https://brown-csci1660.github.io

CS1660: Intro to Computer Systems Security
Spring 2026

Lecture 8: Public-key Cryptography

Instructor: **Nikos Triandopoulos**

February 19, 2026

BROWN

CS1660: Announcements

- Course updates
  - Project 1 “Cryptography” is due today
  - HW 1 is due next Thursday (Feb 26)

Last class

- Cryptography
  - Integrity & reliable communication
    - Message authentication codes (MACs)
    - Authenticated encryption, side-channel attacks
    - Cryptographic hash functions, cryptographic hashing in practice & applications
- Authentication
  - User authentication: something you know, are, have
    - Password security and cracking, more on password cracking
  - The Merkle tree

Today

- Cryptography
  - Introduction to modern cryptography
  - Secure communication & symmetric-key encryption in practice
  - Integrity & reliable communication
  - Public-key encryption & digital signatures
    - Motivation, key management, hybrid encryption, implementation, assumptions

Public-key crypto

- Authentication
  - User authentication: something you know, are, have
    - Password security and cracking, more on password cracking
  - The Merkle tree

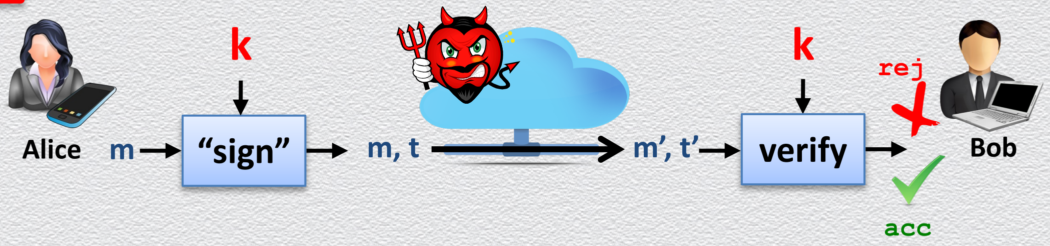
8.1 Public-key encryption & digital signatures

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, hashing, ...
    - e.g., block ciphers, AES, etc.



On “secret key is distributed securely”

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

- “securely obtain” → 1. strong assumption to accept
  - need of a secure channel
- “shared secret key” → 2. challenging problem to manage
  - too many keys

→ **Public-key cryptography to the rescue…**

On “secret key is distributed securely”

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

- “securely obtain” ⇨ 1. 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” ⇨ 2. 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-specific secret keys

- 1. For secure distribution, just **make another (more reasonable) 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)
- 2. For secure management, just **live with it**!

Public-key cryptography to the rescue…

Public-key (or asymmetric) cryptography

disclaimer on names
private = secret

Goal: devise a cryptosystem where key setup is more manageable

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

- user U generates two correlated 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 (run by **other users**)
- employ **private** key U_{sk} for certain "**sensitive/critical**" tasks (run by **user U**)

New assumption

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

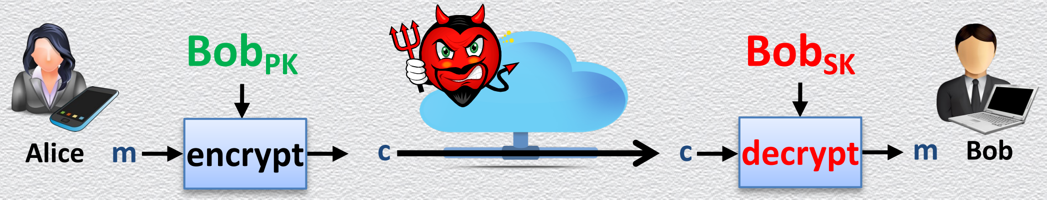
From symmetric to asymmetric encryption

secret-key encryption

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

public-key encryption

- main flexibility
  - **user-specific** keys



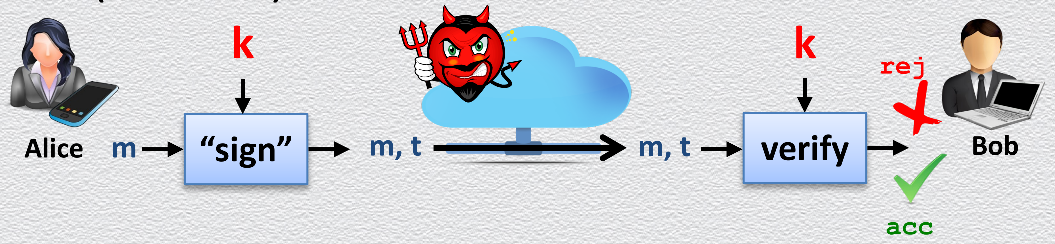
"sensitive" task

- messages encrypted by receiver's PK can (only) be decrypted by receiver's SK

From symmetric to asymmetric message authentication

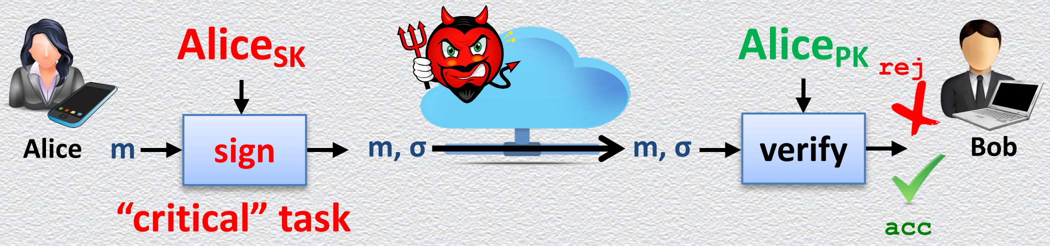
secret-key message authentication (or MAC)

- main limitation
  - session-specific keys



public-key message authentication (or **digital signatures**)

- main flexibility
  - **user-specific** keys



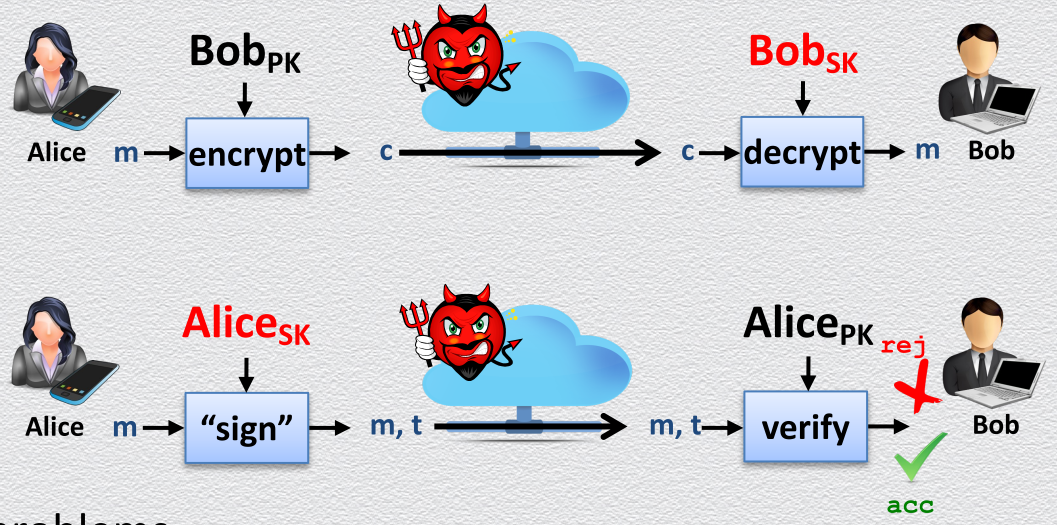
- (only) messages signed by sender’s SK can be verified by sender’s PK

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
  - PKI is needed
  - secret keys remain secret
- trust basis
  - underlying primitives are secure
  - 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 efficient in practice

Asymmetric crypto

- key management
  - more scalable & simpler
- assumptions
  - authenticity (PKI)
  - secure storage
- primitives
  - math assumptions
  - less efficient in practice (2-3 o.o.m.)

