ciphers, SHA-2, etc.**



On “secret key is distributed securely”

Alice & Bob (or 2 individuals) must **securely obtain** a **shared secret key**

- “securely obtain” → **(A) strong assumption** to accept
  - requires secure channel for key distribution (chicken & egg situation)
  - seems impossible for two parties having no prior trust relationship
  - not easily justifiable to hold a priori
- “shared secret key” → **(B) challenging problem** to manage
  - requires too many keys, namely $O(n^2)$ keys for n parties to communicate
  - imposes too much risk to protect all such secret keys
  - entails additional complexities in dynamic settings (e.g., user revocation)

Alternative approaches?

Need to securely distribute, protect & manage many **session-based** secret keys

- (A) for secure distribution, just “make another assumption…”
  - employ **“designated” secure channels**
    - physically protected channel (e.g., meet in a “sound-proof” room)
  - employ **“trusted” party**
    - entities authorized to distribute keys (e.g., key distribution centers (KDCs))
- (B) for secure management, just ‘live with it!”

Public-key cryptography to the rescue…

Public-key (or asymmetric) cryptography **disclaimer on names**

Goal: devise a cryptosystem where key setup is "more" manageable

Main idea: **user-specific** keys (that come in pairs)

- user U generates two keys (U_{pk}, U_{sk})
  - **U_{pk} is public** – it can safely be known by everyone (even by the adversary)
  - **U_{sk} is private** – it must remain secret (even from other users)

Usage

- employ **public** key U_{pk} for certain "**public**" tasks (performed by **other users**)
- employ **private** key U_{sk} for certain "**sensitive**" tasks (performed by **user U**)

Assumption

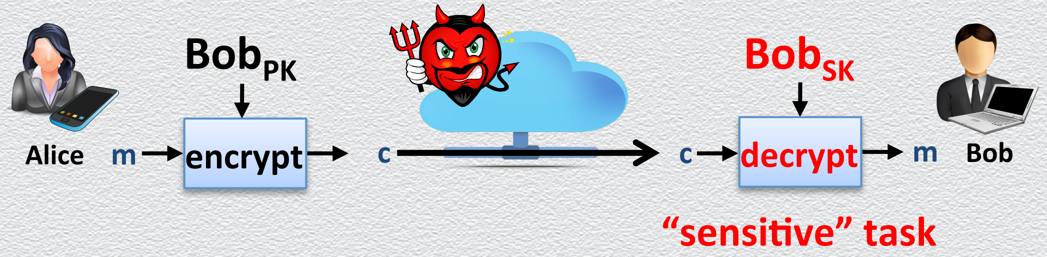
- **public-key infrastructure (PKI)**: public keys become **securely** available to users

From symmetric to asymmetric encryption

secret-key encryption

- main limitation
  - **session-specific** keys

public-key encryption

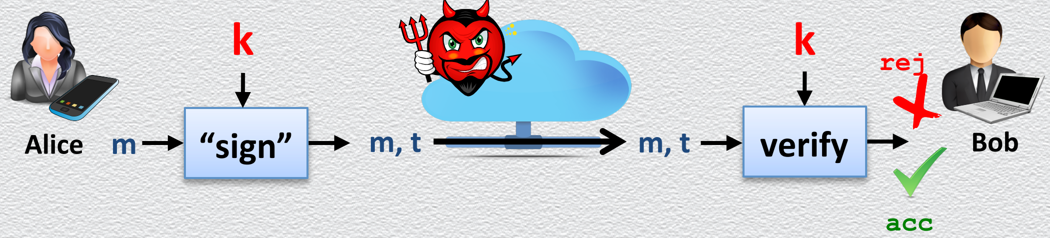


- main flexibility
  - **user-specific** keys
    - each user has two keys: **one public key PK** & **one private SK**
- messages encrypted by receiver’s PK can (only) be decrypted by receiver’s SK

From symmetric to asymmetric message authentication

secret-key message authentication (or MAC)

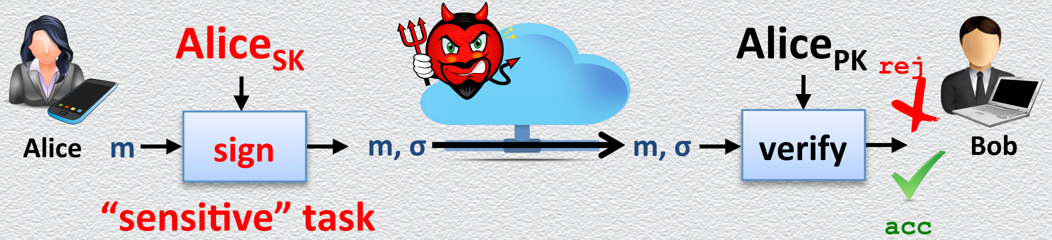
- main limitation
  - **session-specific** keys



public-key message authentication

(or **digital signatures**)

- main flexibility
  - **user-specific** keys
    - each user has two keys: **one public key PK** & **one private SK**
- (only) messages signed by sender’s SK can be verified by sender’s PK



Public-key infrastructure (PKI)

A mechanism for securely managing, in a dynamic multi-user setting, user-specific public-key pairs (to be used by some public-key cryptosystem)

- **dynamic, multi-user**
  - the system is open to anyone; users can join & leave
- **user-specific public-key pairs**
  - each user U in the system is assigned a unique key pair (U_{pk}, U_{sk})
- **secure management** (e.g., authenticated public keys)
  - public keys are authenticated: current U_{pk} of user U is publicly known to everyone

Very challenging to realize

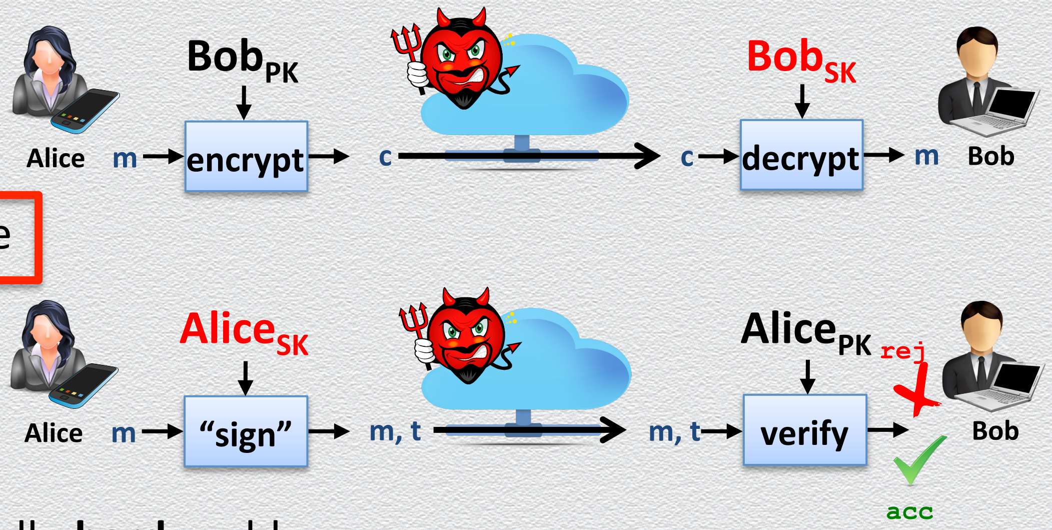
- currently using **digital certificates**; ongoing research towards a better approach…

Thus: Principles of modern cryptography

(A) security definitions, **(B) precise assumptions**, (C) formal proofs

For **asymmetric-key** message encryption/authentication

- adversary
  - types of attacks
- trusted set-up
  - public keys are securely available
  - secret keys remain secret
- trust basis
  - underlying primitives are secure
  - typically, **algebraic** computationally-**hard** problems
    - e.g., **discrete log**, **factoring**, etc.



