Cryptography
CS 555

Week 5:

- Cryptographic Hash Functions
- HMACs
- Generic Attacks
- Random Oracle Model
- Applications of Hashing

Readings: Katz and Lindell Chapter 5, Appendix A.4

Spring 2021

Recap

- Authenticated Encryption + CCA-Security
  - ~~Encrypt and Authenticate [SSL]~~
  - Authenticate then Encrypt [TLS] (Caution Required)
  - Encrypt **then** Authenticate!

$$Enc_K(m) = \left\langle \mathrm{c}, \mathrm{Mac}'_{K_M}(\mathrm{c}) \right\rangle \text{ where } \mathrm{c} = \mathrm{Enc}'_{K_E}(m)$$

- Secure Communication
  - Attacks: Reflection/Replay/Reordering + Defenses
  - AES-GCM
- Cryptographic Hash Functions
  - Definitional Challenges

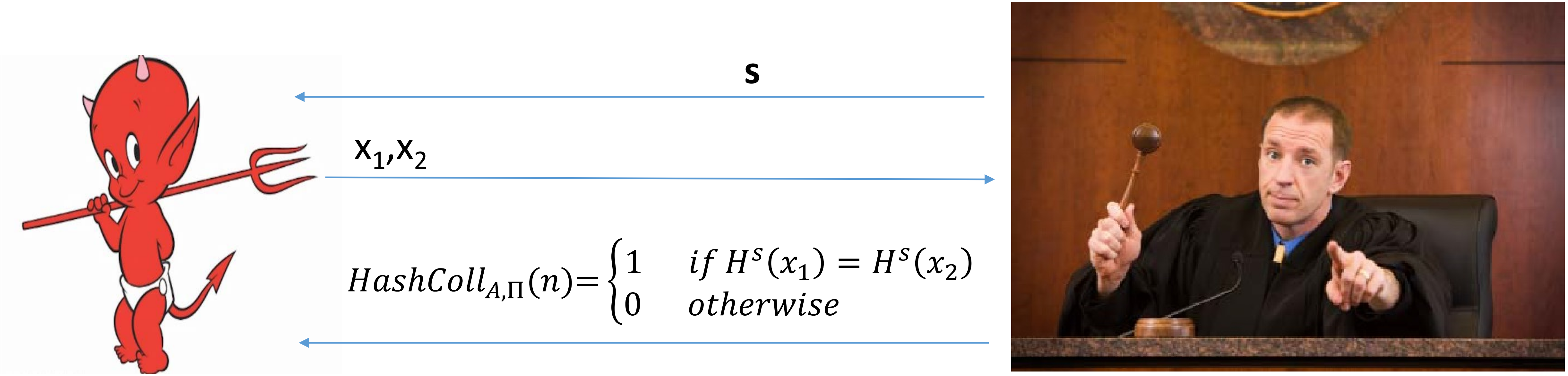
Week 5: Topic 1:
Cryptographic Hash Functions

Keyed Hash Function Syntax

- Two Algorithms
  - $\mathrm{Gen}(1^n; R)$ (Key-generation algorithm)
    - **Input:** Random Bits R
    - **Output:** ~~Secret~~ key s
  - $H^s(m)$ (Hashing Algorithm)
    - **Input:** key s and message $\mathrm{m} \in \{0,1\}^*$ (unbounded length)
    - **Output:** hash value $H^s(m) \in \{0,1\}^{\ell(n)}$

- Fixed length hash function
  - $m \in \{0,1\}^{\ell'(n)}$ with $\ell'(n) > \ell(n)$
  - Example: $m \in \{0,1\}^{2n}$ and $H^s(m) \in \{0,1\}^n$

Collision Experiment ($HashColl_{A,\Pi}(n)$)



s

x_1, x_2

$$HashColl_{A,\Pi}(n) = \begin{cases} 1 & if\ H^s(x_1) = H^s(x_2) \\ 0 & otherwise \end{cases}$$

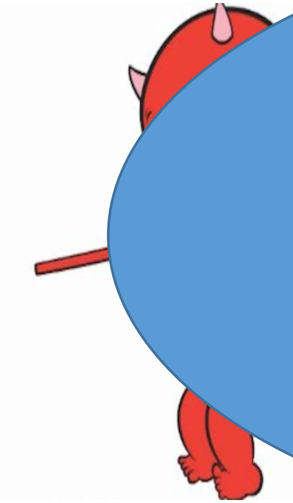
$s = \mathrm{Gen}(1^n; R)$

Definition: (Gen,H) is a collision resistant hash function if

$\forall PPT\ A\ \exists \mu$ (negligible) s.t

$$\Pr\left[HashColl_{A,\Pi}(n) = 1\right] \le \mu(n)$$

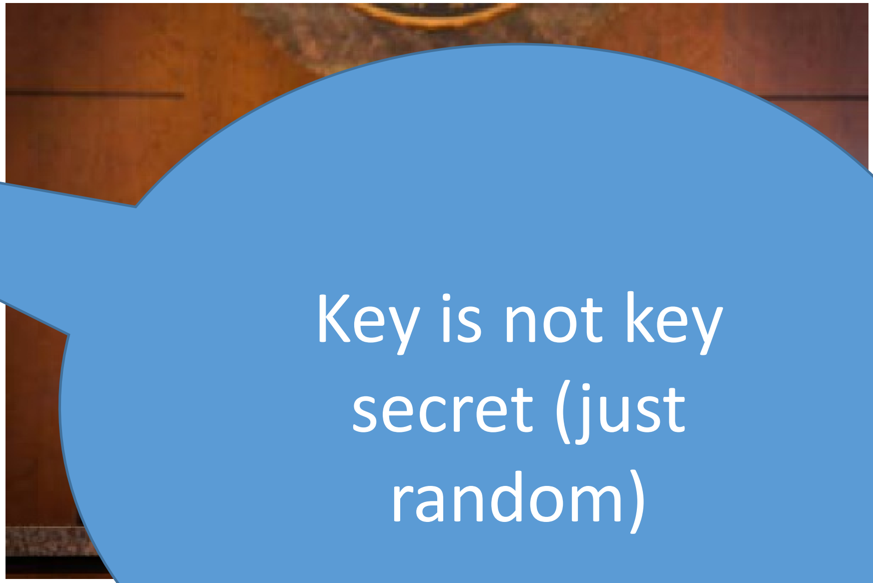
Collision Experiment ($HashColl_{A,\Pi}(n)$)



For simplicity we will sometimes just say that H (or H^s) is a collision resistant hash function

$= H^s(x_2)$

ise



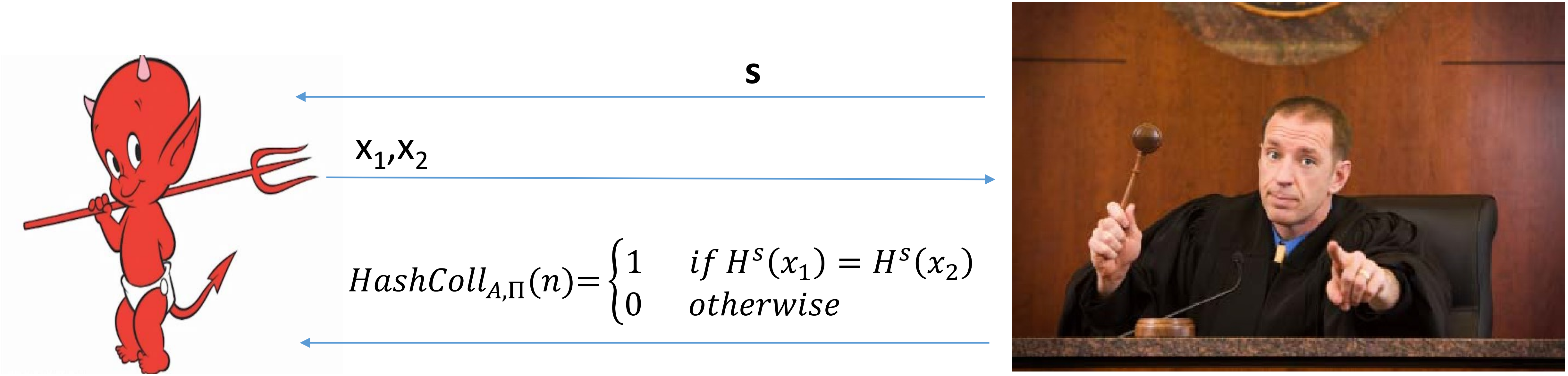
Key is not key secret (just random)

Definition: (Gen,H) is a collision resistant hash function if

$$\forall PPT\ A\ \exists \mu \text{ (negligible) s.t}$$

$$\Pr\left[HashColl_{A,\Pi}(n)=1\right] \leq \mu(n)$$

Concrete Security ($HashColl_{A,\Pi}(n)$)



s

x1,x2

$$HashColl_{A,\Pi}(n)=\begin{cases}1 & if\ H^s(x_1)=H^s(x_2)\\ 0 & otherwise\end{cases}$$

