CSE 3400/ CSE 5850 - Introduction to Computer & Network Security
/ Introduction to Cybersecurity

Lecture 6

Hash Functions – Part I

Ghada Almashaqbeh

UConn

Adapted from textbook slides by Prof. Amir Herzberg

UConn

Outline

❑ Introduction and motivation.

❑ Collision resistant hash functions (CRHF).

❑ CRHF applications.

❑ Other notions of security.

   ❑ TCR, SPR, OWF.

❑ Randomness extraction.

❑ The random oracle model.

Hash Functions

- Input m: binary strings
- Output $h(m)$:
  - 'Short' (n-bit) binary strings
    - Aka **message digest**
- Efficiently computable
- Applications: cryptography, security, efficiency
- Keyed $h_k(m)$, where the key is public, or unkeyed $h(m)$
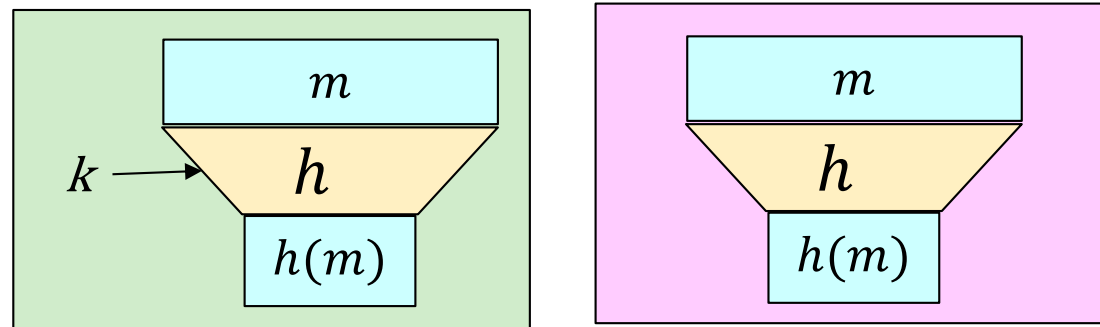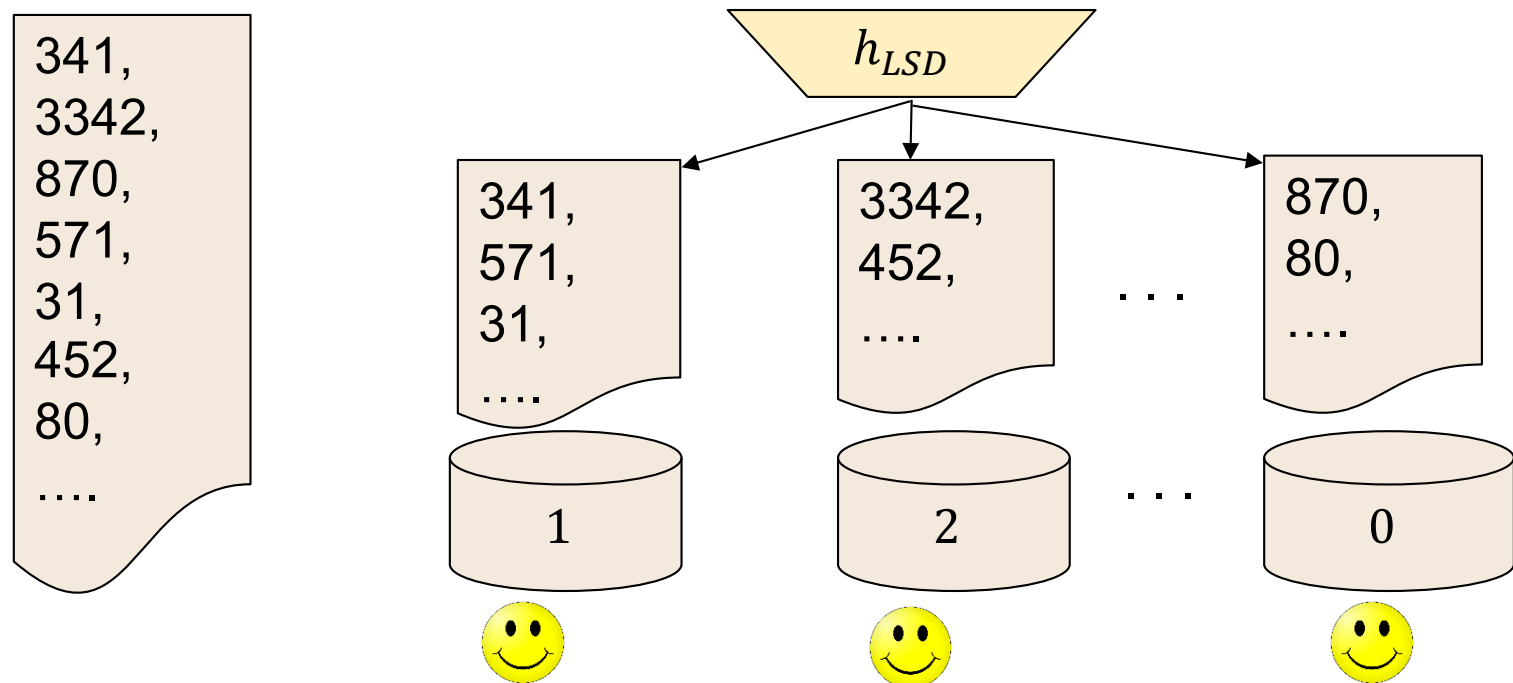
Hash functions: simple examples
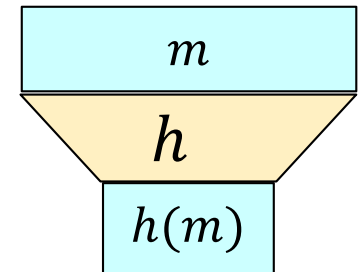
- ## For simplicity: input m is decimal integer

  - View as string of (three) digits

  - For example, $m = 127 \rightarrow m_1 = 1, m_2 = 2, m_3 = 7$

- ## Least Significant Digit hash:
$$h_{LSD}(m) = m_3$$

- ## Sum hash: $\quad h_{Sum}(m) = (m_1 + m_2 + m_3) \bmod 10$

- ## Exercise: $\quad h_{LSD}(117) = \underline{7}$
$$h_{Sum}(117) = \underline{9}$$

