

CSE 3400 - Introduction to Computer & Network Security  
(aka: Introduction to Cybersecurity)

# Lecture 6

## Hash Functions – Part I

Ghada Almashaqbeh  
UConn

From Textbook Slides by Prof. Amir Herzberg  
UConn

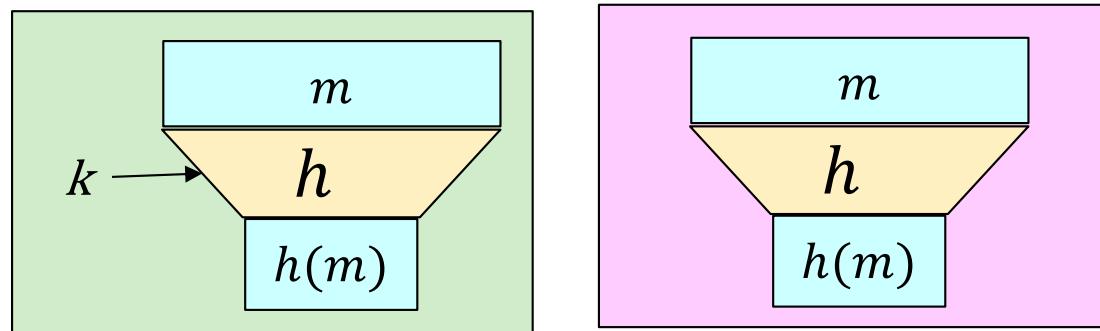
---

# Outline

- ❑ Introduction and motivation.
- ❑ Collision resistant hash functions (CRHF).
- ❑ CRHF applications.
- ❑ Weaker 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:

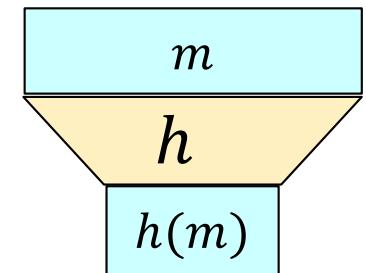
$$h_{LSD}(m) = m_3$$

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

- Exercise:

