https://brown-csci1660.github.io

CS1660: Intro to Computer Systems Security Spring 2025

Lecture 5: Cryptography IV

Co-Instructor: **Nikos Triandopoulos**

February 6, 2025

BROWN

CS1660: Announcements

- Override requests
  - Status update
- Course updates
  - Homework 1, Project 1 have new submission dates
    - To provide more time & better preparation
    - To avoid possible confusion due to specific order/pace of topic coverage
  - Future assignment dates may be updated as well/accordingly
  - Ed Discussion, Top Hat (code: **821033**), Gradescope (set up for Project 1)

updated

Today

- Cryptography
    - Message authentication codes (MACs)
    - Authenticated encryption
    - Public-key encryption and digital signatures (introduction)

5.1 Message authentication

Recall: Integrity

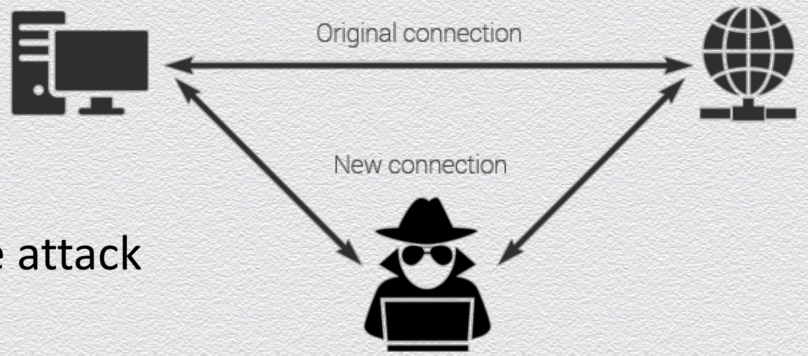
Fundamental security property

- **an asset is modified only by authorized parties**
- "I" in the CIA triad

> *"computer security seeks to prevent* ***unauthorized*** *viewing (confidentiality) or* ***modification (integrity)*** *of* ***data*** *while preserving access (availability)"*

Alteration

- main threat against integrity of **in-transit** data
- e.g., Attacker-In-The-Middle attack



Man in the middle, Phisher, or annonymous proxy

Security problems studied by modern cryptography

- Classical cryptography: **message encryption**
  - early crypto schemes tried to provide **secrecy / confidentiality**

- Modern cryptography: **wide variety** of security problems
  - today we need to study a large set of **security properties** beyond secrecy

- The sibling of message encryption: **message authentication**
  - **another cornerstone** of any secure system aiming to provide **authenticity & integrity**

Message authentication: Motivation

Information has **value**, but only when it is **correct**

- random, incorrect, inaccurate or maliciously altered data is **useless** or **harmful**
  - **message authentication = message integrity + authenticity**
    - while in transit (or at rest), no message should be **modified** by an outsider
    - no outsider can **impersonate** the stated message sender (or owner)

- it is often necessary / worth to protect critical / valuable data
  - **message encryption**
    - while in transit (or at rest), no message should be **leaked** to an outsider

Example 1

Secure electronic banking

- a bank receives an electronic request to transfer $1,000 from Alice to Bob

Concerns

- who ordered the transfer, Alice or an attacker (e.g., Bob)?
- is the amount the intended one or was maliciously modified while in transit?
  - adversarial Vs. random message-transmission errors
    - standard error-correction is **<u>not sufficient</u>** to address this concern

Example 2

Web browser cookies

- a user is performing an online purchase at Amazon
- a “cookie” contains session-related info, as client-server HTTP traffic is stateless
  - stored at the client, included in messages sent to server
  - contains client-specific info that affects the transaction
    - e.g., the user’s shopping cart along with a discount due to a coupon

Concern

- was such state maliciously altered by the client (possibly harming the server)?

Integrity of communications / computations

Highly important

- any unprotected system cannot be assumed to be trustworthy w.r.t.
  - origin/source of information (due to impersonation attacks, phishing, etc.)
  - contents of information (due to man-in-the-middle attacks, email spam, etc.)
  - overall system functionality

Prevention Vs. detection

- unless system is “closed,” adversarial tampering with its integrity **cannot be avoided**!
- goal: identify system components that are not trustworthy
  - **detect tampering** or **prevent undetected tampering**
    - e.g., avoid “consuming” falsified information

Encryption does not imply authentication

A common misconception

"since ciphertext c hides message m, Mallory cannot meaningfully modify m via c"

Why is this incorrect?

- all encryption schemes (seen so far) are based on one-time pad, i.e., masking via XOR
- consider flipping a single bit of ciphertext c; what happens to plaintext m?
    - such property of one-time pad does not contradict the secrecy definitions

Generally, secrecy and integrity are distinct properties

- encrypted traffic generally provides **no integrity** guarantees

5.2 Message authentication codes (MACs)

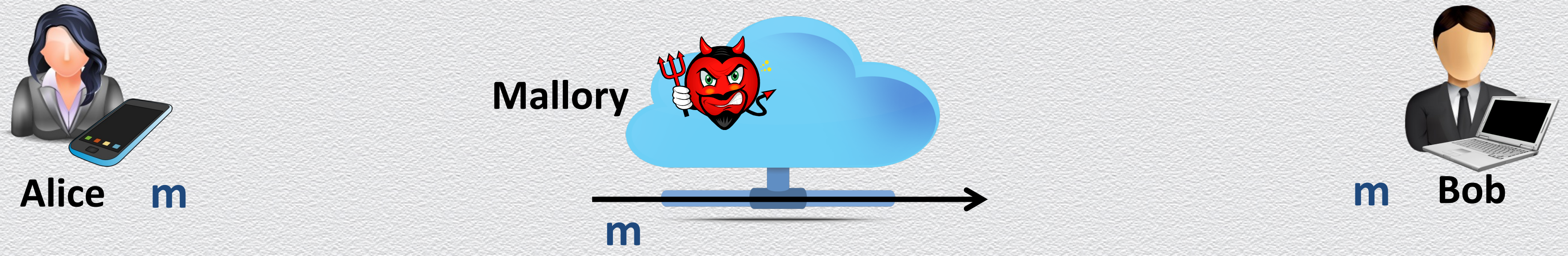
Problem setting: Reliable communication

Two parties wish to communicate over a channel

- Alice (sender/source) wants to send a message m to Bob (recipient/destination)