General comparison

Symmetric crypto

- key management
  - less scalable & riskier
- assumptions
  - secret & authentic communication
  - secure storage
- primitives
  - generic assumptions
  - more efficiently in practice
- adversarial “sampling”
  - oracle access

Asymmetric crypto

- key management
  - more scalable & simpler
- assumptions
  - authenticity (PKI)
  - secure storage
- primitives
  - number-theoretic assumptions
  - less efficiently in practice (2-3 o.o.m.)
- adversarial “sampling”
  - public-key operations & oracle access

Public-key cryptography: Early history

Proposed by Diffie & Hellman

- documented in “New Directions in Cryptography” (1976)
- solution concepts of public-key encryption schemes & digital signatures
- key-distribution systems
  - Diffie-Hellman key-agreement protocol
    - “reduces” symmetric crypto to asymmetric crypto

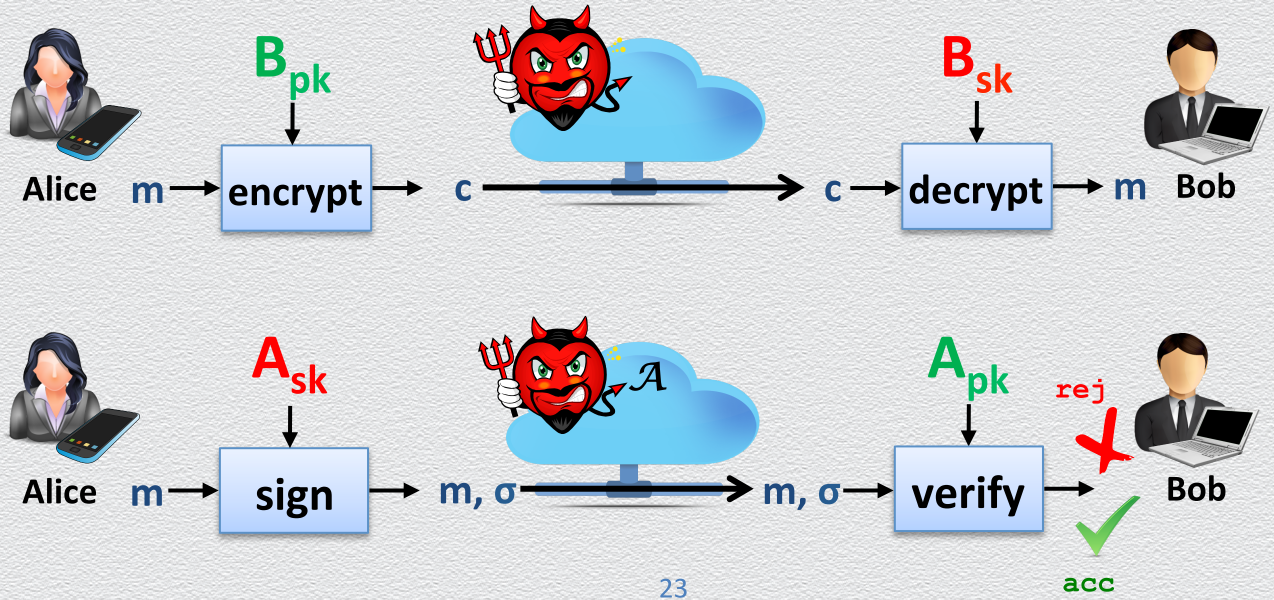
Public-key encryption was earlier (and independently) proposed by James Ellis

- classified paper (1970)
- published by the British Governmental Communications Headquarters (1997)
- concept of digital signature is still originally due to Diffie & Hellman

6.2 Public-key encryption

Recall: Public-key encryption & signatures

- assumes a trusted set-up
  - public keys are securely available (PKI) & secret keys remain secret



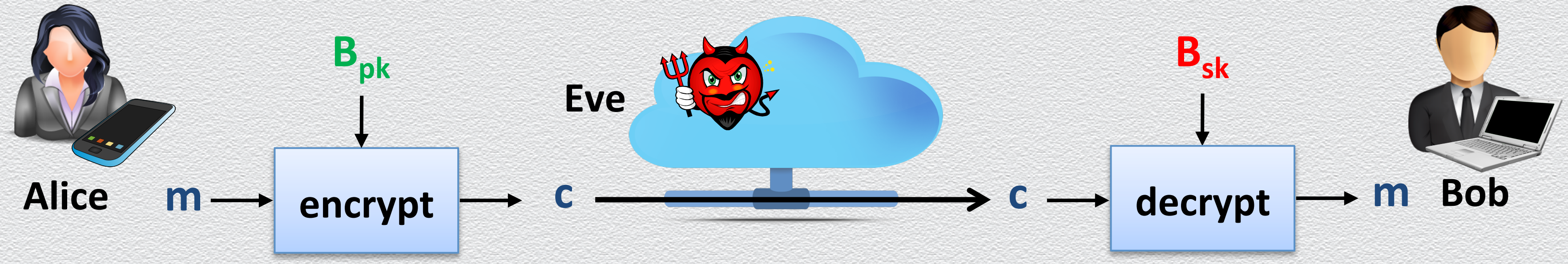
The setting of “asymmetric-key encryption”

Motivation: **Secret communication** amongst two parties

- Alice (sender/source) wants to send a message m to Bob (recipient/destination)
- Eve (attacker/adversary) can eavesdrop sent messages (i.e., unprotected channel)

Solution concept: **Public-key encryption scheme**

- Alice **encrypts** her message m to **ciphertext c**, which is sent instead of **plaintext m**
- Bob **decrypts** received message c to original message m; Eve “**cannot learn**” m from c
- Bob’s **public key B_{pk}** is used for encryption & Bob’s **secret key B_{sk}** for decryption



What's new

Setting: User-specific key pairs

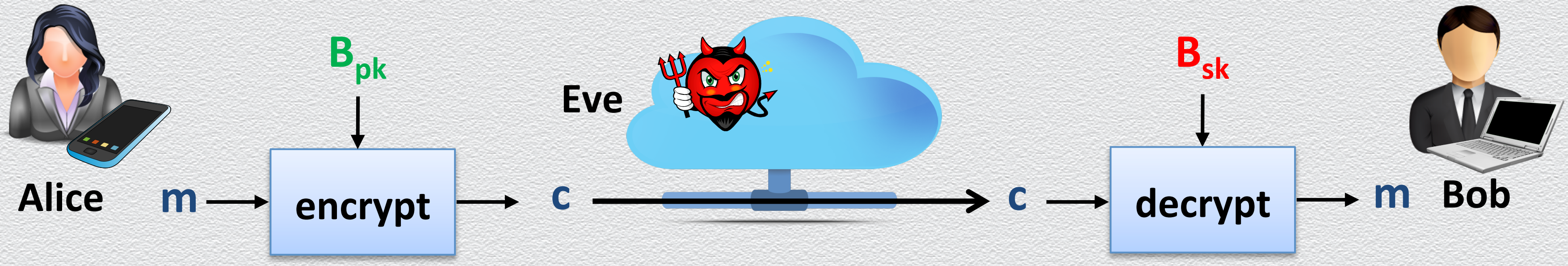
- many parties can encrypt, but only one party can decrypt
- messages encrypted using a user's PK can only be decrypted using that user's SK

Assumption: Public-key infrastructure

- each user U has unique key pair (U_{pk}, U_{sk}) and U_{pk} is publicly known

Solution concept: **Public-key encryption scheme**

- Bob's **public key B_{pk}** is used for encryption & Bob's **secret key B_{sk}** for decryption



Public-key encryption scheme

Abstract crypto primitive

- defined by **message space** M & triplet of algorithms **(Gen, Enc, Dec)**
  - Gen: probabilistic algorithm, outputs a key pair (U_{pk}, U_{sk}) for user U
  - Enc: probabilistic algorithm, on input plaintext m and key U_{pk}, outputs ciphertext c
  - Dec: deterministic algorithm, on input c and key U_{sk}, outputs a plaintext m

Properties

Correctness (as before)

- Enc(U_{pk}, m) = c, Dec(U_{sk}, c) = m, so that **Dec(U_{sk}, End(U_{pk}, m)) = m**
  - U_{pk}, U_{sk} have "canceling" effect, e.g., (U_{pk}, U_{sk}) may be of the form (K, K^{-1})

Security (as before, but with different security notions)

- ciphertext indistinguishability against **CPA-type of attackers**
  - infeasible for computationally adversary to distinguish among pair of ciphertexts
  - **randomized encryption is also required** (why?)

Non-triviality of public-key pairs

- easy to check (U_{pk}, U_{sk}) is a valid key pair
- infeasible to produce U_{sk} from U_{pk}

*A formal computational view of asymmetric encryption

An asymmetric-key encryption scheme Π is defined by

- a triple (**Gen**, **Enc**, **Dec**) of **PPT algorithms** where
- **Gen**: probabilistic key-generation algorithm, defines key space $\mathcal{K}_1$, $\mathcal{K}_2$
  - Gen($\mathbf{1^n}$) $\rightarrow$ pk $\in \mathcal{K}_1$, sk $\in \mathcal{K}_2$, where |pk|≥ n and |sk|≥ n
- **Enc**: probabilistic encryption algorithm, defines ciphertext space C
  - Enc: $\mathcal{K}_1 \times \mathcal{M} \rightarrow C$, for $\mathcal{M} = \{0,1\}^*$, Enc(pk, m) = Enc_{pk}(m) $\rightarrow$ c $\in C$
- **Dec**: deterministic encryption algorithm
  - Dec: $\mathcal{K}_2 \times C \rightarrow \{\mathcal{M}, \perp\}$, Dec(sk, c) = Dec_{sk}(c) := **m** $\in \mathcal{M}$ **or** $\perp$

*the security parameter 1^n is assumed to implicitly be input also to algorithms Enc and Dec

Facts

An attacker can of course possess the public key pk!

- **EAV-security ~ CPA-security**
  - all three of the following attack types **collapse to the same attack type**
    - ciphertext-only attack
    - known-plaintext attack
    - chosen-plaintext attack

EAV-security for asymmetric encryption

Encryption scheme
(Gen, Enc, Dec)

Π $\mathcal{T}$

$\mathrm{Gen}(1^n) \longrightarrow (pk, sk)$

$\{0, 1\} \longrightarrow b$

$\mathrm{Enc}_{pk}(m_b) \longrightarrow c_b$

m_0, m_1 ←

c_b →

b' ←

$\mathcal{A}$, pk

chooses **m_0,m_1**

The scheme is EAV-secure if
any attacker distinguishes among two ciphertexts only negligibly often,
even when it can use the recipient's public key pk.