$$h_{LSD}(117) = \underline{7}$$

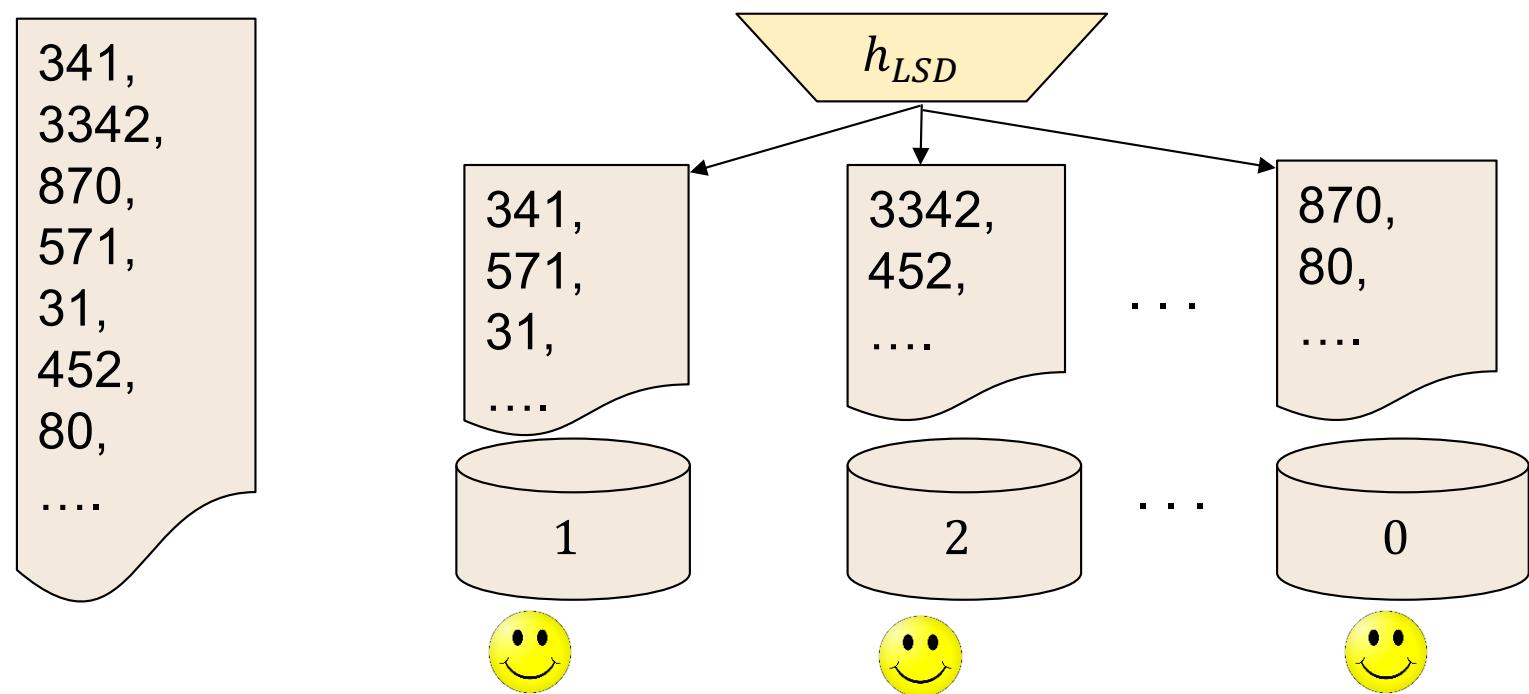
$$h_{Sum}(117) = \underline{9}$$



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

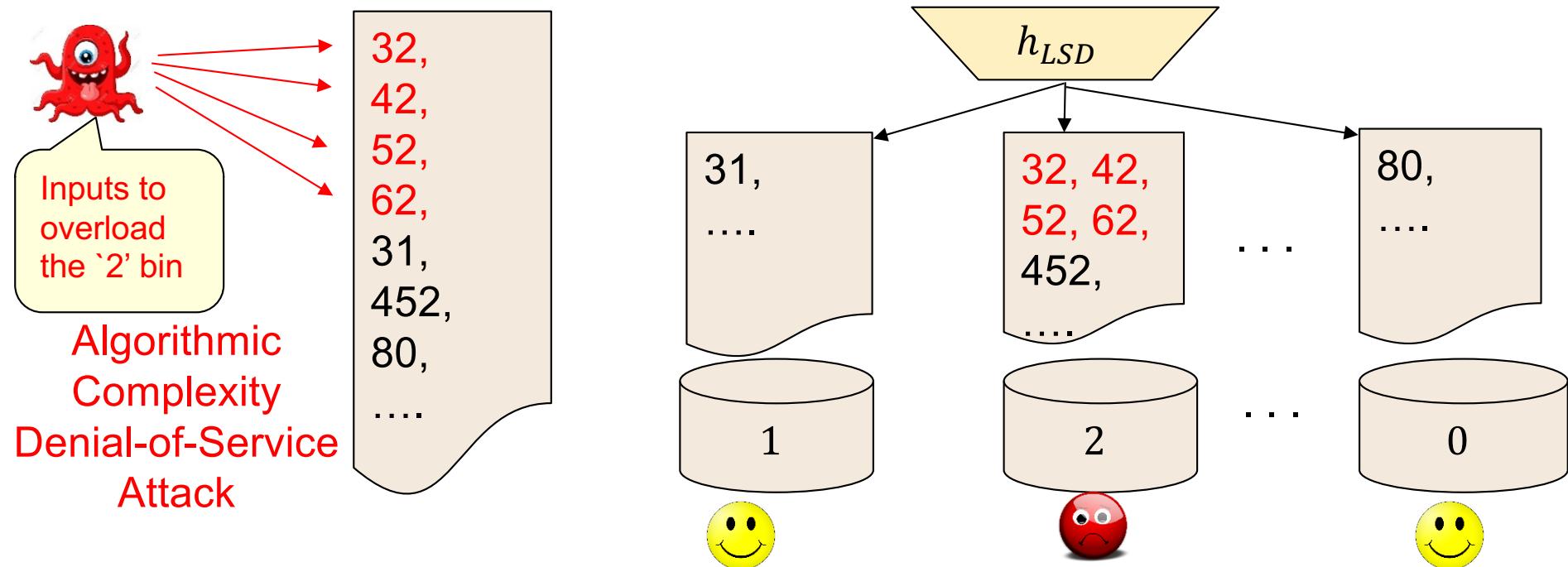
# Motivation: Hashing for efficiency

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



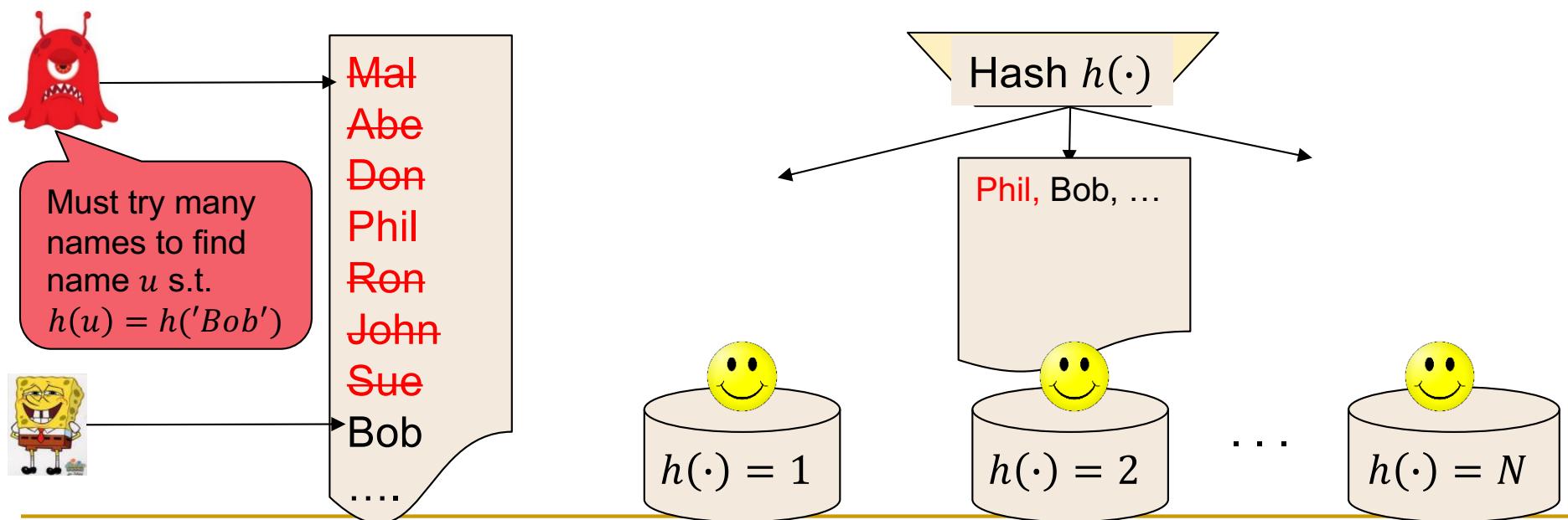
# Collisions?

- 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



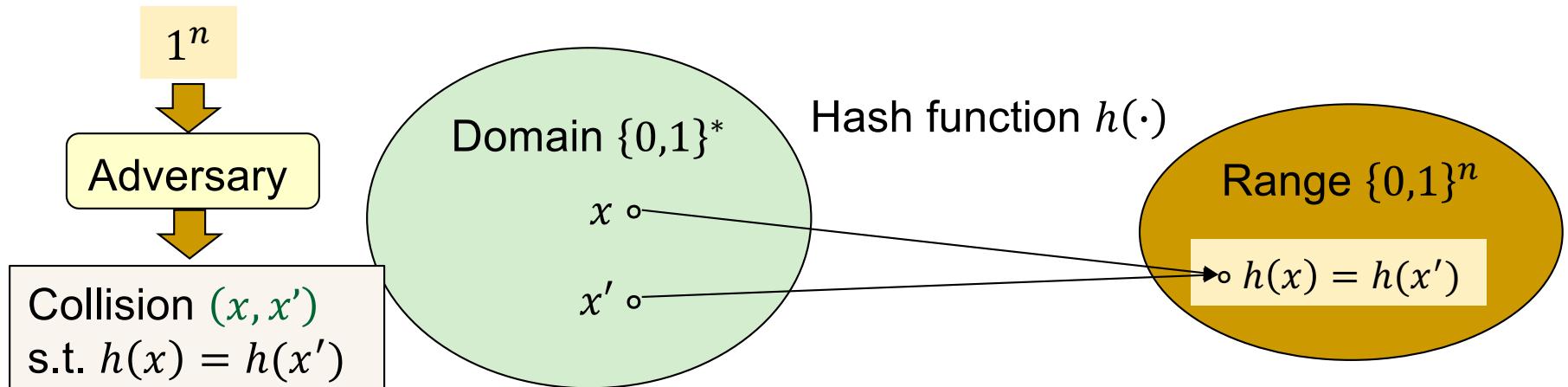
# Security Goal: Collision Resistance

- A **collision**: two inputs (names) with same hash:  
 $h('Bob')=h('Phil')$
- Every hash has collisions, since  $|\text{input}| >> |\text{output}|$  !
- Collision resistance: hard to **find** 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)



# 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.



# 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 4.1** (Keyless Collision Resistant Hash Function (CRHF)). *A keyless hash function  $h^{(n)}(\cdot) : \{0, 1\}^* \rightarrow \{0, 1\}^n$  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  $n \rightarrow \infty$ ), where:*

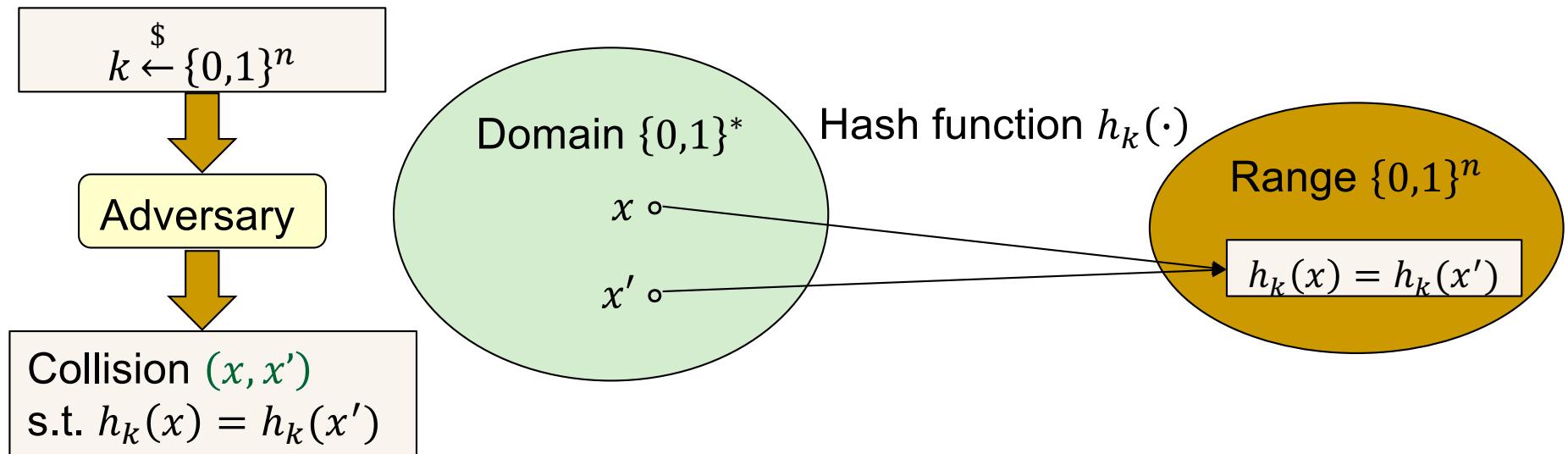
$$\varepsilon_{h, \mathcal{A}}^{CRHF}(n) \equiv \Pr \left[ (x, x') \leftarrow \mathcal{A}(1^n) \text{ s.t. } (x \neq x') \wedge (h^{(n)}(x) = h^{(n)}(x')) \right] \quad (4.1)$$

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

# Keyless CRHF Do Not Exist!

- $|\text{Range}| << |\text{Domain}|$  so there is a collision where  $h(x') = h(x)$ ,  $x \neq x'$
- For a keyless CRHF is a PPT algorithm A that can always output a collision:  $A(1^n) = \{\text{return } x, x'\}$ 
  - 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:**
  - keyed CRHF,
  - Use functions that support weak-collision-resistance,
  - or ignore! (more like asking if the collision is useful for the attacker?)