Underlying channel is unprotected

- Mallory (attacker/adversary) can manipulate any sent messages
- e.g., message transmission via a compromised router



Solution concept: Symmetric-key message authentication

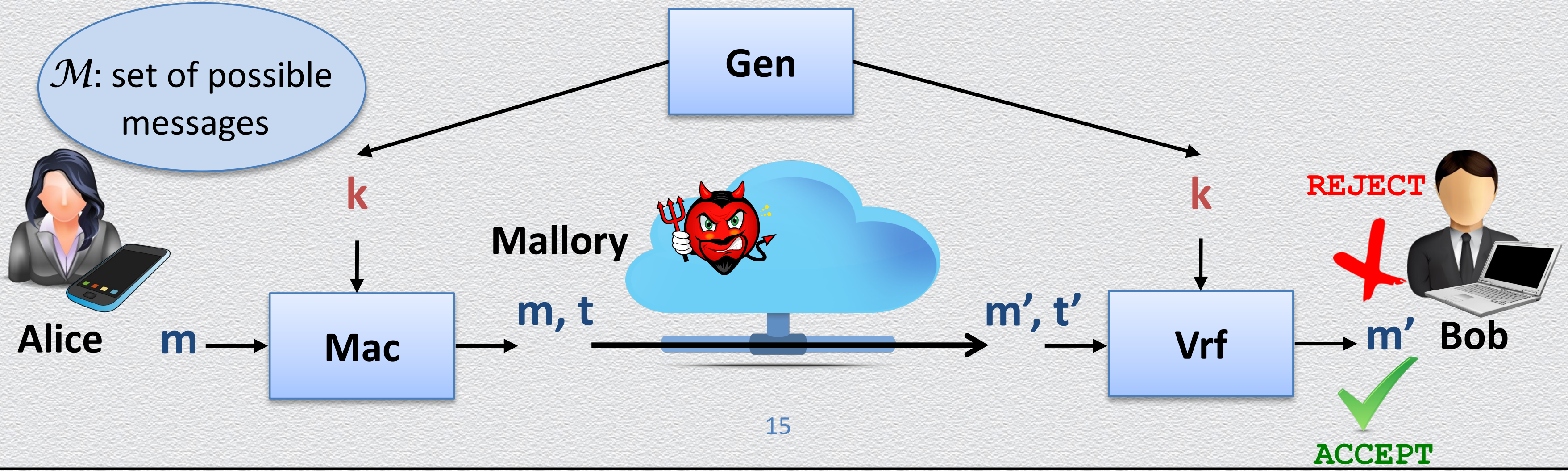
Main idea

- secretly annotate or "sign" message so that it is **unforgeable** while in transit
  - Alice **tags** her message m with **tag t**, which is sent **along** with **plaintext m**
  - Bob **verifies** authenticity of received message using tag t
  - Mallory can manipulate m, t but "**cannot forge**" a fake verifiable pair m', t'
  - Alice and Bob share a **secret key k** that is used for both operations

Security tool: Message Authentication Code

Abstract cryptographic primitive, **a.k.a. MAC**, defined by

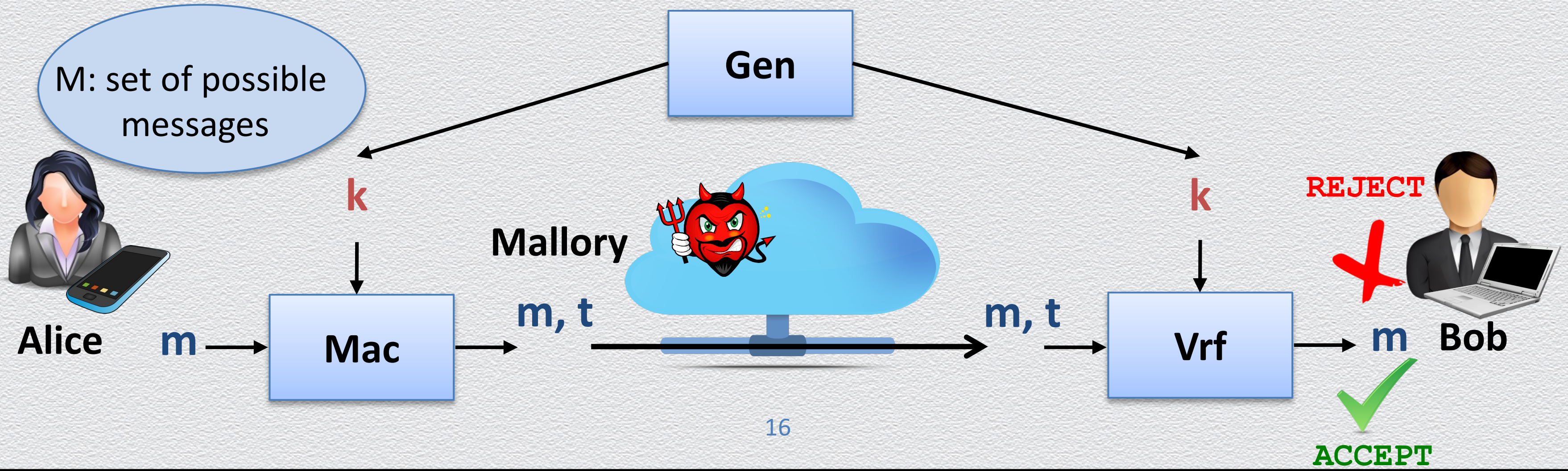
- a **message space** $\mathcal{M}$; and
- a triplet of algorithms **(Gen, Mac, Vrf)**
  - Gen, Mac are probabilistic algorithms, whereas Vrf is deterministic
  - Gen outputs a uniformly random key k (from some key space $\mathcal{K}$)