Note: the above are insecure hash functions, these are just toy examples to grasp the concept of hashing.
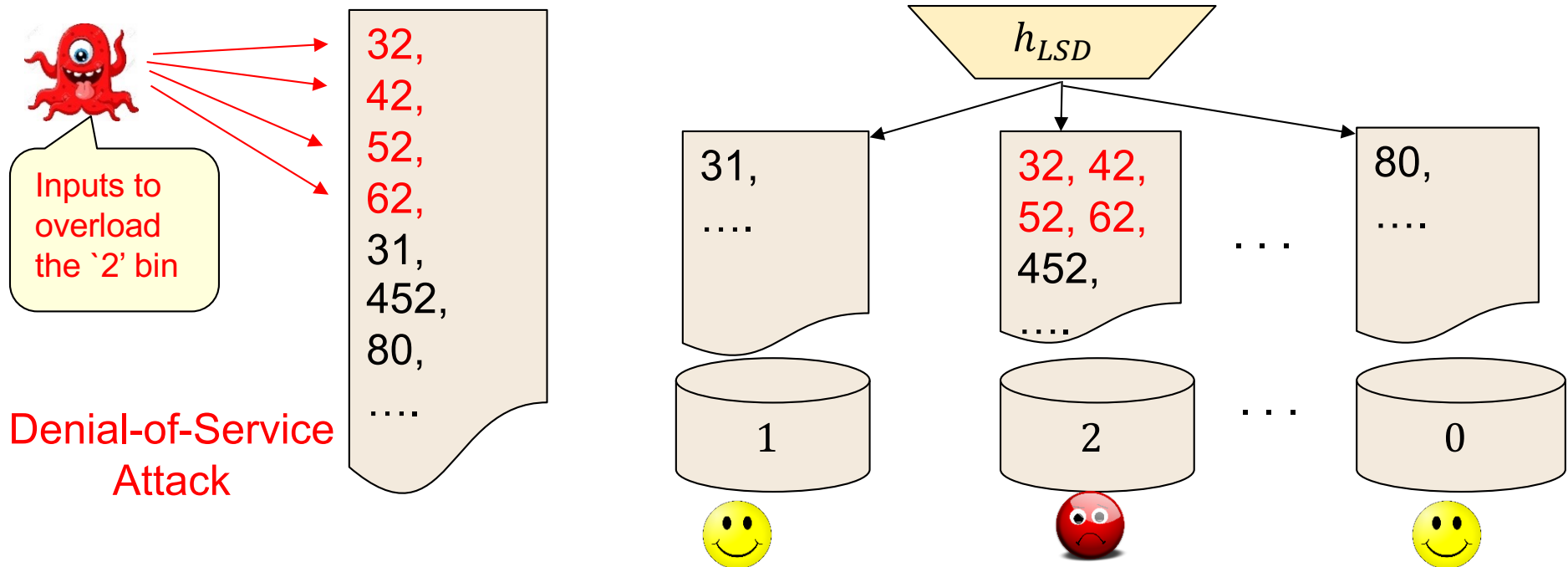
Motivation: Hashing for efficiency

- Input: large set (e.g., integers or strings)
- Goal: map `randomly' to few bins
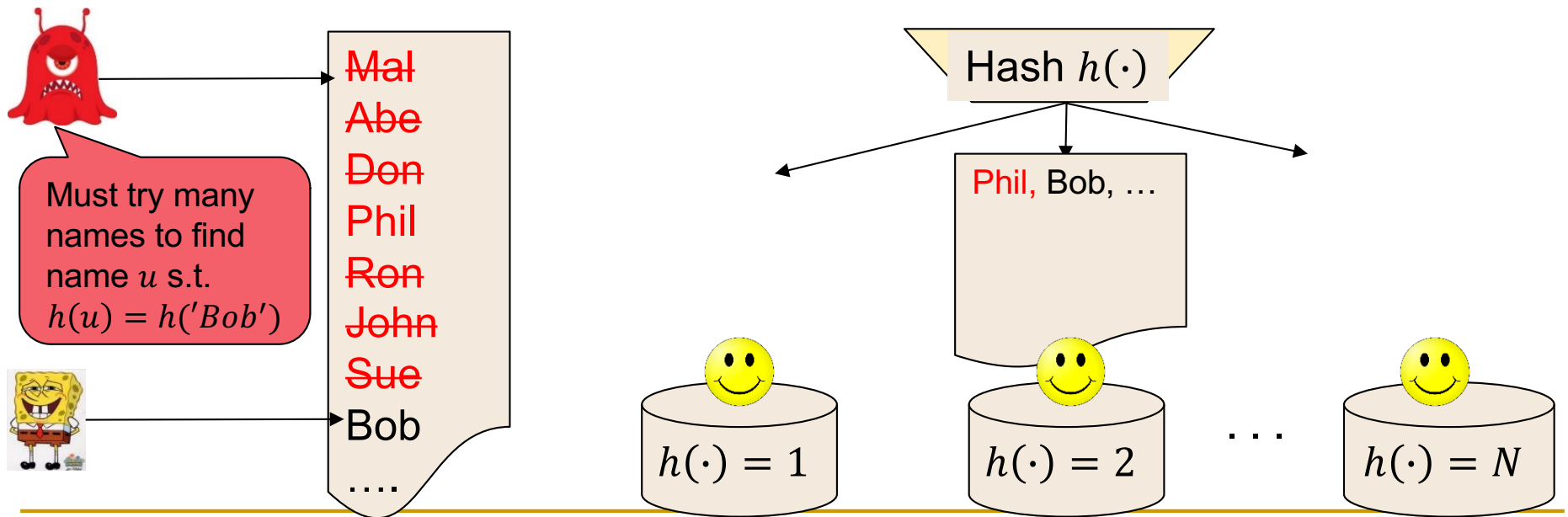 - E.g., to ensure efficiency – load balancing, etc.

341,
3342,
870,
571,
31,
452,
80,
....

h_{LSD}

341,
571,
31,
....

3342,
452,
....

. . .

870,
80,
....

1

2

. . .

0

Collisions?

- Input: large set (e.g., integers or strings)
- Goal: map `randomly' to few bins
  - E.g., to ensure efficiency – load balancing, etc.
 - Adversary chooses inputs that hash to same bin

Inputs to overload the `2' bin

32,
42,
52,
62,
31,
452,
80,
....

Denial-of-Service Attack

h_{LSD}

31,
....

32, 42,
52, 62,
452,
....

. . . .

80,
....

1

2

. . . .

0

Security Goal: Collision Resistance

- A **collision:** two inputs (names) with same hash: $h('Bob')=h('Phil')$

- Every hash has collisions, since |input|>>|output| !

- Collision resistance: hard to **find** these collisions

  - Note: attacker can always try names randomly until a collision is found
  - But this should be ineffective: must try about (on average) N names (number of bins)

Must try many names to find name u s.t. $h(u) = h('Bob')$

Mal
Abe
Don
Phil
Ron
John
Sue
Bob
....

Hash $h(\cdot)$

Phil, Bob, …

$h(\cdot) = 1$ $h(\cdot) = 2$ . . . $h(\cdot) = N$

Collision Resistant Hash Function (CRHF)

- h is CRHF if it is hard to **find** collisions $h(x)=h(x')$
  - Note: attacker can always try inputs randomly till finding collisions
  - But this should be ineffective: must try about $|Range|$ values
- Hard means that the probability that the attacker succeeds in finding a collision is negligible.

1^n

Adversary

Collision (x, x')
s.t. $h(x) = h(x')$

Domain $\{0,1\}^*$

x

x'

Hash function $h(\cdot)$

Range $\{0,1\}^n$
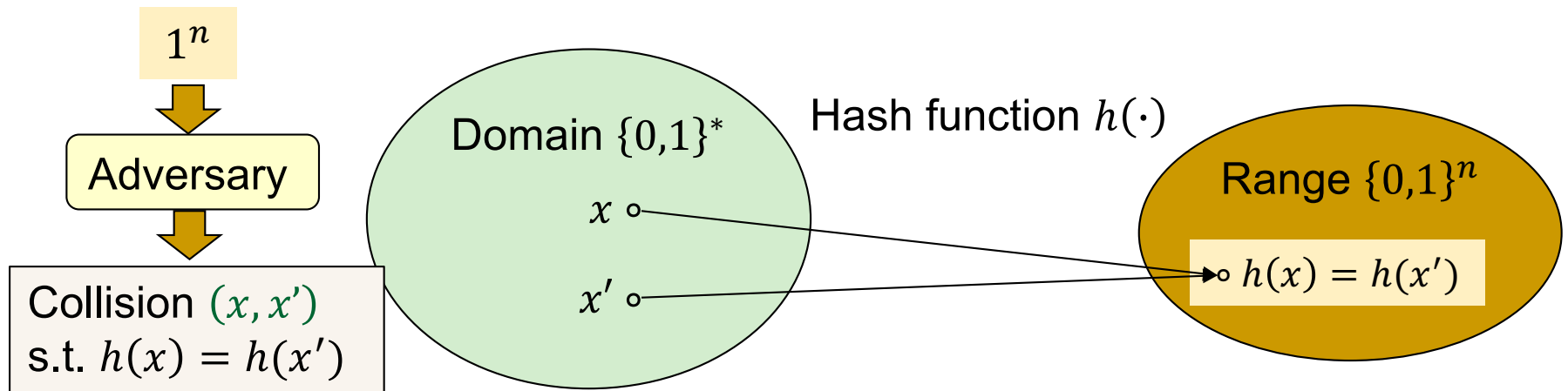
$h(x) = h(x')$

Collision Resistant Hash Function (CRHF)