$\mathrm{s} = \mathrm{Gen}(1^n; R)$

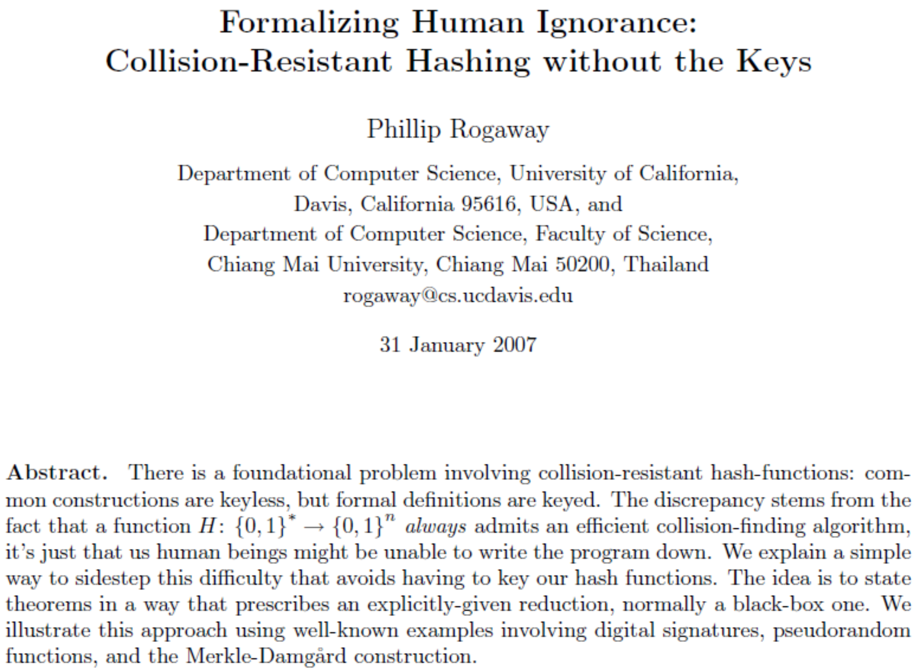
Definition: (Gen,H) is a (t,ε) $-$collision resistant hash function if $\forall\ attackers\ A\ running\ in\ \ time\ at\ most\ t(n)$

$$\Pr\left[HashColl_{A,\Pi}(n)=1\right] \le \varepsilon(n)$$

Theory vs Practice

- Most cryptographic hash functions used in practice are un-keyed
  - Examples: MD5, SHA1, SHA2, SHA3, Blake2B
- Tricky to formally define collision resistance for keyless hash function
  - There is a PPT algorithm to find collisions
  - We just usually can't find this algorithm ☺
  - Guarantee for protocol using H

    If we know an explicit efficient algorithm A breaking our protocol then there is an efficient blackbox reduction transforming A into an efficient collision finding algorithm.



Formalizing Human Ignorance:
Collision-Resistant Hashing without the Keys

Phillip Rogaway

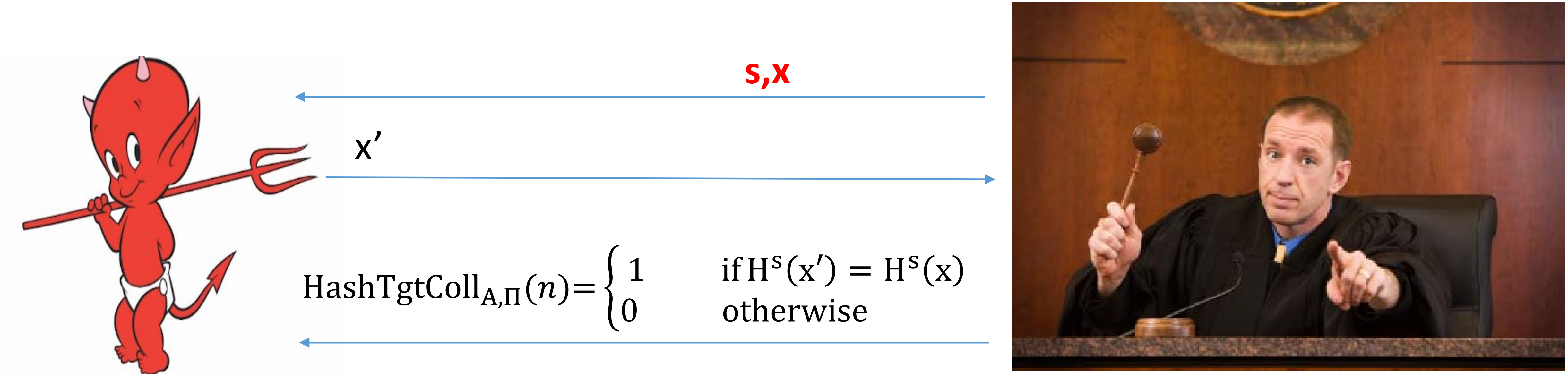
Department of Computer Science, University of California,
Davis, California 95616, USA, and
Department of Computer Science, Faculty of Science,
Chiang Mai University, Chiang Mai 50200, Thailand
rogaway@cs.ucdavis.edu

31 January 2007

Abstract. There is a foundational problem involving collision-resistant hash-functions: common constructions are keyless, but formal definitions are keyed. The discrepancy stems from the fact that a function $H\colon \{0,1\}^* \to \{0,1\}^n$ *always* admits an efficient collision-finding algorithm, it's just that us human beings might be unable to write the program down. We explain a simple way to sidestep this difficulty that avoids having to key our hash functions. The idea is to state theorems in a way that prescribes an explicitly-given reduction, normally a black-box one. We illustrate this approach using well-known examples involving digital signatures, pseudorandom functions, and the Merkle-Damgård construction.

Weaker Requirements for Cryptographic Hash

- Target-Collision Resistance



$$\mathrm{HashTgtColl}_{A,\Pi}(n) = \begin{cases} 1 & \text{if } H^s(x') = H^s(x) \\ 0 & \text{otherwise} \end{cases}$$

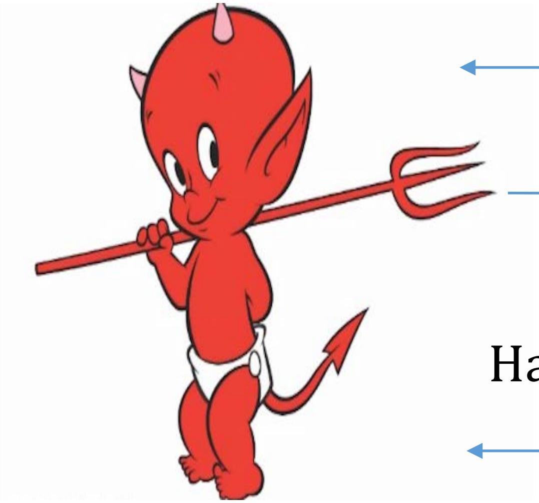
$$s = \mathrm{Gen}(1^n; R)$$

$$x \in \{0,1\}^n$$

Question: Why is collision resistance stronger?

Weaker Requirements for Cryptographic Hash

- Preimage Resistance (One-Wayness)



s, y

x

$$\text{HashPreImgRes}_{A,\Pi}(n) = \begin{cases} 1 & \text{if } H^{s}(x) = y \\ 0 & \text{otherwise} \end{cases}$$

$s = \mathrm{Gen}(1^n; R)$

$y \in \{0,1\}^{\ell(n)}$

Question: Why is collision resistance stronger?

Merkle-Damgård Transform

- Most cryptographic hash functions accept fixed length inputs

- What if we want to hash arbitrary length strings?

Construction: Suppose (Gen,h) fixed length hash function from 2n bits to n bits, define H^s as follows

$$H^s(x_1, \dots, x_d) = h^s\big(h^s\big(h^s\big(h^s\big(\dots h^s(0^n \parallel x_1)\big) \parallel x_{d-1}\big) \parallel x_d\big) \parallel |x|\big)$$

Merkle-Damgård Transform

Construction: (Gen,h) fixed length hash function from 2n bits to n bits

```
$H^s(x)$ =
1. Break x into n bit segments $x_1$,..,$x_d$ (pad last block by 0's)
2. $z_0 = 0^n$ (initialization)
3. For i = 1 to d
    1. $z_i = h^s(z_{i-1} \parallel x_{\mathrm{i}})$
4. Output $z_{d+1} = h^s(z_d \parallel L)$ where $L$ encodes $|x|$ as an n-bit string
```

Merkle-Damgård Transform

Theorem: If (Gen,h) is collision resistant then so is (Gen,H)

Proof: Show that any collision in H^s yields a collision in h^s. Thus a PPT attacker A_H for (Gen,H) can be transformed into PPT attacker A_h for (Gen,h).

Suppose that A_H finds a collision i.e., distinct x and x' such that

$$H^s(x) = H^s(x')$$

(note x and x' may have different lengths)

Merkle-Damgård Transform

Theorem: If (Gen,h) is collision resistant then so is (Gen,H)

Proof: Suppose that $H^s(x) = H^s(x')$. We will extract a collision for h^s.

