CS306: Introduction to IT Security

Fall 2018

Lecture 5: Hash functions

Instructor: **Nikos Triandopoulos**

September 25, 2018

STEVENS
INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY®
1870

Last week

- Symmetric encryption
  - big picture
  - pseudorandomness
  - modes of operations (part II)
- Message authentication
  - motivation & properties
  - message authentication codes & constructions

Today

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

5.0 Announcements

CS306: Announcements

- **HW 1**
  - **due this Friday**
  - start working on it & seek help during office hours!
  - solutions will be discussed in class next week
- **upcoming labs**
  - **Lab 4** (September 27th)
    - covers message authentication & hash functions
  - **Lab 5** (October 4th)
    - covers public-key cryptography
    - **2nd practical assignment**
  - **no lab** the Monday-schedule week (October 11th)*
    - week before midterm

* see possible exception in next slide

CS306: Announcements (continued)

- **HW 2** goes out on Monday next week (October 1st)
  - covers message authentication, hash functions & PK crypto
  - **due in 1 week**, preparation for mid-term exam
  - solutions will be discussed before mid-term*
- **mid-term exam** is in 3 weeks (October 16th)
  - 1-hour closed-book exam
  - covers all materials covered in weeks 1 – 6
- **help session** (tentatively)
  - one 30-min review session with instructor
    - ~20-min review & examples, ~10min Q & A
    - per section or for entire class, depending on room availability*
  - time & location TBA

* possibly during labs schedule on October 11th

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***	**No lab**
7	Oct 16	**Midterm** (closed books)	All materials covered	

*** tentatively**

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

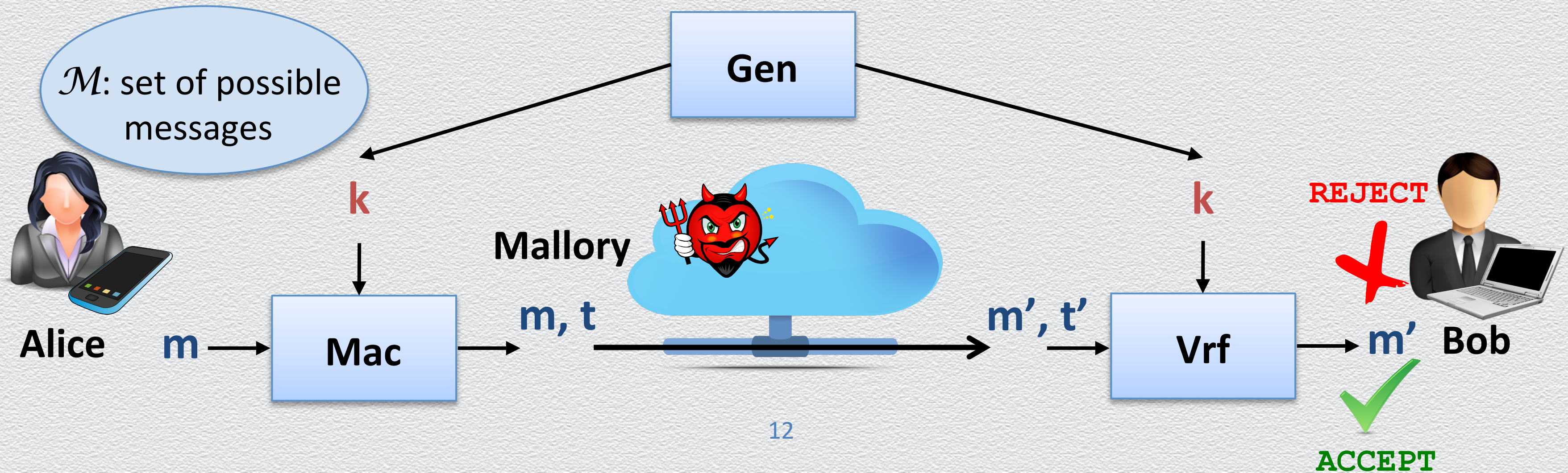
Questions?

5.1 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)**

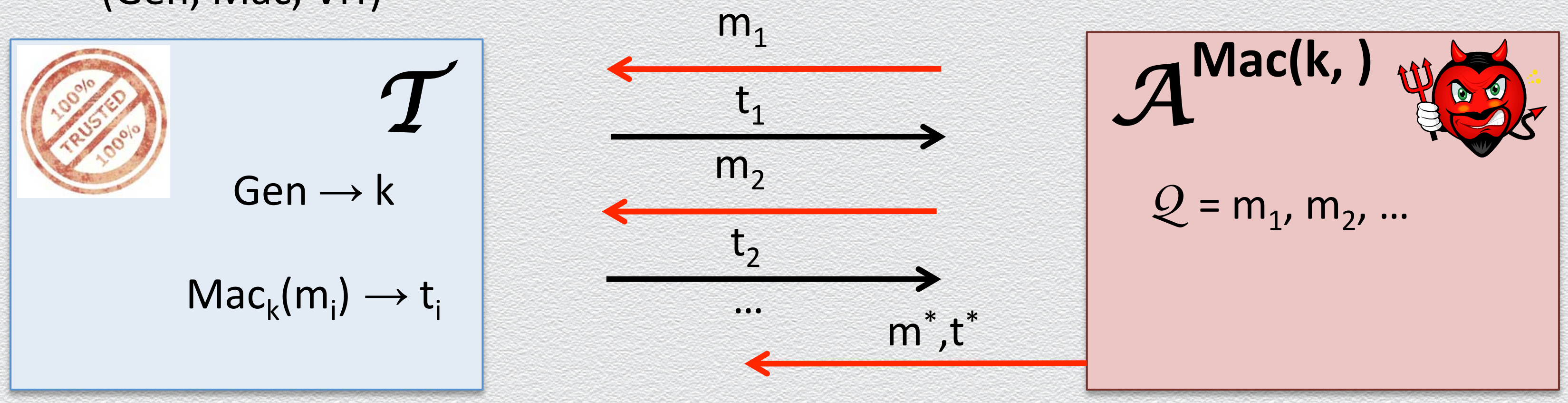


Recall: MAC security

Attacker **wins** the game if

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

MAC scheme
(Gen, Mac, Vrf)



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

Real-life attacker

In practice, an attacker

- observes a traffic of authenticated (and successfully verified) messages
- is able to manipulate or often also to partially influence traffic
  - e.g., with its purchasing behavior in the example of cookies representing shopping carts
- has as goal to forge a verifiable message
  - typically by inserting a <u>new</u> invalid but verifiable message m^*, t^* into the traffic
  - if m^* equals any <u>previously observed</u> message, its goal is <u>trivial</u> to achieve
    - typically known as a **replay attack**
    - it can be harmful in real life but will be excluded now
- is capable of launching a brute-force attack (given that $\mathrm{Mac}_k(m) \rightarrow t$ is publicly known)
  - given any observed pair (m, t), <u>exhaustively search key space $\mathcal{K}$</u> to find the used key k

Threat model – capturing real-world attackers

Mallory is an adversary $\mathcal{A}$ that is

- "active" (on the wire)
  - allow $\mathcal{A}$ to **observe** and/or **manipulate** sent messages
- "well-informed" (w.r.t. its intelligence) via access to oracle $\mathrm{Mac}_k()$
  - allow $\mathcal{A}$ to **request MAC tags** of messages of **its choice** (i.e., to fully control the traffic)
- "replay-attack safe" (w.r.t. its goal)
  - restrict A to target to **forge only new** messages (i.e., not present in the traffic)
- "PPT" (w.r.t. its power)
  - restrict $\mathcal{A}$ to be **computationally bounded** (thus, capturing computational security)
  - new messages may be forged undetectably but they can be found only with **negligible** probability or after an **exponentially** large computation

Notes on security definition

- Is it a rather strong security definition?
  - allowing $\mathcal{A}$ to **query MAC tags for any message**
    - real-world senders will authenticate only "meaningful" messages
  - allowing $\mathcal{A}$ to break the scheme by **forging any new message**
    - similarly, a breach of security should only consider "meaningful" forged messages
- It is the right approach…
  - message "**meaningfulness" depends on higher-level application**
    - text messaging apps require authentication of English-text messages
    - but other apps may require authentication of binary tranfered 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! they constitute a very realistic and serious threat in practice
  - e.g., integrity of e-banking transactions: what if a money transfer is "replayed"?
  - whether replayed messages are valid or safe depends on higher app
- Still, it's preferable to consider a "replay-attack unsafe" security definition
  - better not to assume any semantics regarding the high-level app, but instead delegate the validity or safety check to this app that "consumes" the messages
- 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 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”

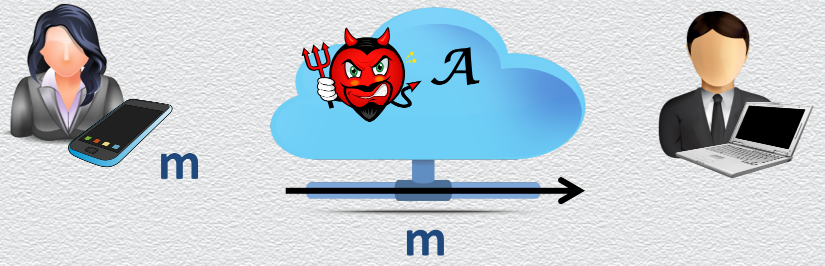
Recall: Yet, they are quite close…

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**!
- threat model
  - **active** adversary