- h is CRHF if it is hard to **find** collisions $h(x)=h(x')$
  - Note: attacker can always try inputs randomly till finding collisions
  - But this should be ineffective: must try about $|Range|$ values
- Hard means that the probability that the attacker succeeds in finding a collision is negligible.

Definition . (Keyless Collision Resistant Hash Function (CRHF)). *A keyless hash function* $h^{(n)}(\cdot) : \{0,1\}^* \to \{0,1\}^n$ is collision-resistant *if for every efficient (PPT) algorithm* $\mathcal{A}$, *the* advantage $\varepsilon_{h,\mathcal{A}}^{CRHF}(n)$ *is* negligible *in n, i.e., smaller than any positive polynomial for sufficiently large n (as $n \to \infty$), where:*
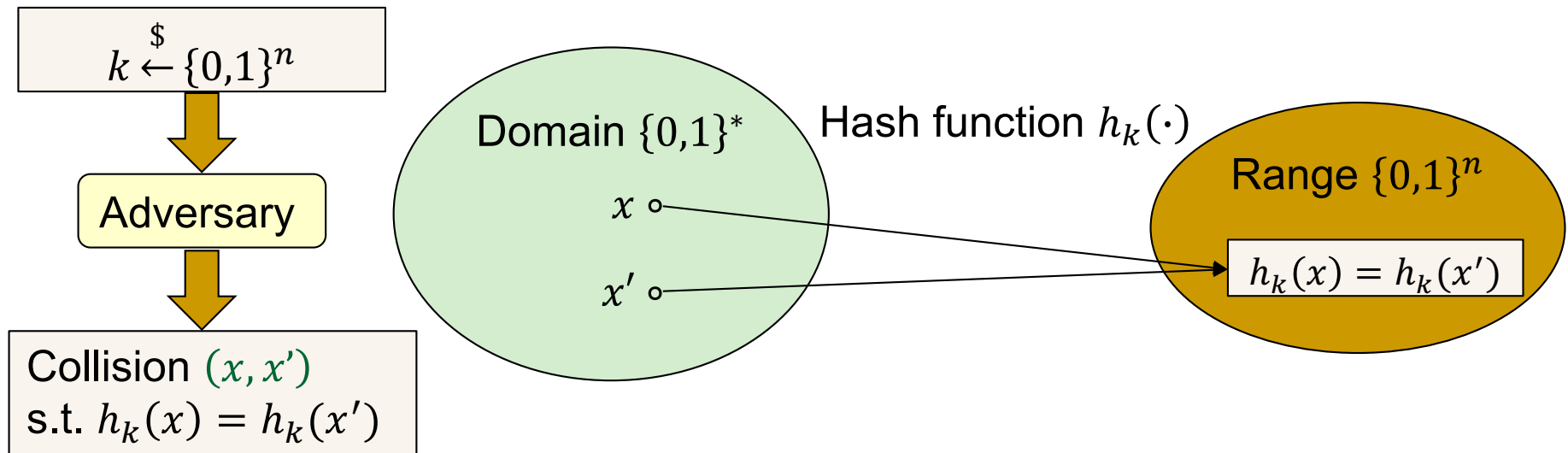
$$\varepsilon_{h,\mathcal{A}}^{CRHF}(n) \equiv \Pr\left[(x,x') \leftarrow \mathcal{A}(1^n) \ s.t. \ (x \neq x') \wedge (h^{(n)}(x) = h^{(n)}(x'))\right]$$

Where the probability is taken over the random coin tosses of $\mathcal{A}$.

Keyless CRHF **Do Not** Exist!

- |Range|<<|Domain| so there <u>is</u> a collision where $h(x')=h(x), x \neq x'$

- For a keyless CRHF there <u>is</u> a PPT algorithm A that can always output a collision: $A(1^n) = \{return\ x, x'\}$

  - Proof: in textbook.

    - Intuitively, since the function is fixed (same input-output mapping), a collision instance can be hardcoded in the attacker algorithm and just output that collision and win the security game.

- Solutions:

  - <u>keyed</u> CRHF,

  - use weaker collision notions,

  - or ignore! (more like asking if the collision is useful for the attacker?)

Keyed CRHF

$$k \xleftarrow{\$} \{0,1\}^n$$

Adversary

Collision (x, x')
s.t. $h_k(x) = h_k(x')$

Domain $\{0,1\}^*$

Hash function $h_k(\cdot)$

Range $\{0,1\}^n$

x

x'

$h_k(x) = h_k(x')$

Adversary knows k but **not in advance** – cannot `know` a collision

Often referred to as **ACR**-hash (**ANY**-collision resistance)

Keyed CRHF - Definition

Definition (Keyed Collision Resistant Hash Function (CRHF)). *Consider a keyed hash function $h_k(\cdot) : \{0,1\}^n \times \{0,1\}^* \to \{0,1\}^n$, defined for any $n \in \mathbb{N}$. We say that h is* collision-resistant *if for every efficient (PPT) algorithm $\mathcal{A}$, the advantage $\varepsilon_{h,\mathcal{A}}^{CRHF}(n)$ is negligible in n, i.e., $\varepsilon_{h,\mathcal{A}}^{CRHF}(n) \in NEGL(n)$, where:*

$$\varepsilon_{h,\mathcal{A}}^{CRHF}(n) \equiv \Pr_{k \leftarrow \{0,1\}^n} \left[(x, x') \leftarrow \mathcal{A}(k) \ s.t. \ (x \neq x') \wedge ((h_k(x) = h_k(x'))) \right]$$

Where the probability is taken over the random coin tosses of the adversary $\mathcal{A}$ and the random choice of k.