# Keyed CRHF



Adversary knows  $k$  but **not in advance** –  
cannot ‘know’ a collision

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

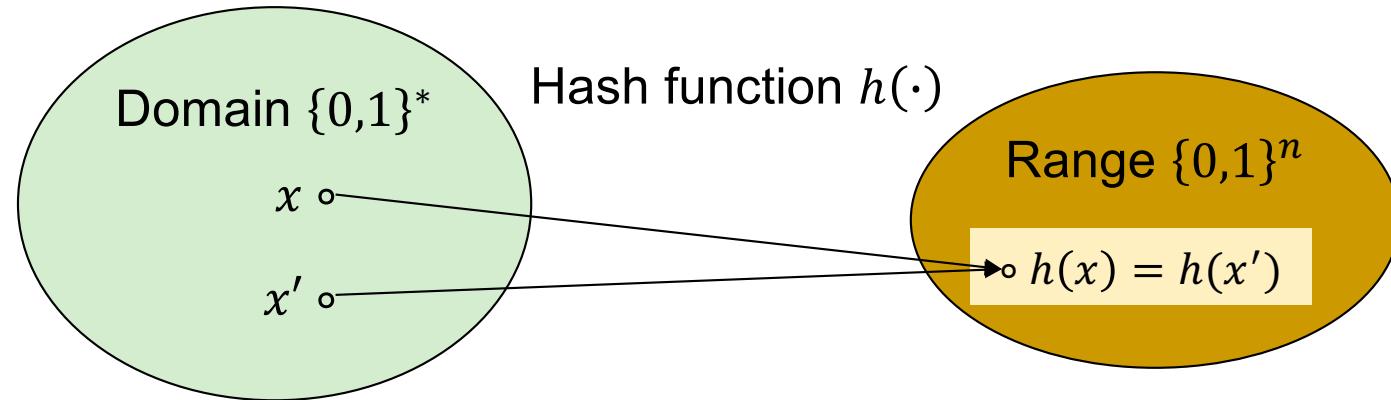
# Keyed CRHF - Definition

**Definition 4.3** (Keyed Collision Resistant Hash Function (CRHF)). *A keyed hash function  $h_k(\cdot) : \{0, 1\}^* \times \{0, 1\}^* \rightarrow \{0, 1\}^*$  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  $n \rightarrow \infty$ ), where:*

$$\varepsilon_{h, \mathcal{A}}^{CRHF}(n) \equiv \Pr_{k \leftarrow \{0, 1\}^n} [(x, x') \leftarrow \mathcal{A}(k) \text{ s.t. } (x \neq x') \wedge (h_k(x) = h_k(x'))] \quad (4.2)$$

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

# Generic Collision Attacks



- 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.

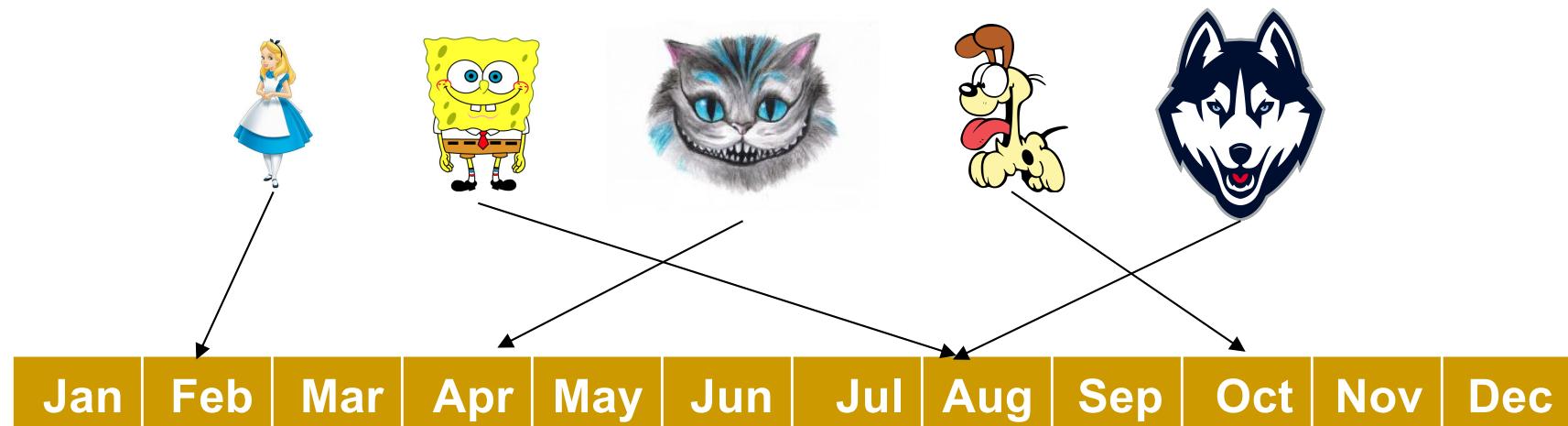
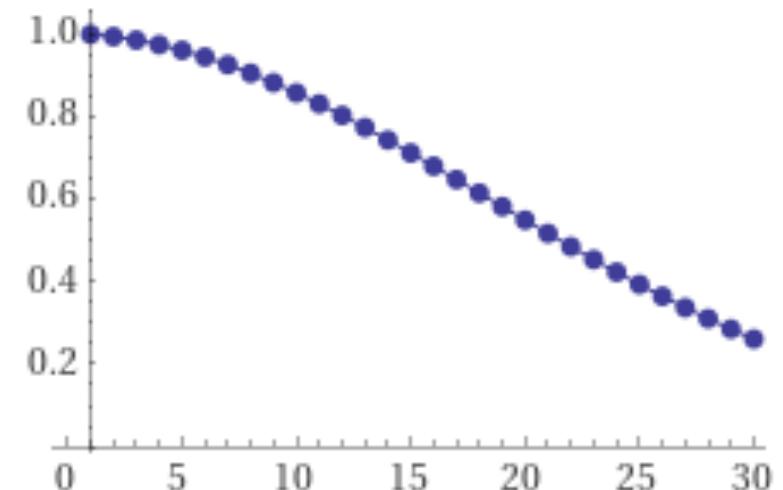
# The Birthday Paradox

- **The birthday paradox** states that expected number  $q$  of hashes until a collision is found is  $O(2^n)$  not  $O(2^n)$ .
  - It is  $q \approx 2^{n/2} \cdot \sqrt{\frac{\pi}{2}} \approx 1.254 \cdot 2^{n/2}$
- 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:  $\prod_{i=1}^{n-1} \frac{365-i}{365}$