Three generic constructions for authenticated encryption

3 generic ways to construct 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 $Vrfy_{km}(m,t)$ 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

Generic AE constructions (2)

- 2. **authenticate-then-encrypt**
  - $\mathrm{Mac}_{km}(m) \longrightarrow t$; $\mathrm{Enc}_{ke}(m||t) \longrightarrow c$; send ciphertext c
  - if $\mathrm{Dec}_{ke}(c) = m||t \neq$ `fail` and $\mathrm{Vrfy}_{km}(m,t)$ accepts, output m; else output `fail`
  - **insecure scheme, generally**

Generic AE constructions (3)

- 3. **encrypt-then-authenticate** (cf. "authenticated encryption")
  - $\text{Enc}_{ke}(m) \rightarrow c$; $\text{Mac}_{km}(c) \rightarrow t$; send ciphertext (c, t)
  - if $\text{Vrfy}_{km}(c,t)$ accepts then output $\text{Dec}_{ke}(c) = m$, else output `fail`
  - **secure scheme, generally** (as long as Π_M is a **"strong" MAC**)

Application: Secure communication sessions

An AE scheme Π_{AE} = (Enc, Dec) enables 2 parties to **communicate securely**

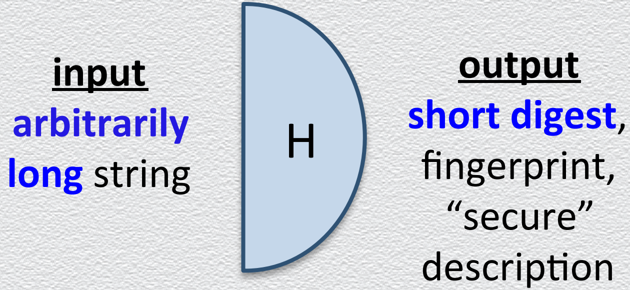
- session: period of time during which sender and receiver maintain state
- idea: send any message m as $c = Enc_k(m)$ & ignore received c that don't verify
- security
  - by applying Π_{AE} messages' **secrecy and integrity is protected**
- remaining possible attacks
  - **re-ordering** attack — counters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

5.3 Hash functions

Cryptographic hash functions

Basic cryptographic primitive

- maps **"objects"** to a **fixed-length** binary strings
- core security property: mapping **avoids collisions**
  - **collision**: distinct objects ($x \neq y$) are mapped to the same hash value ($H(x) = H(y)$)
  - although collisions **necessarily exist**, they are **infeasible to find**



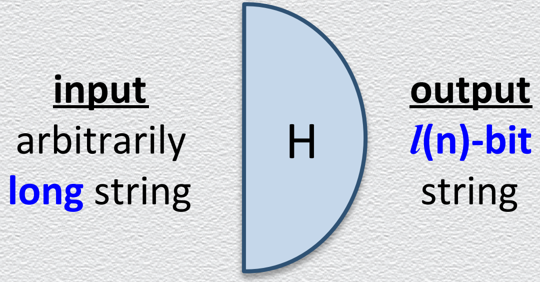
Important role in modern cryptography

- lie between symmetric- and asymmetric-key cryptography
- capture different security properties of "idealized random functions"
- qualitative stronger assumption than PRF

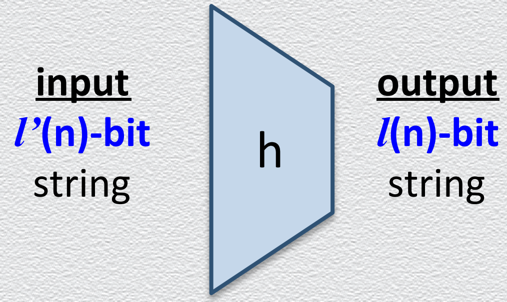
Hash & compression functions

Map messages to short digests

- a **general** hash function H() maps
  - a message of an **arbitrary length** to a **l(n)-bit** string



- a **compression** (hash) function h() maps
  - a **long** binary string to a **shorter** binary string
  - an **l'(n)-bit string** to a **l(n)-bit string**, with **$l'(n) > l(n)$**



Collision resistance (CR)

Attacker wins the game if $x \neq x'$ & $H^s(x) = H^s(x')$

Hash function H $\mathcal{T}$

description of H $\longrightarrow$

$\longleftarrow$ x, x'

$\mathcal{A}$

H is collision-resistant if any PPT $\mathcal{A}$ wins the game only negligibly often.

Weaker security notions

Given a hash function $H: X \rightarrow Y$, then we say that H is

- **preimage resistant** (or **one-way**)
  - if given $y \in Y$, finding a value $x \in X$ s.t. $H(x) = y$ happens negligibly often
- **2-nd preimage resistant** (or **weak collision resistant**)
  - if given a **uniform** $x \in X$, finding a value $x' \in X$, s.t. $x' \neq x$ and $H(x') = H(x)$ happens negligibly often

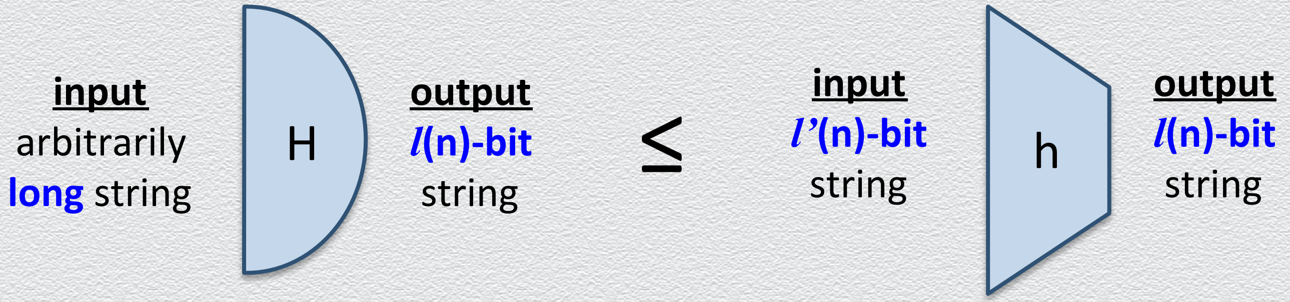
- cf. **collision resistant** (or **strong collision resistant**)
  - if finding two distinct values $x', x \in X$, s.t. $H(x') = H(x)$ happens negligibly often

5.4 Design framework

Domain extension via the Merkle-Damgård transform

General design pattern for cryptographic hash functions

- reduces CR of general hash functions to CR of compression functions



- thus, in practice, it suffices to realize a collision-resistant compression function h
- compressing by 1 single bit is a least as hard as compressing by any number of bits!

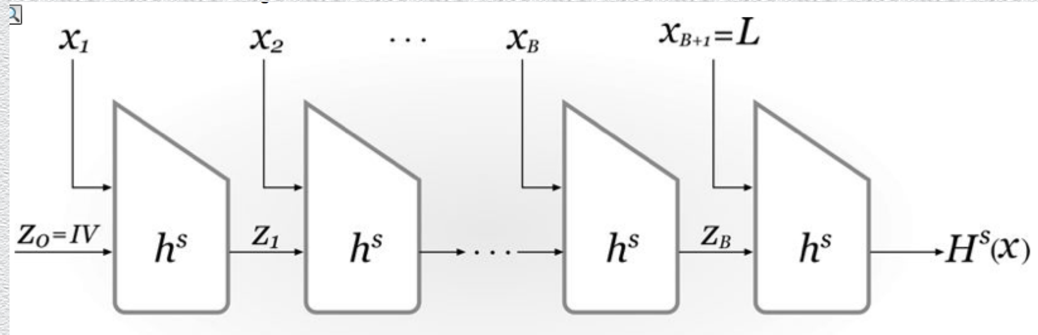
Merkle-Damgård transform: Design

Suppose that h: $\{0,1\}^{2n} \rightarrow \{0,1\}^n$ is a collision-resistant compression function

Consider the general hash function H: $\mathcal{M} = \{x : |x| < 2^n\} \rightarrow \{0,1\}^n$, defined as