Generic Collision Attacks

Domain $\{0,1\}^*$

Hash function $h(\cdot)$

Range $\{0,1\}^n$

x

x'

$h(x) = h(x')$
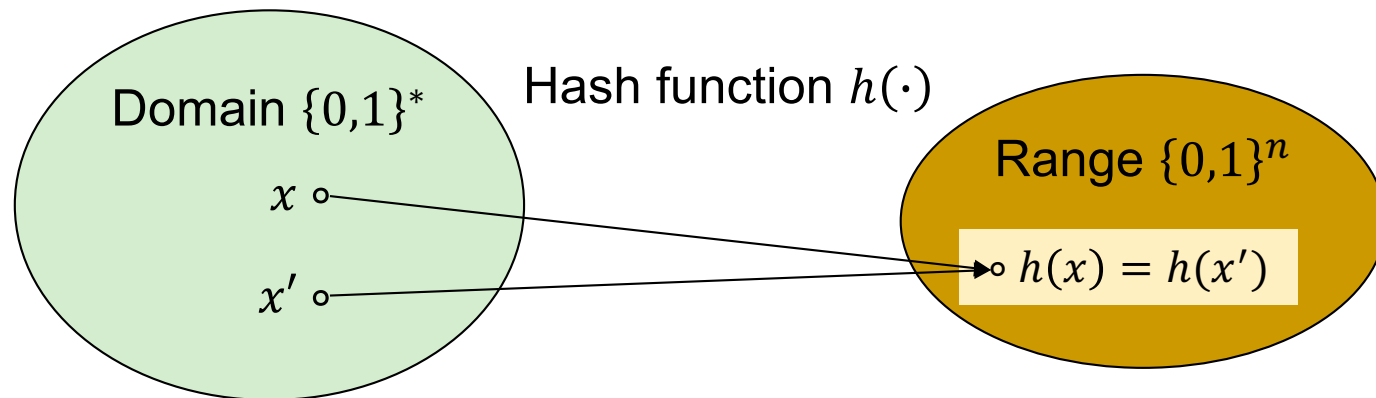
- **An attacker that runs in exponential time can always find a collision (i.e., non PPT attacker)**

  - Easy: find collisions in 2^n time by trying $2^n + 1$ distinct inputs (compute their hash and locate a collision).

- **An attacker finds a collision with 2^{-n} probability (negligible probability).**

 - Choose x and x' at random and check if they produce a collision.

The Birthday Paradox

- **The birthday paradox** states that the expected number q of hashes until a collision is found in $O(2^{n/2})$ not $O(2^n)$.
  - It is $q \lessapprox 2^{n/2} \cdot \sqrt{\dfrac{\pi}{2}} \lessapprox 1.254 \cdot 2^{n/2}$

- For 80 bit of effective security, use n=160 !

  - So to defend against an attacker who can perform 2^{80} hashes set the digest length to be at least 160 bits.

    - So the range has a size of 2^{160} digests.

- Why? Intuition?

The Birthday Attack ('Paradox')

- Probability of NO birthday-collision:

  - Two persons: (364/365)
  - Three persons: (364/365)*(363/365)
  - …
  - n persons: $\prod_{i=1}^{n-1} \frac{365-i}{365}$
 - So for 23 individuals, the probability

  of birthday collision is about 0.5

Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
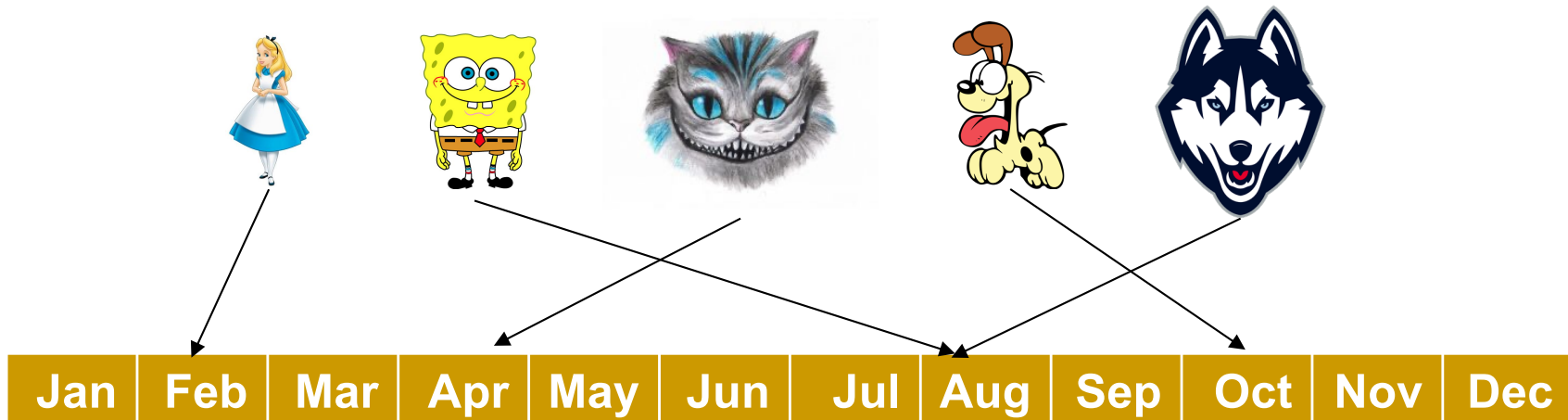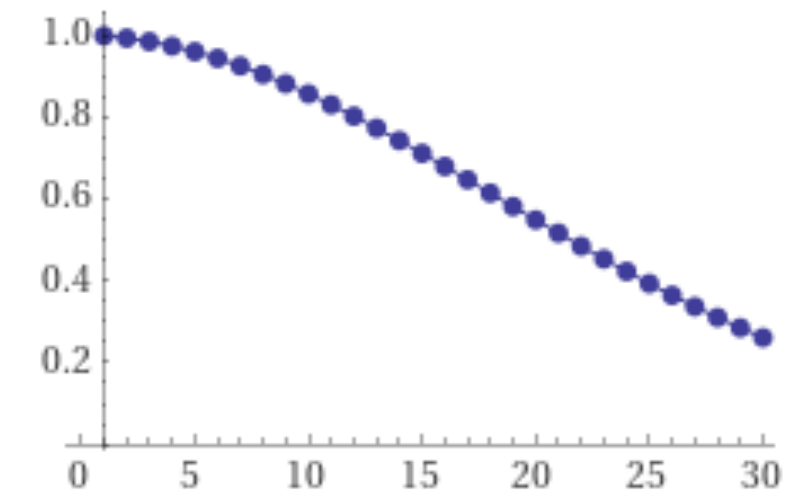
Collision-Resistance: Applications

- **Integrity (of object / file / message)**
  - Send $hash(m)$ securely to validate m
  - Later we will see how a hash function can be used to construct a MAC (called HMAC).

- **Hash-then-Sign**
  - Instead of signing m sign $hash(m)$
    - More efficient!
    - We will explore this in detail once we study digital signatures.

- **Blockchains**
  - Later

CRHF and Software Distribution

- Developer in LA develops a large software m

- Repository in DC obtains a copy of m

- User in NY wants to obtain m – securely and efficiently
  - <u>Don't</u> send m from LA to **both** NY and DC

- How?