Desired properties for MACs

By design, any MAC should satisfy the following

- **efficiency**: key generation & message transformations "are fast"
- **correctness**: for all m and k, it holds that $Vrf_k(m, Mac_k(m))$ = ACCEPT
- **security**: one "cannot forge" a fake verifiable pair m', t'



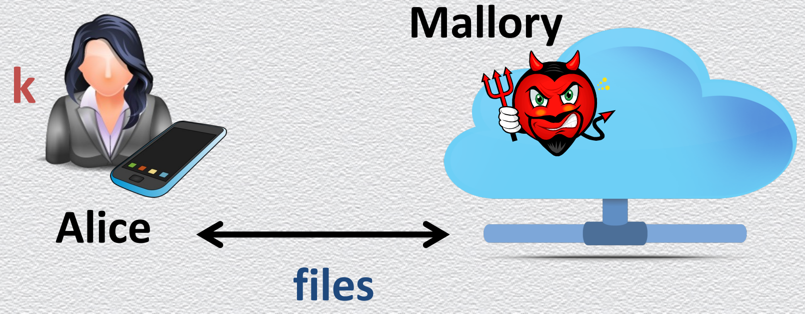
Main application areas

Secure communication

- **verify authenticity of messages** sent among parties
- assumption
  - Alice and Bob **securely generate, distribute and store shared key k**
  - attacker does not learn key k

Secure storage

- **verify authenticity of files** outsourced to the cloud
- assumption
  - Alice **securely generates and stores key k**
  - attacker does not learn key k



Conventions

Random key selection

- typically, Gen selects key k **uniformly at random** from the key space $\mathcal{K}$

Canonical verification

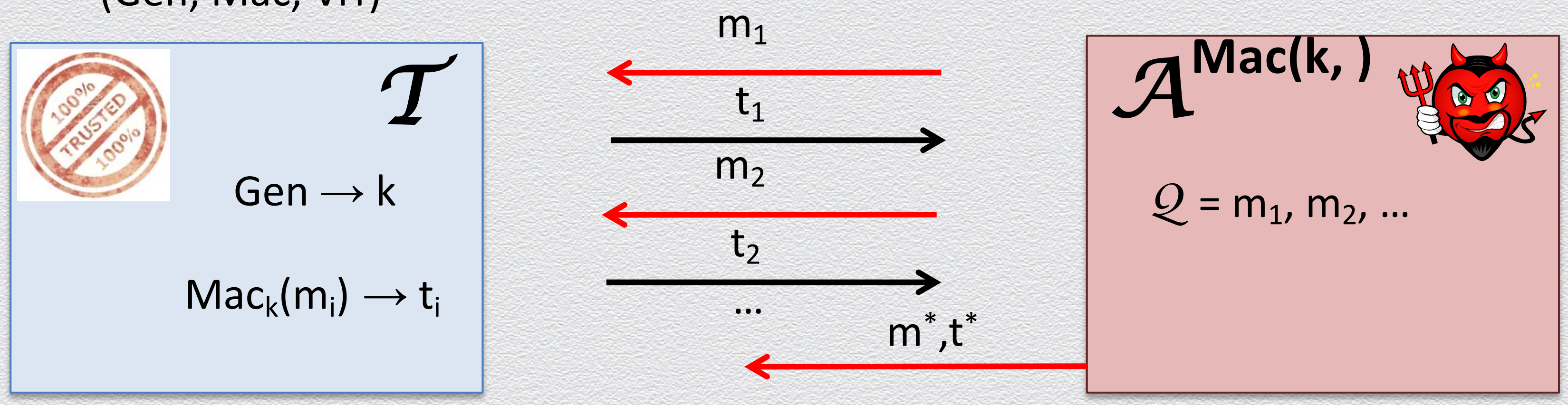
- when Mac is deterministic, Vrf typically amounts to re-computing the tag t
  - $Vrf_k(m, t)$: 1. $t' := Mac_k(m)$ 2. if $t = t'$, output `ACCEPT` else output `REJECT`
- but conceptually the following operations are distinct
  - authenticating m (i.e., running Mac) Vs. verifying authenticity of m (i.e., running Vrf)

MAC security

MAC scheme
(Gen, Mac, Vrf)

Attacker **wins** the game if

1. $Vrf_k(m^*,t^*)$ = ACCEPT &
2. m^* not in $\mathcal{Q}$



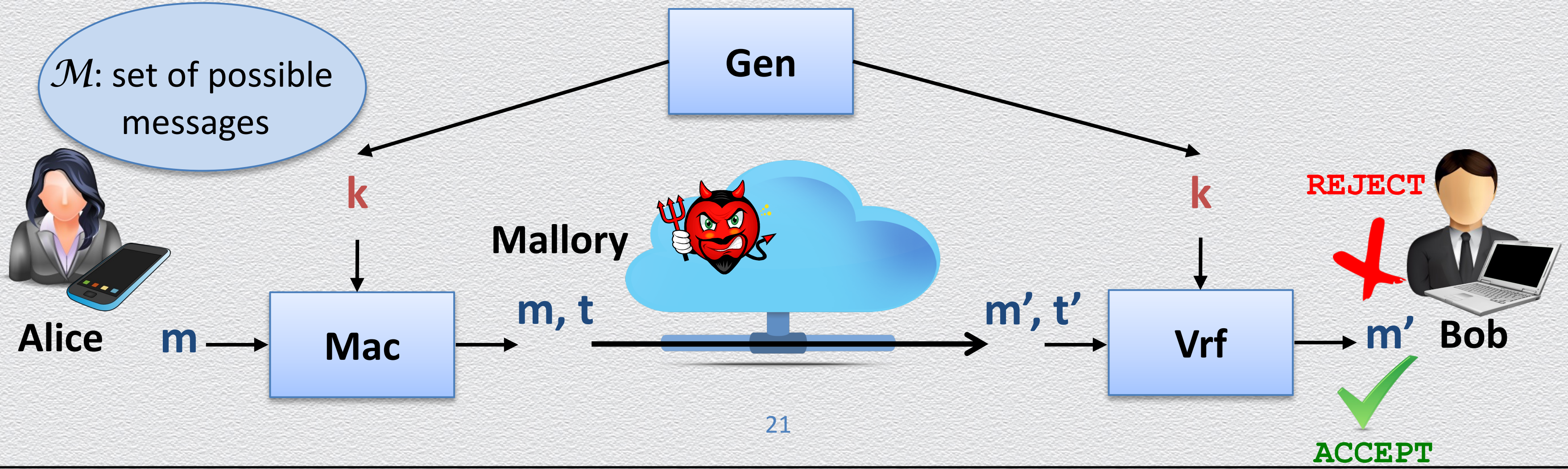
The MAC scheme is **secure** if any PPT $\mathcal{A}$ wins the game only negligibly often.