Merkle-Damgård design

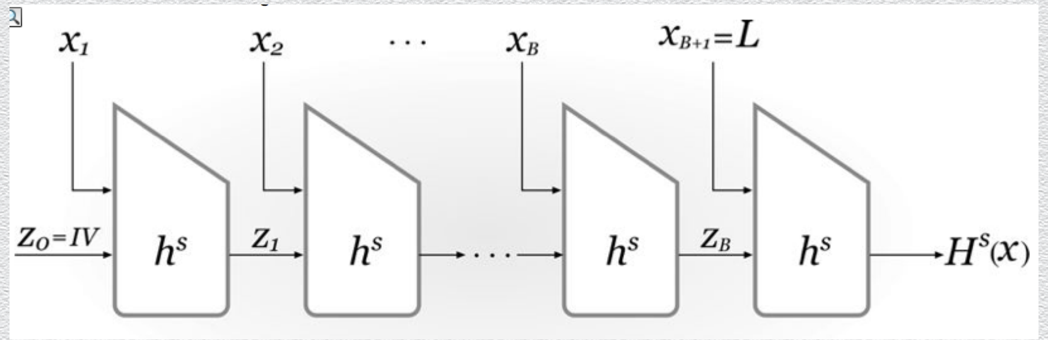
- H(x) is computed by applying h() in a **"chained" manner** over n-bit message blocks



- pad x to define a number, say B, **message blocks $x_1, \ldots, x_B$**, with $|x_i| = n$
- set extra, final, message block **x_{B+1} as an n-bit encoding L of $|x|$**
- starting by initial digest **$z_0 = IV = 0^n$**, output **$H(x) = z_{B+1}$**, where **$z_i = h^s(z_{i-1} || x_i)$**

Merkle-Damgård transform: Security

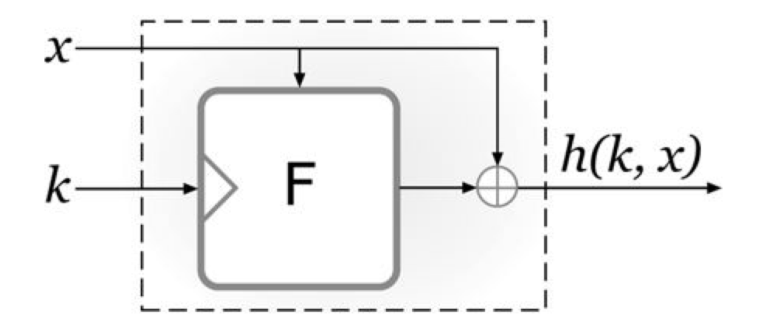
If the compression function h is CR,
then the derived hash function H is also CR!



Alternative design: The Davies-Meyer scheme

Employs PRF w/ key length k & block length n

- define h: $\{0,1\}^{k+n} \longrightarrow \{0,1\}^n$ as **$h(x) = F_k(x)$ XOR x**



Security

- h is CR, if F is an **ideal cipher**

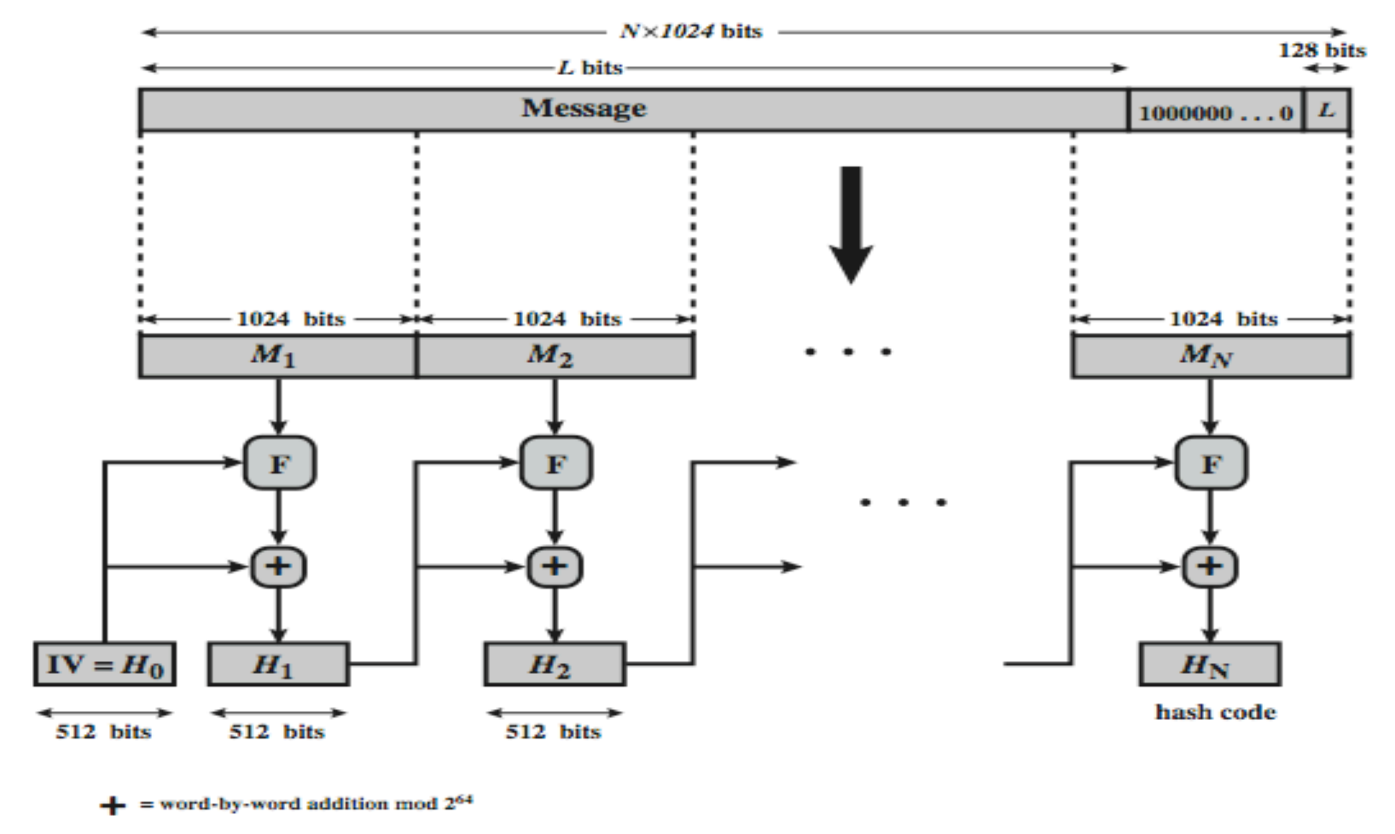
Well known hash functions

- MD5 (designed in 1991)
  - output 128 bits, collision resistance **completely broken** by researchers in 2004
  - today (controlled) collisions can be found in less than a minute on a desktop PC
- SHA1 – the Secure Hash Algorithm (series of algorithms standardized by NIST)
  - output 160 bits, considered **insecure** for collision resistance
  - **broken** in 2017 by researchers at CWI
- SHA2 (SHA-224, SHA-256, SHA-384, SHA-512)
  - outputs 224, 256, 384, and 512 bits, respectively, **no real security concerns yet**
  - based on Merkle-Damgård + Davies-Meyer generic transforms
- SHA3 (Kessac)
  - **completely new philosophy** (sponge construction + unkeyed permutations)

Message Digest Algorithm 5 – MD5

- developed by Ron Rivest in 1991
- uses 128-bit hash values
- still widely used in legacy applications although considered insecure
- various severe vulnerabilities discovered
- collisions found by Marc Stevens, Arjen Lenstra and Benne de Weger

SHA-2 overview



Current hash standards

Algorithm	Maximum Message Size (bits)	Block Size (bits)	Rounds	Message Digest Size (bits)
MD5	2^{64}	512	64	128
SHA-1	2^{64}	512	80	160
SHA-2-224	2^{64}	512	64	224
SHA-2-256	2^{64}	512	64	256
SHA-2-384	2^{128}	1024	80	384
SHA-2-512	2^{128}	1024	80	512
SHA-3-256	unlimited	1088	24	256
SHA-3-512	unlimited	576	24	512

5.5 Generic attacks

Generic attacks against cryptographic hashing

Assume a CR compression function $h : \{0,1\}^{l'(n)} \rightarrow \{0,1\}^{l(n)}$

- **brute-force** attack
  - for each string x in the domain
    - compute and record hash value h(x)
    - if h(x) equals a previously recorded hash h(y) (i.e., $x \neq y$ but $h(x)=h(y)$), halt and output collision on $x \neq y$
- **birthday** attack
  - surprisingly, a more efficient generic attack exists!