CPA-security for asymmetric encryption

$\xleftarrow{m_i}$

$\xrightarrow{c_i}$

no need to explicitly ask for any ciphertext

Encryption scheme (Gen, Enc, Dec)

$\mathrm{Enc}_k(m_i) \rightarrow c_i$

Π $\mathcal{T}$

$\mathrm{Gen}(1^n) \rightarrow (pk, sk)$

$\{0, 1\} \rightarrow b$

$\mathrm{Enc}_{pk}(m_b) \rightarrow c_b$

$\xleftarrow{m_0,\, m_1}$

$\xrightarrow{c_b}$

$\xleftarrow{b'}$

$\mathcal{A}^{\mathbf{Enc(pk,)}}$

chooses $\mathbf{m_0,m_1}$

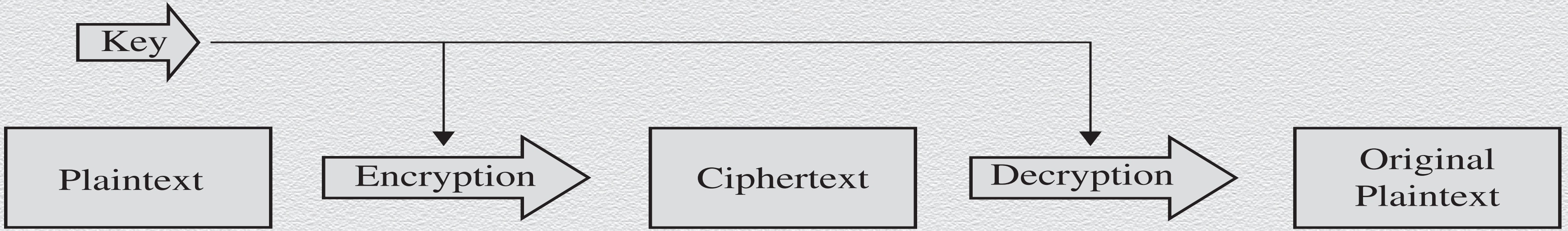
The scheme is EAV-secure if
any attacker distinguishes among two ciphertexts only negligibly often,
even when it learns the encryptions of messages of its choice.

Facts (II)

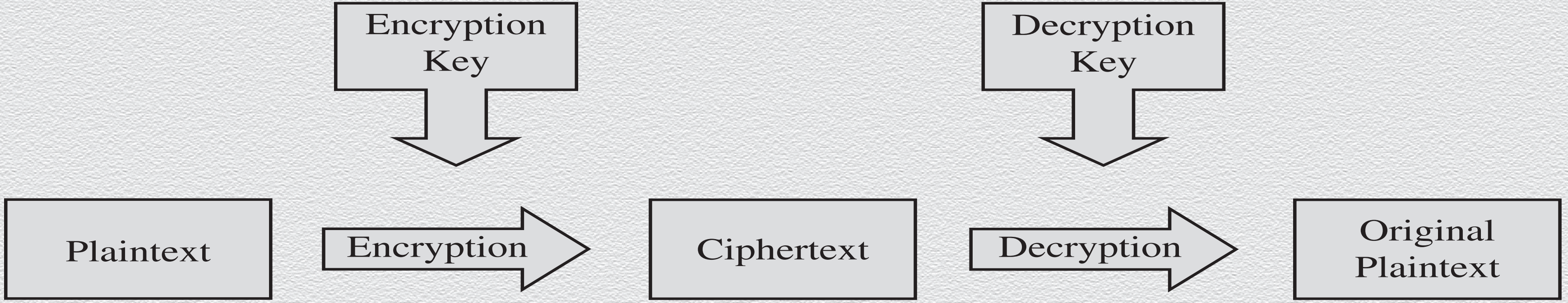
An attacker can of course possess the public key pk

- **EAV-security ~ CPA-security**
  - if a scheme is EAV-secure then it is also CPA-secure
- probabilistic encryption is **necessary** for CPA-security

Symmetric Vs. Asymmetric encryption



(a) Symmetric Cryptosystem



(b) Asymmetric Cryptosystem

Secret-key vs. public-key encryption

	Secret Key (Symmetric)	Public Key (Asymmetric)
Number of keys	1	2
Key size (bits)	56–112 (DES), 128–256 (AES)	Unlimited; typically no less than 256; 1000 to 2000 currently considered desirable for most uses
Protection of key	Must be kept secret	One key must be kept secret; the other can be freely exposed
Best uses	Cryptographic workhorse. Secrecy and integrity of data, from single characters to blocks of data, messages and files	Key exchange, authentication, signing
Key distribution	Must be out-of-band	Public key can be used to distribute other keys
Speed	Fast	Slow, typically by a factor of up to 10,000 times slower than symmetric algorithms

6.3 Hybrid encryption

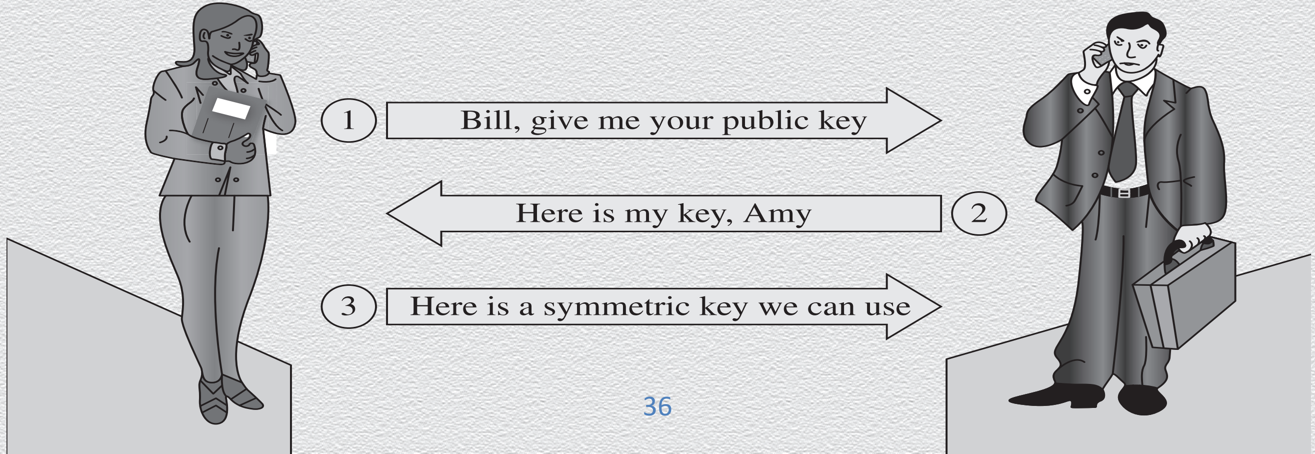
Secret-key cryptography is "reduced" to public-key

Had we established a **secure PKI**, secret-key management would be solved!

- need to overcome many challenges in order to achieve this…

Main idea

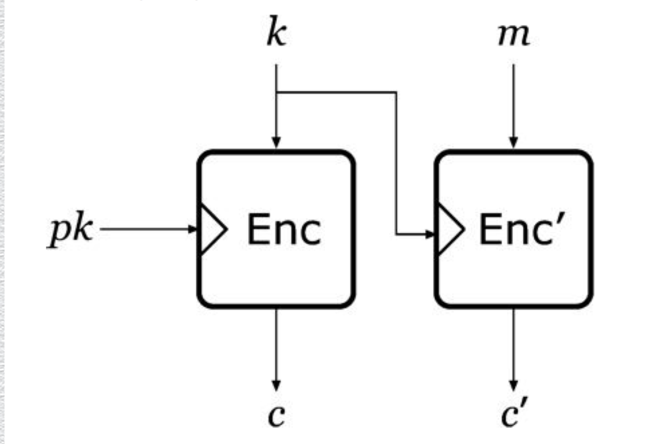
- generate fresh session-specific secret key &
- securely share it with the intended party using PK encryption



Hybrid encryption

"Reduces" public-key crypto to secret-key crypto

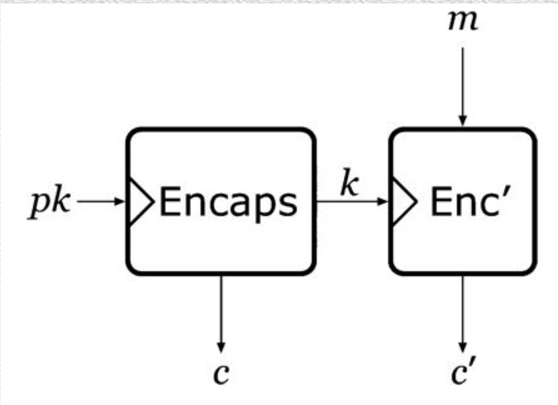
- better performance than block-based public-key CPA-encryption
- main idea
  - apply PK encryption on random key k
  - use k for secret-key encryption of m