Case 1: L=|x|=|x'|=L' (proof for case two is similar)

$$H^s(x) = z_{d+1} = h^s(z_d \parallel L) = H^s(x') = z'_{d+1} = h^s(z'_d \parallel L')$$

$$z_d \parallel L =?\, z'_d \parallel L'$$

No → Found collision

$h^s(z_d \parallel L) = h^s(z'_d \parallel L')$

Yes?

$$z_d = h^s(z_{d-1} \parallel x_d) = h^s(z'_{d-1} \parallel x'_d) = z'_d$$

Merkle-Damgård Transform

Theorem: If (Gen,h) is collision resistant then so is (Gen,H)

Proof: Suppose that

$$H^s(x) = H^s(x')$$

Case 1: L=|x|=|x'|=L' (proof for case two is similar)

$$z_d = h^s(z_{d-1} \parallel x_d) = h^s(z'_{d-1} \parallel x'_d) = z'_d$$

$$z_{d-1} \parallel x_d =? \, z'_{d-1} \parallel x'_d$$

No → Found collision

$h^s(z_{d-1} \parallel x_d) = h^s(z'_{d-1} \parallel x'_d)$

Yes?

$$z_{d-1} = h^s(z_{d-2} \parallel x_{d-1}) = h^s(z'_{d-2} \parallel x'_{d-1}) = z'_{d-1}$$

Merkle-Damgård Transform

Theorem: If (Gen,h) is collision resistant then so is (Gen,H)

Proof: Suppose that

$$H^s(x) = H^s(x')$$

Case 1: |x|=|x'| (proof for case two is similar)

If for some i we have $z_{i-1} \parallel x_i \neq z'_{i-1} \parallel x'_i$ then we will find a collision

But x and x' are different so we must have $x_i \neq x'_i$ for some $i \leq d$!

Merkle-Damgård Transform

Theorem (Concrete Version): If (Gen,h) is (t, ε)-collision resistant then (Gen,H) is is (t', ε)-collision resistant for where $t' = O(t)$

Analysis: Run attacker A_H to get pair x and x’ (time t), then compute z_i (resp. $z_i{}'$) values to extract collision.

$$H^s(x) = z_{d+1} = h^s(z_d \parallel L) = H^s(x') = z'_{d+1} = h^s(z'_d \parallel L')$$

$$z_d \parallel L =?\, z'_d \parallel L'$$

No → Found collision

$$h^s(z_d \parallel L) = h^s(z'_d \parallel L')$$

Yes?

$$z_d = h^s(z_{d-1} \parallel x_d) = h^s(z'_{d-1} \parallel x'_d) = z'_d$$

Week 5: Topic 2: HMACs and Generic Attacks

MACs for Arbitrary Length Messages

$\text{Mac}_K(m)$=

- Select random n/4 bit string r
- Let $t_i = \mathrm{Mac}'_K(r \parallel \ell \parallel i \parallel m_i)$ for i=1,…,d
  - (Note: encode i and ℓ as n/4 bit strings)
- **Output** $\langle r, t_1, \dots, t_d \rangle$

Theorem 4.8: If Π’ is a secure MAC for messages of fixed length n, above construction $\Pi = (\mathrm{Mac}, \mathrm{Vrfy})$ is secure MAC for arbitrary length messages.

MACs for Arbitrary Lengt

Disadvantage 1: Long output

Disadvantages: Lose Strong-MAC Guarantee (Multiple valid MACs of same message)

ri

‖ i ‖

- (Note: e and ℓ as n/4 bi

- **Output** $\langle r, t_1, \ldots, t_d \rangle$

Randomized Construction (no canonical verification). Disadvantage?

Theorem 4.8: If Π' i above constructio messages.

Hash and MAC Construction

Start with $\Pi = (\mathrm{Mac}, \mathrm{Vrfy})$, a secure MAC for messages of fixed length, and (Gen$_H$,H) a collision resistant hash function and define Π'

$$Mac'_{\langle K_M, S \rangle}(m) = Mac_{K_M}\left(H^s(m)\right)$$

$$Vrfy'_{\langle K_M, S \rangle}(m, t) = Vrfy_{K_M}(H^s(m), t)$$

Theorem 5.6: Π' is a secure MAC for arbitrary length message assuming that Π is a secure MAC and (Gen$_H$,H) is collision resistant.

Note: If $\mathrm{Vrfy}_{K_M}(m, t)$ is canonical then $\mathrm{Vrfy}'_{\langle K_M, S \rangle}(m, t)$ is canonical.

Hash and MAC Construction

Start with (Mac,Vrfy) a MAC for messages of fixed length and (Gen$_H$,H) a collision resistant hash function

$$Mac'_{\langle K_M, S\rangle}(m) = Mac_{K_M}\left(H^s(m)\right)$$

Theorem 5.6: Above construction is a secure MAC.

Proof Intuition: If attacker successfully forges a valid MAC tag t' for unseen message m' then either

- **Case 1:** $H^s(m') = H^s(m_i)$ for some previously requested message m$_i$
- **Case 2:** $H^s(m') \neq H^s(m_i)$ for every previously requested message m$_i$

Hash and MAC Construction

Theorem 5.6: Above construction is a secure MAC.

Proof Intuition: If attacker successfully forges a valid MAC tag t' for unseen message m' then either

- **Case 1:** $H^s(m') = H^s(m_i)$ for some previously requested message m_i
  - **Attacker can find hash collisions!**
- **Case 2:** $H^s(m') \neq H^s(m_i)$ for every previously requested message m_i
  - **Attacker forged a valid new tag on the "new message" $\boldsymbol{H^s(m')}$**
  - **Violates security of the original fixed length MAC**

Hash and MAC Construction

Start with (Mac,Vrfy) a MAC for messages of fixed length and (Gen$_H$,H) a collision resistant hash function

$$Mac'_{\langle K_M,S\rangle}(m) = Mac_{K_M}\left(H^s(m)\right)$$

Theorem 5.6 (Concrete Version): If Mac is $(t, q_{MAC}, \varepsilon_{MAC}) -$ secure and (Gen$_H$,H) is (t, ε_{Hash}) −collision resistant then $Mac'_{\langle K_M,S\rangle}$ is $(O(t), q_{MAC}, \varepsilon_{MAC} + \varepsilon_{Hash}) - secure$

Proof Intuition: When A succeeds we either get a hash collision (case 1) or a Mac_{K_M} forgery (case 2)

if $\Pr[\text{case } 2] > \varepsilon_{MAC}$ **we could violate** $(t, q_{MAC}, \varepsilon_{MAC}) - \text{secure for } Mac_{K_M}$

Simulate $Mac'_{\langle K_M,S\rangle}$ attacker A

when attacker makes a query $Mac'_{\langle K_M,S\rangle}(m)$ we

1. compute $H^s(m)$ and
2. forward $H^s(m)$ to Mac_{K_M}oracle to get back $Mac_{K_M}\left(H^s(m)\right)$

A's tag yields a Mac_{K_M} forgery for new message with probability at least $\Pr[\text{case } 2] > \varepsilon_{MAC}$

Similar argument If $\Pr[\text{case } 1] > \varepsilon_{HASH}$ **we could violate** (t, ε_{Hash}) −collision resistance for $H^s(.)$

Therefore, A succeeds with probability at most $\varepsilon_{MAC} + \varepsilon_{Hash}$

Recap

- Definition of Collision Resistant Hash Functions (Gen,H)
  - Definitional challenges
  - Gen(1^n) outputs a public seed.

- Merkle-Damgård construction to hash arbitrary length strings
  - Proof of correctness

- Hash and MAC construction
  - Proof of correctness

MAC from Collision Resistant Hash

- Failed Attempt:

$$Mac_{\langle k,S\rangle}(m) = H^s(k \parallel m)$$

Broken if H^suses Merkle-Damgård Transform. Let m_3 encode length of $\mathrm{k} \parallel m_1 \parallel m_2$ and L_3 encode the length of$\mathrm{k} \parallel m_1 \parallel m_2 \parallel m_3$

$$Mac_{\langle k,S\rangle}(m_1 \parallel m_2 \parallel m_3) = h^s(h^s(h^s(h^s(h^s(0^n \parallel k) \parallel m_1) \parallel m_2) \parallel m_3) \parallel L_3)$$
$$= h^s\left(Mac_{\langle k,S\rangle}(m_1 \parallel m_2) \parallel L_3\right)$$

Why does this mean $Mac_{\langle k,S\rangle}$ is broken?

MAC from Collision Resistant Hash

- Failed Attempt: $Mac_{\langle k,S \rangle}(m) = H^s(k \parallel m)$

Broken if H^suses Merkle-Damgård Transform. Let m_3 encode length of $\mathrm{k} \parallel m_1 \parallel m_2$

$$Mac_{\langle k,S \rangle}(m_1 \parallel m_2 \parallel m_3) = h^s(h^s(h^s(h^s(h^s(0^n \parallel k) \parallel m_1) \parallel m_2) \parallel m_3) \parallel L_3)$$
$$= h^s\left(Mac_{\langle k,S \rangle}(m_1 \parallel m_2) \parallel L_3\right)$$

Why does this mean $\boldsymbol{Mac_{\langle k,S \rangle}}$ is broken?

1. **Attacker asks for $\boldsymbol{\tau} = Mac_{\langle k,S \rangle}(m_1 \parallel m_2)$**
2. **Attacker computes $\boldsymbol{\tau}' = h^s(\boldsymbol{\tau} \parallel L_3)$ which is a forgery for the message $m_1 \parallel m_2 \parallel m_3$**

HMAC

$$Mac_{\langle k,S\rangle}(m) = H^s\Big((k\oplus\text{opad}) \parallel H^s\big((k\oplus\text{ipad}) \parallel m\big)\Big)$$

ipad?

HMAC

$$Mac_{\langle k,S\rangle}(m) = H^s\left((k\oplus\text{opad}) \parallel H^s\left((k\oplus\text{ipad}) \parallel m\right)\right)$$

$$\text{ipad} = \text{inner pad}$$
$$\text{opad} = \text{outer pad}$$

Both ipad and opad are fixed constants.