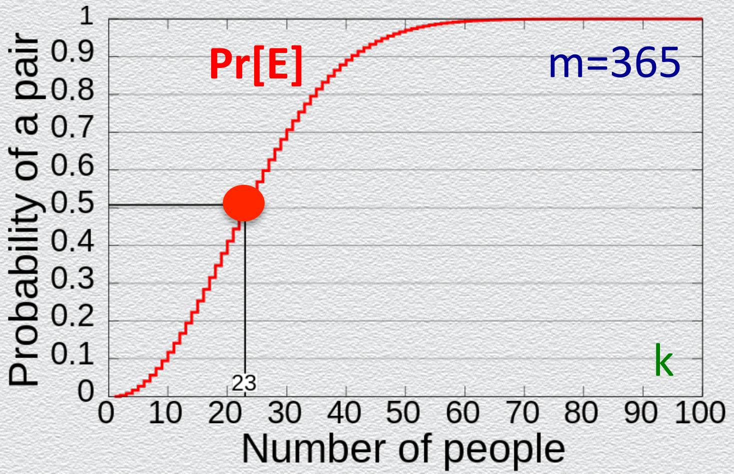
Birthday paradox

"In any group of 23 people (or more), it is **more likely** (than not) that **at least two** individuals have their birthday on the **same** day"

- based on probabilistic analysis of a random "balls-into-bins" experiment:

  "k balls are each, independently and randomly, thrown into one out of m bins"

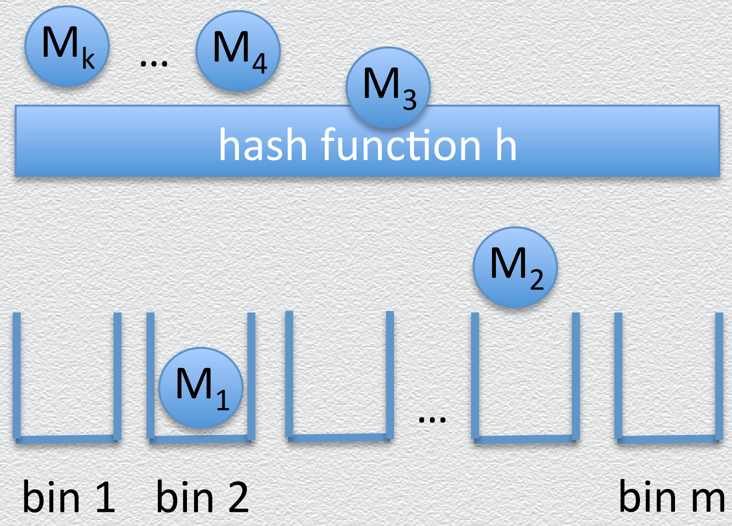
- captures likelihood that event E = "**two balls land into the same bin**" occurs
- analysis shows: $\Pr[E] \approx 1 - e^{-k(k-1)/2m}$ (1)
  - if **Pr[E] = 1/2**, Eq. (1) gives **$k \approx 1.17\ m^{½}$**
  - thus, for m = 365, k is around 23 (!)
    - assuming a uniform birth distribution



Birthday attack

Applies "birthday paradox" against cryptographic hashing

- exploits the likelihood of finding collisions for hash function h using a **randomized** search, rather than an **exhausting** search
- analogy
  - k balls: distinct messages chosen to hash
  - m bins: number of possible hash values
  - independent & random throwing
    - how is this achieved?
    - message selection, hash mapping



Probabilistic analysis

Experiment

- k balls are each, independently and randomly, thrown into one out of m bins

Analysis

- the probability that the i-th ball lands in an empty bin is: $1 - (i - 1)/m$
- the probability F_k that after k throws, no balls land in the same bin is:

$$F_k = (1 - 1/m)\ (1 - 2/m)\ (1 - 3/m)\ \dots\ (1 - (k - 1)/m)$$

- by the standard approximation $1 - x \approx e^{-x}$: $F_k \approx e^{-(1/m + 2/m + 3/m + \dots + (k-1)/m)} = e^{-k(k-1)/2m}$
- thus, two balls land in same bin with probability $Pr[E] = 1 - F_k = 1 - e^{-k(k-1)/2m}$
- **lower bound** – Pr[E] increases if the bin-selection distribution is not uniform

What birthday attacks mean in practice…

- approximate number of hash evaluations for finding hash collisions with prob. p for various digest lengths (or hash ranges H)

Bits	Possible outputs (2 s.f.) (H)	Desired probability of random collision (2 s.f.) (p)									
		10^{-18}	10^{-15}	10^{-12}	10^{-9}	10^{-6}	0.1%	1%	25%	50%	75%
16	65,536	<2	<2	<2	<2	<2	11	36	190	300	430
32	4.3×10^{9}	<2	<2	<2	3	93	2900	9300	50,000	77,000	110,000
64	1.8×10^{19}	6	190	6100	190,000	6,100,000	1.9×10^{8}	6.1×10^{8}	3.3×10^{9}	5.1×10^{9}	7.2×10^{9}
128	3.4×10^{38}	2.6×10^{10}	8.2×10^{11}	2.6×10^{13}	8.2×10^{14}	2.6×10^{16}	8.3×10^{17}	2.6×10^{18}	1.4×10^{19}	2.2×10^{19}	3.1×10^{19}
256	1.2×10^{77}	4.8×10^{29}	1.5×10^{31}	4.8×10^{32}	1.5×10^{34}	4.8×10^{35}	1.5×10^{37}	4.8×10^{37}	2.6×10^{38}	4.0×10^{38}	5.7×10^{38}
384	3.9×10^{115}	8.9×10^{48}	2.8×10^{50}	8.9×10^{51}	2.8×10^{53}	8.9×10^{54}	2.8×10^{56}	8.9×10^{56}	4.8×10^{57}	7.4×10^{57}	1.0×10^{58}
512	1.3×10^{154}	1.6×10^{68}	5.2×10^{69}	1.6×10^{71}	5.2×10^{72}	1.6×10^{74}	5.2×10^{75}	1.6×10^{76}	8.8×10^{76}	1.4×10^{77}	1.9×10^{77}

- additionally, for large enough $|H|=m$, it can be approximated that **the first hash collision** will be found **on average** after **$t(m) = 1.25(m)^{1/2}$** hash evaluations

Overall

Assume a CR function h producing hash values of size $l(n)$

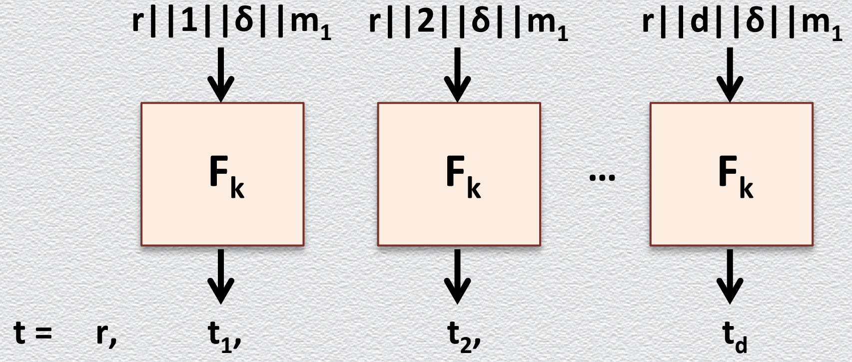
- **brute-force** attack
  - evaluate h on $\mathbf{2^{l(n)} + 1}$ distinct inputs
  - by the "pigeon hole" **principle**, at least 1 collision **will be** found
- **birthday** attack
  - evaluate h on (much) **fewer** distinct inputs that hash to **random** values
  - by "balls-into-bins" **probabilistic analysis**, at least 1 collision will **likely** be found
  - when hashing **only half** distinct inputs, it's **more likely** to find a collision!
  - thus, in order to get **k-bit security**, we (**at least**) need **hash values of length 2k**

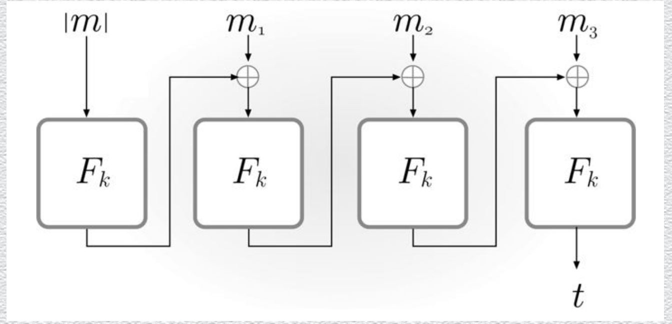
5.6 Applications in Cryptography

Hash functions enable efficient MAC design!