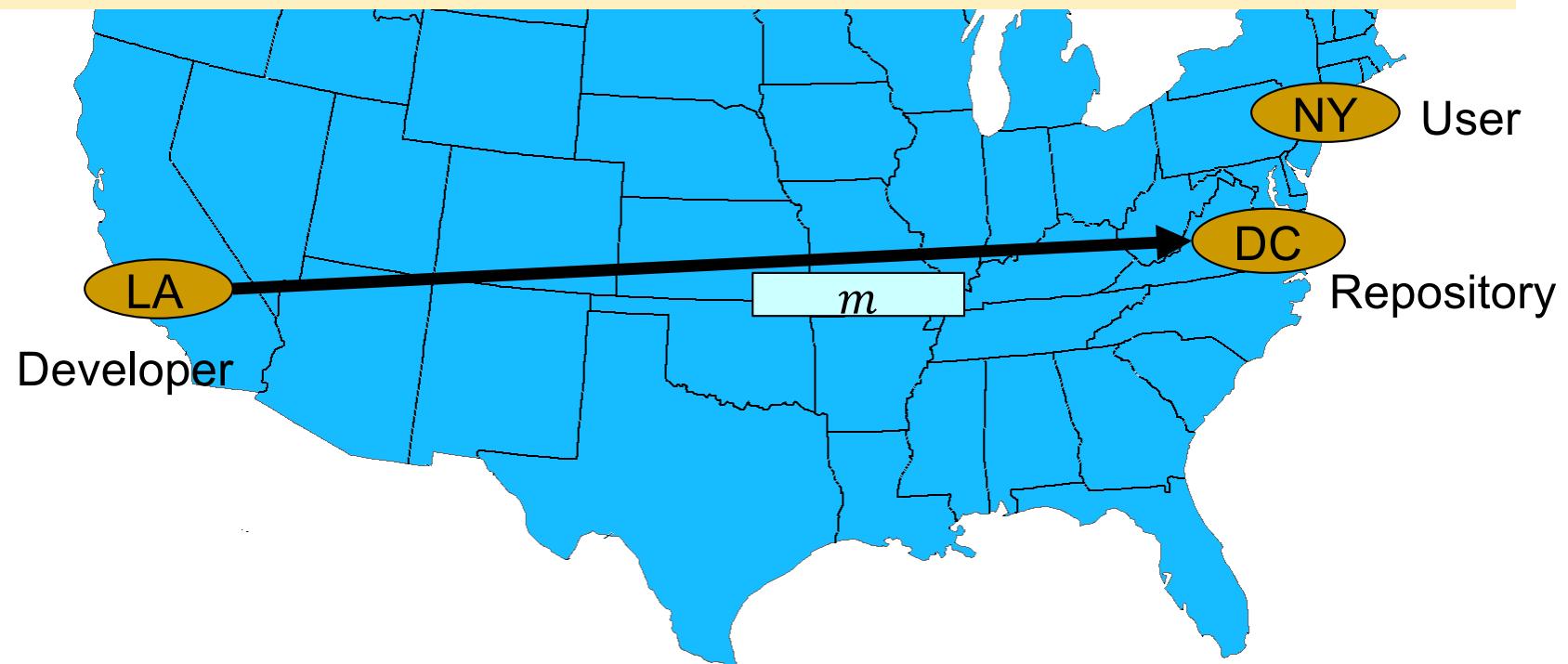


# Collision-Resistance: Applications

- 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

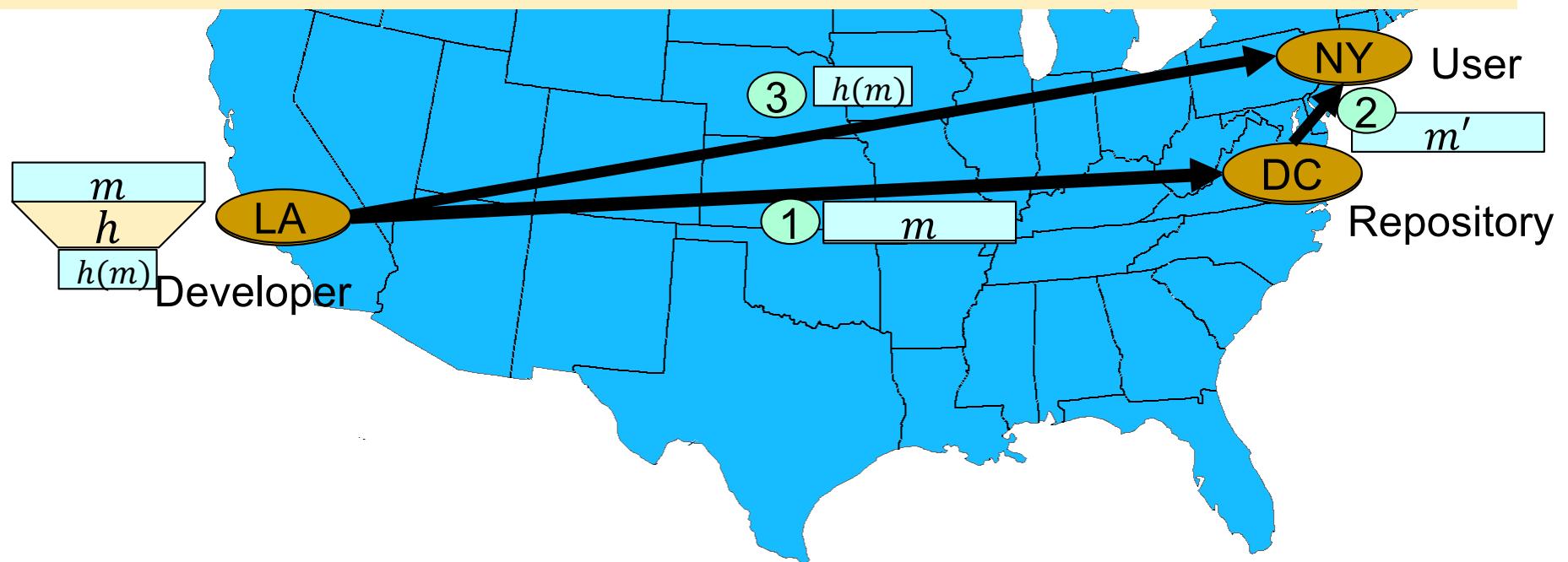
# CRHF and Software Distribution

- ❑ Developer in LA develops large software  $m$
- ❑ Repository in DC obtains copy of  $m$
- ❑ User in NY wants to obtain  $m$  – securely and efficiently
  - Don't send  $m$  from LA to **both** NY and DC
- ❑ How?



# CRHF: secure, efficient SW distribution

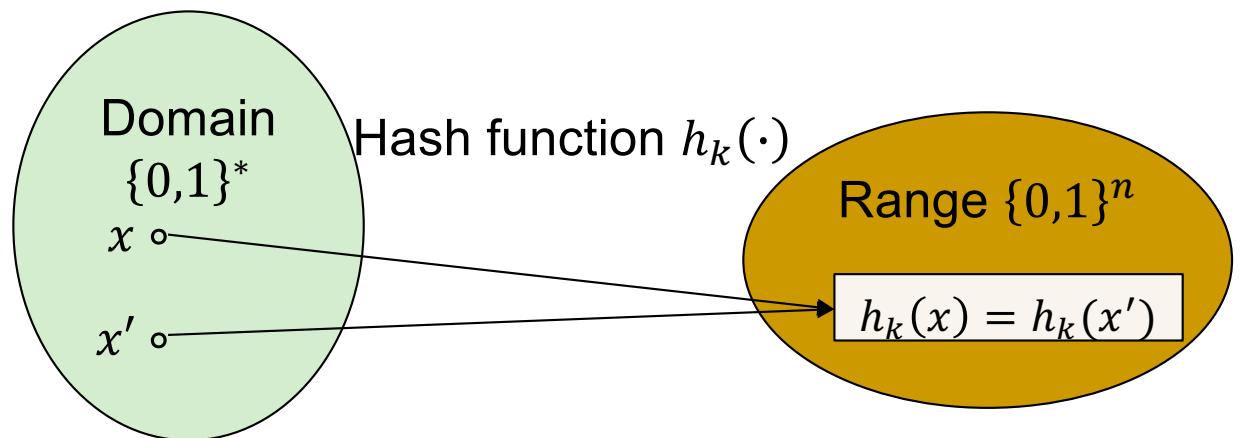
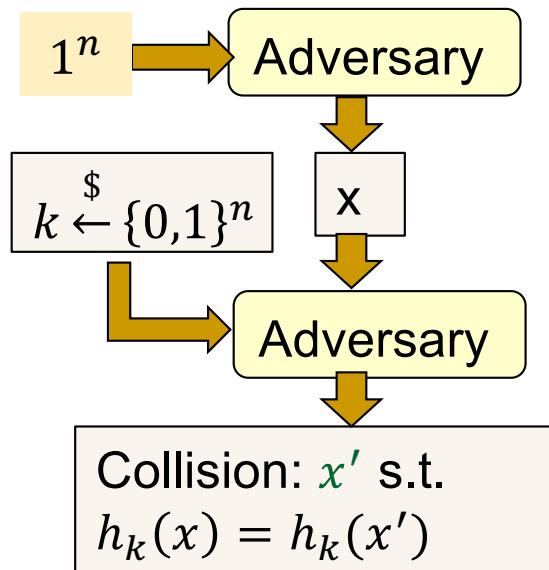
1. Repository in DC downloads software  $m$  from developer in LA
2. User download from (nearby) repository; receives  $m'$ 
  - ❑ Is  $m' = m$  ? User should validate! How?
3. User securely downloads  $h(m)$  directly from developer
4. User validates:  $h(m) = h(m') \rightarrow m = m'$



# Weaker Notions of Security

- Collision resistance provide 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.
- Weaker security notions (but sufficient for many applications):
  - Target collision resistance (TCR).
  - Second preimage (or target-collision) 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!