5.2.1 Replay attacks

Recall: MAC

Abstract cryptographic primitive, **a.k.a. MAC**, defined by

- a **message space** $\mathcal{M}$; and
- a triplet of algorithms **(Gen, Mac, Vrf)**

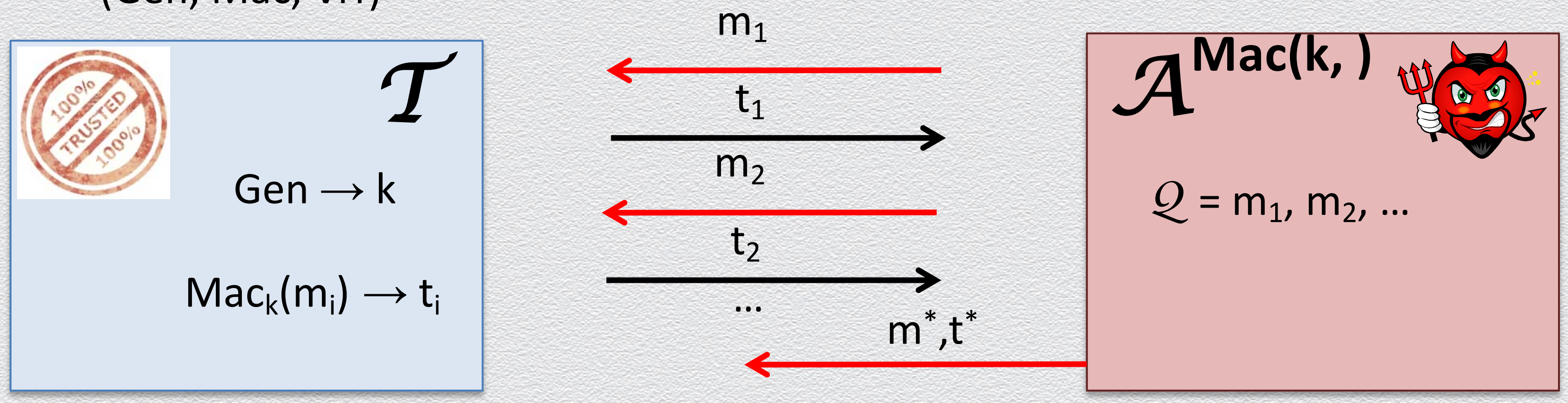


Recall: MAC security

MAC scheme
(Gen, Mac, Vrf)

Attacker **wins** the game if

1. $\mathrm{Vrf}_k(m^*, t^*) = \texttt{ACCEPT}$ &
2. m^* not in $\mathcal{Q}$



The MAC scheme is **secure** if any PPT $\mathcal{A}$ wins the game only negligibly often.

Real-life attacker

In practice, an attacker may

- observe a traffic of authenticated (and successfully verified) messages
- manipulate (or often also partially influences) traffic
  - aims at inserting an invalid but verifiable message m^*, t^* into the traffic
    - interesting case: forged message is a <u>new</u> (unseen) one
    - trivial case: forged message is a <u>previously observed</u> one, a.k.a. a **replay attack**
- launch a **brute-force attack** (given that $Mac_k(m) \rightarrow t$ is publicly known)
  - given any observed pair m, t, exhaustively search key space to find the used key k

Threat model

In the security game, Mallory is an adversary $\mathcal{A}$ who is

- "active" (on the wire)
  - we allow $\mathcal{A}$ to **observe** and **manipulate** sent messages
- "well-informed"
  - we allow $\mathcal{A}$ to **request MAC tags** of messages of **its choice**
- "replay-attack safe"
  - we restrict $\mathcal{A}$ to **forge only new** messages
- "PPT"
  - we restrict $\mathcal{A}$ to be **computationally bounded**
  - new messages may be forged undetectably only **negligibly** often

Notes on security definition

Is it a rather strong security definition?

- we allow $\mathcal{A}$ to **query MAC tags for any message**
  - but real-world senders will authenticate only “meaningful” messages
- we allow $\mathcal{A}$ to break the scheme by **forging any new message**
  - but real-world attackers will forge only “meaningful” messages

Yes, it is the right approach…

- message “**meaningfulness” depends on higher-level application**
  - text messaging apps require authentication of English-text messages
  - other apps may require authentication of binary files
  - security definition should better be **agnostic** of the specific higher application

Notes on security definition (II)

Are replay attacks important in practice?

- absolutely yes: a **very realistic & serious threat!**
  - e.g., what if a money transfer order is “replayed”?