Hybrid encryption using the KEM/DEM approach

"Reduces" public-key crypto to secret-key crypto

- main idea
  - **encapsulate** secret key k into c
  - use k for secret-key encryption of m
  - KEM: key-encapsulation mechanism - Encaps
  - DEM: data encapsulation mechanism - Enc'



- KEM/DEM scheme
  - CPA-secure if KEM is CPA-secure and Enc' EAV-secure
  - CCA-secure if KEM and Enc' are CCA-secure

6.4 The ElGamal encryption scheme

Discrete logarithm problem

Setting

- consider cyclic group Z^*_p with order $\phi(p) = p-1$ that is generated by element g in Z^*_p
- (g, p) are public information; for any x in Z^*_p, it is easy to compute x^k mod p for any integer k

Problem

- given a in Z^*_p, compute an integer k such that g^k mod p = a
- it is believed to be a hard problem to solve

Example

- Z^*_{17} is a cyclic group with order 16, 3 is a generator of the group
- if k = 1580212, then $3^k = 13$ mod 17 is easy to compute
- but finding an integer k such that $3^k = 13$ mod 17 is hard (for large values of p, that is)

ElGamal encryption

- Let p be a prime, g be a generator of Z_p, message space Z_p
- **Secret key:** random number $a \in Z^*_p$
- **Public key:** $A = g^a$
- **Encryption**:
  - Pick a random $r \in Z^*_{p-1}$ and set $R = A^r = g^{ar}$, $c_1 = g^r$
  - $\mathbf{E_{PK}(m) = (c_1,c_2)}$ where $\mathbf{c_2 = mR \bmod p}$
- **Decryption:** $\mathbf{D_{SK}(c_1,c_2) = c_2\ (1/c_1^a) \bmod p}$ where $\mathbf{c_1^a = g^{ar}}$
- Security depends on Computational Diffie-Hellman (CDH) assumption
  - given (g, g^a, g^b) it is hard to compute g^{ab}
  - the same r should not be used twice
- A signature scheme can be also derived based on above discussion

6.5 Digital signatures

Real-world signatures

A real-life example

- a buyer pays by credit card and signs a bill/receipt
- the buyer, however, later can potentially deny his signature

Vendors or credit card providers typically accept such denied changes

- e.g., credit cards can be stolen, signatures are easy to fake

Goal

- devise a service in the electronic world that implements **digital signatures** that are, however, **infeasible to fake**

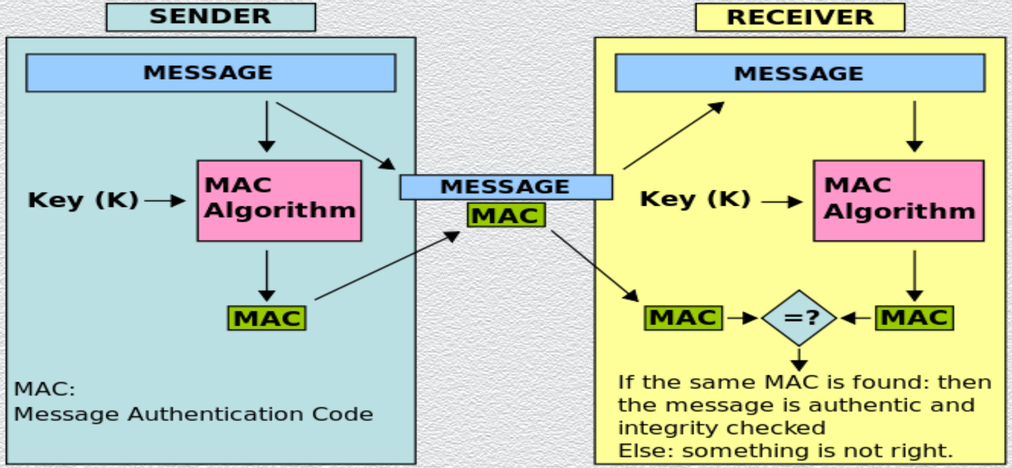
MACs for digital signing

Straightforward approach

- two parties share a secret key K
- one party generates MAC on the message to be signed, using K
- message digest serves as a signature
- the other party verifies integrity of "signature" using K

Properties

- authentication
- data integrity



Public-key message authentication – digital signatures

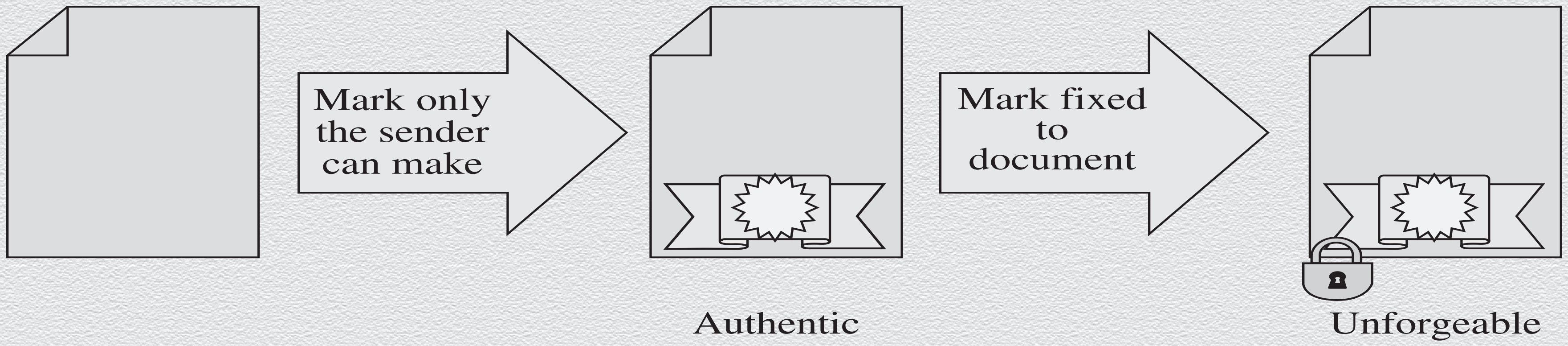
Main idea

- messages signed using a user's secret key can be only verified using the user's (corresponding) public key
- one party generates digital signature, many parties can verify

Digital signatures paradigm

- PKI is assumed
  - user U has unique key pair (U_{pk}, U_{sk})
  - U_{pk} is publicly known
- employ **private key** U_{sk} for **signing** sent messages (performed by user U)
- employ **public key** U_{pk} for **verifying** messages signed by U (performed by any user)

Schematically: Digital signatures



Digital signature scheme

Abstract crypto primitive

- defined by **message space** M & triplet of algorithms **(Gen, Sign, Vrfy)**
  - Gen: probabilistic algorithm, outputs a key pair (U_{pk}, U_{sk}) for user U
  - Sign: probabilistic algorithm, on input plaintext m and key U_{sk}, outputs signature σ
  - Dec: deterministic algorithm, on input m, σ and key U_{pk},
    outputs 1, i.e., 'accept', or 0, i.e., `reject'

Properties

Correctness (as with MAC schemes)

- Sign(U_{sk}, m) = σ, Vrfy(U_{pk}, m, σ) = 0/1, so that **Vrfy(U_{pk}, m, Sign(U_{sk}, m)) = 1**
  - U_{pk}, U_{sk} have "canceling" effect, e.g., (U_{pk}, U_{sk}) may be of the form (K, K^{-1})

Security (as with MAC schemes)

- existential unforgeability
  - infeasible for computationally adversary to forge an invalid signature on a new message of its choice, even if the attacker learns signatures of messages of its choice
  - discussion: how security must be defined?

Non-triviality of public-key pairs

- easy to check (U_{pk}, U_{sk}) is a valid key pair
- infeasible to produce U_{sk} from U_{pk}

Existential unforgeability for digital signatures

Signature scheme
(Gen, Sign, Vrfy)

Σ $\mathcal{T}$

$\text{Gen}(1^n) \to (\text{sk}, \text{pk})$

$\text{Sign}_{\text{sk}}(m_i) \to \sigma_i$

$\xrightarrow{\text{pk}}$

$\xleftarrow{m_1}$

$\xrightarrow{t_1}$

$\xleftarrow{m_2}$

$\xrightarrow{t_2}$

…

$\xleftarrow{m^*, t^*}$

$\mathcal{A}^{\text{Sign(sk,)}}$

$\mathcal{Q} = m_1, m_2, \ldots$

The scheme is **secure** if any attacker finds a new forgery non-negligibly often.

Attacker wins if and only if

1. $\text{Vrfy}_{\text{pk}}(m^*, t^*) = 1$ **successful forgery**

2. m^* not in $\mathcal{Q}$ **replay-attack safe**

High-level security properties of digital signatures

authentication: verify (authenticity of) source of message m

- the receiver can determine that the signature really came from the signer

integrity/unforgeability: verify (correctness of) contents of message m

- no one other than the signer can produce the signature without the signer's private key

non-repudiation: ensure that someone **cannot deny** something

- typically, it refers to the ability to ensure that a party cannot deny the authenticity of their signature on a document

Do MACs offer non-repudiation?