Why use key twice?

Allows us to prove security from *weak collision resistance* of H^s

HMAC Security

$$Mac_{\langle k,S\rangle}(m) = H^s\left((k\oplus\mathrm{opad}) \parallel H^s\big((k\oplus\mathrm{ipad}) \parallel m\big)\right)$$

Theorem (Informal): Assuming that H^s is weakly collision resistant and that (certain other plausible assumptions hold) this is a secure MAC.

Weak Collision Resistance: Give attacker oracle access to $f(m) = \ H^s(k \parallel m)$ (secret key k remains hidden).

Attacker Goal: Find distinct m, m' such that $f(m) = f(m')$

HMAC in Practice

- MD5 can no longer be viewed as collision resistant

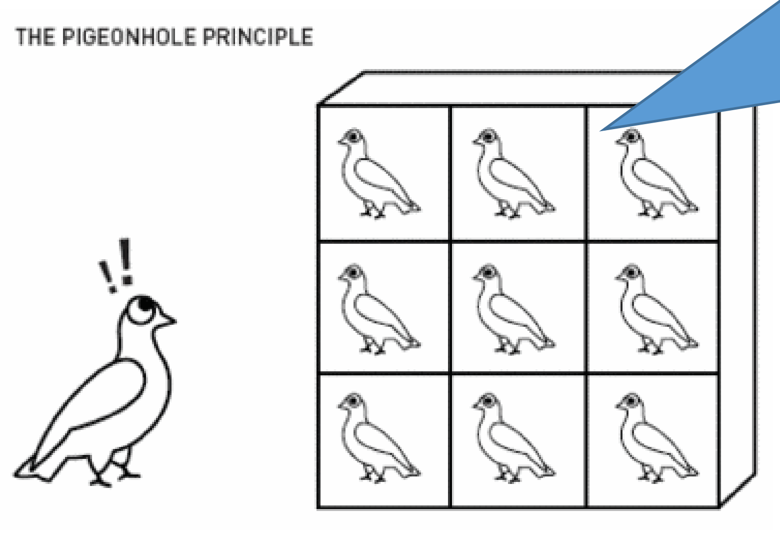
- However, HMAC-MD5 remained unbroken after MD5 was broken
  - Gave developers time to replace HMAC-MD5
  - Nevertheless, don’t use HMAC-MD5!
- HMAC-SHA1 still seems to be okay (temporarily), despite collision
- HMAC is efficient and unbroken
  - CBC-MAC was not widely deployed because it is “too slow”
  - Instead practitioners often used heuristic constructions (which were breakable)

Finding Collisions

- Ideal Hashing Algorithm
  - Random function H from $\{0,1\}^*$ to $\{0,1\}^{\ell}$
  - Suppose attacker has oracle access to H(.)

- **Attack 1:** Evaluate H(.) on $2^{\ell}+1$ distinct inputs.

Can we do better?



Birthday Attack for Finding Collisions

- Ideal Hashing Algorithm
  - Random function H from {0,1}* to $\{0,1\}^{\ell}$
  - Suppose attacker has oracle access to H(.)

- **Attack 2:** Evaluate H(.) on $q = 2^{(\ell/2)+1} + 1$ distinct inputs $x_1,\ldots,x_q$.

$$\Pr[\text{No Collision}] = \Pr[\forall i < j. H(\mathrm{x}_i) \neq H(\mathrm{x}_j)]$$

$$= \Pr[\boldsymbol{D}_2] \prod_{i=3}^{q} Pr[\boldsymbol{D}_i | \boldsymbol{D}_{i-1}, \ldots, \boldsymbol{D}_2]$$

$$\boldsymbol{D}_i = \boldsymbol{event\ that}\ H(x_i) \neq H(x_{i-1}), \ldots, H(x_1)$$

Birthday Attack for Finding Collisions

- Ideal Hashing Algorithm
  - Random function H from {0,1}* to $\{0,1\}^{\ell}$
  - Suppose attacker has oracle access to H(.)

- **Attack 2:** Evaluate H(.) on $q = 2^{(\ell/2)+1} + 1$ distinct inputs $x_1,\ldots,x_q$.

$$\Pr[\forall i < j.\, H(x_i) \neq H(x_j)] =$$

$$1 \times \overbrace{\left(1-\frac{1}{2^{\ell}}\right)}^{\Pr[\overbrace{H(x_2)\neq H(x_1)}^{D_2}]} \times \overbrace{\left(1-\frac{2}{2^{\ell}}\right)}^{\Pr[D_3 \mid D_2]} \times \cdots \times \overbrace{\left(1-\frac{2^{(\ell/2)+1}}{2^{\ell}}\right)}^{\Pr\left[D_q \middle| D_{q-1},\ldots,D_2\right]}$$

Birthday Attack for Finding Collisions

- Ideal Hashing Algorithm
  - Random function H from {0,1}* to $\{0,1\}^{\ell}$
  - Suppose attacker has oracle access to H(.)

- **Attack 2:** Evaluate H(.) on $q = 2^{(\ell/2)+1} + 1$ distinct inputs $x_1,\ldots,x_q$.

$$\Pr[\forall i < j. H(x_i) \neq H(x_j)] = 1\left(1 - \frac{1}{2^{\ell}}\right)\left(1 - \frac{2}{2^{\ell}}\right)\left(1 - \frac{3}{2^{\ell}}\right)\ldots\left(1 - \frac{2^{(\ell/2)+1}}{2^{\ell}}\right)$$

$$\approx \exp\left(\frac{-q(q-1)}{2^{\ell+1}}\right)$$

Birthday Attack for Finding Collisions

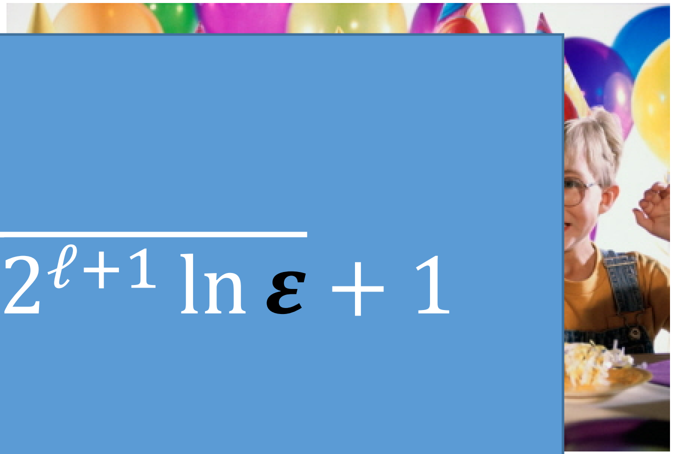
- Ideal Hashing Algorithm
  - Random function H from {0,1}* to $\{0,1\}^{\ell}$
  - Suppose attacker has oracle access to H(.)

- **Attack 2:** Evaluate H(.) on $q = 2^{(\ell/2)+1} + 1$ distinct inputs $x_1,\ldots,x_q$.

$$\Pr[\forall i < j.\, H(x_i) \neq H(x_j)] = 1\left(1 - \frac{1}{2^{\ell}}\right)\left(1 - \frac{2}{2^{\ell}}\right)\left(1 - \frac{3}{2^{\ell}}\right)\ldots\left(1 - \frac{2^{(\ell/2)+1}}{2^{\ell}}\right)$$

$$\approx \exp\left(\frac{-q(q-1)}{2^{\ell+1}}\right) < \exp\left(\frac{-42^{\ell}}{2^{\ell+1}}\right) = e^{-2} < \frac{1}{2}$$

Birthday Attack for Finding Collisions



- Ideal Hashing Algorit
  - Random function H f
  - Suppose attacker has

$$\exp\left(\frac{-q(q-1)}{2^{\ell+1}}\right) < \boldsymbol{\varepsilon} \text{ for } q > \sqrt{2^{\ell+1}\ln \boldsymbol{\varepsilon}} + 1$$

- **Attack 2:** Evaluate H(

$$\Pr[\forall i < j.\, H(\mathrm{x}_i) \neq H(\mathrm{x}_j)] = \left(1 - \frac{1}{2^{\ell}}\right)\left(1 - \frac{2}{2^{\ell}}\right)\left(1 - \frac{3}{2^{\ell}}\right)\ldots\left(1 - \frac{2^{(\ell/2)+1}}{2^{\ell}}\right)$$

$$\approx \exp\left(\frac{-q(q-1)}{2^{\ell+1}}\right) < \exp\left(\frac{-4 2^{\ell}}{2^{\ell+1}}\right) = e^{-2} < \frac{1}{2}$$

Recap

- Collision Resistant Hash Functions
- Merkle–Damgård Construction
- Applications to MACs
  - Hash and MAC
  - Failed MAC: $Mac_{\langle k,S \rangle}(m) = H^s(k \parallel m)$
  - HMAC
- Birthday Attack: Finds collision in time $q = 2^{(\ell/2)+1} + 1$ (and space q)