Back to problem of designing **<u>secure MAC for messages of arbitrary lengths</u>**

- so far, we have seen two solutions
  - block-based "tagging"
    - based on PRFs
    - inefficient
  - CBC-MAC
    - also based on PRFs
    - more efficient

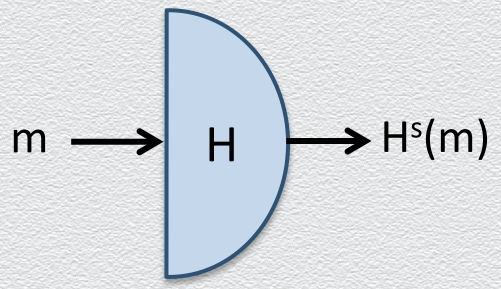


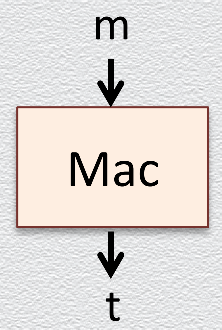


[1] Hash-and-MAC: Design

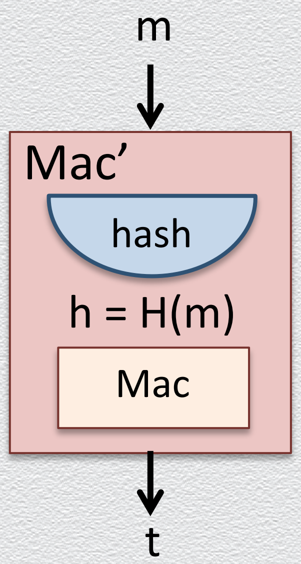
Generic method for designing **secure MAC for messages of arbitrary lengths**

- based on **CR hashing** and **any fix-length secure MAC**





- new MAC (Gen', Mac', Vrfy') **as the name suggests**
  - Gen': **instantiate** H and Mac_k with key k
  - Mac': **hash** message m into $h = H^s(m)$, output **Mac_k**-tag t on h
  - Vrfy': **canonical** verification



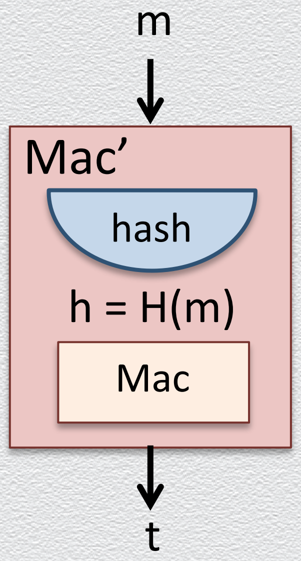
[1] Hash-and-MAC: Security

The Hash-and-MAC construction is a secure as long as

- H is **collision resistant**; and
- the underlying MAC is **secure**

Intuition

- since **H is CR**:
  authenticating **digest** **H(m)** is **a good as** authenticating **m itself**!



[2] Hash-based MAC

- so far, MACs are based on block ciphers
- can we construct a MAC based on CR hashing?

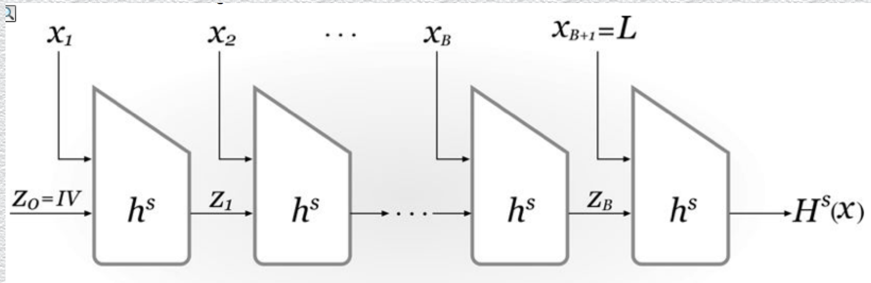
[2] A naïve, insecure, approach

Set tag t as:

$$\mathrm{Mac}_k(m) = \mathbf{H}(k||\mathbf{m})$$

- intuition: given H(k||m) it should be infeasible to compute H(k||m’), m’≠m

Insecure construction



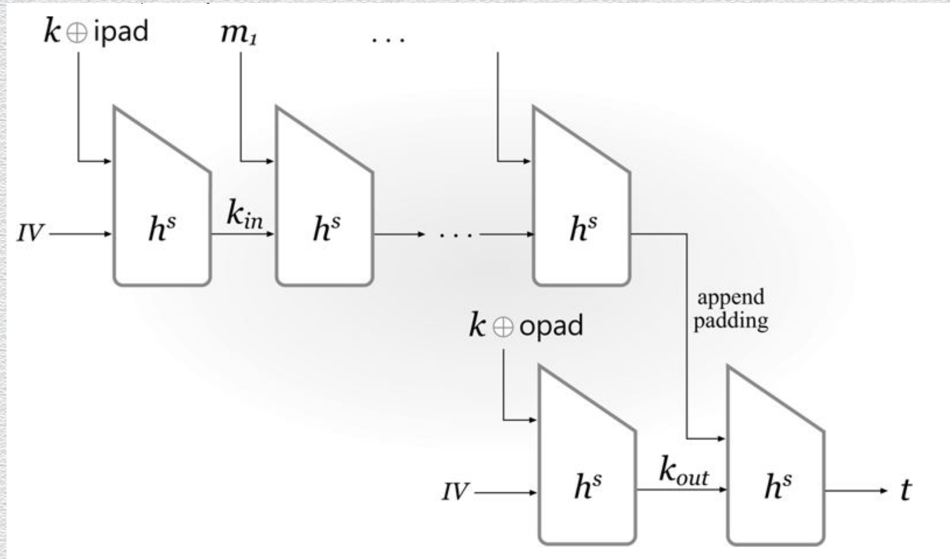
- practical CR hash functions employ the Merkle-Damgård design
- length-extension attack
  - knowledge of $H(m_1)$, make it feasible to compute $H(m_1||m_2)$
  - knowing of length of message m_1 can retrieve internal state s_k even without knowing k!

[2] HMAC: Secure design

Set tag t as:

$$HMAC_k[m] \quad = \quad \mathbf{H\,[}\ \ (k \oplus opad)\ \ ||\ \ \mathbf{H[}\ \ (k \oplus ipad)\ \ ||\ \ \mathbf{m}\ \ \mathbf{]}\ \ \mathbf{]}$$

- intuition: **<u>instantiation of hash & sign paradigm</u>**
- two layers of hashing H
  - **upper layer**
    - $y = H^s(\ (k \oplus ipad)\ ||\ m\ )$
    - $y = H^{s\prime}(m)$ – "hash"
  - **lower layer**
    - $t = H^s\ (\ (k \oplus opad)\ \ ||\ y'\ )$
    - $t = Mac'(k_{out}, y')$ – "sign"



[2] HMAC: Security

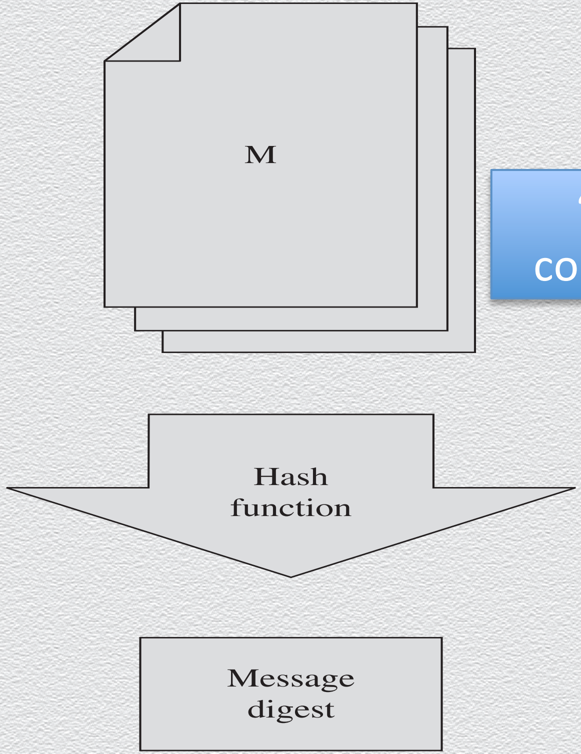
If used with a secure hash function and according to specs, HMAC is secure

- no practical attacks are known against HMAC
  - recent attacks on MD5 did not affect the security of HMAC-MD5
    - because dependence on weak-CR, not CR!

5.7 Applications in security

Generally: Message digests

Short secure description of data primarily used to detect changes



“one way compression”

Application 1: Secure cloud storage

- Bob has files f_1, f_2,…,f_n
- Bob sends to a cloud storage provider
  - the hashes $h(r||f_1)$, $h(r||f_2)$,…, $h(r||f_n)$
  - files f_1, f_2,…,f_n
- Bob stores locally randomness r and keeps it secret
- every time Bob **reads** a file f_i, he also reads $h(r||f_i)$ and verifies the integrity of f_i
- any problems with **writes**?

Application 2: Fairness (I)

Suppose Alice, Bob, Charlie are bidders in an online auction

- Alice plans to bid A, Bob B and Charlie C
  - they do not trust that bids will be secret
  - nobody is willing to submit their bid
- solution
  - Alice, Bob, Charlie submit **hashes** h(A), h(B), h(C) of their bids
  - all received hashes are posted online
  - then parties' bids A, B and C revealed
- analysis
  - "hiding:" hashes do not reveal bids (which property?)
  - "binding:" cannot change bid after hash sent (which property?)