Other properties signatures can satisfy

Digital signatures can also satisfy

- **not alterable signatures**
  - no signer, receiver, or any interceptor can modify the signature without the tampering being evident
- **not reusable signatures (replay-attack safeness)**
  - any attempt to reuse a previous signature will be detected by receiver

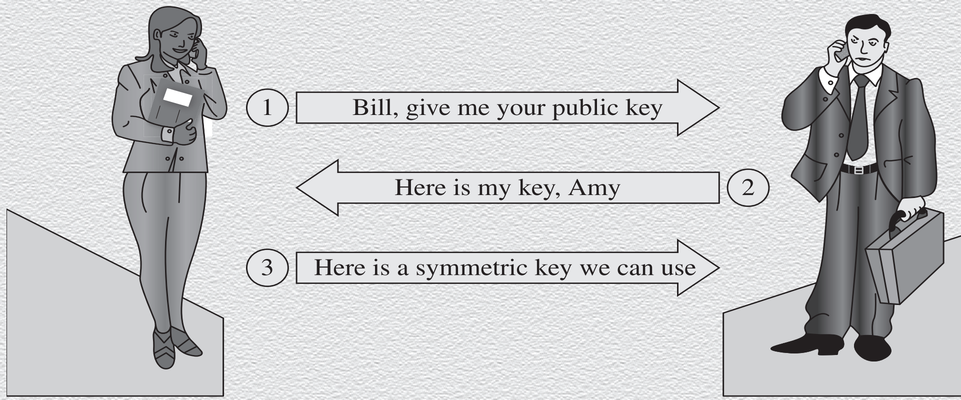
6.6 Key agreement

2 approaches for securely distributing a shared secret key

Refers to key-agreement problem

- Approach 1
  - use public-key encryption
- Approach 2
  - Diffie-Hellman key-agreement protocol

Key agreement

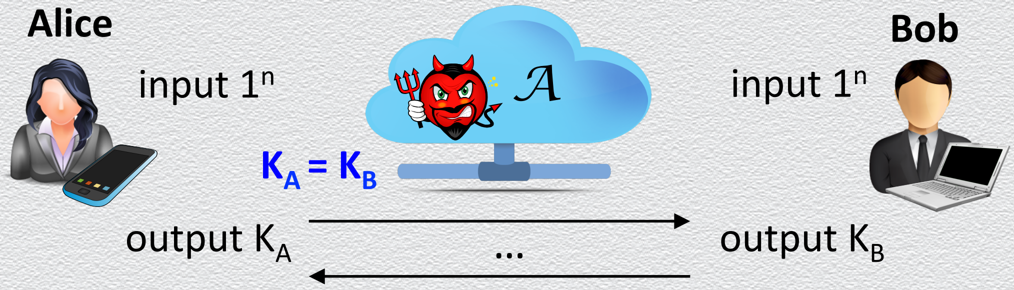


1) From a secure PKI

- if a PKI exists, then we can “solve” the secret-key distribution problem
- MITM attacks

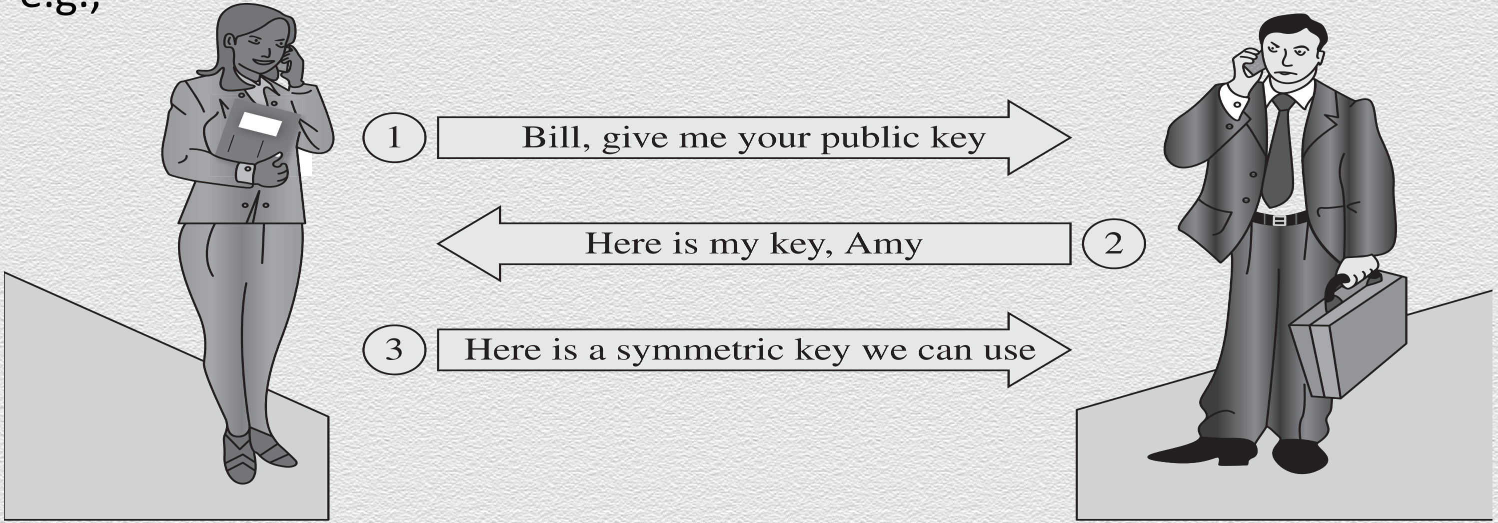
2) From specific 2-party cryptographic schemes

- key-agreement (KA) protocols
- parties agree on a secret key, using individual randomness



1) Secret-key cryptography is “reduced” to public-key

- had we established a **secure PKI**, secret-key management is solved
  - generate fresh session-specific secret keys and securely share them **once** with the intended party using PK encryption
  - e.g.,