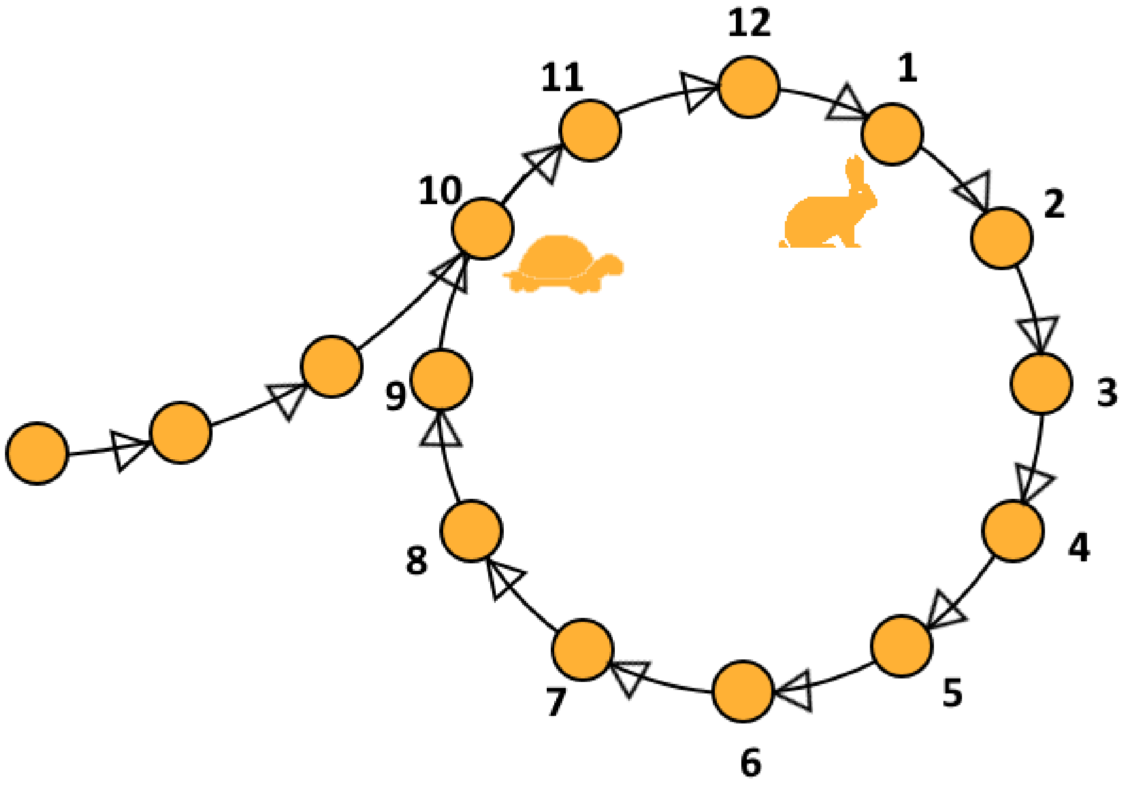
- **Reminder:** Homework 2 Due Tonight
- **Final Exam:** Monday, May 3 at 10:30AM (FRNY B124)

Birthday Attack for Finding Collisions

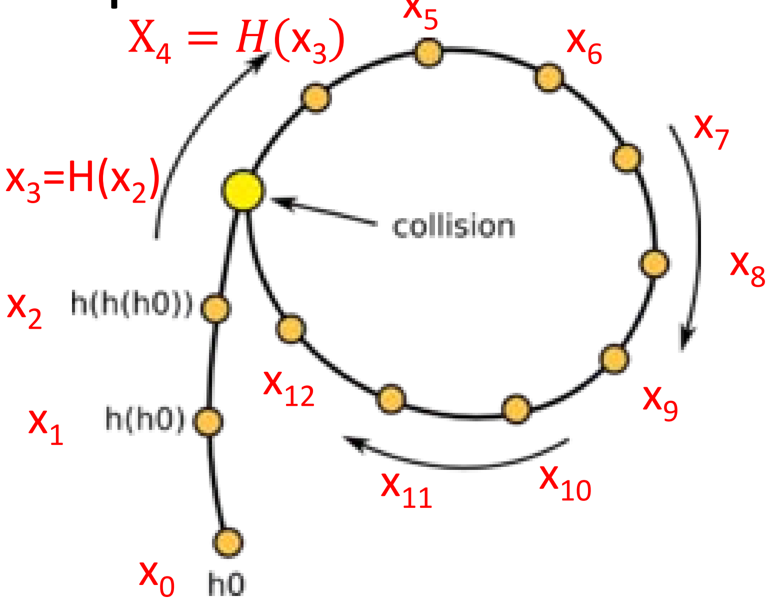
- Ideal Hashing Algorithm
  - Random function H from {0,1}* to $\{0,1\}^{\ell}$
  - Suppose attacker has oracle access to H(.)

- **Attack 2:** Evaluate H(.) on $q = 2^{(\ell/2)+1} + 1$ distinct inputs $x_1,\ldots,x_q$.
- Store values $(x_i, H(x_i))$ in a hash table of size q
  - Requires time/space $O(q) = O\left(\sqrt{2^{\ell}}\right)$
  - Can we do better?

Floyd’s Cycle Finding Algorithm



- A cycle denotes a hash collision
    - $x_3=H(x_2)=H(x_{12})$
- Occurs after $O\left(2^{\ell/2}\right)$ steps by birthday paradox
- First attack phase detects cycle
- Second phase identifies collision



- Analogy: Cycle detection in linked list
- Can traverse “linked list” by computing H

Small Space Birthday Attack

- **Attack 2:** Select random x_0, define $x_i := H(x_{i-1})$
  - Initialize: $x=x_0$ and $x'=x_0$
  - Repeat for i=1,2,…
    - $x:=H(x)$ now $x = x_i$
    - $x':=H(H(x'))$ now $x' = x_{2i}$
    - If $x=x'$ then break
  - Reset $x=x_0$ and set $x'=x$ and remember i
  - Repeat for j=1 to i
    - If $H(x) = H(x')$ then output x,x'
    - Else $x:= H(x)$, $x' = H(x)$

Now $x=x_j$ AND $x' = x_{i+j}$

Claim: for some $k \leq i$ the collision is

$x_k=H(x_{k-1})=H(x_{k+i-1})$

Proof: Let C be length of cycle,

Let k= #steps before cycle

2i-k = i-k mod C ➔ i= mod C

Hare takes 2i-k total steps inside cycle, looping around before ending in same place

Tortoise takes i-k steps inside cycle (equivalent to k backwards steps)

Initially, for phase 2 we have $x'=x_i$ and $x = x_0$ after j=k-1 steps we have $x=x_{k-1}$ and

$x'=x_{i+K-1}=x_{k+C-1}$

Small Space Birthday Attack

- **Attack 2:** Select random x_0, define $x_i = H(x_{i-1})$
  - Initialize: $x=x_0$ and $x'=x_0$
  - Repeat for i=1,2,…
    - $x:=H(x)$ now $x = x_i$
    - $x':=H(H(x'))$ now $x' = x_{2i}$
    - If x=x' then break
  - Reset $x=x_0$ and set $x'=x$
  - Repeat for j=1 to i
    - If H(x) = H(x') then output x,x'
    - Else $x:= H(x)$, $x' = H(x)$ Now $x=x_j$ AND $x' = x_{i+j}$

Finds collision after $O\left(2^{\ell/2}\right)$ steps in expectation

Small Space Birthday Attack

- Can be adapted to find "meaningful collisions" if we have a large message space O$\left(2^{\ell}\right)$

- **Example**: $\mathrm{S} = S_1 \cup S_2$ with $|S_1| = |S_2| = 2^{\ell-1}$
  - S_1 = Set of positive recommendation letters
  - S_2 = Set of negative recommendation letters

- **Goal**: find $z_1 \in S_1, z_2 \in S_2$, such that H(z$_1$) = H(z$_2$)

- Can adapt previous attack by defining an injective mapping $\mathrm{b}\colon \{0,1\}^{\ell} \rightarrow S$

$$\mathrm{x_i} = H(\mathrm{b}(\mathrm{x_{i-1}}))$$

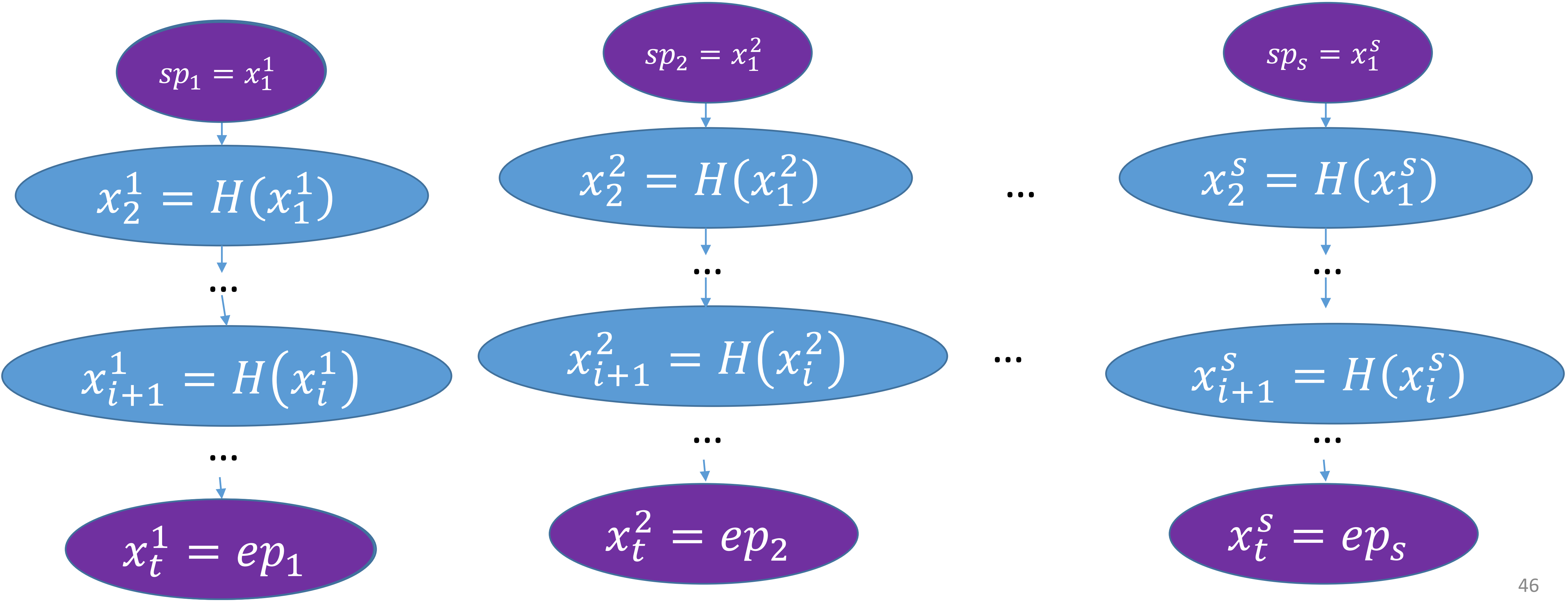
- If $\mathrm{x_i} = \mathrm{x_{i+j}}$ then $H\left(\mathrm{b}(\mathrm{x_{i-1}})\right) = H\left(\mathrm{b}(\mathrm{x_{i+j-1}})\right)$ ➔ Colliding inputs are both in S