Yet, a “replay-attack safe” security definition is preferable

- again, whether replayed messages are valid depends on higher-lever app
- better to delegate to this app the specification of such details
  - e.g., semantics on traffic or validity checks on messages before they’re “consumed”

Eliminating replay attacks

- use of counters (i.e., common shared state) between sender & receiver
- use of timestamps along with a (relaxed) authentication window for validation

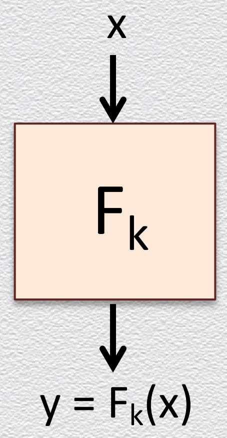
5.2.2 MAC constructions

Three generic MAC constructions

- fixed-length MAC
  - direct application of a PRF for tagging
  - limited applicability
- domain extension for MACs
  - straightforward secure extension of fix-length MAC
  - inefficient
- CBC-MAC
  - resembles CBC-mode encryption
  - efficient

1. Fixed-length MAC

- based on use of a PRF
  - employ a PRF F_k in the obvious way to compute and canonically verify tags
  - set tag t to be the pseudorandom string derived by evaluating F_k on message m
- secure, provided that F_k is a secure PRF



<u>MAC scheme Π</u>

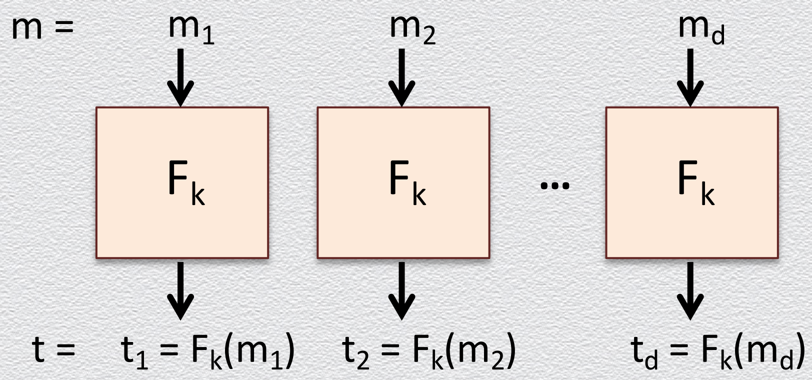
$Gen(1^n)$: $\{0,1\}^n \longrightarrow k$

$Mac_k(m)$: set $t = F_k(m)$

$Vrfy_k(m,t)$: return 1 iff $t = F_k(m)$

2. Domain extension for MACs (I)

- suppose we have the previous fix-length MAC scheme
- how can we authenticate a message m of arbitrary length?
- naïve approach
  - pad m and view it as d blocks $m_1, m_2, \ldots, m_d$
  - separately apply MAC to block m_i



- security issues
  - reordering attack; verify block index, $t = F_k(m_i || i)$
  - truncation attack; verify message length $\delta = |m|$, $t = F_k(m_i || i || \delta)$
  - mix-and-match attack; randomize tags (using message-specific fresh nonce)

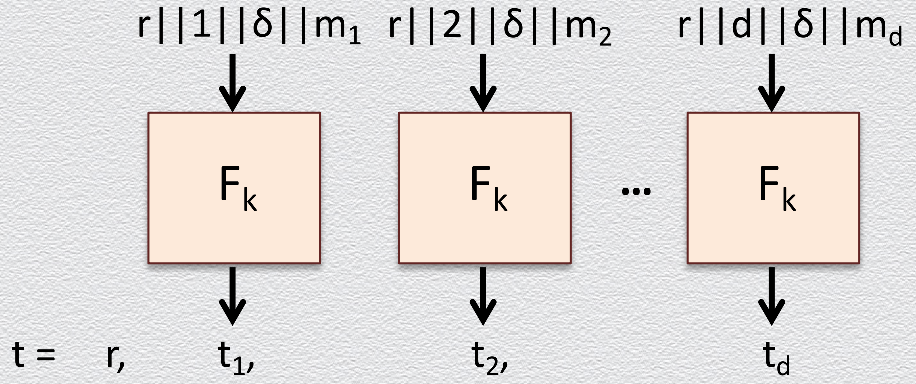
2. Domain extension for MACs (II)

Final scheme

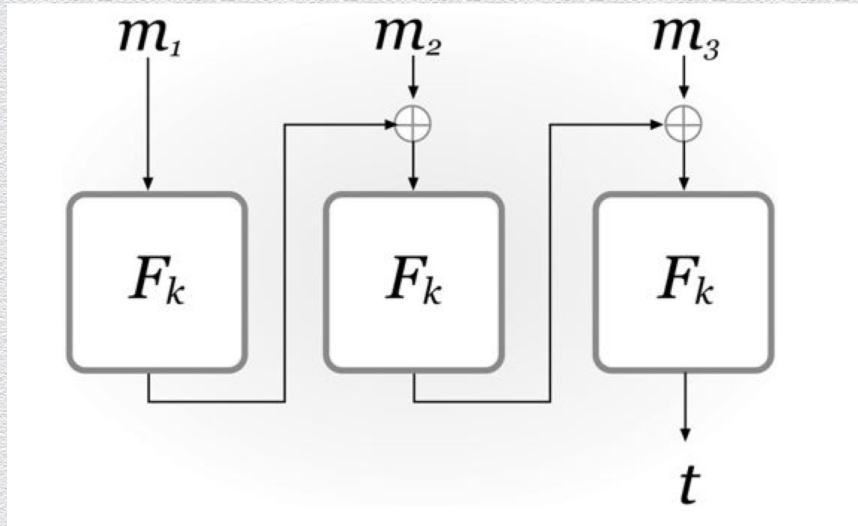
- assumes a secure MAC scheme for messages of size n
- set tag of message m of size δ at most $2^{n/4}$ as follows
  - choose fresh random nonce r of size n/4; view m as d blocks of size n/4 each
  - separately apply MAC on each block, authenticating also its index, δ and nonce r

Security

- extension is secure, if F_k is a secure PRF



3. CBC-MAC



Idea

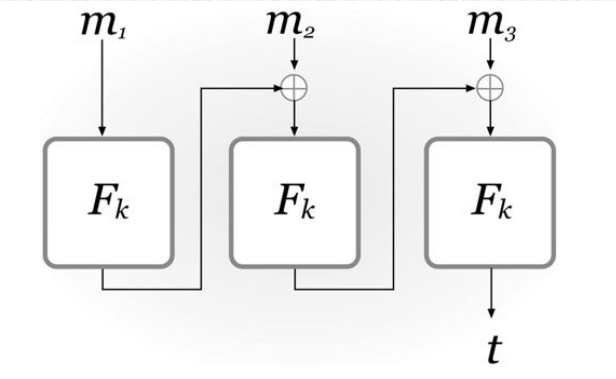
- employ a PRF in a manner similar to CBC-mode encryption