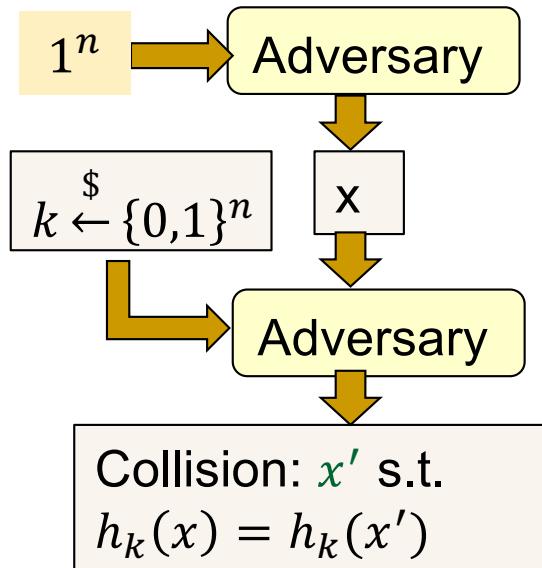
# Target CRHF (TCR Hash Function)



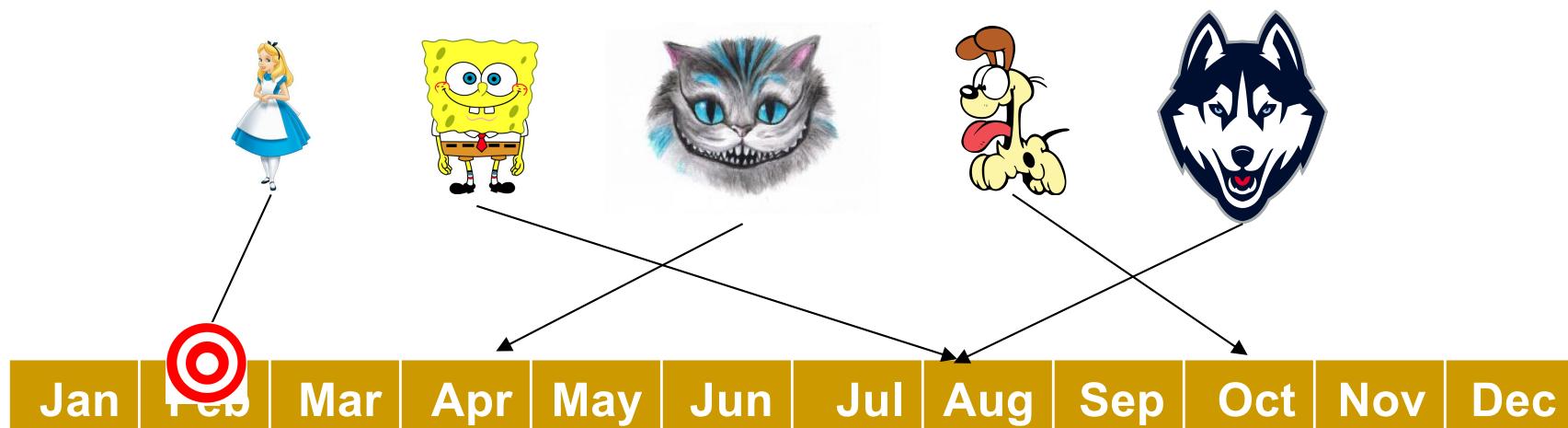
Adversary has to select target **before** knowing key

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

# TCR and Birthday Paradox?



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

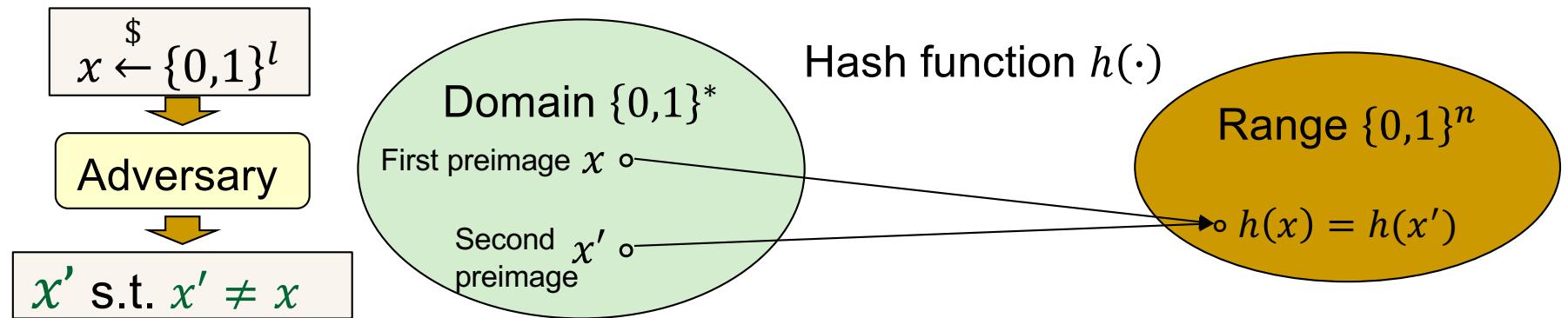


# We (mostly) focus on keyless hash...

- 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 and TCR
    - Why?
  - Modifying to ACR is quite trivial
    - Just make it keyed!
  - Usually used in practice: libraries, standards, ...

# 2<sup>nd</sup>-Preimage-Resistant Hash (SPR)

- Hard to find collision with a specific random  $x$ .

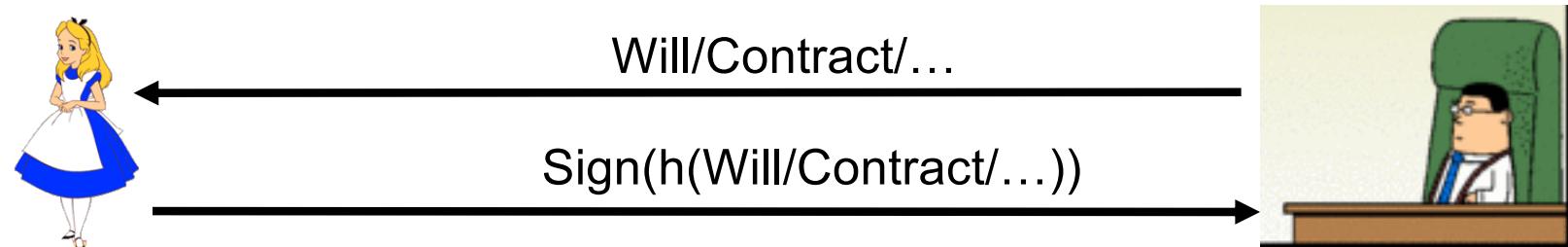


$$\varepsilon_{h, \mathcal{A}}^{SPR}(n) \equiv \Pr_{x \xleftarrow{\$} \{0,1\}^{A(1^n)}} [x' \leftarrow \mathcal{A}(x) \text{ s.t. } x \neq x' \wedge h(x) = h(x')]$$

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

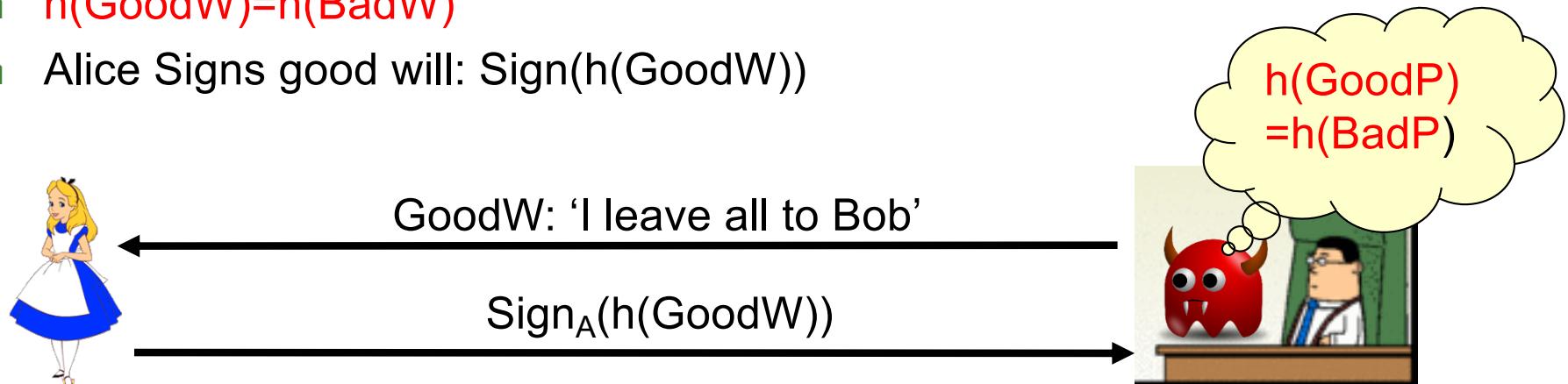
# CRHF/SPR vs. Applications

- 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!



# 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
  - $h(\text{GoodW})=h(\text{BadW})$
  - Alice Signs good will:  $\text{Sign}_A(h(\text{GoodW}))$



- Later... Mal presents to the court:

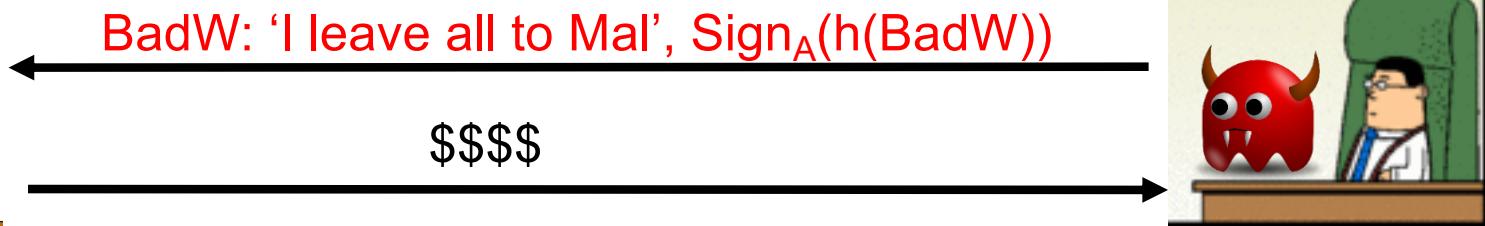


# SPR: collisions to chosen message

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

Is such attack realistic?