Public-key infrastructure (PKI)

A system 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 currently assigned a unique key pair (U_{pk}, U_{sk})
- **secure management**
  - public keys are authenticated: correct current U_{pk} of user U is known to everyone

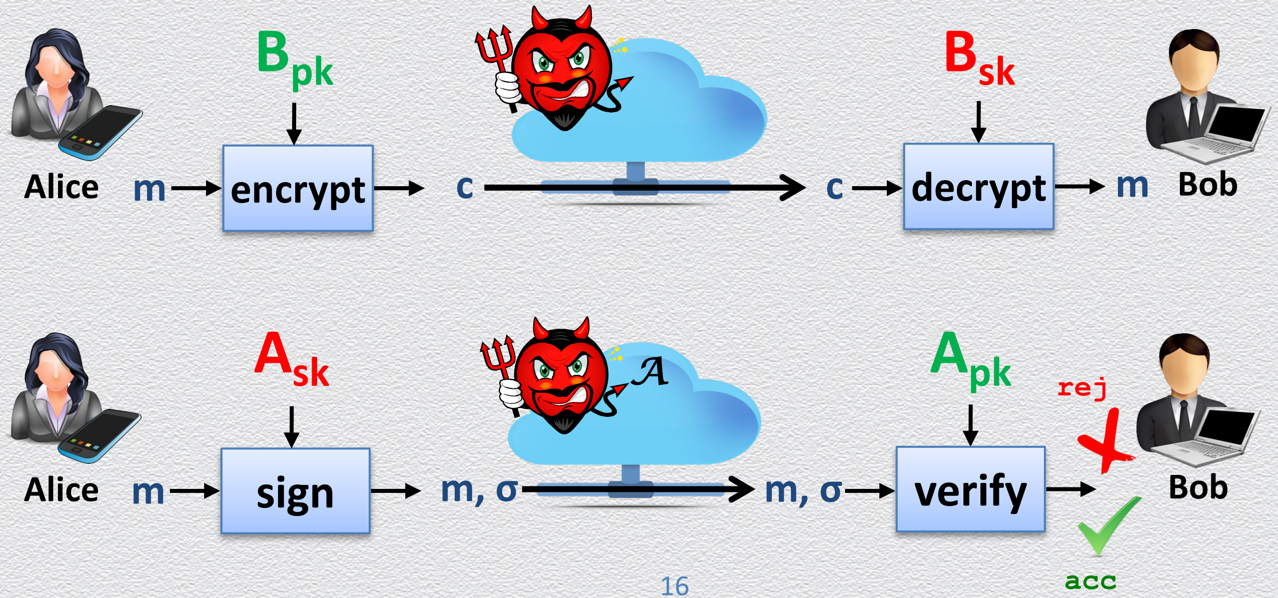
Very challenging to realize

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

Overall: Public-key encryption & signatures

Assume a trusted set-up

- public keys are securely available (PKI) & secret keys remain secret



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

8.2 Public-key certificates

How to set up a 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 A_{PK} (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}

Problem statement

A PKI entails binding user identities to public keys

How can we maintain the invariant that

- any given **user** U is **assigned** a **unique** public-private key pair; and
- any other user may learn U's **current** public key?
  - secret keys can be lost, stolen or they should be revoked

Recall

- PK cryptosystems come with a Gen 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.

Distribution of public keys

Public announcement

- Users distribute public keys to recipients or broadcast to community at large

Publicly available directory

- Users register public keys to a public directory

Both approaches have problems and are vulnerable to forgeries

Do you trust a public key?

A PKI entails binding user identities to public keys

One is what their public key “claims to be”

- Impostor wants to claim to be a true party
  - true party has a public and private key
  - impostor also has a public and private key
- Impostor manages to send impostor’s own public key to the sender/verifier
  - claims, “This is the true party’s public key”
    - critical step in the deception
  - succeeds in decrypting/forging a message as received/signer

Certificates: Trustable identities & public keys

Certificate

- a public key & an identity **bound** together
- in a document **signed by** a certificate authority

Certificate authority (CA)

- an authority that users **trust** to securely bind identities to public keys
  - CA **verifies identities** before generating certificates for these identities
    - E.g., domain, organization or extended validation
  - secure binding via **digital signatures**
    - **ASSUMPTION**: The authority's PK CA_{PK} is authentic

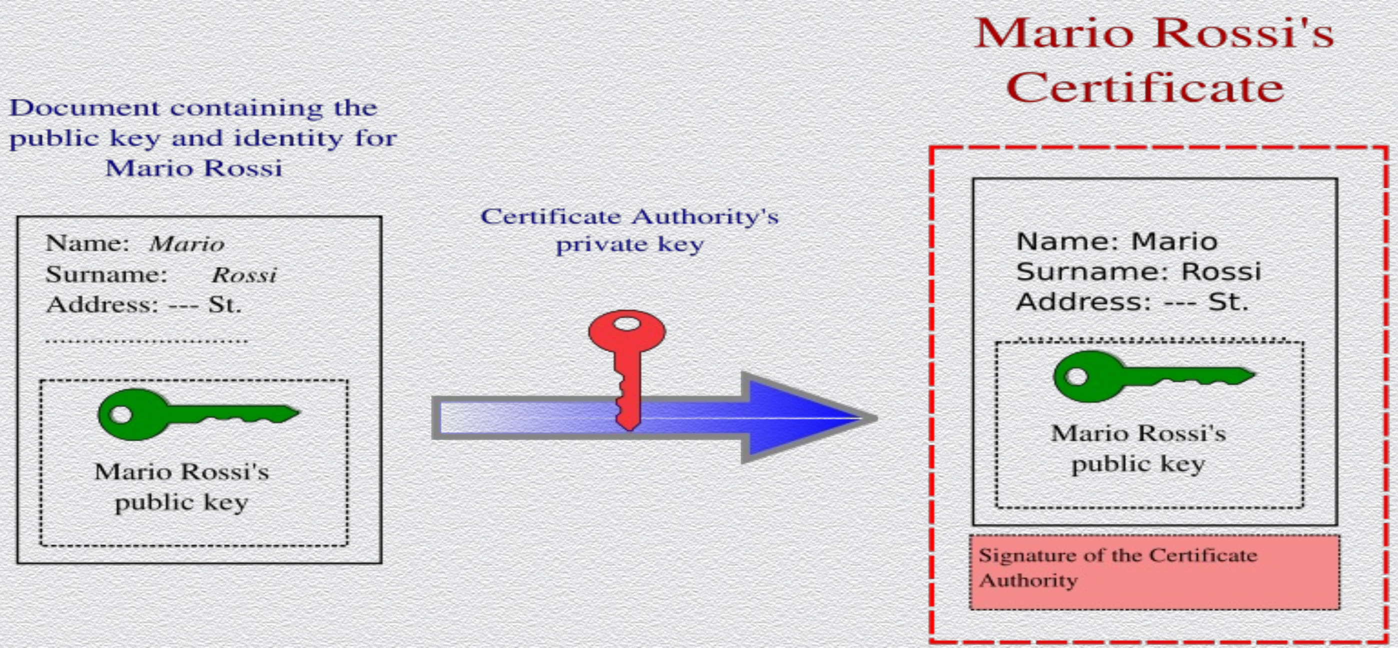
Public-key certificates in practice

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

- everybody trusts a Certificate Authority (CA)
  - everybody knows CA_{PK} & trusts that CA knows/protects corresponding secret key CA_{SK}
- a certificate binds identities to public keys in a CA-signed statement
  - e.g., Alice obtains a signature on the statement "Alice's public key is 1032xD"
- users query CA for public keys of intended recipients or signers
  - e.g., when Bob wants to send an encrypted message to Alice
    - he first **obtains & verifies** a certificate of Alice's public key
  - e.g., when Alice wants to verify the latest software update by Company
    - she first **obtains & verifies** a certificate of Company's public key