Application 2: Fairness (II)

- due to the small search space, this protocol is not secure!
- a forward search attack is possible
  - e.g., Bob computes h(A) for the most likely bids A
- how to prevent this?
  - increase search space
  - e.g., Alice computes h(A||R), where R is randomly chosen
    - at the end, Alice must reveal A and R
    - but before he chooses B, Bob cannot try all A and R combination

Application 2: Digital envelops

Commitment schemes

- two operations
- commit(x, r) = C
  - i.e., put message x into an envelop (using randomness r)
  - e.g., commit(x, r) = h(x || r)
  - **hiding property**: you cannot see through an (opaque) envelop
- open(C, m, r) = ACCEPT or REJECT
  - i.e., open envelop (using r) to check that it has not been tampered with
  - e.g., open(C, m, r): check if h(x || r) =? C
  - **binding property**: you cannot change the contents of a sealed envelop

Application 2: Security properties

Hiding: perfect opaqueness

- similar to indistinguishability; commitment reveals nothing about message
  - adversary selects two messages x_1, x_2 which he gives to challenger
  - challenger randomly selects bit b, computes (randomness and) commitment C_i of x_i
  - challenger gives C_b to adversary, who wins if he can find bit b (better than guessing)

Binding: perfect sealing

- similar to unforgeability; cannot find a commitment "collision"
  - adversary selects two distinct messages x_1, x_2 and two corresponding values r_1, r_2
  - adversary wins if $\text{commit}(x_1, r_1) = \text{commit}(x_2, r_2)$

Example 2: Fair decision via coin flipping

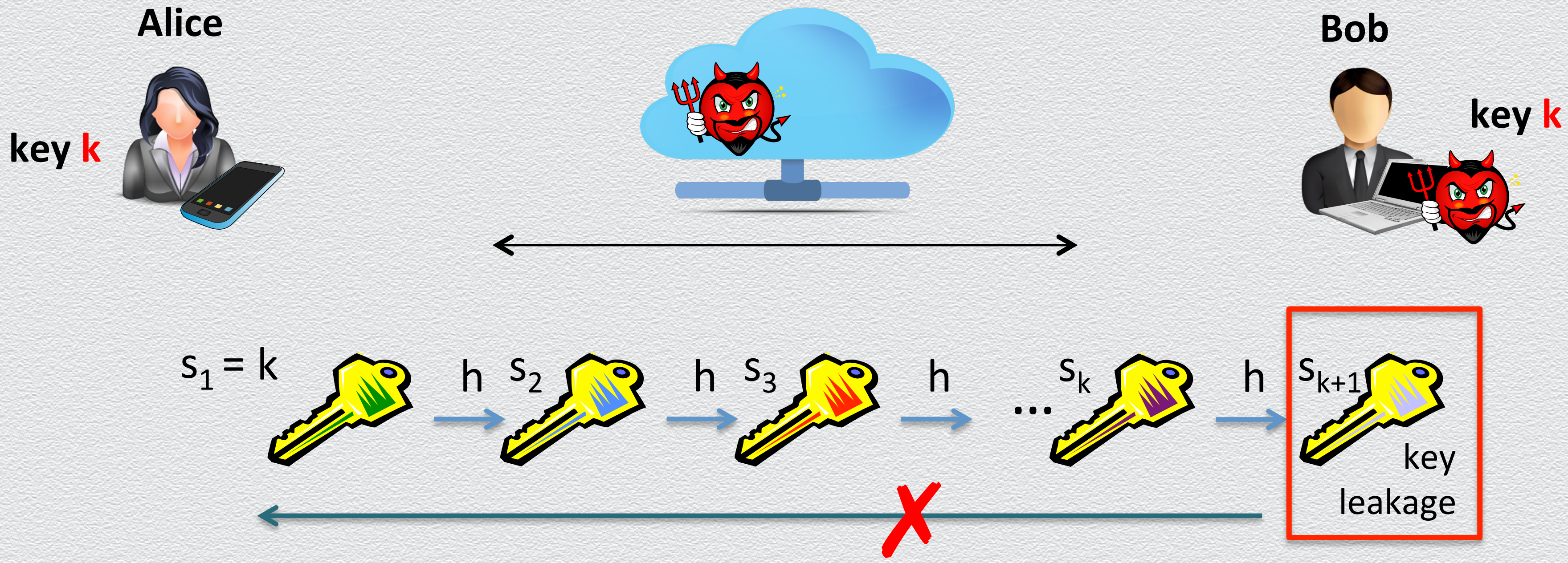
Alice is to “call” the coin flip and Bob is to flip the coin

- to decide who will do the dishes…
- problem: Alice may change her mind, Bob may skew the result
- protocol
  - Alice "calls" the coin flip but only tells Bob a commitment to her call
  - Bob flips the coin and reports the result
  - Alice reveals what she committed to
  - Bob verifies that Alice's call matches her commitment
  - If Alice's revelation matches the coin result Bob reported, Alice wins
- hiding: Bob does not get any advantage by seeing Alice commitment
- binding: Alice cannot change her mind after the coin is flipped

Application 3: Forward-secure key rotation

Alice and Bob secretly communicate using symmetric encryption

- Eve intercepts their messages and later breaks into Bob's machine to steal the shared key



Application 4: Hash values as file identifiers

Consider a cryptographic hash function H applied on a file F

- the hash (or digest) H(M) of F serves as a **unique** identifier for F
  - "uniqueness"
    - if another file F' has the same identifier, this contradicts the security of H
  - thus
    - the hash H(F) of F is like a fingerprint
    - one can check whether two files are equal by comparing their digests

Many real-life applications employ this simple idea!

Examples

4.1 Virus fingerprinting

- When you perform a virus scan over your computer, the virus scanner application tries to identify and block or quarantine programs or files that contain viruses
- This search is primarily based on comparing the digest of your files against a database of the digests of already known viruses
- The same technique is used for confirming that is safe to download an application or open an email attachment

4.2 Peer-to-peer file sharing

- In distributed file-sharing applications (e.g., systems allowing users to contribute contents that are shared amongst each other), both shared files and participating peer nodes (e.g., their IP addresses) are uniquely mapped into identifiers in a hash range
- When a given file is added in the system it is consistently stored at peer nodes that are responsible to store files those digests fall in a certain sub-range
- When a user looks up a file, routing tables (storing values in the hash range) are used to eventually locate one of the machines storing the searched file

Example 4.3: Data deduplication

Goal: Elimination of duplicate data

- Consider a cloud provider, e.g., Gmail or Dropbox, storing data from numerous users.
- A vast majority of stored data are duplicates; e.g., think of how many users store the same email attachments, or a popular video…
- Huge cost savings result from deduplication:
  - a provider stores identical contents possessed by different users once!
  - this is completely transparent to end users!

Idea: Check redundancy via hashing

- Files can be reliably checked whether they are duplicates by comparing their digests.
- When a user is ready to upload a new file to the cloud, the file's digest is first uploaded.
- The provider checks to find a possible duplicate, in which case a pointer to this file is added.
- Otherwise, the file is being uploaded literally
- This approach saves both storage and bandwidth!

Example 4.4: Password hashing

Goal: User authentication

- Today, passwords are the dominant means for user authentication, i.e., the process of verifying the identity of a user (requesting access to some computing resource).
- This is a “something you know” type of user authentication, assuming that only the legitimate user knows the correct password.
- When you provide your password to a computer system (e.g., to a server through a web interface), the system checks if your submitted password matches the password that was initially stored in the system at setup.

Problem: How to protect password files

- If password are stored at the server in the clear, an attacker can steal the password file after breaking into the authentication server – this type of attack happens routinely nowadays…
- Password hashing involved having the server storing the hashes of the users passwords.
- Thus, even if a password file leaks to an attacker, the onewayness of the used hash function can guarantee some protections against user-impersonation simply by providing the stolen password for a victim user.

Password storage

Identity	Password
Jane	qwerty
Pat	aaaaaa
Phillip	oct31witch
Roz	aaaaaa
Herman	guessme
Claire	aq3wm$oto!4

Plaintext

Identity	Password
Jane	0x471aa2d2
Pat	0x13b9c32f
Phillip	0x01c142be
Roz	0x13b9c32f
Herman	0x5202aae2
Claire	0x488b8c27

Concealed via hashing

Application 5: Hash-and-digitally-sign (looking ahead)