Or SPR is enough 'in practice'?



P)  
P)

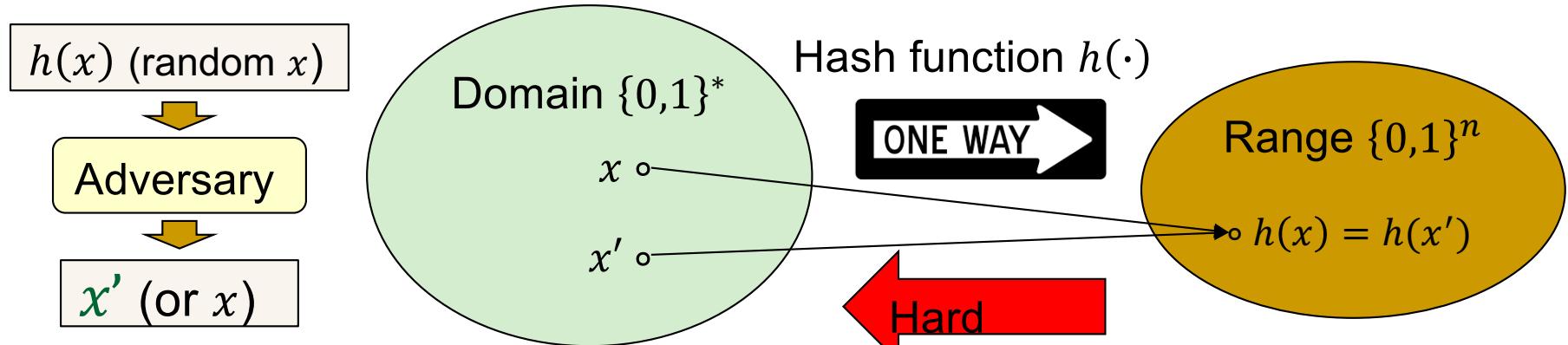
# SPR & Chosen-prefix vulnerability

- Chosen-prefix vulnerability :
  - Mal selects ‘prefix string’  $p$
  - Efficient alg finds :
$$x \neq x' \text{ s.t. } h(p||x) = h(p||x')$$
    - Or, also for any 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
  - Example of possible attack on HtS with SPR

# Chosen-prefix attack

- Let  $x < x'$  be collision for prefix:  $p = \text{'Pay Mal $'}$
- Mal tricks Alice into signing him an IOU for  $\$x$
- Alice signs, sends  $s = S_s^h(m)$  where  $m = \text{'Pay Mal $'} || x$ 
  - $S_s^h(m) = S_s^h(h(p || x)) = S_s^h(h(p || x')) = S_s^h(\text{m}')$
  - $\text{m}' = \text{'Pay Mal $'} || x'$
- Mal sends  $s, \text{m}'$  to Alice's bank
  - Bank validates "Ok" =  $\text{Verify}_{Alice.v}(\text{m}', s)$
- 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

# One-Way Function (OWF)

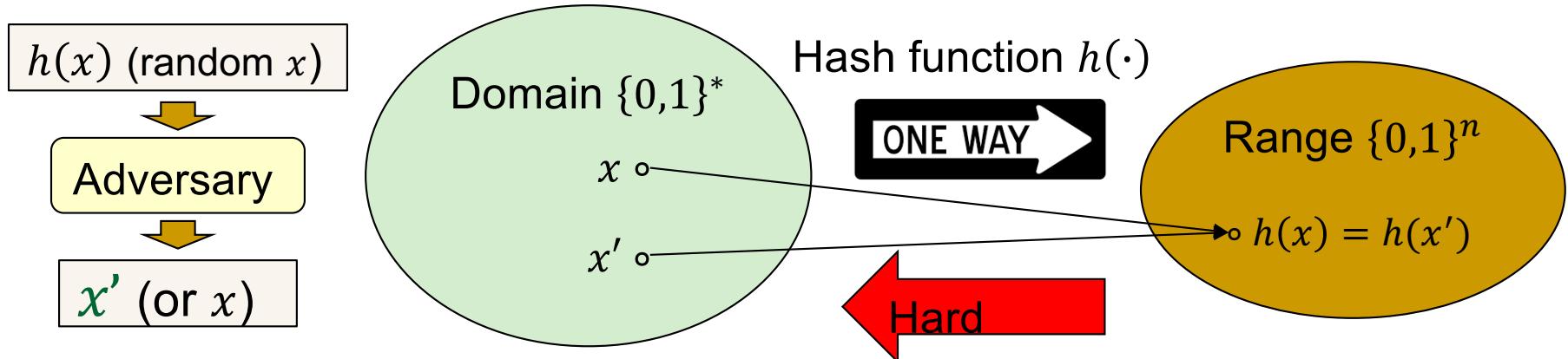


$$\varepsilon_{h,\mathcal{A}}^{OWF}(n) \equiv \Pr_{x \xleftarrow{\$} \{0,1\}^n} [x' \leftarrow \mathcal{A}(h(x)) \text{ s.t. } h(x) = h(x')]$$

Compare to:

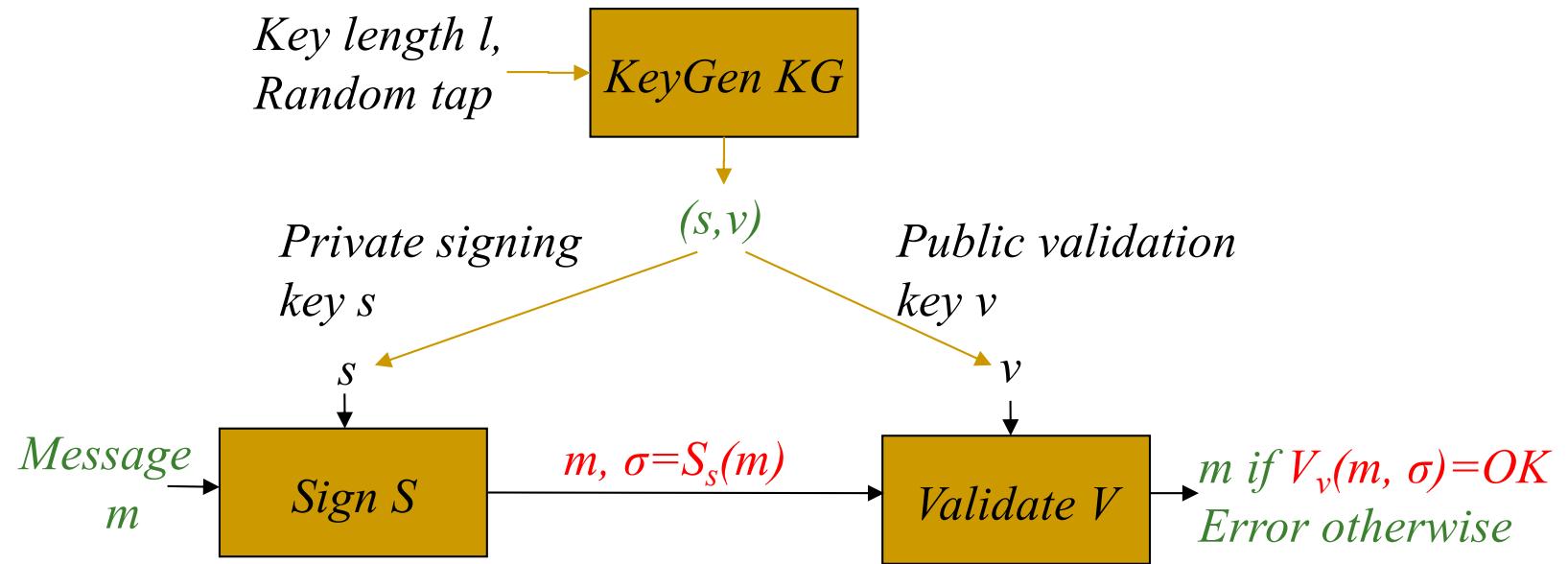
- **Collision-Resistance (CR):** hard to find collision, i.e.,  $(x, x')$  s.t.  $h(x') = h(x)$ ,  $x \neq x'$
- **Second-preimage resistance (SPR):** hard to find collision with random  $x$ , i.e.,  $x'$  s.t.  $h(x') = h(x)$ ,  $x \neq x'$

# OWF Application: One-time PW



- **One-way function (aka preimage resistance):**  
given  $h(x)$  for random  $x$ , it is hard to find  $x$ ,  
or any  $x'$  s.t.  $h(x')=h(x)$
- **One-time password authentication:**
  - Select random  $x$  : 'one-time password' (keep secret!)
  - Validate using non-secret 'one-time validation token':  $h(x)$
- Extend to one-time public-key signatures...