Security

- extension is secure, if
  - F_k is a secure PRF; and
  - only **fixed-length** messages are authenticated
- messages of length equal to any multiple of n can be authenticated
  - but this length need be fixed in advance
  - insecure, otherwise

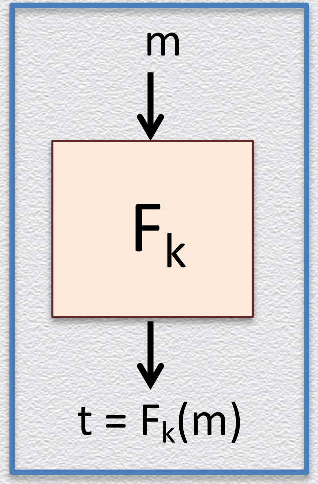
3. CBC-MAC Vs. previous schemes

- can authenticate longer messages than basic PRF-based scheme (1)

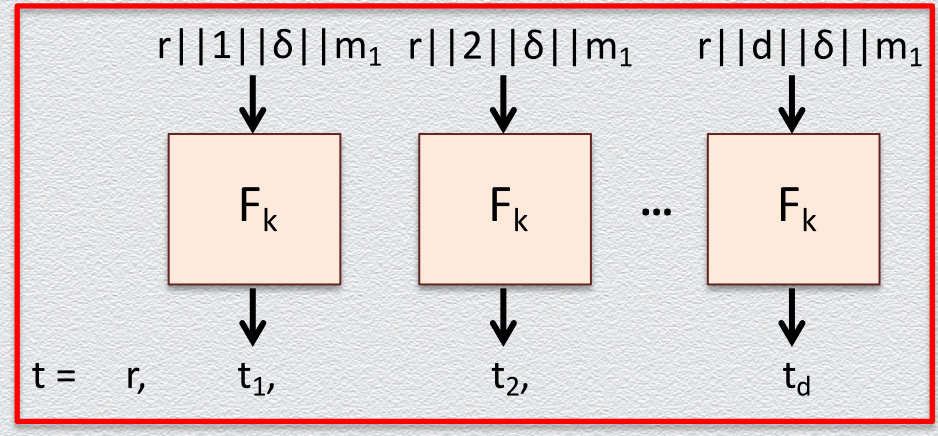


- more efficient than domain-extension MAC scheme (2)

Scheme (1)



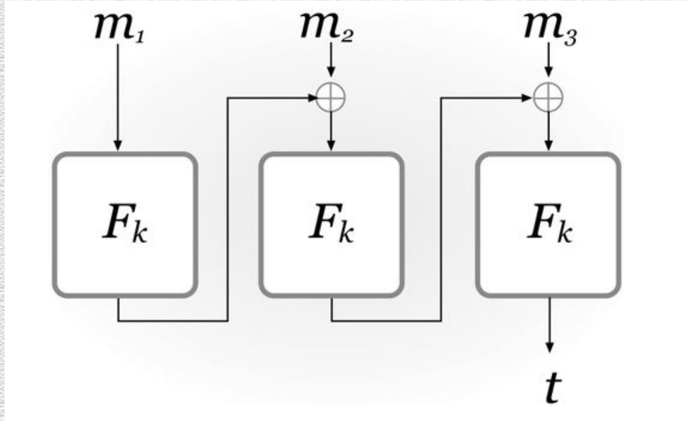
Scheme (2)



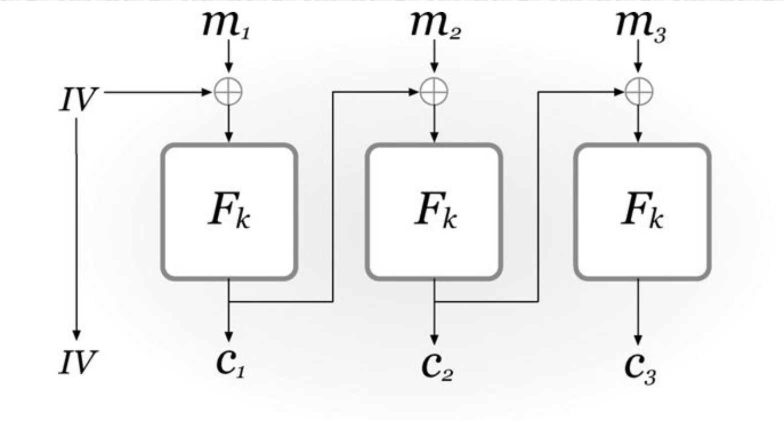
3. CBC-MAC Vs. CBC-mode encryption

- crucially for their security
  - CBC-MAC uses **no IV** (or uses an IV set to 0) and only the **last PRF output**
  - CBC-mode encryption uses a **random IV** and **all PRF outputs**
  - “simple”, innocent modification can be catastrophic…

CBC-MAC



CBC-mode encryption



5.3 Authenticated encryption

Recall: Two distinct properties

Secrecy

- **sensitive** information has value
  - if **leaked**, it can be **risky**
- **specific** scope / **general** semantics
- **prevention**
- does **not** imply integrity
  - e.g., bit-flipping "attack"

Integrity

- **correct** information has value
  - if **manipulated**, it can **harmful**
  - **random Vs. adversarial** manipulation
- **wider** scope / **context-specific** semantics
  - source Vs. content authentication
  - replay attacks
- **detection**
- does **not** imply secrecy
  - e.g., user knows cookies' "contents"

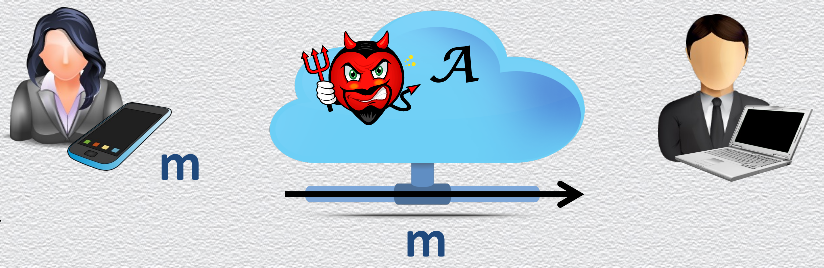
Recall: Yet, they are quite close…

Common setting

- communication (storage) over an "**open**," i.e., **unprotected**, channel (medium)