Very often digital signatures are used with hash functions

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

Signing message M

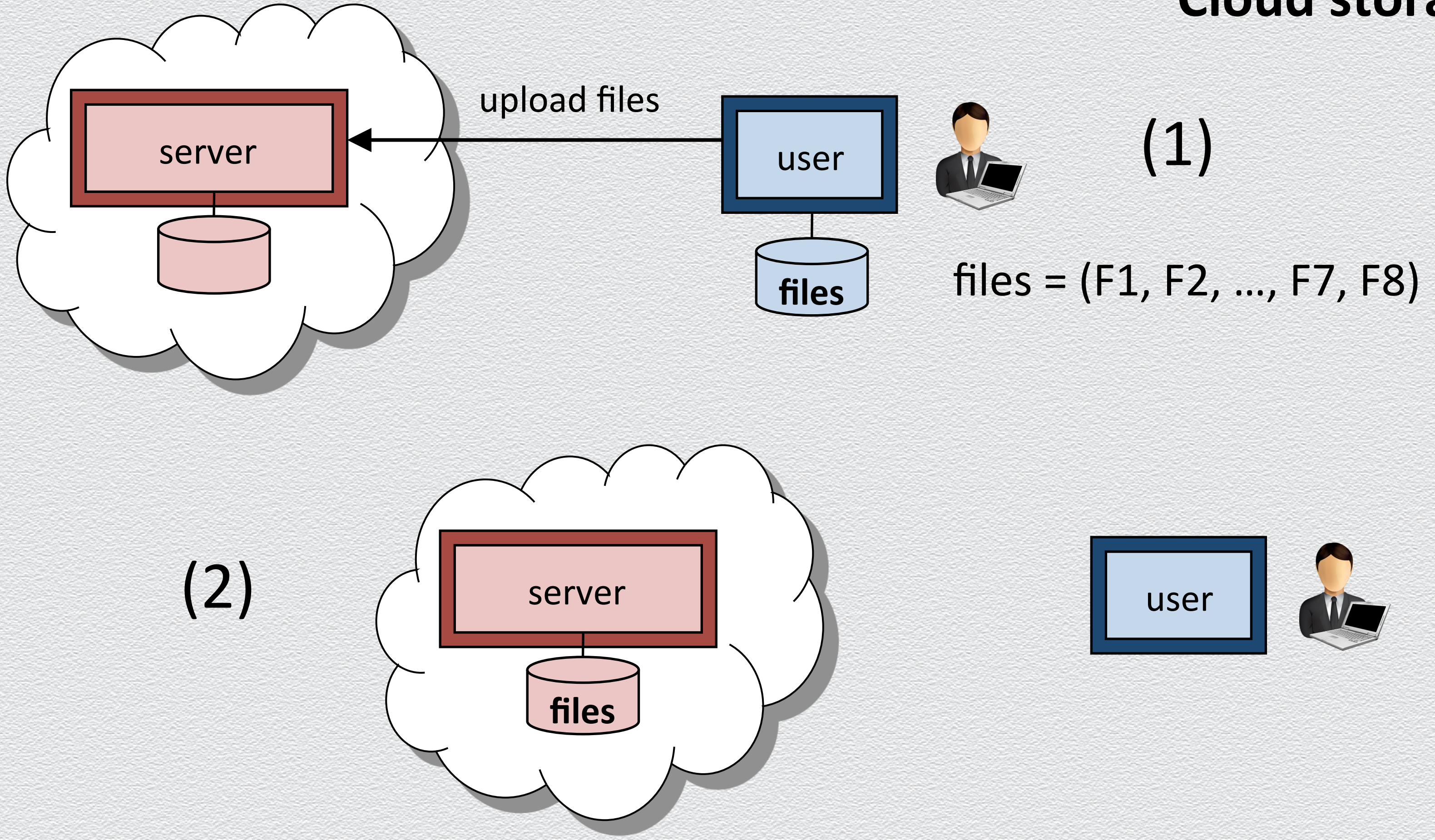
- let h be a cryptographic hash function, assume RSA setting (n, d, e)
- compute signature $\sigma = h(M)^d \bmod n$
- send σ, M

Verifying signature σ

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

5.8 Cloud storage security

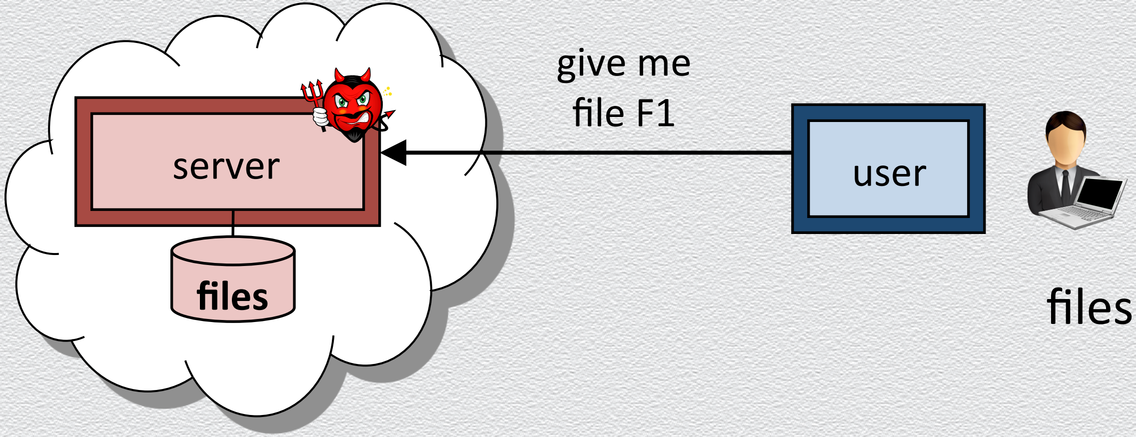
Cloud storage model



Cloud storage model

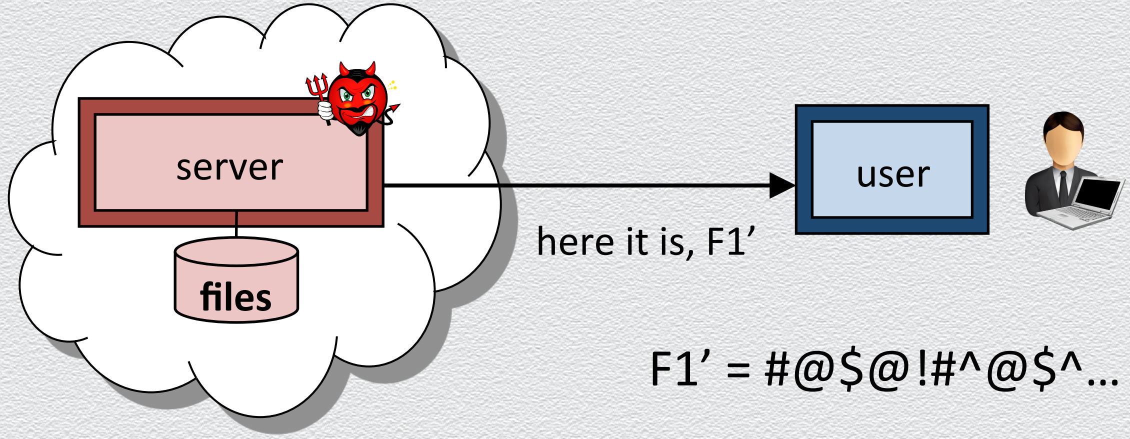
- attack by malicious server

(3)



files = (F1, F2, …, F7, F8)

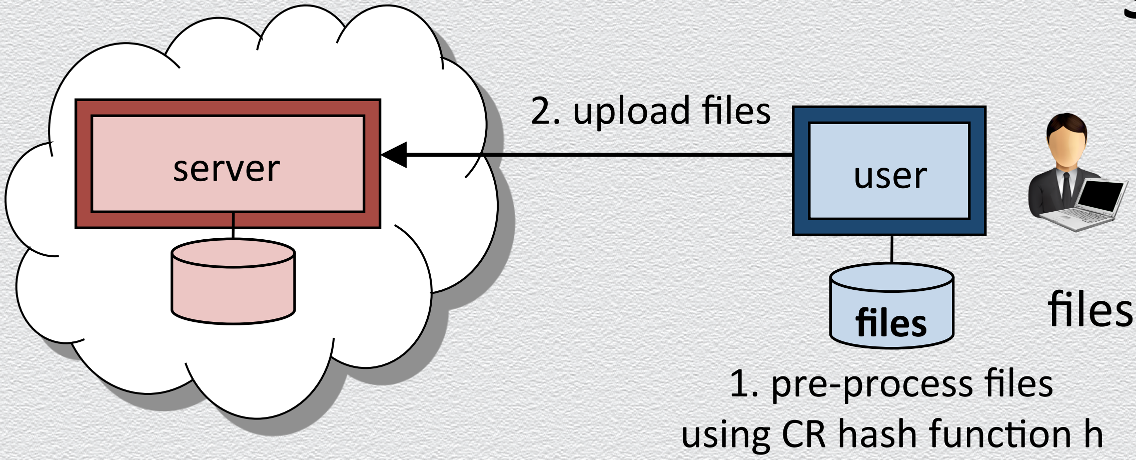
(4)



F1’ = #@$@!#^@$^…   (altered)

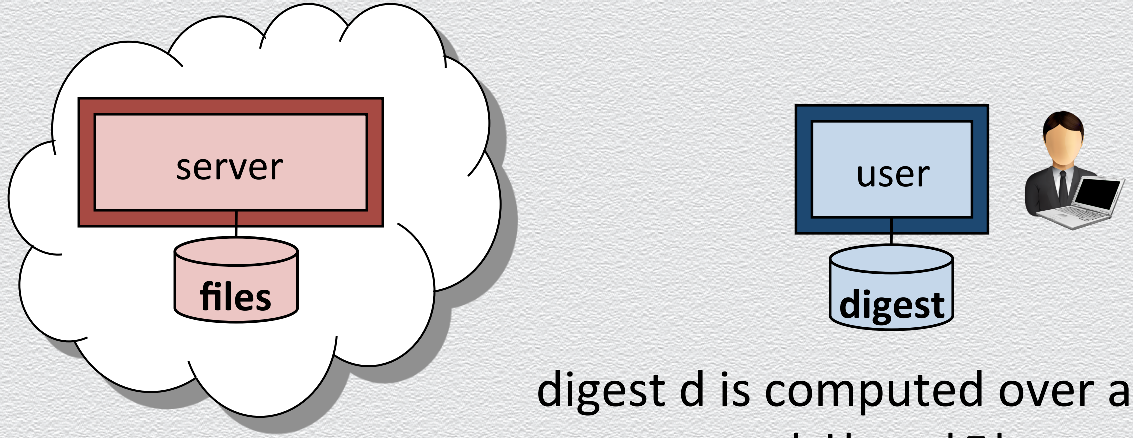
Secure cloud storage model
- integrity protection via hashing