NY User

DC

LA

Repository

m
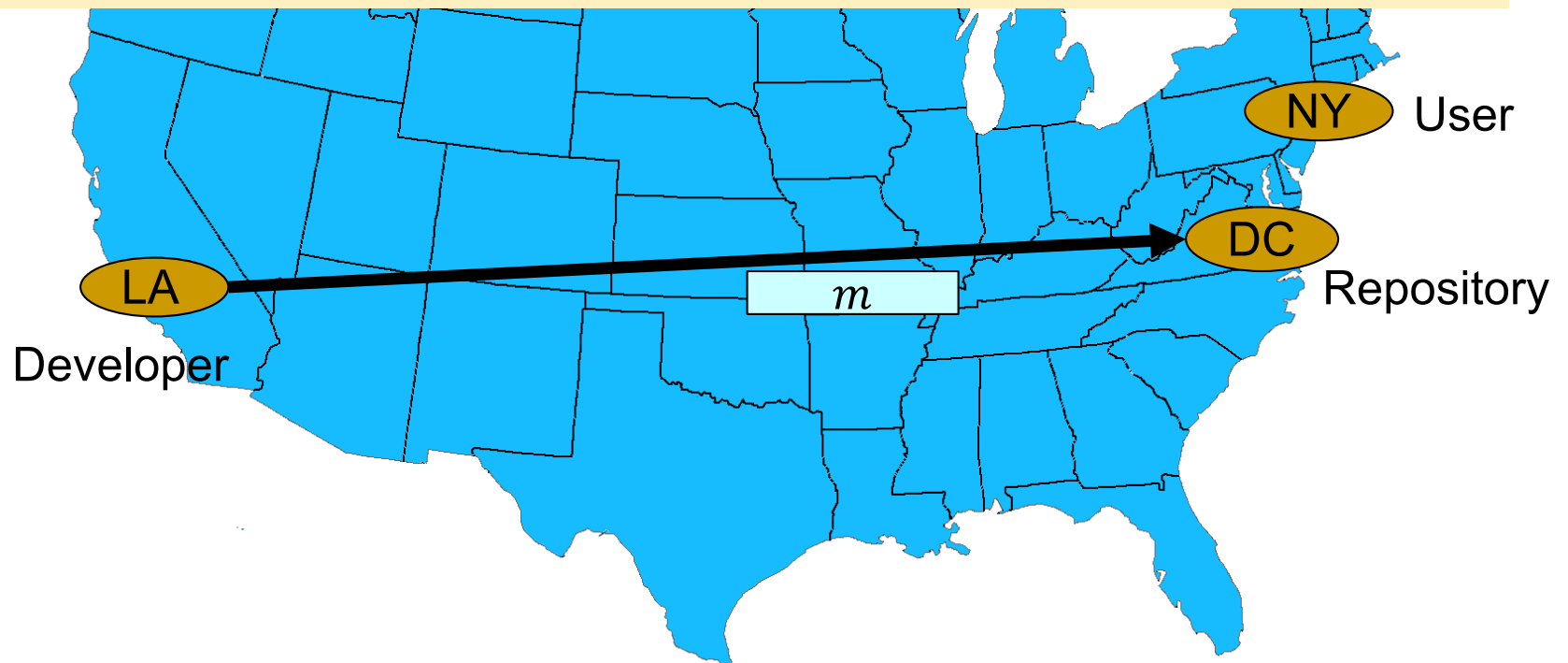
Developer

CRHF: secure, efficient SW distribution

1. Repository in DC downloads software m from developer in LA
2. User downloads from (nearby) repository; receives m'
   - Is $m' = m$? User should validate! How?
3. User securely downloads $h(m)$ directly from developer
   - Digest $h(m)$ is short – much less overhead than downloading m
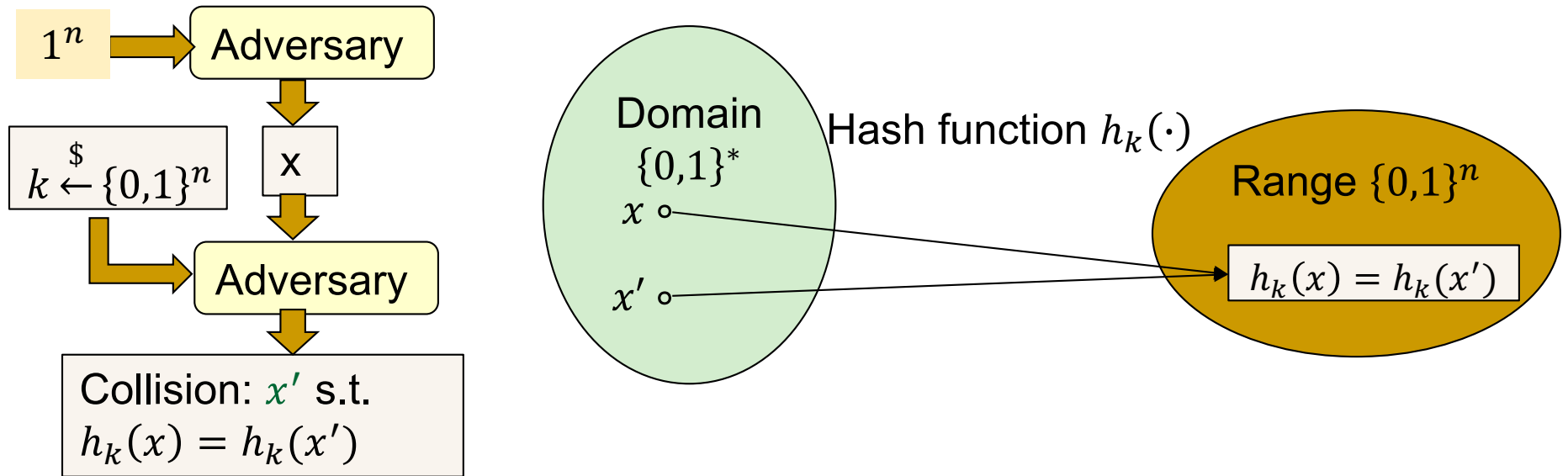4. User validates: $h(m) = h(m') \Rightarrow m = m'$
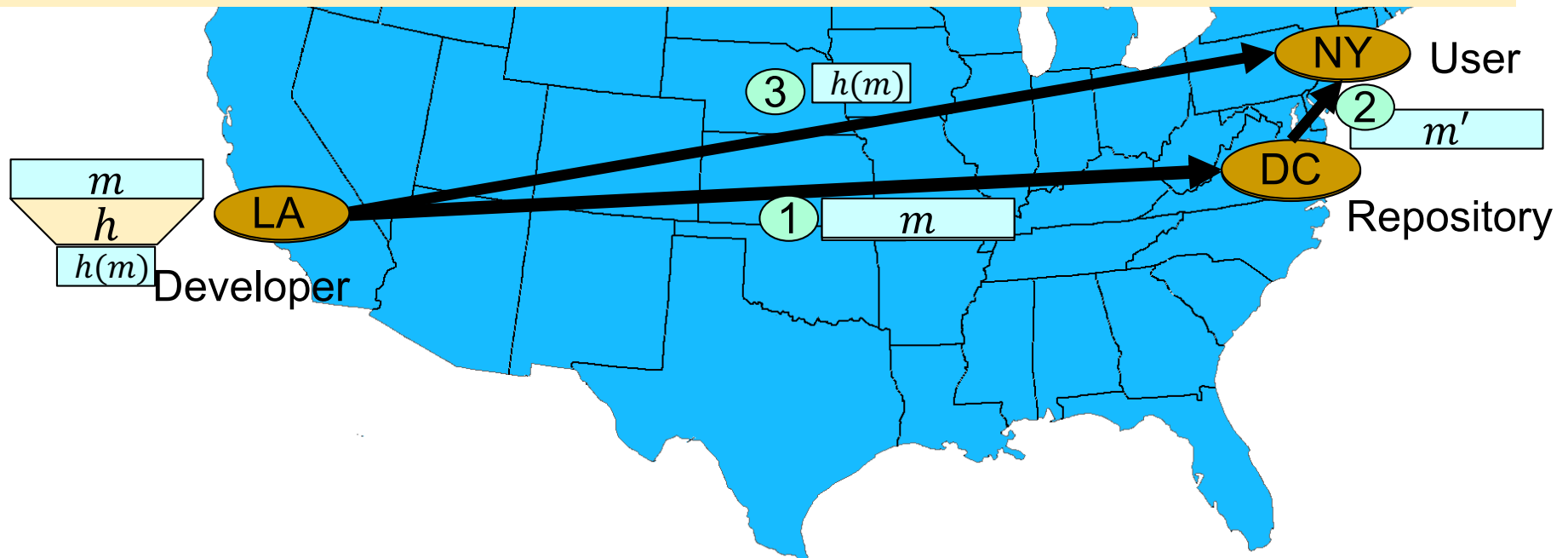
Other Notions of Security

- **Collision resistance provides the strongest guarantee.**

  - Gives more freedom to the adversary; the adversary wins if it finds any two inputs with the same digest.