Example

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



document signed by 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 certificate authorities

- root CA signs certificates for intermediate CAs, they sign certificates for lower-level CAs, etc.
  - certificate "chain of trust"
    - $sign_{SK\_Symantec}$ ("Brown", PK_{Brown})
    - $sign_{SK\_Brown}$ ("faculty", $PK_{faculty}$)
    - $sign_{SK\_faculty}$ ("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:

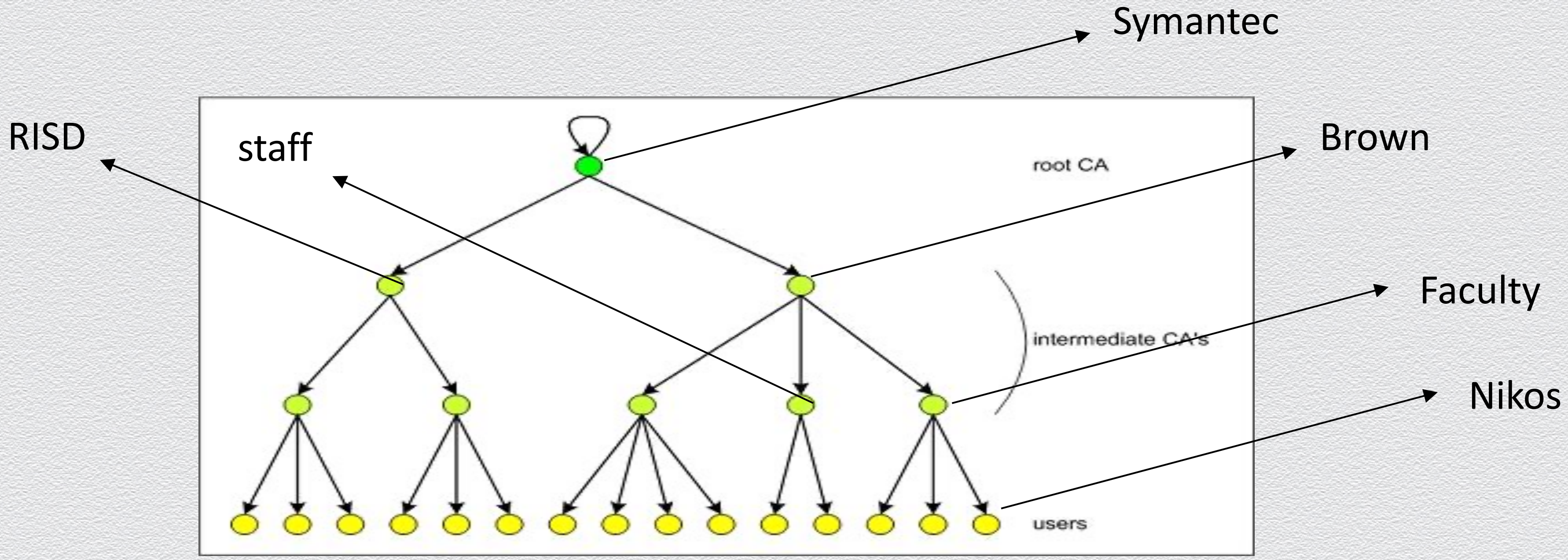
Name: Delwyn Position: Dept Manager Public key: 3AB3882C ...	hash value 48CFA

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

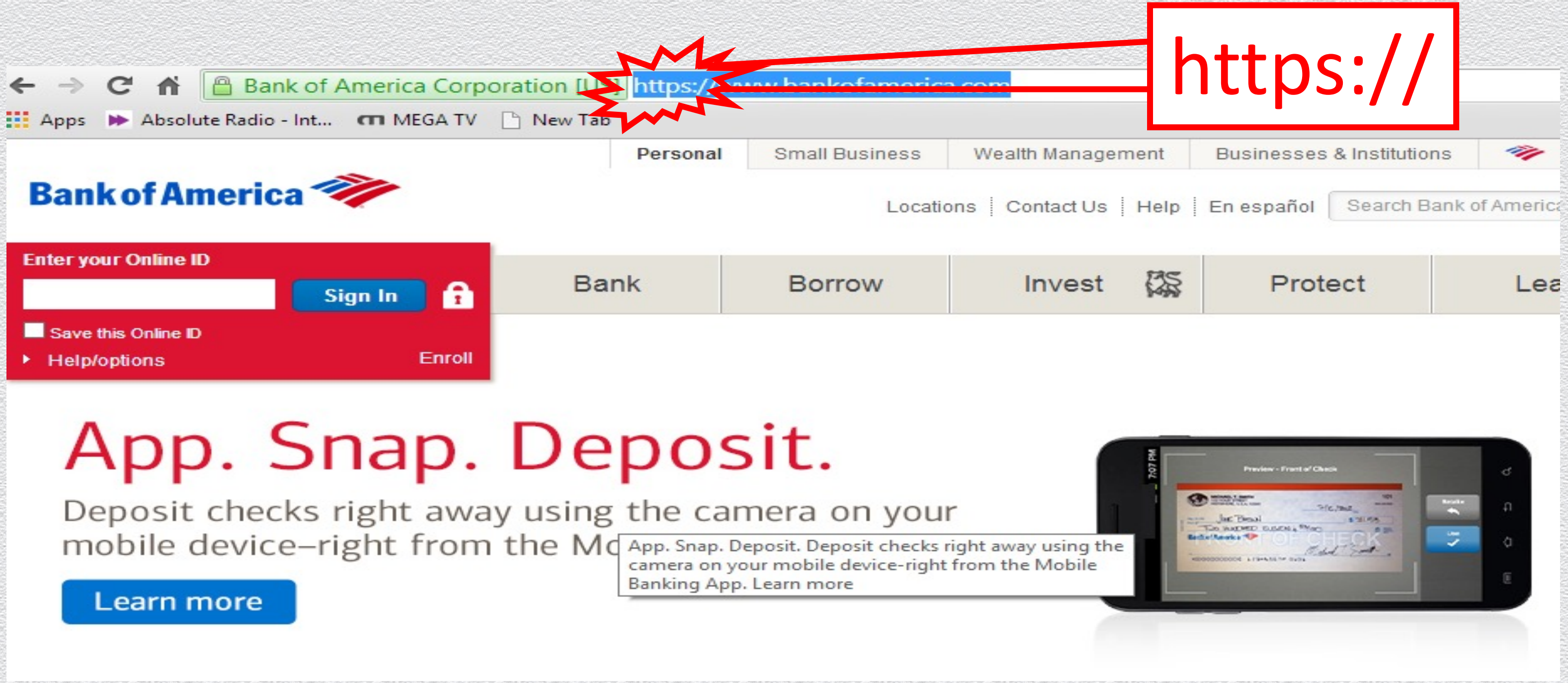
Which is Delwyn's certificate.