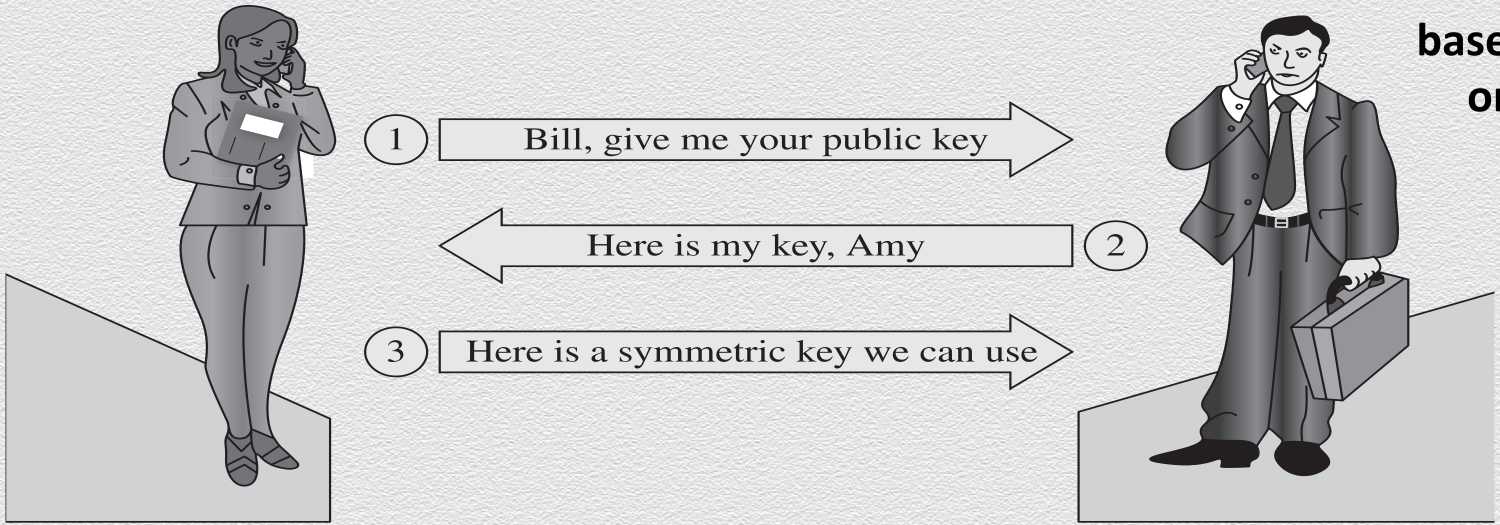


Using PK-encryption to exchange secret keys: Challenges

- **many challenges**, e.g., make sure that you're talking to the intended user…

public-key U_{PK} ⟷ **user U**

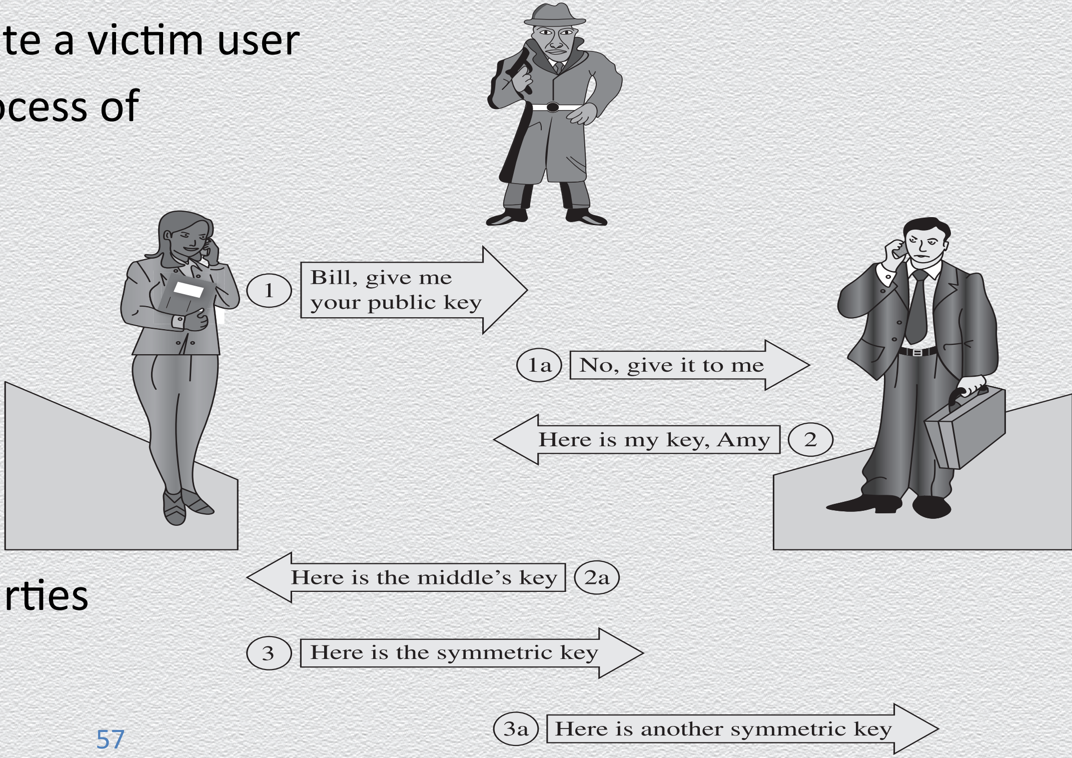
trust with U is established based solely on U_{PK}!



Things can go wrong…

General threat

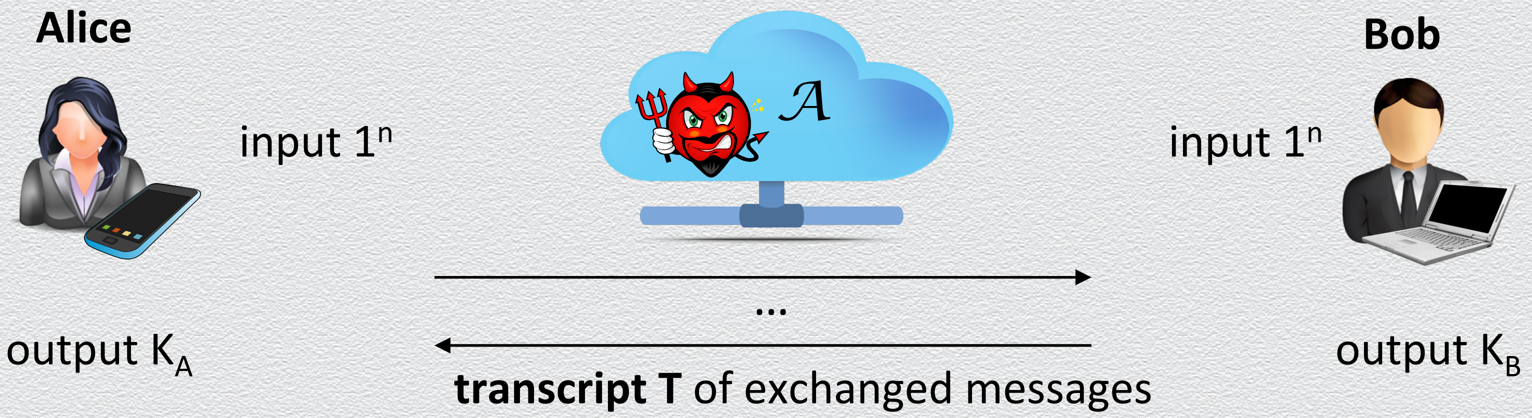
- the adversary will try to impersonate a victim user by faking U_{PK} or circumvent the process of establishing a secure PKI set-up
- e.g., by launching a Man-In-The-Middle (MITM) attack
  - the attacker can covertly actively control the entire communication between the parties



2) Key-agreement (KA) scheme

Alice and Bob want to securely establish a **shared key** for secure chatting over an **insecure** line

- instead of meeting in person in a secret place, they want to use the insecure line…
- **KA scheme**: 2-party key-agreement protocol Π s.t. both contribute to a shared key K



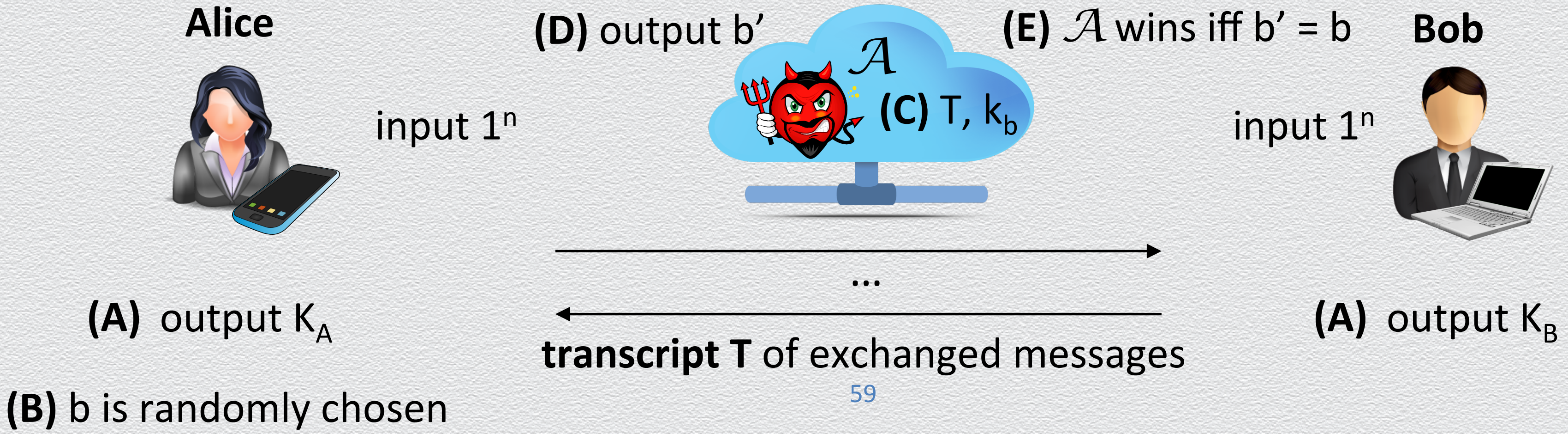
- correctness: $K = K_A = K_B$
- security: no PPT adversary $\mathcal{A}$, given T, can distinguish K from a trully random one

Game-based security definition for KA schemes

- (A) key-agreement scheme $\Pi(1^n)$ runs to generate $K = K_A = K_B$ and transcript T
- (B) bit b is randomly chosen
- (C) adversary $\mathcal{A}$ is given T and k_b, where **$k_0 = K$**, else **k_1 is random** (both n-bit long)
- (D) A outputs bit b' and wins if b' = b;

Π is **secure** if no PPT $\mathcal{A}$ can win non-negligibly better than guessing, i.e.,

$$\forall \text{ PPT adversary } \mathcal{A},\ \exists \text{ a negl. } \nu(n) \text{ s.t. } \Pr[\mathcal{A} \text{ wins above game}] \leq ½ + \nu(n)$$



The Diffie-Hellman key-agreement protocol

Alice and Bob want to securely establish a **shared key** for secure chatting over an **insecure** line

- DH KA scheme Π
  - discrete log setting: prime p, public generator g with <g> = Z^*_p,

Alice

input 1^n

Bob

input 1^n

(1) randomly pick secret a

(2) randomly pick secret b

(3) send g^a mod p →

← **(4)** send g^b mod p

(5) set **K = g^{ab} mod p** = $(g^b \bmod p)^a \bmod p$

(6) set **K = g^{ab} mod p** = $(g^a \bmod p)^b \bmod p$

The discrete logarithm problem

Discrete logarithm setting

- let p be an odd prime, then $G = (Z_p^*, \cdot)$ is a cyclic group of order $p - 1$
- let g be a generator of the group, then $|\langle g \rangle| = p - 1$
  - for any element a in the group , we have: **$g^k = a \bmod p$, for some integer k**
  - k is called the **discrete logarithm** of a (mod p)