    - No conditions on these two inputs other than being in the domain of the hash function.

- **Other security notions (but sufficient for many applications):**

  - Target collision resistance (TCR).

  - Second preimage resistance.

  - First preimage resistance.

- **Birthday paradox (or attack) does not work against these weaker notions.**

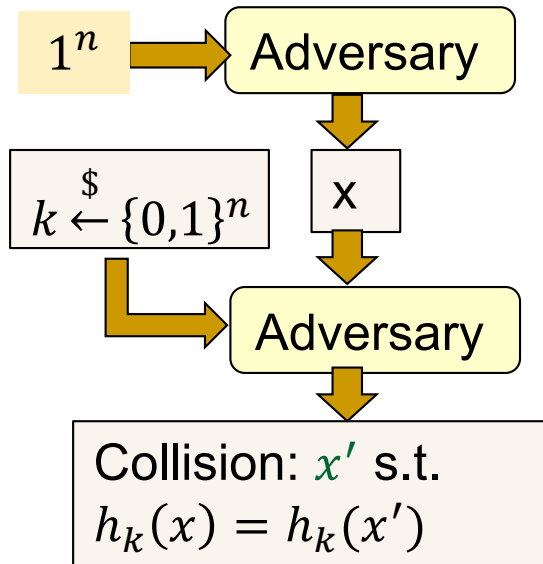 - It is for collision resistance; find *any* two inputs that collide!

Target CRHF (TCR Hash Function)

1^n ➡ Adversary

$k \xleftarrow{\$} \{0,1\}^n$ x

Adversary

Collision: x' s.t.
$h_k(x) = h_k(x')$

Domain $\{0,1\}^*$

Hash function $h_k(\cdot)$

x

x'

Range $\{0,1\}^n$

$h_k(x) = h_k(x')$

Adversary has to select target **before** knowing key

$$\varepsilon_{h,\mathcal{A}}^{TCR}(n) \equiv \Pr_{k \leftarrow \{0,1\}^n}\left[\left\{ \begin{array}{l} x \leftarrow A(1^n); \\ x' \leftarrow A(x,k) \end{array} \right\} \; s.t. \; (x \neq x') \wedge (h_k(x) = h_k(x')) \right]$$
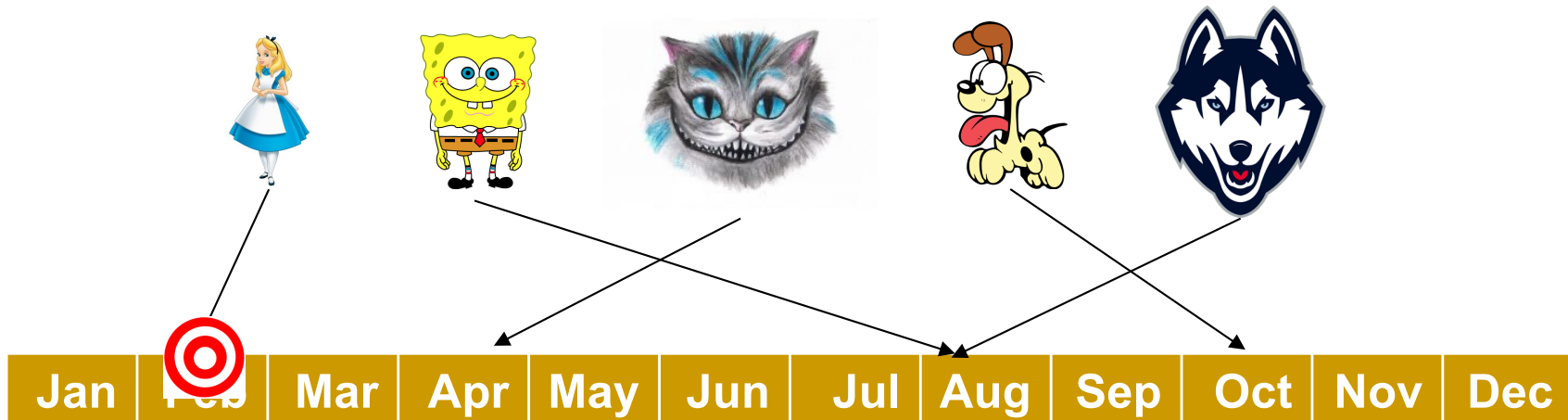
TCR and Birthday Paradox?

1^n → **Adversary**

$k \overset{\$}{\leftarrow} \{0,1\}^n$ **x**

Adversary

Collision: x' s.t. $h_k(x) = h_k(x')$

- **First:** adversary selects **x**
- Probability for NO birthday-collision **with x:**
  - Two persons: (364/365)
  - Three persons: (364/365)*(36**4**/365)
  - …
  - n persons: $\prod_{i=1}^{n-1} \frac{364}{365} = \left(\frac{364}{365}\right)^{n-1}$

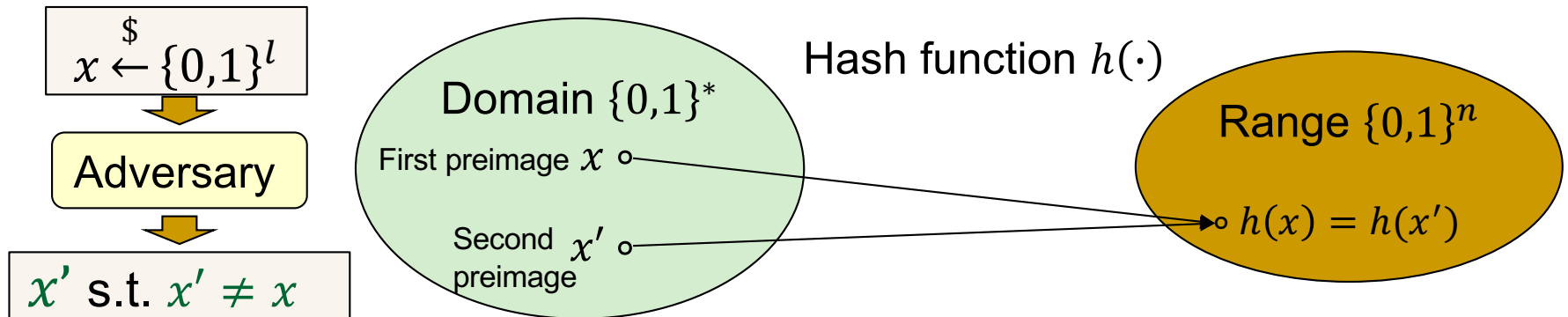| Jan | Feb | Mar | Apr | May | Jun | Jul | Aug | Sep | Oct | Nov | Dec |

We (mostly) focus on keyless hash…

- Although there are no CRHFs
- And theory papers focus on keyed hash
- But…
  - It's a bit less complicated and easier to work with.
  - No need to consider both ACR (this stands for Any Collision Resistance, which is the same as CRHF) and TCR
    - Why?
  - Modifying to CRHF is quite trivial
    - Just make it keyed!
  - Usually used in practice: libraries, standards, …

2nd-Preimage-Resistant Hash (SPR)

- Hard to find collision with a <u>specific random</u> x.

$$x \overset{\$}{\leftarrow} \{0,1\}^l$$

Adversary

$$x' \text{ s.t. } x' \neq x$$

Hash function $h(\cdot)$

Domain $\{0,1\}^*$

First preimage x

Second preimage x'

Range $\{0,1\}^n$

$h(x) = h(x')$

$$\varepsilon_{h,\mathcal{A}}^{SPR}(n) \equiv \Pr_{x \overset{\$}{\leftarrow} \{0,1\}^{A(1^n)}} \left[x' \leftarrow \mathcal{A}(x) \ s.t. \ x \neq x' \ \wedge \ h(x) = h(x') \right]$$

Use with care!
(think carefully about the security you want to achieve and see if SPR suffices)

CRHF/SPR vs. Applications

- CRHF secure for signing, SW-distribution

- How about SPR hash (weak-CRHF)?