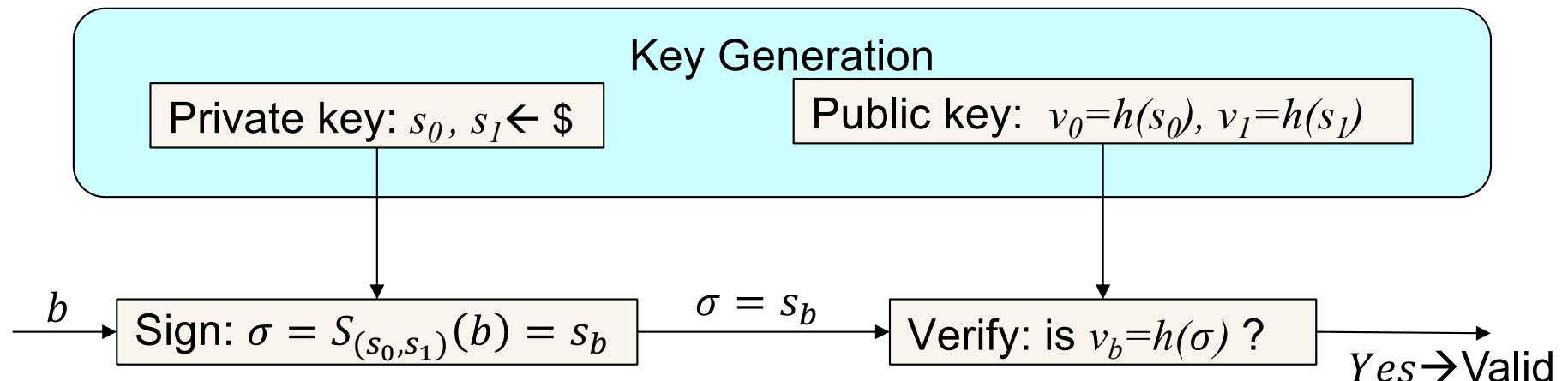
# Recall: Public Key Signatures



- **Assuming limitations:**
  - **Knowledge limitations:** key  $s$  is secret (unknown to attacker)
  - **Resource limitations:** can't find key  $s$  by trying all keys
- Then attacker cannot forge signature
  - I.e., find 'signature'  $\sigma$  for unsigned-message  $m$  s.t.  $V_v(m, \sigma)=OK$

# OWF-based One-Time Signing

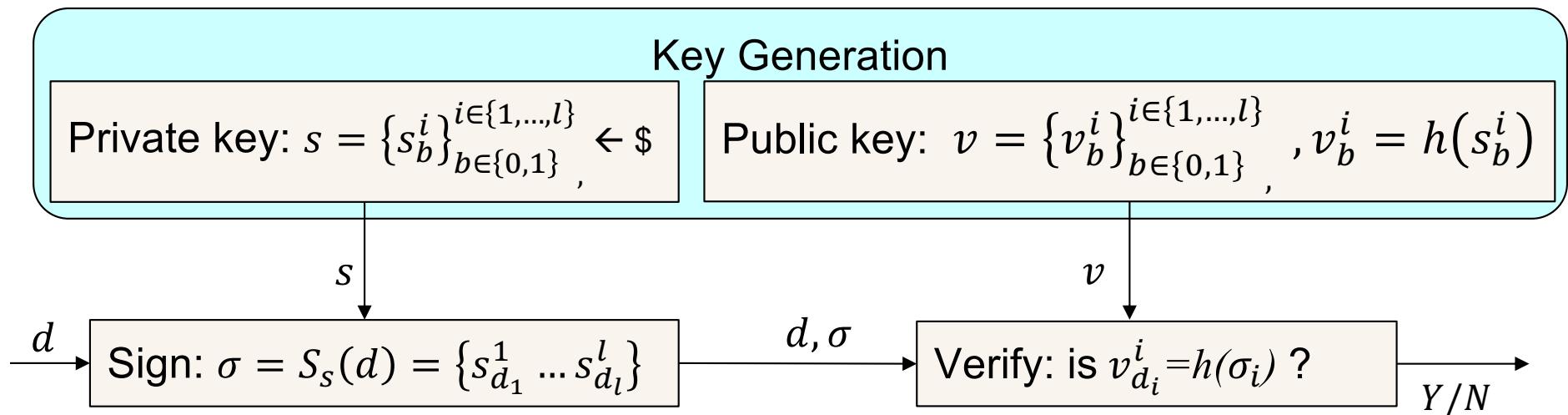
- One-time signature scheme – for single bit  $b$  message:
  - Private key: two random ‘one-time PWs’:  $s_0, s_1$
  - Public key: two strings  $v_0 = h(s_0), v_1 = h(s_1)$
  - Signature: to sign bit  $b$  send signature  $\sigma = S_{(s_0, s_1)}(b) = s_b$
  - Validate:  $V_{(v_0, v_1)}(b, \sigma) = \{1 \text{ if } v_b = h(\sigma), 0 \text{ else}\}$



- How to extend to a (one-time) signatures for **any string** ?

# FIL One-Time Signing

- One-time signature – for FIL **string**  $d \in \{0,1\}^l$  (of  $l$  bits):
  - Private key:  $2l$  random ‘one-time PWs’:  $s = \{s_b(i)\}_{b \in \{0,1\}, i \in \{1, \dots, l\}}$
  - Public key:  $2l$  strings  $v = \{v_b(i)\}_{b \in \{0,1\}, i \in \{1, \dots, l\}}$ ,  $v(i)_b = h(s(i)_b)$
  - Signature of  $d = \{d_1 \dots d_l\}$  is  $\sigma = S_s(d) = \{s_{d_1}(1) \dots s_{d_l}(l)\}$
  - Validate:  $V_v(d, \sigma) = \{1 \text{ if } (\forall i) v_{d_i}(i) = h(\sigma_i), 0 \text{ else}\}$

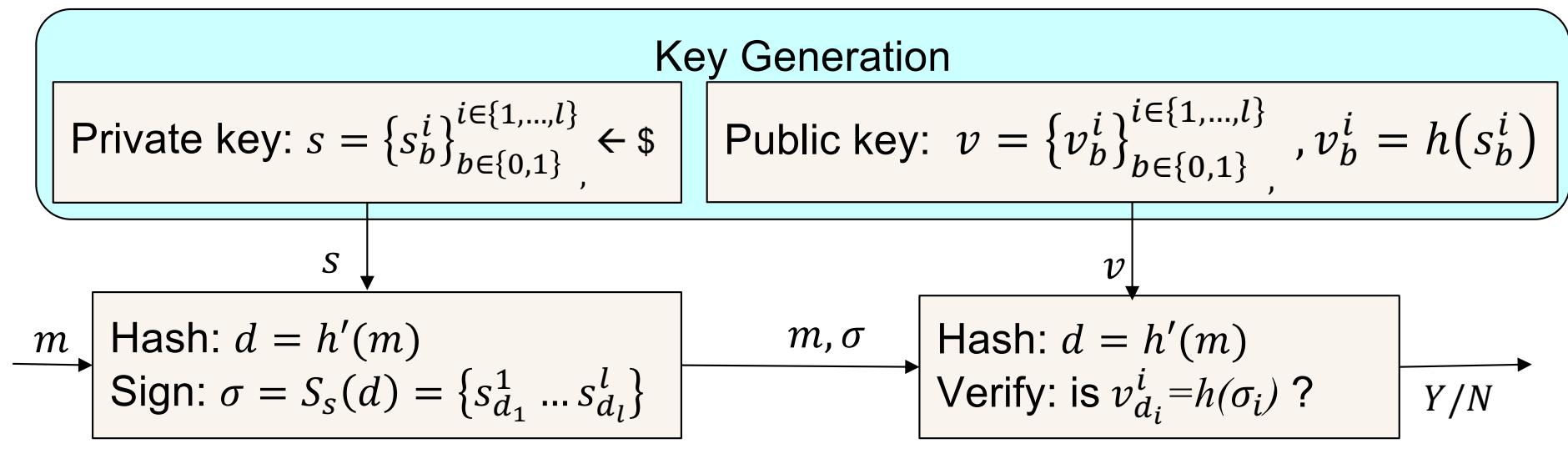


How to extend to one-time-sign any message  $m \in \{0,1\}^*$  ?