Fundamental security problems

- while in transit (at rest)
  - no message (file) should be **leaked** to $\mathcal{A}$
  - no message (file) should be **modified** by $\mathcal{A}$



Core cryptographic protections

- **encryption schemes** provide **secrecy / confidentiality**
- **MAC schemes** provide **integrity / unforgeability**

Can we achieve both at once in the symmetric-key setting? **Yes!**

Authenticated Encryption (AE): Catch 2 birds w/ 1 stone

Cryptographic primitive that realizes an "**ideally secure**" communication channel

- motivation
  - important in practice as real apps often **need both**
  - **good security hygiene**
    - even if a given app "asks" only/more for secrecy or integrity than the other, it's always better **to achieve both**!

Three generic AE constructions

Constructions of a **secure authenticated encryption** scheme Π_{AE}

- they all make use of
  - a **CPA-secure** encryption scheme Π_E = (Enc, Dec); and
  - a **secure MAC** Π_M = (Mac, Vrf)
  - which are instantiated using **independent** secret keys ke, km

  - …but the **order** with which these are used matters!

Generic AE constructions (1)

1. **encrypt-and-authenticate**

- $Enc_{ke}(m) \longrightarrow c$; $Mac_{km}(m) \longrightarrow t$; send ciphertext (c, t)
- if $Dec_{ke}(c) = m \neq$ `fail` and $Vrf_{km}(m,t)$ accepts, output m; else output `fail`
- **insecure scheme, generally**
  - e.g., MAC tag t may leak information about m
  - e.g., if MAC is deterministic (e.g., CBC-MAC) then Π_{AE} is not even CPA-secure
  - used in SSH

Generic AE constructions (2)

2. **authenticate-then-encrypt**

- $Mac_{km}(m) \rightarrow t$; $Enc_{ke}(m||t) \rightarrow c$; send ciphertext c
- if $Dec_{ke}(c) = m||t \neq$ `fail` and $Vrf_{km}(m,t)$ accepts, output m; else output `fail`
- **insecure scheme, generally**
  - used in TLS, IPsec

Generic AE constructions (3)

3. **encrypt-then-authenticate** (cf. “authenticated encryption”)

- $Enc_{ke}(m) \longrightarrow c$; $Mac_{km}(c) \longrightarrow t$; send ciphertext (c, t)
- if $Vrf_{km}(c,t)$ accepts then output $Dec_{ke}(c) = m$, else output `fail`
- **secure scheme, generally** (as long as Π_M is a **“strong” MAC**)
  - used in TLS, SSHv2, IPsec

Application: Secure communication sessions

An AE scheme Π_{AE} = (Enc, Dec) enables two parties to **communicate securely**

- session: period of time during which sender and receiver maintain state
- idea: send any message m as c = $Enc_k(m)$ & ignore received c that don't verify
- security: **secrecy & integrity are protected**
- remaining possible attacks
  - **re-ordering** attack — counters can be used to eliminate reordering/replays
  - **reflection** attack — directional bit can be used to eliminate reflections
  - **replay** attack — $c = Enc_k(b_{A \to B} | ctr_{A,B} || m)$; $ctr_{A,B}$++

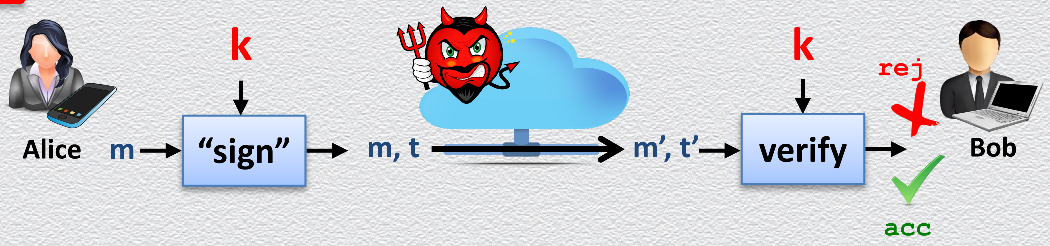
5.4 Public-key encryption & digital signatures

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” ⇒ **strong assumption** to accept
  - need of a secure channel
- “shared secret key” ⇒ **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” ⇨ **(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
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 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/critical**" tasks (performed by **user U**)

Assumption

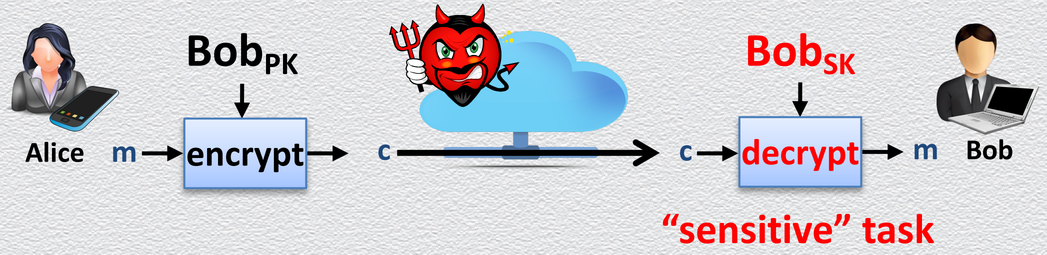
- **public-key infrastructure (PKI)**: public keys become **securely** available to users

From symmetric to asymmetric encryption

secret-key encryption

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

public-key encryption



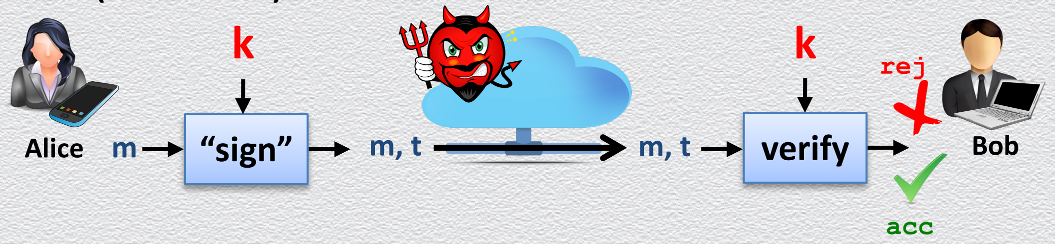
- main flexibility
  - **user-specific** keys