Pre-Computation Attacks for Targeted Collision

- **Challenger:** Picks random x and sends y=H(x) to attacker
- **Attacker's Goal:** Find some x' (not necessarily x) s.t. y=H(x')
- **Brute-Force Attack:** Requires $2^{\ell-1}$ queries to H on average.
- **Pre-Computation Attack:** What if we know we will need to do this multiple times?
  - Pre-Processing Cost (one-time cost): $O(2^{\ell})$
  - Post-Processing Cost: $\ll 2^{\ell}$ (is this possible?)
- **Applications:**
  - Targeted Hash Inversion, MAC forgery, Signature Forgery, Key-Recovery, Password Cracking etc…

Pre-Computation Attacks for Targeted Collision

- Precomputation ($t \times s$ steps, $2s$ memory)



$sp_1 = x_1^1$

$x_2^1 = H(x_1^1)$

…

$x_{i+1}^1 = H(x_i^1)$

…

$x_t^1 = ep_1$

$sp_2 = x_1^2$

$x_2^2 = H(x_1^2)$

…

$x_{i+1}^2 = H(x_i^2)$

…

$x_t^2 = ep_2$

…

…

$sp_s = x_1^s$

$x_2^s = H(x_1^s)$

…

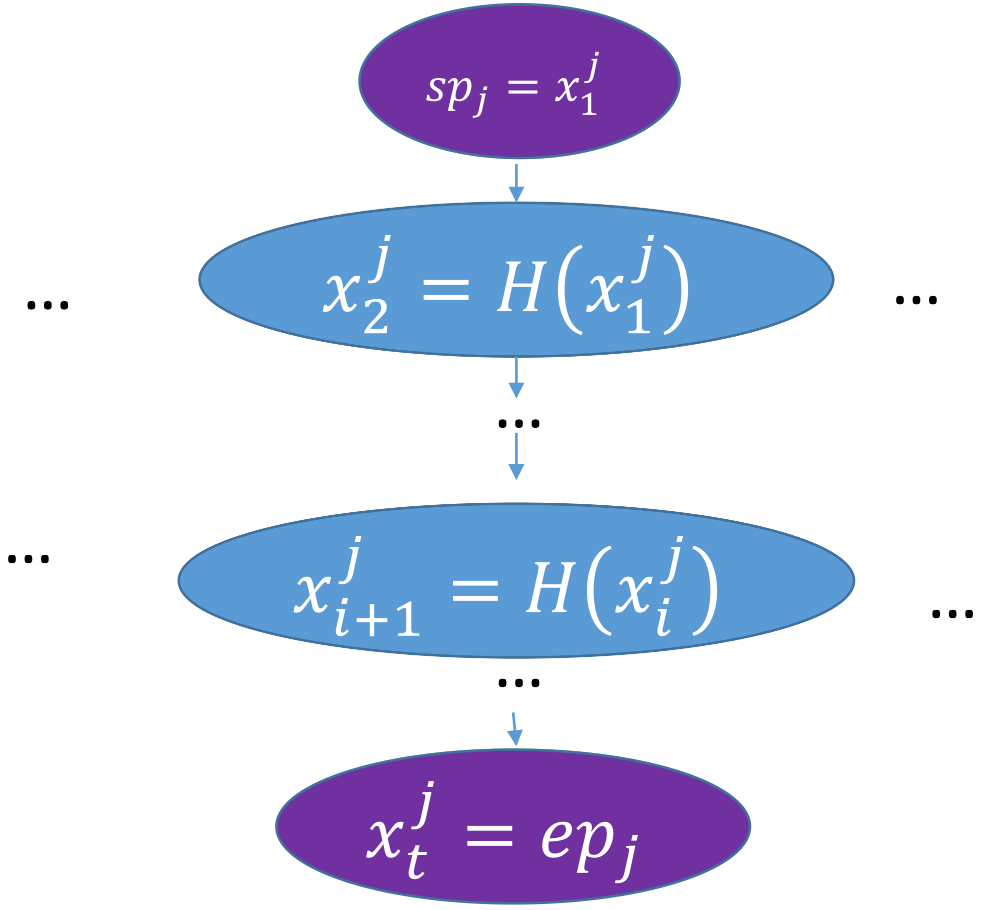
$x_{i+1}^s = H(x_i^s)$

…

$x_t^s = ep_s$

Pre-Computation Attacks for Targeted Collision

- Precomputation ($t \times s$ steps, $2s \times \ell$ memory)



$sp_j = x_1^j$

… $x_2^j = H\left(x_1^j\right)$ …

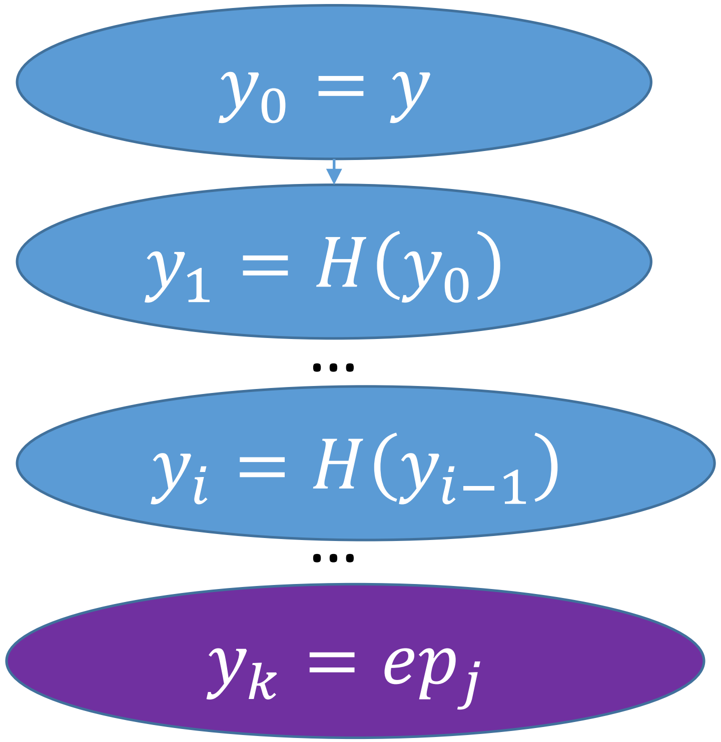
…

… $x_{i+1}^j = H\left(x_i^j\right)$ …

…

$x_t^j = ep_j$

- **Goal:** Find collision for target $y = H(x)$



$y_0 = y$

$y_1 = H(y_0)$

…

$y_i = H(y_{i-1})$

…

$y_k = ep_j$

Pre-Computation Attack sion

- Precomputation ($t \times s$ steps, $2s \times$

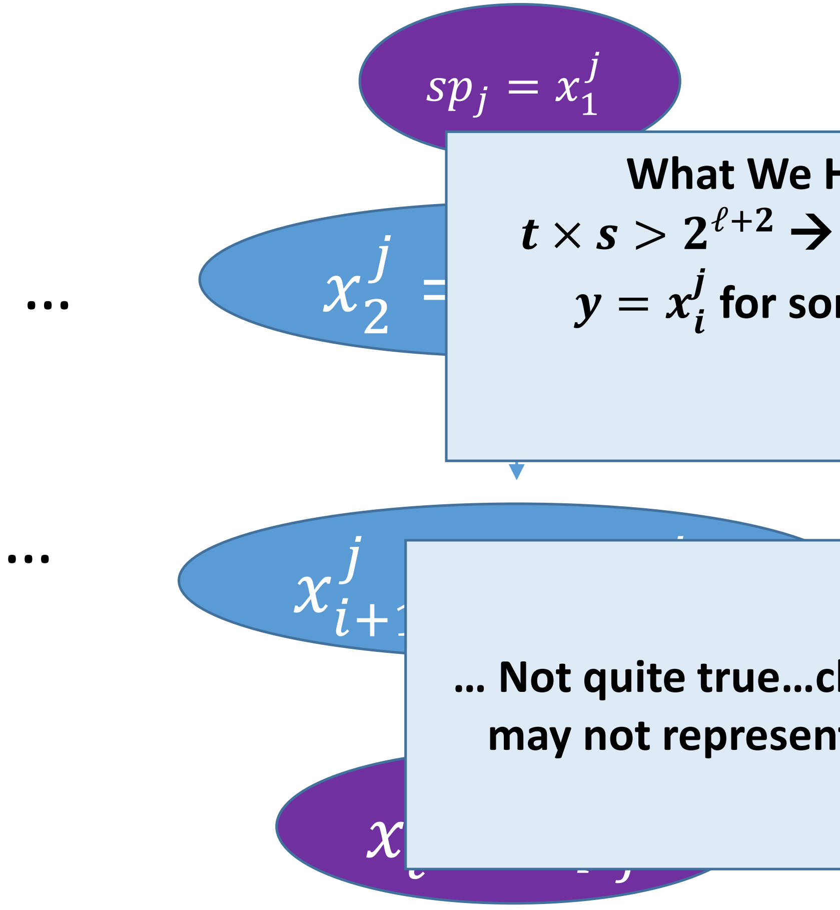
Suppose $y = x_i^j$ for some $i \le t, j \le s$