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 a CA

Used by numerous applications: SSL

Example: see certificates accepted by your browser

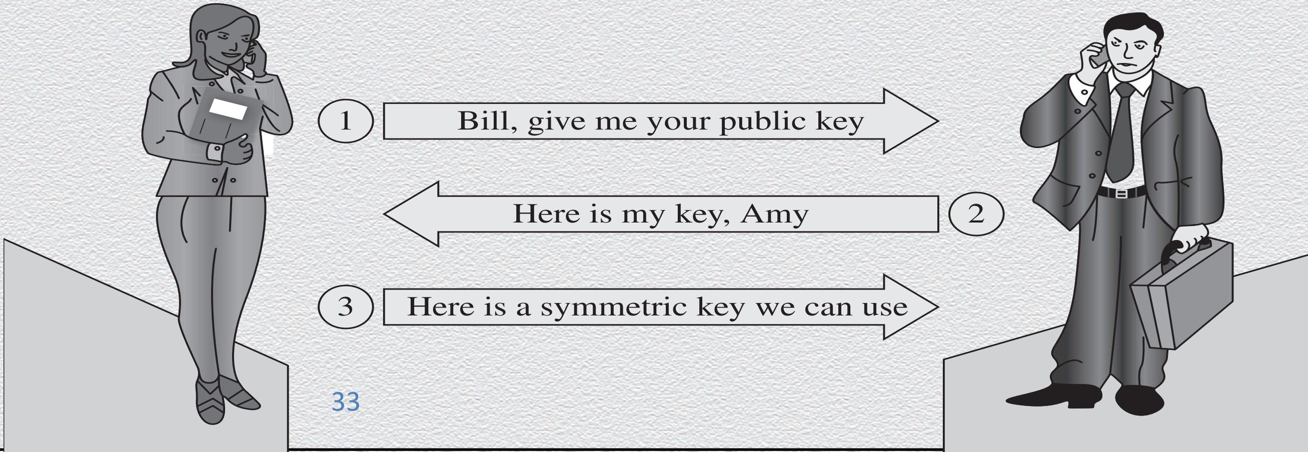
8.3 Hybrid encryption

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

PK encryption can be used "on-the-fly" to securely distribute session keys

Main idea: Leverage PK encryption to securely distribute session keys

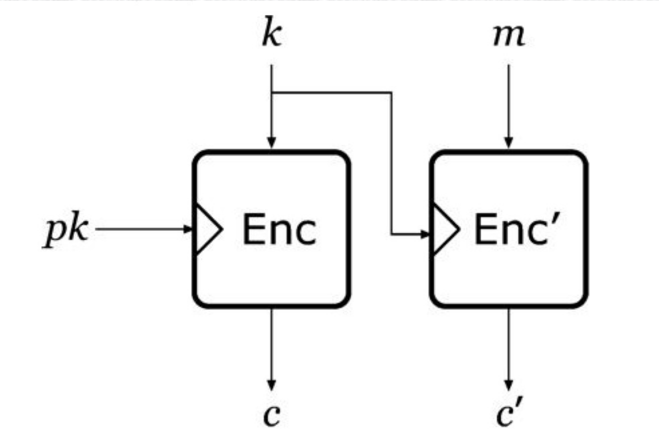
- sender generates a fresh session-specific secret key k and **learns** receiver's public key R_{pk}
- session key k is sent to receiver encrypted under key R_{pk}
- session key k is employed to run symmetric-key crypto
  - e.g., how **not** to run above protocol



Hybrid encryption

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

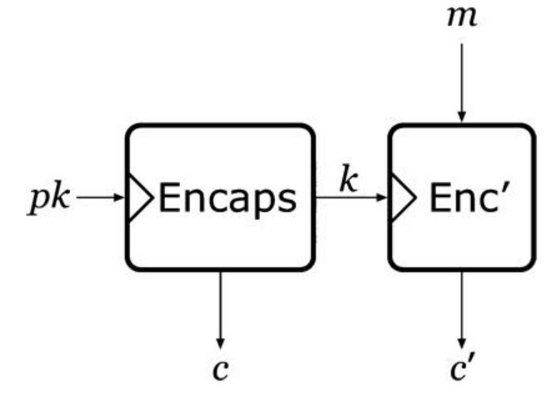
- better performance than block-based public-key CPA-encryption
- main idea
  - apply PK encryption on random key k
  - use k for secret-key encryption of m



Hybrid encryption using the KEM/DEM approach

"Reduces" secret-key crypto to public-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

8.4 Number theory

Multiplicative inverses

The residues modulo a positive integer n comprise set $Z_n = \{0,1,2,\ldots,n - 1\}$

- let x and y be two elements in Z_n such that x y mod n = 1
  - we say: y is the multiplicative inverse of x in Z_n
  - we write: $y = x^{-1}$

Theorem

An element x in Z_n has a multiplicative inverse iff x, n are relatively prime

Multiplicative inverses (cont.)

- e.g., multiplicative inverses of the residues **modulo 10** are 1, 3, 7, 9

x	0	1	2	3	4	5	6	7	8	9
x^{-1}		1		7				3		9

- e.g., multiplicative inverses of the residues **modulo 11** are all non-zero elements

x	0	1	2	3	4	5	6	7	8	9	10
x^{-1}		1	6	4	3	9	2	8	7	5	10

Computing multiplicative inverses

Fact

- given two numbers **a** and **b**, there exist integers x, y s.t.

$$\mathbf{x\ a + y\ b = gcd(a,b)}$$

which can be computed efficiently by the extended Euclidean algorithm.

Thus

- the multiplicative inverse of a in Z_b exists iff gcd(a, b) = 1
- i.e., iff the extended Euclidean algorithm computes x and y s.t. $\mathbf{x\ a + y\ b = 1}$
- in this case, the multiplicative inverse of a in Z_b is **x**

Euclidean GCD algorithm

Computes the greater common divisor
by repeatedly applying the formula

gcd(a, b) = gcd(b, a mod b)

```
Algorithm EuclidGCD(a, b)
    Input integers a and b
    Output gcd(a, b)

    if b = 0
        return a
    else
        return EuclidGCD(b, a mod b)
```

- example
  - gcd(412, 260) = 4

a	412	260	152	108	44	20	4
b	260	152	108	44	20	4	0

Extended Euclidean algorithm

Theorem

If, given positive integers **a** and **b**, **d** is the smallest positive integer s.t. **d** = **ia** + **jb**, for some integers **i** and **j**, then **d** = gcd(**a, b**)

- example
  - **a** = 21, **b** = 15
  - **d** = 3, **i** = 3, **j** = -4
  - 3 = 3·21 + (-4)·15 = 63 - 60 = 3

Algorithm Extended-Euclid(a, b)
  Input integers **a** and **b**
  Output gcd(**a**, **b**), i and j
    s.t. ia+jb = gcd(a,b)
  if b = 0
    return (a,1,0)
  (d′, x′, y′) = Extended-Euclid(b, a mod **b)**
  (d, x, y) = (d′, y′, x′ - [a/b]y′)
  return (d, x, y)

Multiplicative group

A set of elements where multiplication • is defined

- closure, associativity, identity & inverses
- multiplicative groups Z^*_n, defined w.r.t. Z_n (residues modulo n)
  - subsets of Z_n containing all integers that are relative prime to n
  - **CASE 1**: if n is a prime number, then all non-zero elements in Z_n have an inverse
    - $Z^*_7 = \{1,2,3,4,5,6\}$, n = 7
    - 2 • 4 = 1 (mod 7), 3 • 5 = 1 (mod 7), 6 • 6 = 1 (mod 7), 1 • 1 = 1 (mod 7)
  - **CASE 2**: if n is not prime, then not all integers in Z_n have an inverse
    - $Z^*_{10} = \{1,3,7,9\}$, n = 10
    - 3 • 7 = 1 (mod 10), 9 • 9 = 1 (mod 10), 1 • 1 = 1 (mod 10)