- 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)

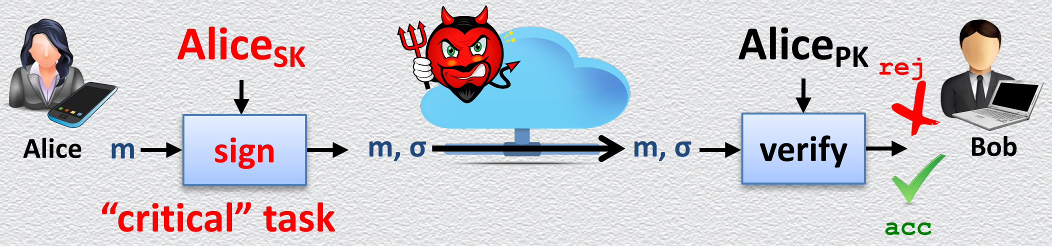
- main limitation
  - session-specific keys



public-key message authentication

(or **digital signatures**)

- main flexibility
  - user-specific keys



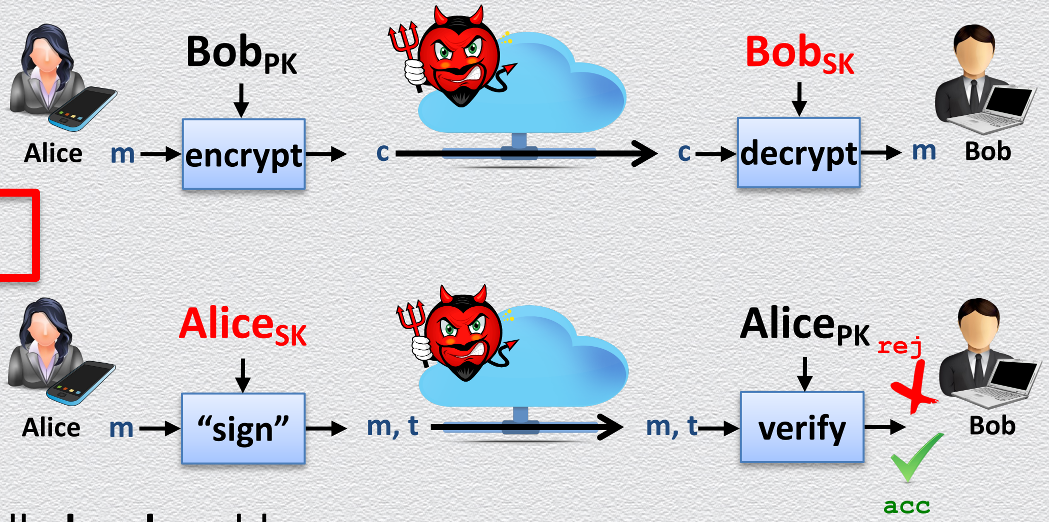
- (only) messages signed by sender’s SK can be verified by sender’s PK

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
  - 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

Asymmetric crypto

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

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

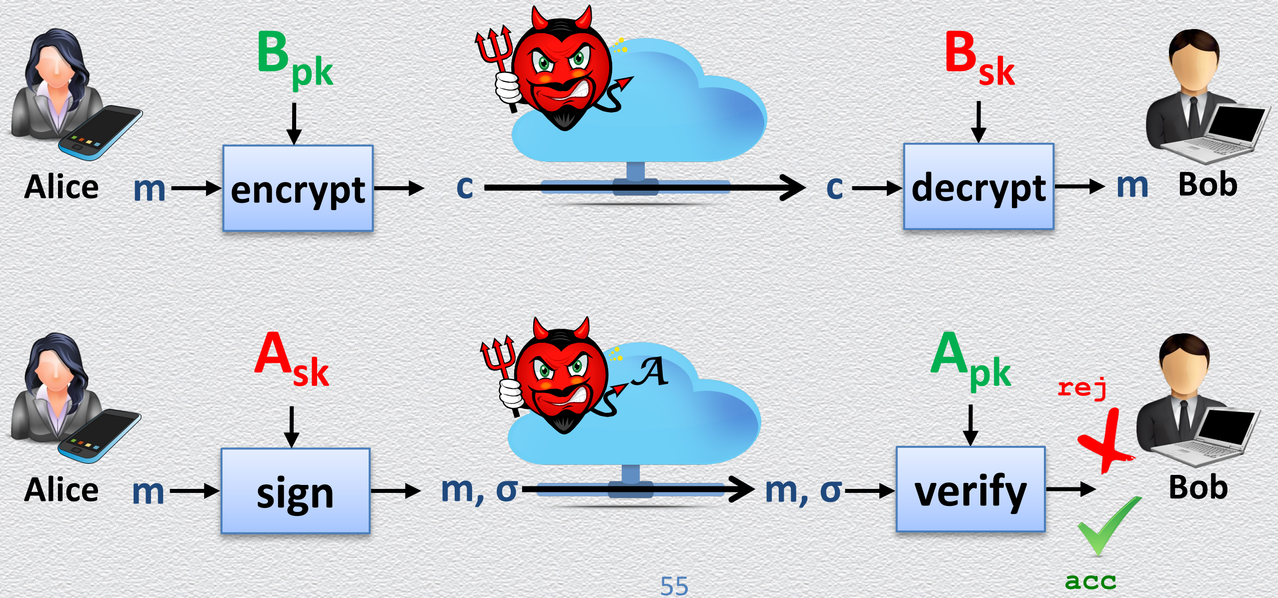
Very challenging to realize

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

Overall: Public-key encryption & signatures

Assume a trusted set-up

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



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

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