  - SW-distribution? *YES*

  - Hash-then-sign? *NO*

- Why?

  - Attacker can't impact SW to be distributed

  - But… attacker may be able to impact signed msg!

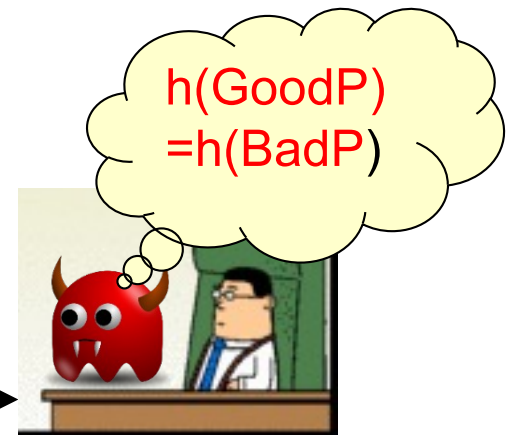Will/Contract/…

Sign(h(Will/Contract/…))
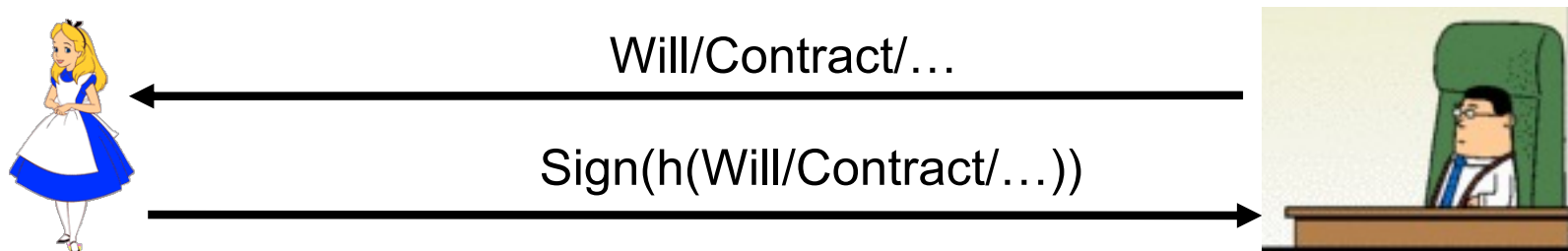
SPR: Collisions to **Chosen** Messages

- Or: Alice and Mal, the corrupt lawyer

- Mal finds two `colliding wills', GoodW and BadW:

  - GoodW: contents agreeable to Alice

  - h(GoodW)=h(BadW)

  - Alice Signs good will: Sign(h(GoodW))
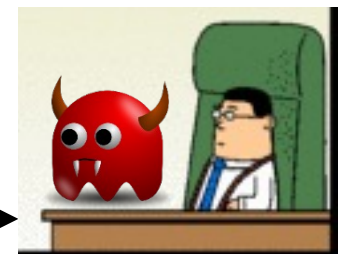
h(GoodP) =h(BadP)

GoodW: 'I leave all to Bob'

$Sign_A(h(GoodW))$

- Later… Mal presents to the court:

BadW: 'I leave all to Mal', $Sign_A(h(BadW))$

$$$$

SPR: collisions to **chosen** message

- Or: Alice and Mal, the corrupt lawyer
- Mal finds two `colliding wills', GoodW and BadW:
  - GoodW: contents agreeable to Alice

P)

P)

Is such attack realistic?
Or SPR is enough 'in practice'?

BadW: 'I leave all to Mal', Sign$_A$(h(BadW))

$$$$

SPR & Chosen-prefix Vulnerability

- **Chosen-prefix vulnerability:**
  - Mal selects `prefix string' p
  - Efficient algorithm finds:

  $x \neq x'$ s.t. $h(p||x) = h(p||x')$

  - Or, also for <u>any</u> suffix: $(\forall s)h(p||x||s) = h(p||x'||s)$
- Hash may be SPR yet allow chosen-prefix attacks
- Such attacks found for several proposed, standard cryptographic hash function, e.g., MD5 and SHA1
- We show chosen prefix attack on HtS
  - <u>Example of possible attack on HtS with SPR</u>

Chosen-prefix Attack

- Let $x < x'$ be collision for prefix: $p=$ `Pay Mal $'
- Mal tricks Alice into signing him an IOU for $\$x$
- Alice signs, sends $s=S_s^h(m)$ where $m=$ `Pay Mal $'$||x$
    - $S_s^h(m)=S_s(h(p||x))=S_s(h(p||x'))=S_s^h(m')$
        - $m'=$ `Pay Mal $'$||x'$
- Mal sends s, m' to Alice's bank
    - Bank validates $"Ok"=Verify_{Alice.v}(m',s)$
- Bank gives $\$x'$ of Alice to Mal!!
- This attack is simplified:
    - Mal has to find `good' collision (high profit, convince Alice to sign)
    - People sign (PDF) files, not plain text…
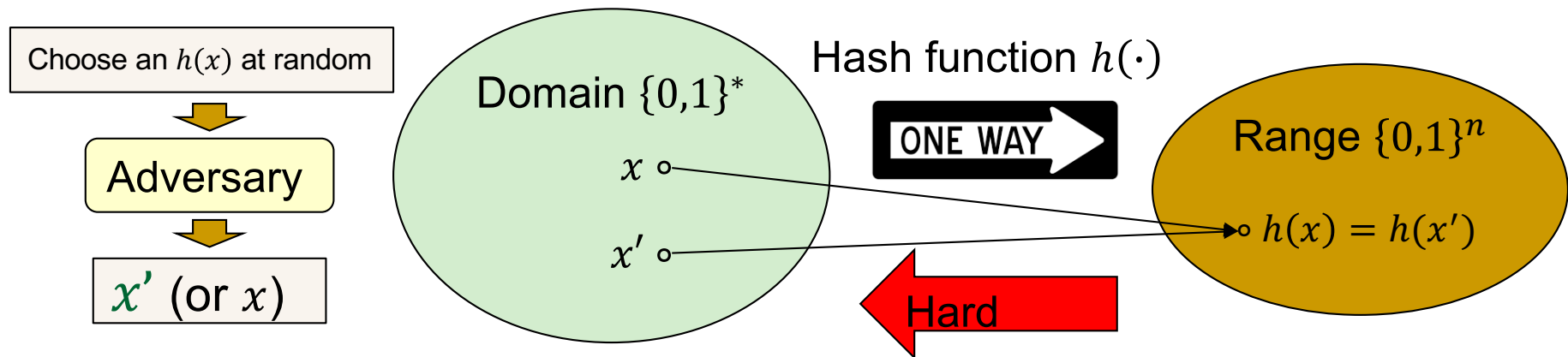- In reality, attacker also chooses suffix ➜ stronger attack

Examples

1)   Let h_k be a keyed CRHF. Is $h_k' = h_k(h_k(x))$ a CRHF? Why?