Order of a multiplicative group

Order of a group = cardinality of the group

- multiplicative groups for Z^*_n
- the totient function $\phi(n)$ denotes the order of Z^*_n , i.e., $\phi(n) = |Z^*_n|$
  - if **n = p is prime**, then the order of $Z^*_p=\{1,2,\ldots,p-1\}$ is p-1, i.e., $\phi(n) = p-1$
    - e.g., $Z^*_7 = \{1,2,3,4,5,6\}$, $n = 7$, $\phi(7) = 6$
  - if **n is not prime**, $\phi(n) = n(1-1/p_1)(1-1/p_2)\ldots(1-1/p_k)$, where $n = p^{e1}{}_1 p^{e2}{}_2 \ldots p^{ek}{}_k$
    - e.g., $Z^*_{10} = \{1,3,7,9\}$, $n = 10$, $\phi(10) = 4$

- if n = p q, where p and q are distinct primes, then $\phi(n) = (p-1)(q-1)$ **Factoring problem**
  - difficult problem: given n = pq, where p, q are primes, find p and q or $\phi(n)$

Fermat's Little Theorem

Theorem

If **p is a prime**, then for each nonzero residue x in Z_p, we have $x^{p-1} \bmod p = 1$

- example (p = 5):

  $1^4 \bmod 5 = 1$ $\quad$ $2^4 \bmod 5 = 16 \bmod 5 = 1$

  $3^4 \bmod 5 = 81 \bmod 5 = 1$ $\quad$ $4^4 \bmod 5 = 256 \bmod 5 = 1$

Corollary

If **p is a prime**, then the multiplicative inverse of each x in Z^*_p is $x^{p-2} \bmod p$

- proof: $x(x^{p-2} \bmod p) \bmod p = xx^{p-2} \bmod p = x^{p-1} \bmod p = 1$

Euler's Theorem

Theorem

For each element x in Z^*_n, we have $x^{\phi(n)}$ mod n = 1

- example (**n = 10**)
  - $Z^*_{10} = \{1,3,7,9\}$, n = 10, $\phi(10) = 4$
  - $3^{\phi(10)}$ mod 10 = 3^4 mod 10 = 81 mod 10 = 1
  - $7^{\phi(10)}$ mod 10 = 7^4 mod 10 = 2401 mod 10 = 1
  - $9^{\phi(10)}$ mod 10 = 9^4 mod 10 = 6561 mod 10 = 1

Computing in the exponent

For the multiplicative group Z^*_n, we can reduce the exponent modulo $\phi(n)$

- $x^y \bmod n = x^{k\,\phi(n) + r} \bmod n = (x^{\phi(n)})^k x^r \bmod n = x^r \bmod n = x^{y \bmod \phi(n)} \bmod n$

Corollary: For Z^*_p, we can reduce the exponent modulo p-1

- example
  - $Z^*_n = \{1,3,7,9\}$, $n = 10$, $\phi(10) = 4$
  - $3^{1590} \bmod 10 = 3^{1590 \bmod 4} \bmod 10 = 3^2 \bmod 10 = 9$
- example
  - $Z^*_p = \{1,2,\ldots,p - 1\}$, $p = 19$, $\phi(19) = 18$
  - $15^{39} \bmod 19 = 15^{39 \bmod 18} \bmod 19 = 15^3 \bmod 19 = 12$

Modular powers

Repeated squaring algorithm

Speeds up computation of $\mathbf{a^p}$ mod $\mathbf{n}$

- write the exponent $\mathbf{p}$ in binary
  $\mathbf{p} = \mathbf{p}_{\mathbf{b}-1}\,\mathbf{p}_{\mathbf{b}-2} \ldots \mathbf{p}_1\,\mathbf{p}_0$
- start with $\mathbf{Q}_1 = \mathbf{a}^{\mathbf{p}_{\mathbf{b}-1}}$ mod $\mathbf{n}$
- repeatedly compute
  $\mathbf{Q}_\mathbf{i} = ((\mathbf{Q}_{\mathbf{i}-1})^2 \bmod \mathbf{n})\mathbf{a}^{\mathbf{p}_{\mathbf{b}-\mathbf{i}}} \bmod \mathbf{n}$
- obtain $\mathbf{Q_b} = \mathbf{a^p}$ mod $\mathbf{n}$

Total **O** (log **p**) arithmetic operations

Example

- 3^{18} mod 19 (18 = 10010)
- $\mathbf{Q}_1 = 3^1$ mod 19 = 3
- $\mathbf{Q}_2 = (3^2$ mod 19$)3^0$ mod 19 = 9
- $\mathbf{Q}_3 = (9^2$ mod 19$)3^0$ mod 19 = 81 mod 19 = 5
- $\mathbf{Q}_4 = (5^2$ mod 19$)3^1$ mod 19 =
  (25 mod 19)3 mod 19 = 18 mod 19 = 18
- $\mathbf{Q}_5 = (18^2$ mod 19$)3^0$ mod 19 = (324 mod 19) mod 19 = 17·19 + 1 mod 19 = 1

Powers

Let p be a prime

- the sequences of successive powers of the elements in Z^*_p exhibit repeating subsequences
- the sizes of the repeating subsequences and the number of their repetitions are the divisors of p – 1
- example, p = 7

x	x^2	x^3	x^4	x^5	x^6
1	1	1	1	1	1
2	4	1	2	4	1
3	2	6	4	5	1
4	2	1	4	2	1
5	4	6	2	3	1
6	1	6	1	6	1

8.5 The Discrete Log problem & its applications

The discrete logarithm problem

Setting

- if p be an odd prime, then G = (Z_p^*, ·) is a cyclic group of order p – 1
  - Z_p^* = {1, 2, 3, …, p-1}, generated by some g in Z_p^*
    - for i = 0, 1, 2, …, p-2, the process **g^i mod p** produces all elements in Z_p^*
  - for any x in the group , we have that **g^k mod p = x**, for some integer k
  - k is called the **discrete logarithm** (or log) of x (mod p)

Example

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

Computational assumption

Discrete-log setting

- cyclic $G = (Z_p^*, \cdot)$ of order $p - 1$ generated by g, prime p of length t ($|p|=t$)

Problem

- given G, g, p and x in Z_p^*, compute the discrete log k of x (mod p)
- we know that $x = g^k \bmod p$ for some unique k in {0, 1, …, p-2}… but

Discrete log assumption

- for groups of specific structure, **solving the discrete log problem is infeasible**
- any efficient algorithm finds discrete logs negligibly often (prob = $2^{-t/2}$)

Brute force attack

- cleverly enumerate and **check $O(2^{t/2})$ solutions**

ElGamal encryption

Assumes discrete-log setting (cyclic $G = (Z_p^*, \cdot) = $ <g>, prime p, message space Z_p)

Gen

- secret key: random number $x \in Z^*_p$ public key: $A = g^x \bmod p$, along w/ G, g, p

Enc

- pick a fresh random $r \in Z^*_p$ and set $R = A^r$ $(= g^{xr})$
- send ciphertext $\mathbf{Enc_{PK}(m) = (c_1, c_2)}$ where $\mathbf{c_1 = g^r}$, $\mathbf{c_2 = m \cdot R \bmod p}$

Dec

- $\mathbf{Dec_{SK}(c_1,c_2) = c_2\,(1/c_1^x) \bmod p}$ where $\mathbf{c_1^x = g^{xr}}$