(5)



files = F = (F1, F2, …, F7, F8)

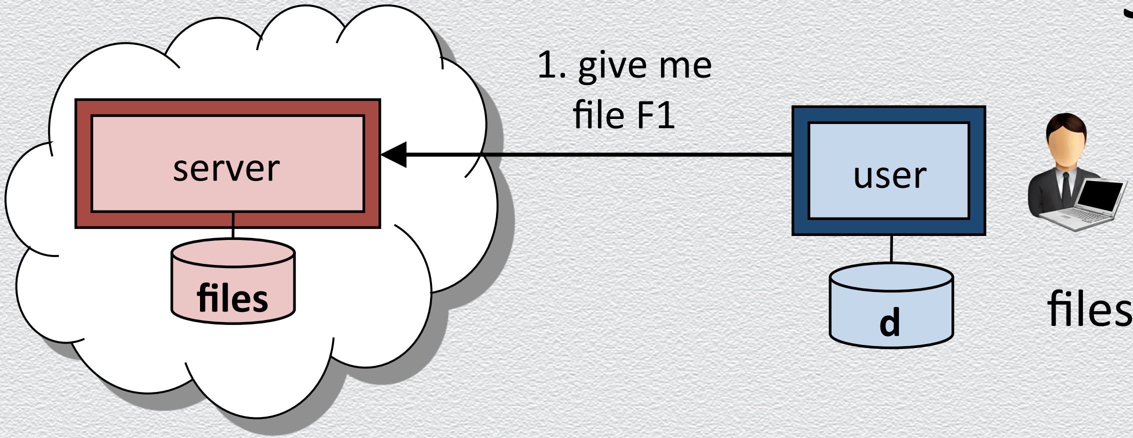
(6)



digest d is computed over all files

$|d| \ll |F|$

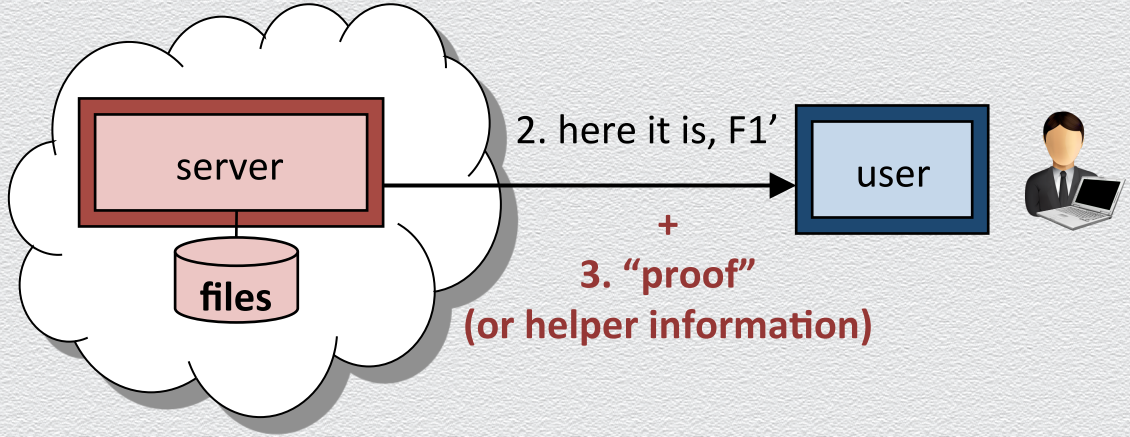
Secure cloud storage model
- how verification works



(7)

files = (F1, F2, …, F7, F8)

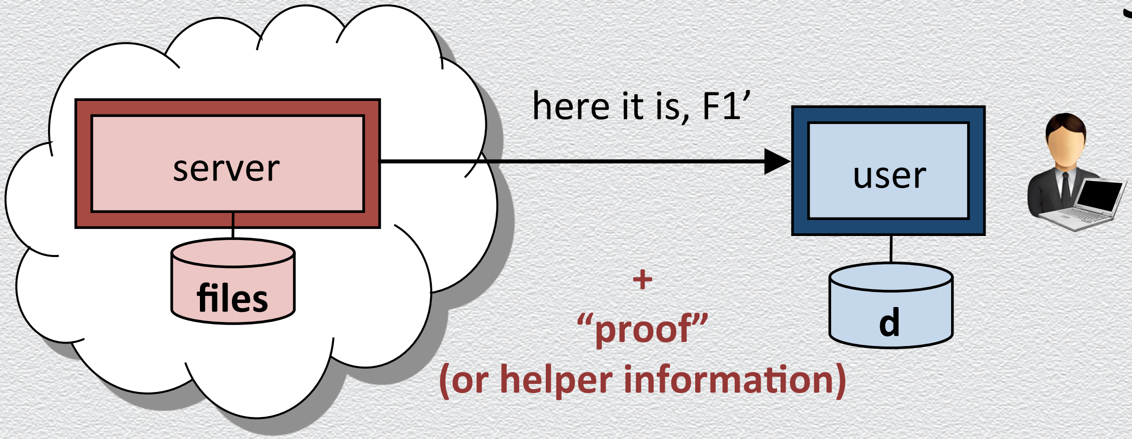
(8)



4. verification

“is F1’ intact?”

Secure cloud storage model
- verification via hashing



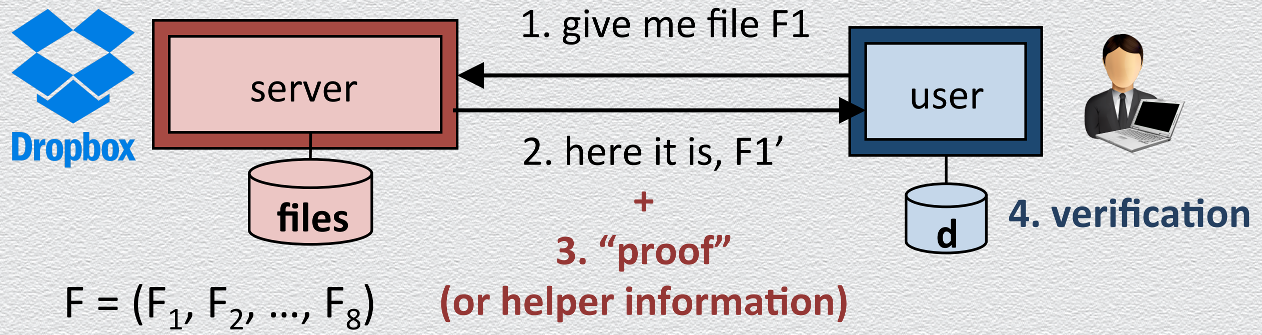
(9)

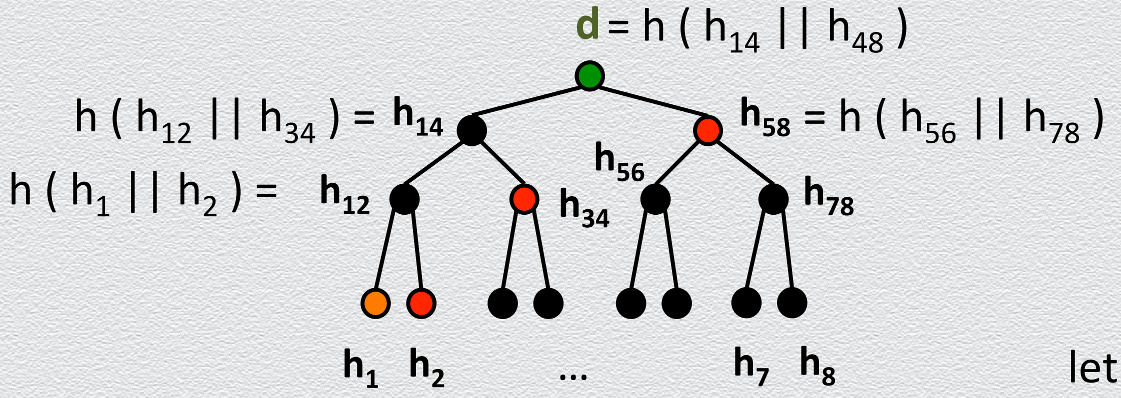
verification

“is F1’ intact?”

- user has
  - authentic digest d (locally stored)
  - file F1’ (to be checked/verified as it can be altered)
  - **proof** (to help checking integrity, but it can be maliciously chosen)
- verification involves (performed locally at user)
  - combine the file F1’ with the proof to re-compute candidate digest d’
  - check if d’ = d
  - if yes, then F1 is intact; otherwise tampering is detected!

Application 2: The Merkle tree





let $h_i = h(F_i)$, $1 \leq i \leq n$