Computational assumption: **Computing discrete logs is computationally hard**

- i.e., if we know (a, g, p), it is hard to compute $k = \log_g a \pmod p$

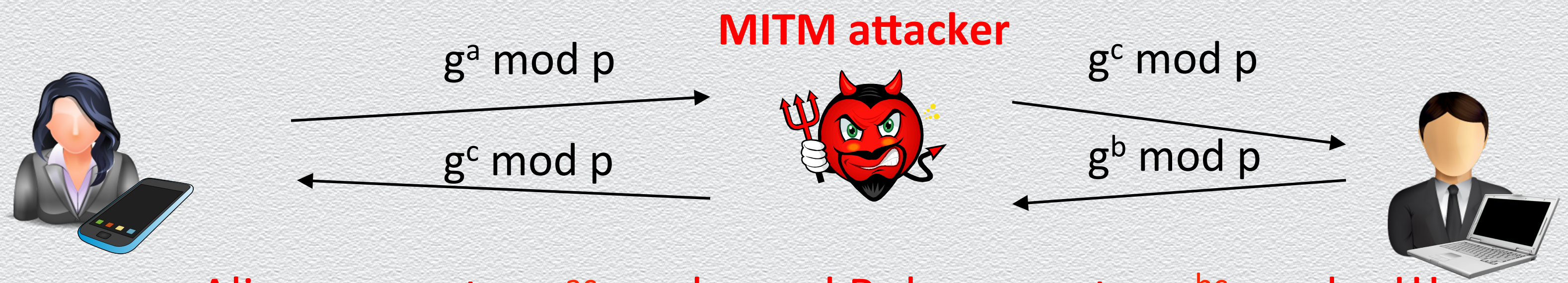
Example

- $(Z_{17}^*, \cdot)$ is a cyclic group G with order 16, 3 is the generator of G and $3^{16} = 1 \bmod 17$
- let $k = 4$, $3^4 = 13 \bmod 17$ (which is easy to compute)
- the inverse problem: if $3^k = 13 \bmod 17$, what is k? what about **large p**?

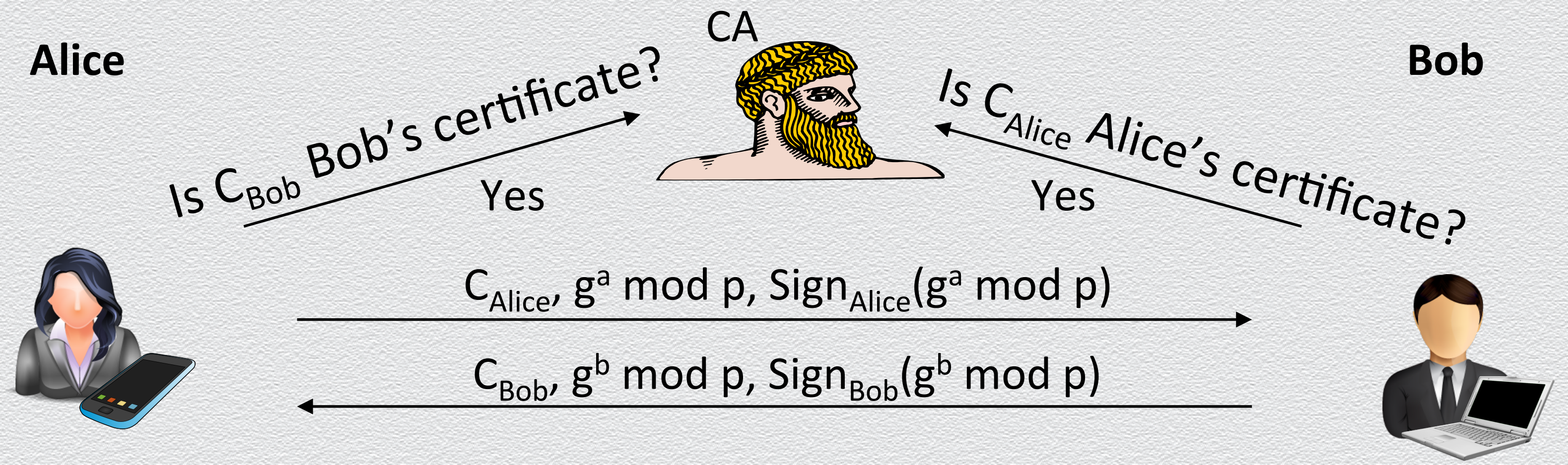
Security

- discrete log assumption is necessary but not sufficient
- decisional DH assumption
  - given g, g^a and g^b, g^{ab} is computationally indistinguishable from uniform

Authenticated Diffie-Hellman



Alice computes g^{ac} mod p and Bob computes g^{bc} mod p !!!



6.7 Public-key certificates

How to set up a PK setting (or PKI)?

- How are public keys stored? How to obtain a user's public key?
- How does Bob know or 'trust' that A_{PK} is Alice's public key?
- How APK (a bit-string) is securely bound to an entity (user/identity)

public key: A_{PK}
secret key: A_{SK}

public key: B_{PK}
secret key: B_{SK}

Achieving a PKI…

How can we maintain the invariant that at all times

1. any given **user** U is **assigned** a **unique** public-private key pair; and
2. any other user known U’s **current** public key?

- this remains a challenging problem even today!
  - we will explore aspects later – why is challenging?
- PK cryptosystems come with a set-up algorithm which is run by U
  - on input a security-strength parameter it outputs a random valid key pair for U
- public keys can be made publicly available
  - e.g., sent by email, published on web page, added into a public directory, etc.

entails binding users/identities to public keys

Distribution of public keys

- **Public announcement**
  - users distribute public keys to recipients or broadcast to community at large
- **Publicly available directory**
  - can obtain greater security by registering keys with a public directory
- Both approaches have problems, and are vulnerable to forgeries

Do you trust your public key?

- Impostor claims to be a true party
  - True party has a public and private key
  - Impostor also has a public and private key

- Impostor sends impostor’s own public key to the verifier
  - Says, “This is the true party’s public key”
  - This is the critical step in the deception

Certificates: Trustable identities & public keys

- a certificate is a public key and an identity bound together and signed by a certificate authority
- a certificate authority is an authority that users trust to accurately verify identities before generating certificates that bind those identities to keys

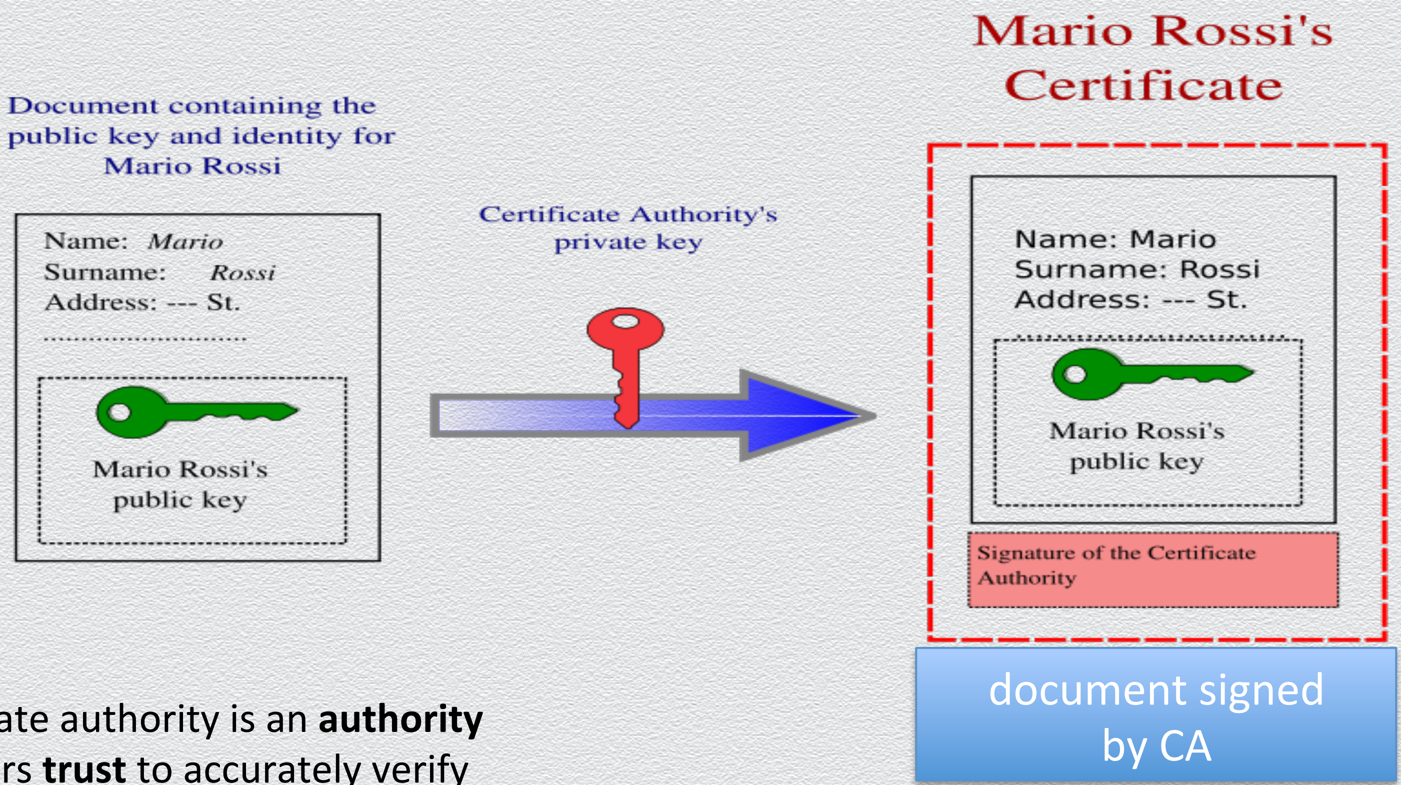
Public-key certificates

Current (imperfect) practice for achieving **trustable identities & public keys**