Security is based on **Computational Diffie-Hellman** (CDH) assumption

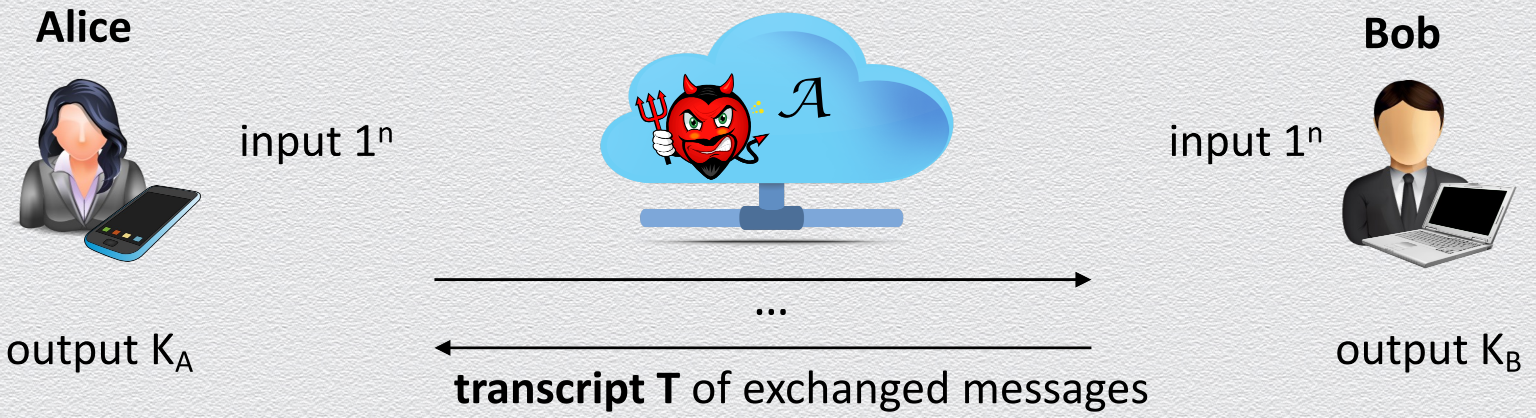
- given (g, g^a, g^b) it is hard to compute g^{ab}

A signature scheme can be also derived based on above discussion

Application: Key-agreement (KA) scheme

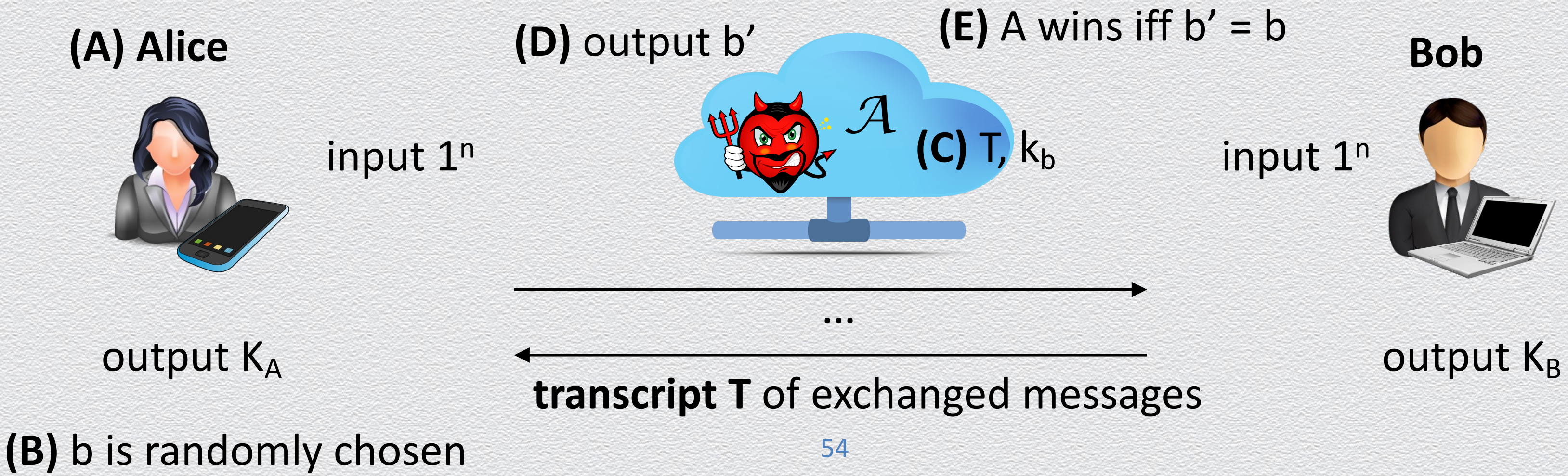
Alice and Bob want to securely establish a **shared key** for secure chatting over an **insecure** line

- instead of meeting in person in a secret place, they want to use the insecure line…
- KA scheme: they run a key-agreement protocol Π to contribute to a shared key K
- correctness: $K_A = K_B$
- security: no PPT adversary $\mathcal{A}$, given T, can distinguish K from a trully random one



Key agreement: Game-based security definition

- scheme $\Pi(1^n)$ runs to generate $K = K_A = K_B$ and transcript T; random bit b is chosen
- adversary $\mathcal{A}$ is given T and k_b; if b = 1, then k_b = K, else k_b is random (both n-bit long)
- $\mathcal{A}$ outputs bit b' and wins if b' = b
- then: **Π is secure if no PPT A wins non-negligibly often**



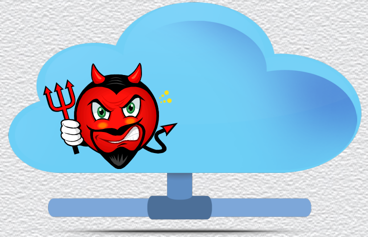
The Diffie-Hellman key-agreement protocol

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

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

Alice

input 1^n



Bob

input 1^n

(1) randomly pick secret a

(3) send g^a mod p →

(2) randomly pick secret b

← **(4)** send g^b mod p

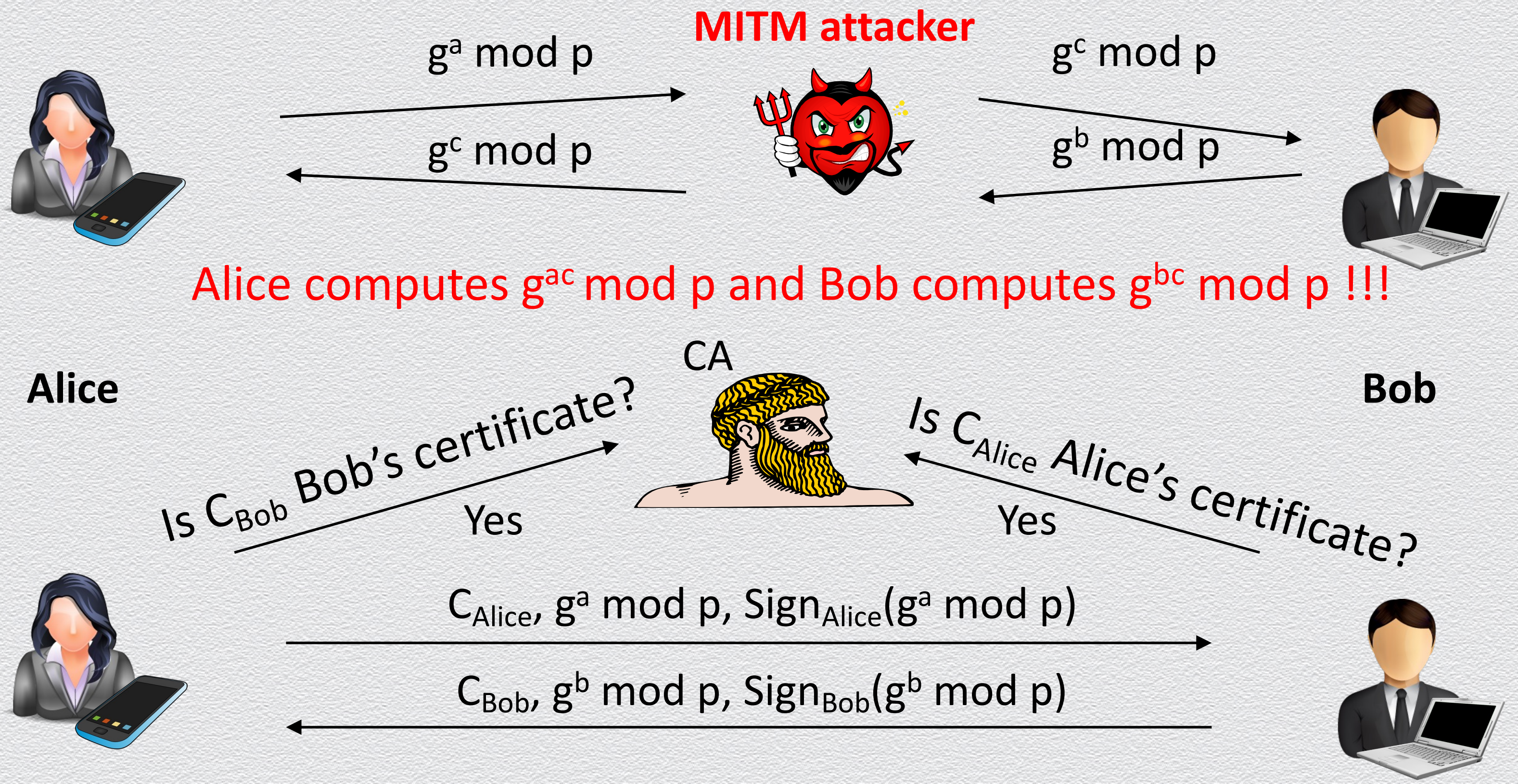
(5) set **K = g^{ab} mod p** = (g^b mod p)^a mod p