# One-Time Signing: any string (VIL)

- One-time signature – for **arbitrary-length**  $m \in \{0,1\}^*$ :

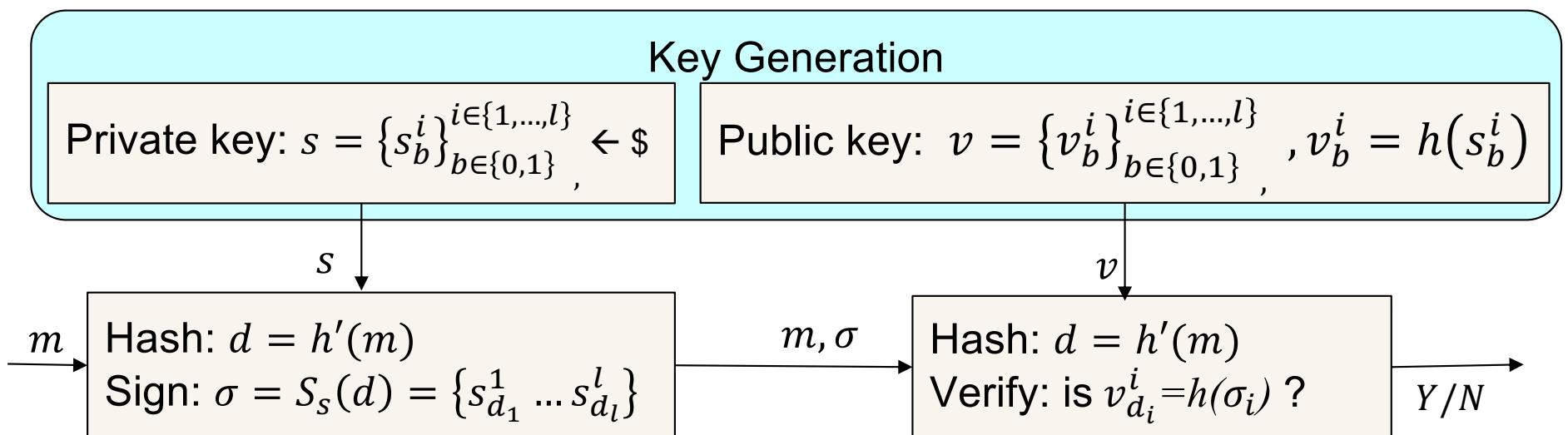
- Private key  $s = \{s_b(i)\}_{b \in \{0,1\}, i \in \{1, \dots, l\}}$
- Public key  $v = \{v_b(i)\}_{b \in \{0,1\}, i \in \{1, \dots, l\}}$ ,  $v(i)_b = h(s(i)_b)$
- Use also a CRHF  $h': \{0,1\}^* \rightarrow \{0,1\}^l$
- Signature of  $m$  is  $\sigma = S_s(h'(m)) = \{s(1)_{h'(m)_1} \dots s(l)_{h'(m)_l}\}$
- Validate:  $V_v(m, \sigma) = \{1 \text{ if } (\forall i) v(i)_{h'(m)_i} = h(\sigma_i), 0 \text{ else}\}$



# One-Time Signing: any string (VIL)

- One-time signature – for **arbitrary-length**  $m \in \{0,1\}^*$ :

- Private key  $s = \{s_b(i)\}_{b \in \{0,1\}, i \in \{1, \dots, l\}}$
- Public key  $v = \{v_b(i)\}_{b \in \{0,1\}, i \in \{1, \dots, l\}}$ ,  $v(i)_b = h(s(i)_b)$



# Recycling One-Time Signatures?

- One-time signatures are ‘disposable’ (duh!)
- Can we recycle them??
- **Exercise: design sequence signatures:**
  - **Generate** sign, validate keypair  $(s, v)$
  - Sign sequence of messages:  $m_1, m_2, \dots$ 
    - Signatures:  $\sigma_1, \sigma_2, \dots$
  - Validate message  $m_i$  with  $\sigma_1, \sigma_2, \dots \sigma_i$
  - How?
    - One bonus point for (correct) solution
    - Another for well-written in latex

# Examples

- ...

# Crypto-Hash functions

- Introduction to cryptographic hash
- Integrity: collision-resistance and blockchains
  - Hash  $h(m)$  allows verification of  $m$
- Confidentiality: one-way functions (OWF)
  - OWF-hash  $h(m)$  does not expose  $m$
  - And: Proof-of-Work (PoW)
- **Pseudo-randomness**
  - **If  $m$  contains some random bits, then  $h(m)$  is (pseudo)random**

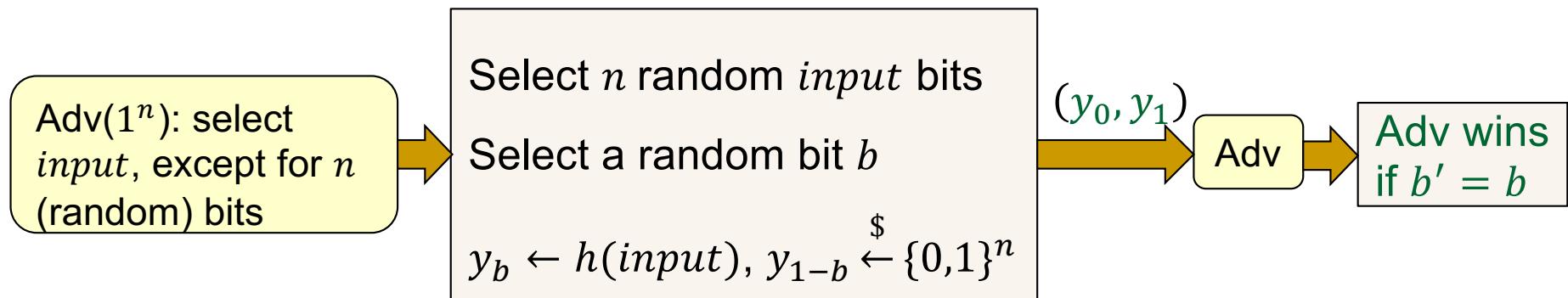
# Von Neuman's Randomness Extraction

- 'If input is sufficiently random, then output is random'
- Multiple 'sufficiently random' models
  - Learn crypto for strong (but complex) models, results
  - We'll see two simplified models
- **Von Neuman's model:**

Assume each bit is result of flip of coin with fixed bias
- Von Neuman's solution:
  - Arrange input in pairs of bits:  $\{(x_i, y_i)\}$
  - Remove pairs where bits are the same, so now  $x_i \neq y_i$
  - Output  $x_i$
- If assumption holds – output is uniform !
- Exercise in textbook.

# Bitwise Randomness Extraction

- ‘If input is sufficiently random, then output is random’
- Simple model: if any  $n$  input bits are random,
  - all  $n$  output bits are pseudorandom
  - For sufficiently large  $n$
- Simplified process:



Hash  $h$  is BRE if for PPT adversary  $A$ ,

$$\varepsilon_{h,A}^{BRE}(n) \equiv \Pr [1 = BRE_{A,h}(1, n)] - \Pr [1 = BRE_{A,h}(0, n)]$$

is negligible.

---

**Algorithm 1** Bitwise-Randomness Extraction  
 $BRE_{A,h}(b, n)$ .

---

$$(m, M) \leftarrow A(1^n)$$

**If**  $|m| \neq |M|$  or  $\sum_{i=1}^{|M|} M(i) < n$  **return**  $\perp$

$$m' \xleftarrow{\$} \{0, 1\}^{|m|}$$

$$y_b \leftarrow h(m \oplus (m' \wedge M))$$

$$y_{1-b} \xleftarrow{\$} \{0, 1\}^n$$

**return**  $A(y_0, y_1, m, M)$

---

# Crypto hash functions: Goals

Potion book hint:

‘... or just shove a bezoar down there...’  
[ The half-blood prince]

Hash functions variant:

‘... or just use the Random Oracle Model!’

i.e.,  $(x, x')$  s.t.  $h(x')=h(x)$ ,  $x \neq x'$

# Random Oracle Method

- Use a fixed, keyless hash function  $h$
- Analyze as if 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
  - Not proof, but exclude ‘structural weaknesses’
- In practice: assume random oracle and use a standard hash functions
  - Or better: replace by ‘standard’ crypto schemes
- More details in textbook

# Covered Material From the Textbook

- ❑ Chapter 4
  - ❑ To be finished later.

# Thank You!