2)   For x parsed as x = $x_1 || x_2 || x_3$, let $h(x_1 || x_2 || x_3) = x_1 + x_2 + x_3 \bmod p$, is h a CRHF? Why? Is it an SPR? Why?

3)   Let $h_k(m)$ be a TCR function. Construct $h_k'(m) = 0^n$ if $m[1: |k|] = k$ and $h_k(m)$ otherwise (recall m, the hash function input, can be of any length).

   ❑ Is h_k' a CRHF? Why?
   ❑ Is h_k' a TCR? Why?

One-Way Function (OWF)

Choose an $h(x)$ at random

Adversary

x' (or x)

Hash function $h(\cdot)$

Domain $\{0,1\}^*$

x

x'

ONE WAY

Hard

Range $\{0,1\}^n$

$h(x) = h(x')$

- **One-way function** or **first preimage resistance**: given $h(x)$ for a <u>random</u> x, it is hard to find x, <u>or any x'</u> s.t. $h(x')=h(x)$
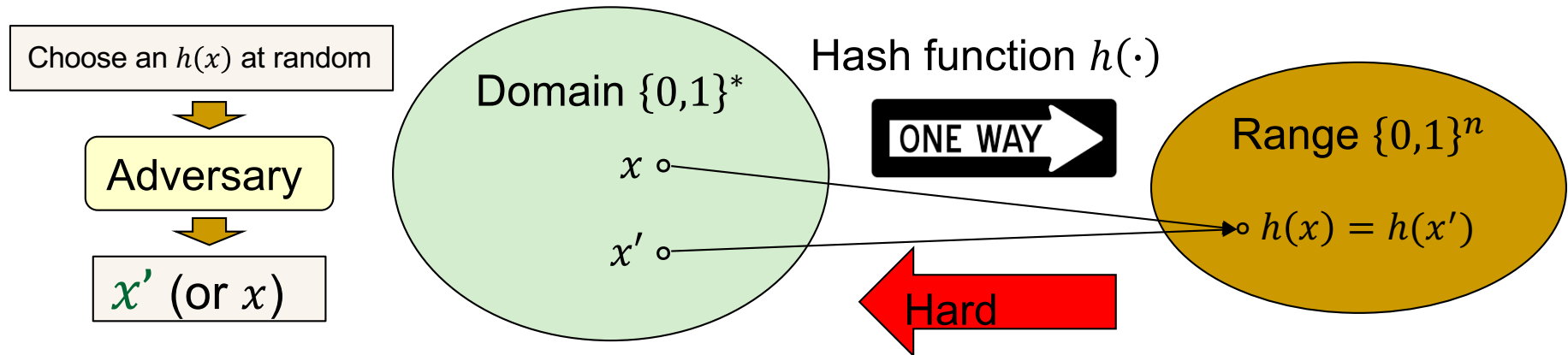
Compare to:

- Collision-Resistance (CRHF): hard to find any collision, i.e., any (x, x') s.t. $h(x')=h(x), x \neq x'$

- Second-preimage resistance (SPR): hard to find a collision with <u>random</u> x, i.e., x' s.t. $h(x')=h(x), x \neq x'$

Note: *the textbook has a typo in formula 3.17, page 201. The attacker A takes h(x) as input not x. Also, the above notion is a simplified version of what is in the textbook!*

Application: One-time Password Authentication

Choose an $h(x)$ at random

Adversary

x' (or x)

Domain $\{0,1\}^*$

x

x'

Hash function $h(\cdot)$

ONE WAY

Hard

Range $\{0,1\}^n$

$h(x) = h(x')$

❑ **One-time password authentication:**

- Select random x : 'one-time password' (keep secret!)

- Validate using non-secret 'one-time validation token': *h(x)*

Exercise

- Let $h_1,\ h_2$ be <u>both</u> CRHF and OWF
- Use them to construct:
    - h_{CRHF} - CRHF but <u>not</u> OWF
    - h_{OWF} - OWF but not CRHF

Exercise

- Let h_1, h_2 be <u>both</u> CRHF and OWF
- Use them to construct:
  - h_{CRHF} - CRHF but <u>not</u> OWF
  - h_{OWF} - OWF but not CRHF
- One possible solution:
  - $h_{CRHF}(m) = \begin{cases} 1\,\|m \ \ if \ |m| = n \\ 0\|h_1(m) \ \ otherwise \end{cases}$

  - $h_{OWF}(m) = \begin{cases} h_1(m) \ if \ |m| = n \\ h_1\big(m_{1..n} \oplus h_2(m')\big) \ if \ m = m_{1..n}\|m' \end{cases}$
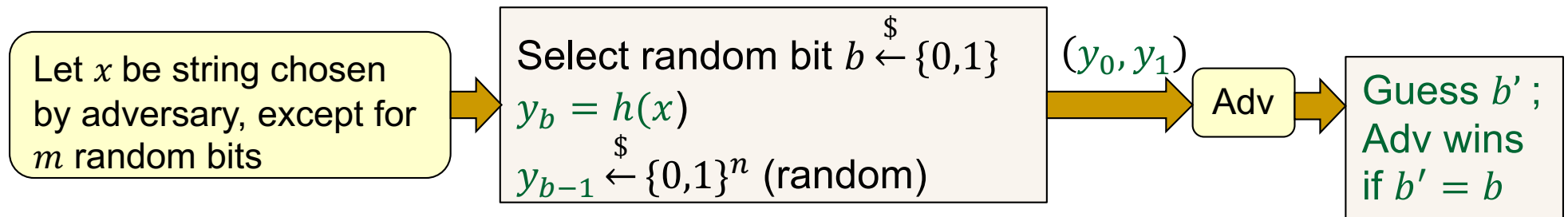
What does all of that tell you about …

- *… the relation between CRHF, TCR, SPR, and OWF?*
  - CRHF vs. TCR?
  - CRHF vs. SPR?
  - TCR vs. SPR?
  - CRHF vs. OWF?
  - SPR vs. OWF?

Randomness Extraction

Let x be string chosen by adversary, except for m random bits

Select random bit $b \overset{\$}{\leftarrow} \{0,1\}$
$y_b = h(x)$
$y_{b-1} \overset{\$}{\leftarrow} \{0,1\}^n$ (random)

(y_0, y_1)

Adv

Guess b' ;
Adv wins
if $b' = b$

- ❑ 'If input is sufficiently random, then output is random'

- ❑ Multiple `sufficiently random' models

- ❑ **Randomness extraction**: if any m input bits are random ➔ all n output bits are pseudorandom

  - ▪ For sufficiently large m

  - ▪ **Pseudorandom:** it is not computationally-feasible to distinguish between these bits and truly random bits

Random Oracle Model (ROM)

- Use a fixed, keyless hash function h
- Analyse security <u>as if</u> hash $h()$ is a *random function*
  - An invalid assumption: $h()$ is fixed!
  - Whenever $h()$ is used, use oracle (black box) for random function
- Good for screening insecure solutions
  - Random oracle security ➜ many attacks fail
- In practice: assume random oracle and use a standard hash function
  - It was shown that in some cases the construction will become insecure.
- Better to have security with standard assumption than the non-standard ROM.

Covered Material From the Textbook

- Chapter 3
  - Section 3.1,
  - Section 3.2 (except 3.2.6 and 3.2.7),
  - Section 3.3,
  - Section 3.4 (except 3.4.2),
  - Section 3.5 (only its introduction, i.e., all text before 3.5.1),
  - Section 3.6 (only what was covered in class)

Thank You!