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



8.6 The RSA algorithm

The RSA algorithm (for encryption)

General case

Setup (run by a given user)

- $\mathbf{n} = \mathbf{p} \cdot \mathbf{q}$, with **p** and **q** primes
- **e** relatively prime to $\boldsymbol{\phi}(n) = (\mathbf{p} - 1)(\mathbf{q} - 1)$
- **d** inverse of **e** in $\mathbf{Z}_{\boldsymbol{\phi}(\mathbf{n})}$

Keys

- public key is $\mathbf{K_{PK}} = (\mathbf{n}, \mathbf{e})$
- private key is $\mathbf{K_{SK}} = \mathbf{d}$

Encryption

- $\mathbf{C} = \mathbf{M}^e \bmod \mathbf{n}$ for plaintext **M** in $\mathbf{Z_n}$

Decryption

- $\mathbf{M} = \mathbf{C}^d \bmod \mathbf{n}$

Example

Setup

- $\mathbf{p} = 7$, $\mathbf{q} = 17$, $\mathbf{n} = 7 \cdot 17 = 119$
- $\mathbf{e} = 5$, $\boldsymbol{\phi}(\mathbf{n}) = 6 \cdot 16 = 96$
- $\mathbf{d} = 77$

Keys

- public key is (119, 5)
- private key is 77

Encryption

- $\mathbf{C} = 19^5 \bmod 119 = 66$ for $\mathbf{M} = 19$ in $\mathbf{Z_{119}}$

Decryption

- $\mathbf{M} = 66^{77} \bmod 119 = 19$

Another complete example

- Setup
  - **p** = 5, **q** = 11, **n** = 5 · 11 = 55
  - **ϕ**(**n**) = 4 · 10 = 40
  - **e** = 3, **d** = 27 (3·27 = 81 = 2·40 + 1)
- Encryption
  - **C** = **M**^{3} mod 55 for **M** in $\mathbf{Z}_{55}$
- Decryption
  - **M** = **C**^{27} mod 55

M	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
C	1	8	27	9	15	51	13	17	14	10	11	23	52	49	20	26	18	2
M	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
C	39	25	21	33	12	19	5	31	48	7	24	50	36	43	22	34	30	16
M	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54
C	53	37	29	35	6	3	32	44	45	41	38	42	4	40	46	28	47	54

Correctness of RSA

Given

Setup

- $\mathbf{n} = \mathbf{p} \cdot \mathbf{q}$, with **p** and **q** primes
- **e** relatively prime to $\phi(n) = (\mathbf{p} - 1)(\mathbf{q} - 1)$
- **d** inverse of **e** in $\mathbf{Z}_{\phi(n)}$ **(1)**

Encryption

- $\mathbf{C} = \mathbf{M}^e \bmod \mathbf{n}$ for plaintext **M** in $\mathbf{Z}_n$

Decryption

- $\mathbf{M} = \mathbf{C}^d \bmod \mathbf{n}$

Fermat's Little Theorem **(2)**

- for prime p, non-zero x: $x^{p-1} \bmod p = 1$

Analysis

Need to show

- $M^{ed} = M \bmod p \cdot q$

Use **(1)** and apply **(2)** for prime p

- $M^{ed} = M^{ed-1} M = (M^{p-1})^{h(q-1)} M$
- $\mathbf{M^{ed}} = 1^{h(q-1)} M \bmod p = \mathbf{M \bmod p}$

Similarly (w.r.t. prime q)

- $\mathbf{M^{ed} = M \bmod q}$

Thus, since p, q are co-primes

- $\mathbf{M^{ed} = M \bmod p \cdot q}$

A useful symmetry

[1] RSA setting

- modulo $\mathbf{n} = \mathbf{p} \cdot \mathbf{q}$, p & q are **primes**, public & private keys (e,d): $\mathbf{d \cdot e = 1 \bmod (p-1)(q-1)}$

[2] RSA operations involve **exponentiations**, thus they are **interchangeable**

- $\mathbf{C} = \mathbf{M^e} \bmod \mathbf{n}$ (encryption of plaintext **M in** $\mathbf{Z_n}$)
- $\mathbf{M} = \mathbf{C^d} \bmod \mathbf{n}$ (decryption of ciphertext **C in** $\mathbf{Z_n}$)

Indeed, their order of execution does not matter: $(M^e)^d = (M^d)^e \bmod n$

[3] RSA operations involve exponents that **"cancel out",** thus they are **complementary**

- $x^{(p-1)(q-1)} \bmod \mathbf{n} = \mathbf{1}$ (Euler's Theorem)

Indeed, they invert each other: $(M^e)^d = (M^d)^e = M^{ed} = M^{k(p-1)(q-1)+1} \bmod n$

$= (M^{(p-1)(q-1)})^k \cdot M = 1^k \cdot M = M \bmod n$

Signing with RSA

RSA functions are complementary & interchangeable w.r.t. order of execution

- **core property**: $M^{ed} = M \bmod p \cdot q$ for any message **M in** Z_n

RSA cryptosystem lends itself to a **signature scheme**

- 'reverse' use of keys is possible : $(M^d)^e = M \bmod p \cdot q$
- signing algorithm **Sign(M,d,n)**: $\sigma = M^d \bmod n$ for message **M** in Z_n
- verifying algorithm **Vrfy(σ,M,e,n)**: return $M == \sigma^e \bmod n$

The RSA algorithm (for signing)

General case

Setup (run by a given user)

- $\mathbf{n} = \mathbf{p} \cdot \mathbf{q}$, with **p** and **q** primes
- **e** relatively prime to $\boldsymbol{\phi}(n) = (\mathbf{p} - 1)(\mathbf{q} - 1)$
- **d** inverse of **e** in $\mathbf{Z}_{\boldsymbol{\phi}(\mathbf{n})}$

Keys (same as in encryption)

- public key is $\mathbf{K_{PK}} = (\mathbf{n}, \mathbf{e})$
- private key is $\mathbf{K_{SK}} = \mathbf{d}$

Sign

- $\boldsymbol{\sigma} = \mathbf{M}^{\mathbf{d}}$ mod **n** for message **M** in $\mathbf{Z_n}$