- a certificate is a signed statement **binding** identities to public keys
  - e.g., a signature on the statement "Alice's public key is 1032xD"
- contents are digitally signed by a trusted Public-Key or **Certificate Authority (CA)**
  - can be verified by anyone who knows the CA's public-key
- when Bob wants to send an encrypted message to Alice
  - he first **obtains & verifies** a certificate of Alice's public key

An example

a certificate is a public key and an identity bound together and signed by a certificate authority (CA)



a certificate authority is an **authority** that users **trust** to accurately verify identities before generating certificates that bind those identities to keys

Symantec

Certificate hierarchy

- Single CA certifying every public key is impractical
- Instead, use trusted root authorities
- Root CA signs certificates for intermediate CAs, they sign certificates for lower-level CAs, etc.
  - Certificate "chain of trust"
    - $sig_{Symantec}$("Stevens", $PK_{Stevens}$)
    - sig_{UMD}("faculty", $PK_{faculty}$)
    - $sig_{faculty}$("Nikos", PK_{Nikos})

Example 1: Certificate signing & hierarchy

To create Diana's certificate:

Diana creates and delivers to Edward:

Name: Diana Position: Division Manager Public key: 17EF83CA ...

Edward adds:

Name: Diana Position: Division Manager Public key: 17EF83CA ...	hash value 128C4

Edward signs with his private key:

Name: Diana Position: Division Manager Public key: 17EF83CA ...	hash value 128C4

Which is Diana's certificate.

To create Delwyn's certificate:

Delwyn creates and delivers to Diana:

Name: Delwyn Position: Dept Manager Public key: 3AB3882C ...

Diana adds:

Name: Delwyn Position: Dept Manager Public key: 3AB3882C ...	hash value 48CFA

Diana signs with her private key:

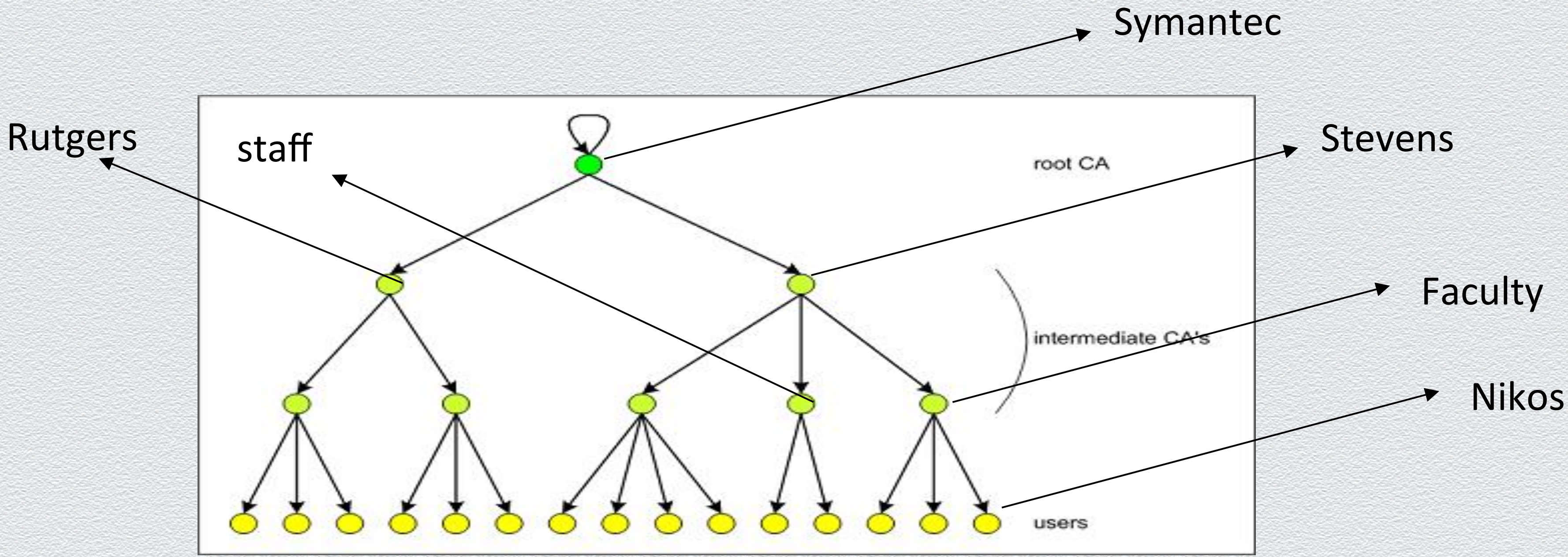
Name: Delwyn Position: Dept Manager Public key: 3AB3882C ...	hash value 48CFA

And appends her certificate:

Name: Delwyn Position: Dept Manager Public key: 3AB3882C ...	hash value 48CFA
Name: Diana Position: Division Manager Public key: 17EF83CA ...	hash value 128C4

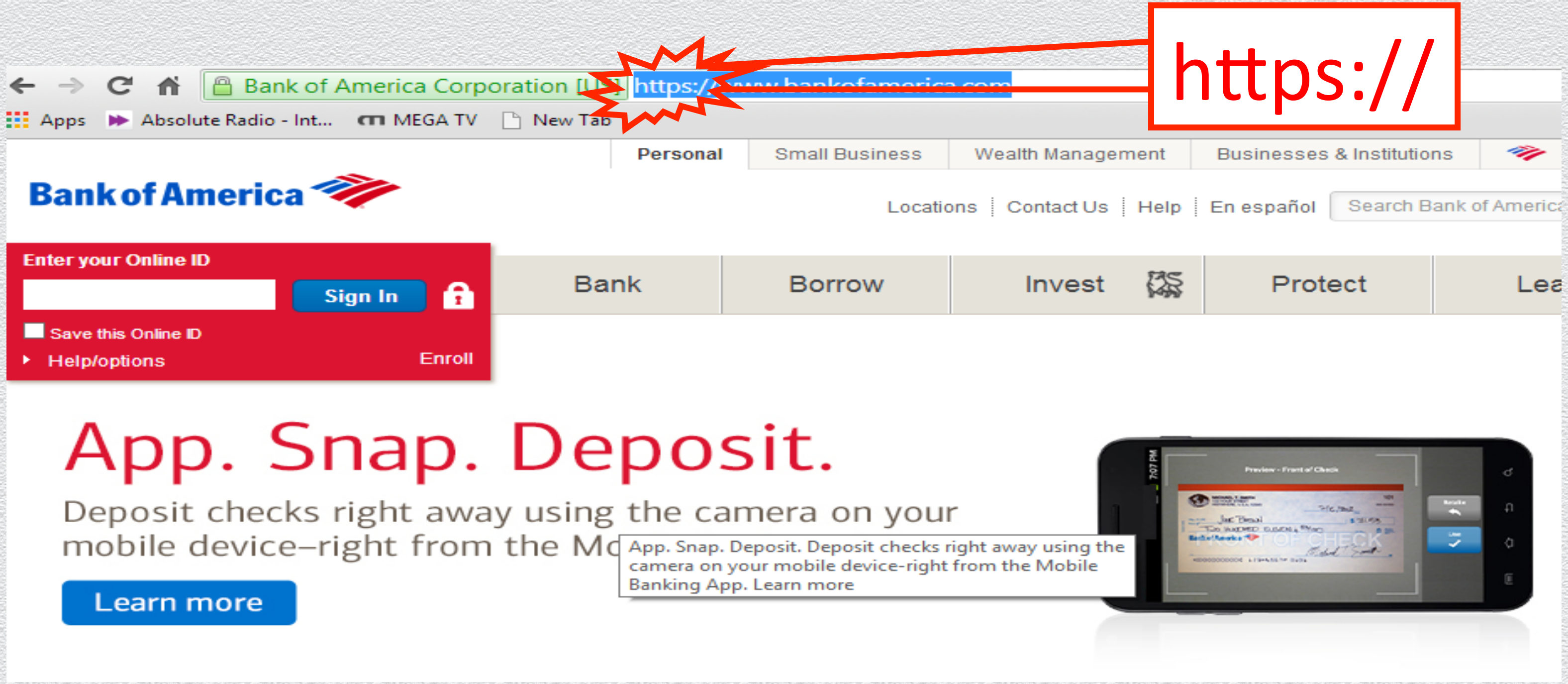
Which is Delwyn's certificate.

Example 2



What bad things can happen if the root CA system is compromised?

Secure communication over the Internet



What cryptographic keys are used to protect communication?

X.509 certificates

- Defines framework for authentication services
  - defines that public keys stored as **certificates** in a public directory
  - certificates are **issued and signed** by an entity called **certification authority (CA)**
- Used by numerous applications: SSL
- Example: see certificates accepted by your browser