→

$y = H\left(x_{i-1}^j\right) = H^{i-1}(sp_j)$

(takes t steps to recover x_{i-1}^j from sp_j)

- **Goal:** F collision for target $y = H(x)$

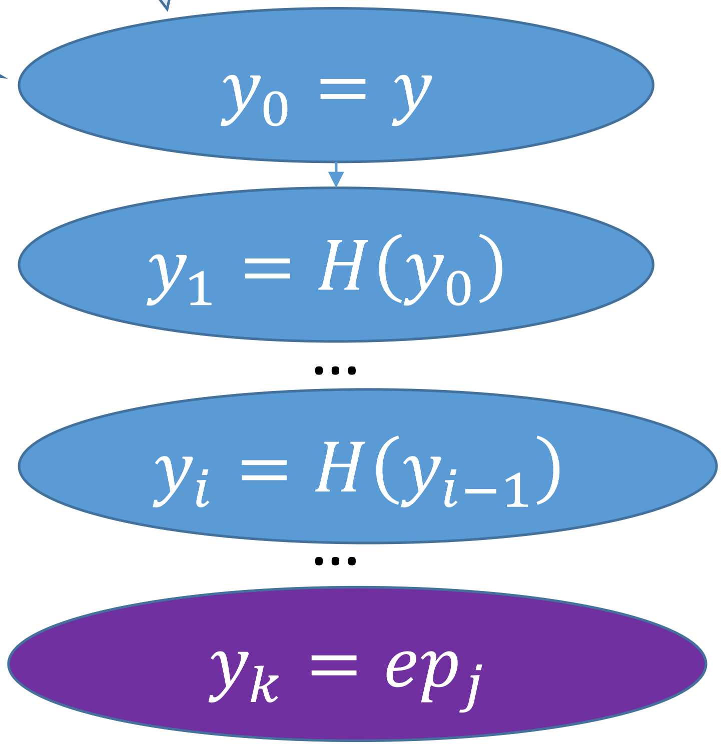


What We Hope is True:

$t \times s > 2^{\ell+2}$ → good chance that

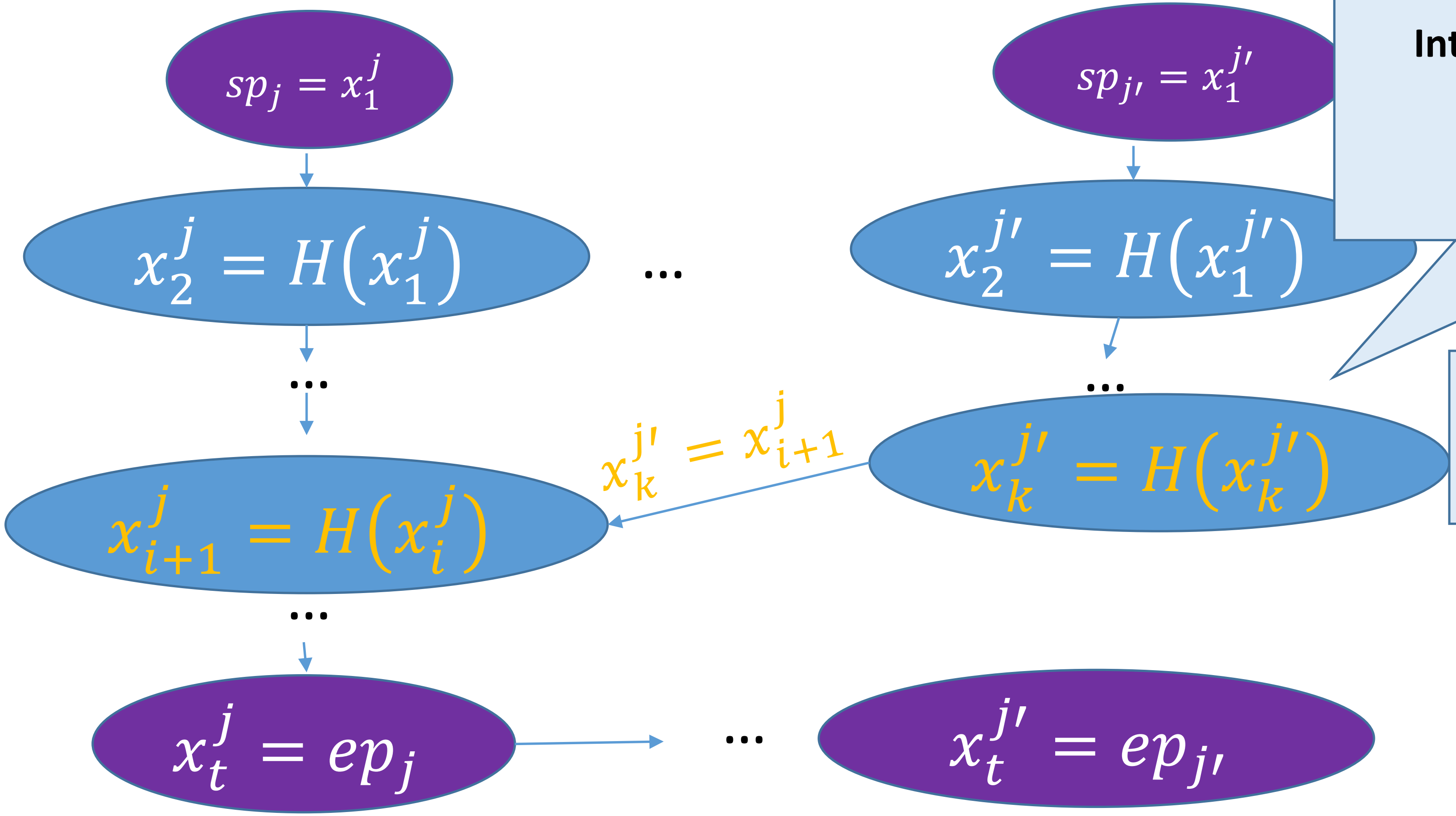
$y = x_i^j$ for some $i \le t, j \le s$

… Not quite true…chains can intersect and may not represent $t \times s$ *distinct* points



Intersecting Chains

- Precomputation ($t \times s$ steps, $2s$ memory)



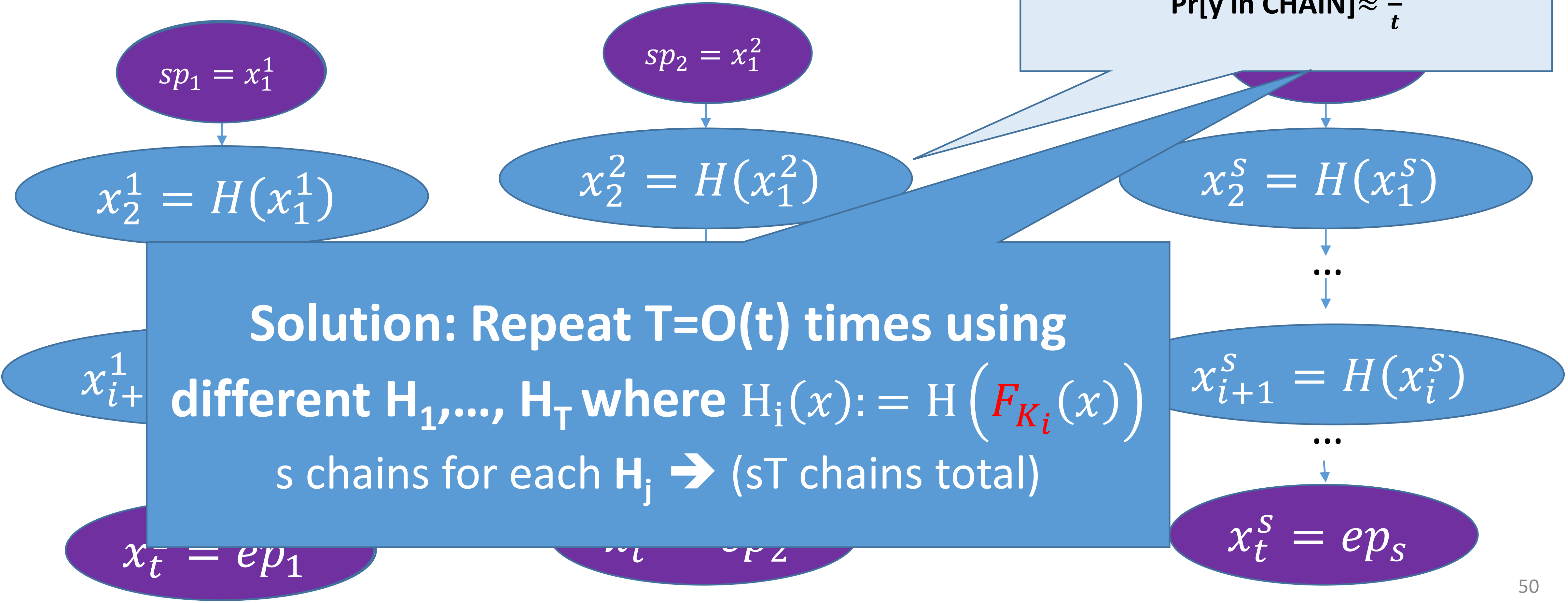
Intersecting chains contain $\ll st$ distinct points.