Verify

- Check if $\mathbf{M} = \boldsymbol{\sigma}^{\mathbf{e}}$ mod **n**

Example

Setup

- $\mathbf{p} = 7$, $\mathbf{q} = 17$, $\mathbf{n} = 7 \cdot 17 = 119$
- $\mathbf{e} = 5$, $\boldsymbol{\phi}(\mathbf{n}) = 6 \cdot 16 = 96$
- $\mathbf{d} = 77$

Keys

- public key is (119, 5)
- private key is 77

Signing

- $\boldsymbol{\sigma} = 66^{77}$ mod 119 = 19 for $\mathbf{M} = 66$ in $\mathbf{Z_{119}}$

Verification

- Check if $\mathbf{M} = 19^5$ mod 119 = 66

Digital signatures & hashing

Very often digital signatures are used with hash functions

- the hash of a message is signed, instead of the message itself

Signing message M

- let h be a cryptographic hash function, assume RSA setting (n, d, e)
- compute signature σ on message M as: $\sigma = h(M)^d \bmod n$
- send σ, M

Verifying signature σ

- use public key (e, n) to compute (candidate) hash value $H = \sigma^e \bmod n$
- if H = h(M) output ACCEPT, else output REJECT

Security of RSA

Based on difficulty of **factoring** large numbers (into large primes), i.e., n = p · q into p, q

- note that for RSA to be secure, both p and q must be large primes
- widely believed to hold true
  - since 1978, subject of extensive cryptanalysis without any serious flaws found
  - best known algorithm takes exponential time in security parameter (key length |n|)
- how can you break RSA if you can factor?

Current practice is using 2,048-bit long RSA keys (617 decimal digits)

- estimated computing/memory resources needed to factor an RSA number within one year

Length (bits)	PCs	Memory
430	1	128MB
760	215,000	4GB
1,020	342×10^{6}	170GB
1,620	1.6×10^{15}	120TB

RSA challenges

Challenges for breaking the RSA cryptosystem of various key lengths (i.e., |n|)

- known in the form RSA-`key bit length' expressed in bits or decimal digits
- provide empirical evidence/confidence on strength of specific RSA instantiations

Known attacks

- RSA-155 (**512-bit**) factored in **4 mo**. using 35.7 CPU-years or 8000 Mips-years (**1999**) and 292 machines
  - 160 175-400MHz SGI/Sun, 8 250MHz SGI/Origin, 120 300-450MHz Pent. II, 4 500MHz Digital/Compaq
- RSA-**640** factored in **5 mo**. using 30 2.2GHz CPU-years (**2005**)
- RSA-220 (**729-bit**) factored in **5 mo**. using 30 2.2GHz CPU-years (**2005**)
- RSA-232 (**768-bit**) factored in **2 years** using **parallel** computers 2K CPU-years (1-core 2.2GHz AMD Opteron) (**2009**)

Most interesting challenges

- prizes for factoring RSA-**1024**, RSA-**2048** is $100K, $200K – estimated at 800K, 20B Mips-centuries

Deriving an RSA key pair

- public key is pair of integers (e,n), secret key is (d, n) or d
- the value of n should be quite large, a product of two large primes, p and q
- often p, q are nearly 100 digits each, so n ~= 200 decimal digits (~512 bits)
  - but 2048-bit keys are becoming a standard requirement nowadays
- the larger the value of n the harder to factor to infer p and q
  - but also the slower to process messages
- a relatively large integer e is chosen
  - e.g., by choosing e as a prime that is larger than both $(p - 1)$ and $(q - 1)$
  - why?
- d is chosen s.t. $e \cdot d = 1 \bmod (p - 1)(q - 1)$
  - how?

Discussion on RSA

- Assume **p** = 5, **q** = 11, **n** = 5 · 11 = 55, **ϕ(n)** = 40, **e** = 3, **d** = 27
  - why encrypting small messages, e.g., **M** = 2, 3, 4 is tricky?
  - recall that the ciphertext is $\mathbf{C} = \mathbf{M}^3 \bmod 55$ for **M** in $\mathbf{Z_{55}}$

M	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
C	1	8	27	9	15	51	13	17	14	10	11	23	52	49	20	26	18	2
M	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
C	39	25	21	33	12	19	5	31	48	7	24	50	36	43	22	34	30	16
M	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54
C	53	37	29	35	6	3	32	44	45	41	38	42	4	40	46	28	47	54

Discussion on RSA

- Assume $\mathbf{p} = 5$, $\mathbf{q} = 11$, $\mathbf{n} = 5 \cdot 11 = 55$, $\boldsymbol{\phi}(\mathbf{n}) = 40$, $\mathbf{e} = 3$, $\mathbf{d} = 27$
  - why encrypting small messages, e.g., $\mathbf{M} = 2, 3, 4$ is tricky?
  - recall that the ciphertext is $\mathbf{C} = \mathbf{M}^3 \bmod 55$ for $\mathbf{M}$ in $\mathbf{Z}_{55}$
- Assume n = 204343943843555343435454289434834343356091 = p · q
  - can e be the number 4343253453434536?
- Are there problems with applying RSA in practice?
  - what other algorithms are required to be available to the user?
- Are there problem with respect to RSA security?
  - does it satisfy CPA (advanced) security?

Algorithmic issues

The implementation of the RSA cryptosystem requires various algorithms

- Main issues
  - representation of integers of arbitrarily large size; and
  - arithmetic operations on them, namely computing modular powers
- Required algorithms (at setup)
  - generation of **random numbers** of a given number of bits (to compute candidates **p**, **q**)
  - **primality testing** (to check that candidates **p**, **q** are prime)
  - computation of the **GCD** (to verify that **e** and **ϕ(n)** are relatively prime)
  - computation of the **multiplicative inverse** (to compute **d** from **e**)

Pseudo-primality testing

Testing whether a number is prime (**primality testing**) is a difficult problem

An integer $\mathbf{n} \geq 2$ is said to be a base-**x** **pseudo-prime** if

- $\mathbf{x}^{\mathbf{n}-1} \bmod \mathbf{n} = 1$ (Fermat's little theorem)

- Composite base-**x** pseudo-primes are rare
  - a random 100-bit integer is a composite base-2 pseudo-prime with probability less than 10^{-13}
  - the smallest composite base-2 pseudo-prime is 341
- Base-**x** pseudo-primality testing for an integer **n**
  - check whether $\mathbf{x}^{\mathbf{n}-1} \bmod \mathbf{n} = 1$
  - can be performed efficiently with the repeated squaring algorithm

Security properties

- Plain RSA is deterministic
  - why is this a problem?
- Plain RSA is also homomorphic
  - what does this mean?
  - multiply ciphertexts to get ciphertext of multiplication!
  - $[(m_1)^e \bmod N][(m_2)^e \bmod N] = (m_1 m_2)^e \bmod N$
  - however, not additively homomorphic

Real-world usage of RSA

- Randomized RSA
  - to encrypt message M under an RSA public key (e,n), generate a new random session AES key K, compute the ciphertext as $[K^e \bmod n, AES_K(M)]$
  - prevents an adversary distinguishing two encryptions of the same M since K is chosen at random every time encryption takes place
- Optimal Asymmetric Encryption Padding (OAEP)
  - roughly, to encrypt M, choose random r, encode M as $M' = [X = M \oplus H_1(r)\ ,\ Y = r \oplus H_2(X)\ ]$ where H_1 and H_2 are cryptographic hash functions, then encrypt it as $(M')^e \bmod n$

Summary of message-authentication crypto tools

	Hash (SHA2-256)	MAC	Digital signature
Integrity	Yes	Yes	Yes
Authentication	No	Yes	Yes
Non-repudiation	No	No	Yes
Crypto system	None	Symmetric (AES)	Asymmetric (e.g., RSA)