After initial intersection the chains merge together ☹

Targeted Collision Attacks

- Precomputation ($t \times s$ steps, $2s$ memory)

Fact: If $t^2 \times s < 2^\ell$ then chains contain $\Omega(ts)$ *distinct* points, but then Pr[y in CHAIN]$\approx \frac{1}{t}$

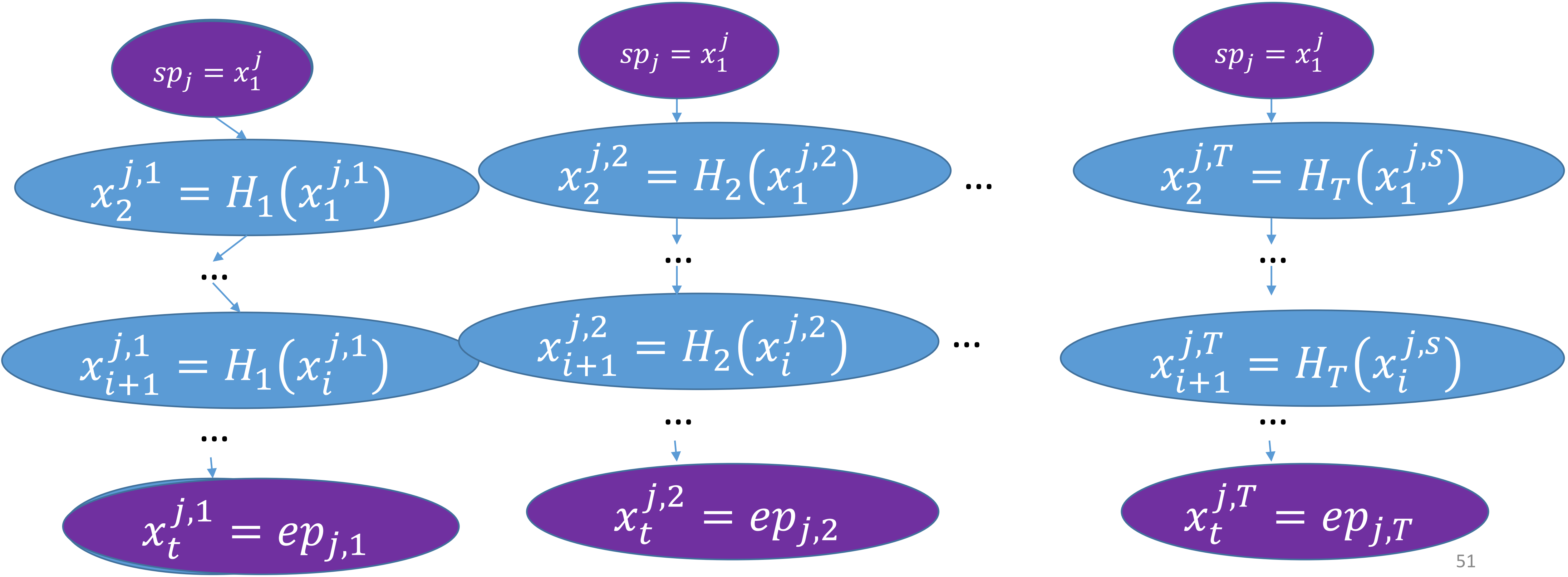


Solution: Repeat T=O(t) times using different H$_1$,…, H$_T$ where $\mathrm{H_i}(x) := \mathrm{H}\left(F_{K_i}(x)\right)$

s chains for each **H$_j$** ➔ (sT chains total)

Targeted Collision Attacks

- Precomputation (stT steps, $2sT$ memory)



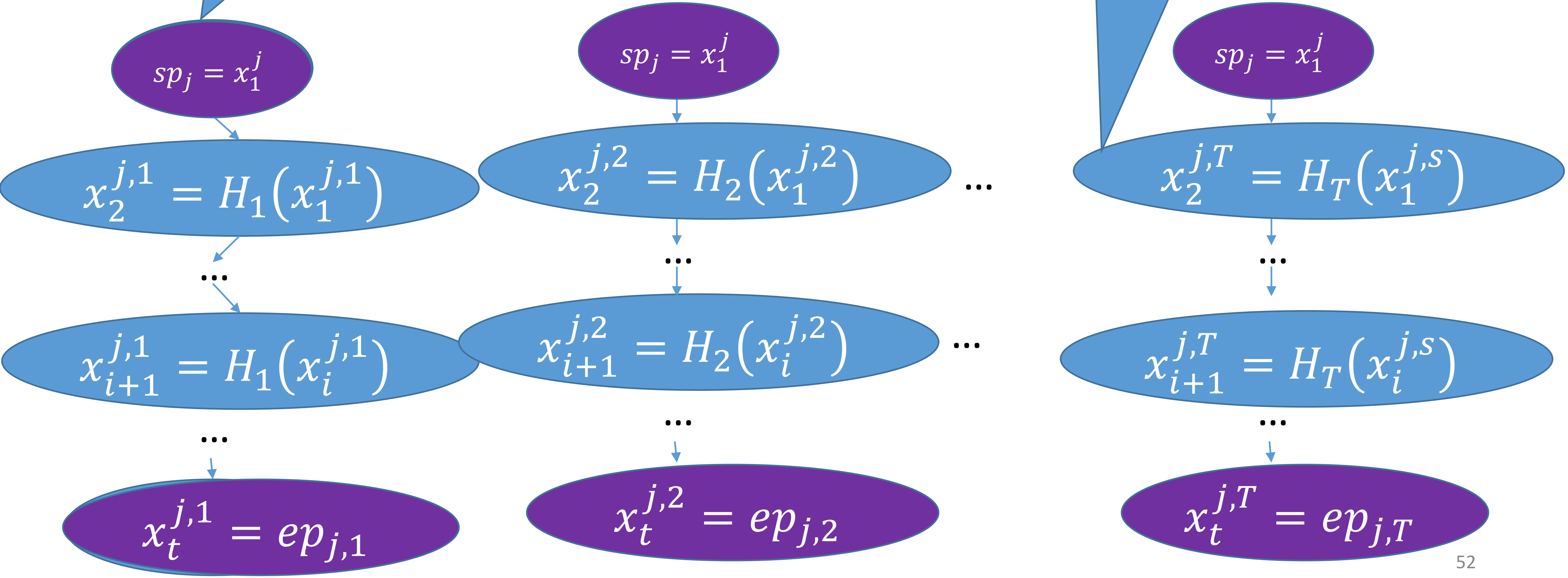
$sp_j = x_1^j$
$x_2^{j,1} = H_1\left(x_1^{j,1}\right)$
…
$x_{i+1}^{j,1} = H_1\left(x_i^{j,1}\right)$
…
$x_t^{j,1} = ep_{j,1}$
$sp_j = x_1^j$
$x_2^{j,2} = H_2\left(x_1^{j,2}\right)$
…
$x_{i+1}^{j,2} = H_2\left(x_i^{j,2}\right)$
…
$x_t^{j,2} = ep_{j,2}$
…
…
$sp_j = x_1^j$
$x_2^{j,T} = H_T\left(x_1^{j,s}\right)$
…
$x_{i+1}^{j,T} = H_T\left(x_i^{j,s}\right)$
…
$x_t^{j,T} = ep_{j,T}$

T Attacks

Repeat for each starting point sp_j with $1 \leq j \leq s$

$H_i(x) = H\left(F_{K_i}(x)\right)$

- Pre omputation (stT steps, $2sT$ memory)



$sp_j = x_1^j$

$x_2^{j,1} = H_1\left(x_1^{j,1}\right)$

…

$x_{i+1}^{j,1} = H_1\left(x_i^{j,1}\right)$

…

$x_t^{j,1} = ep_{j,1}$

$sp_j = x_1^j$

$x_2^{j,2} = H_2\left(x_1^{j,2}\right)$

…

$x_{i+1}^{j,2} = H_2\left(x_i^{j,2}\right)$

…

$x_t^{j,2} = ep_{j,2}$

…

…

$sp_j = x_1^j$

$x_2^{j,T} = H_T\left(x_1^{j,s}\right)$

…

$x_{i+1}^{j,T} = H_T\left(x_i^{j,s}\right)$

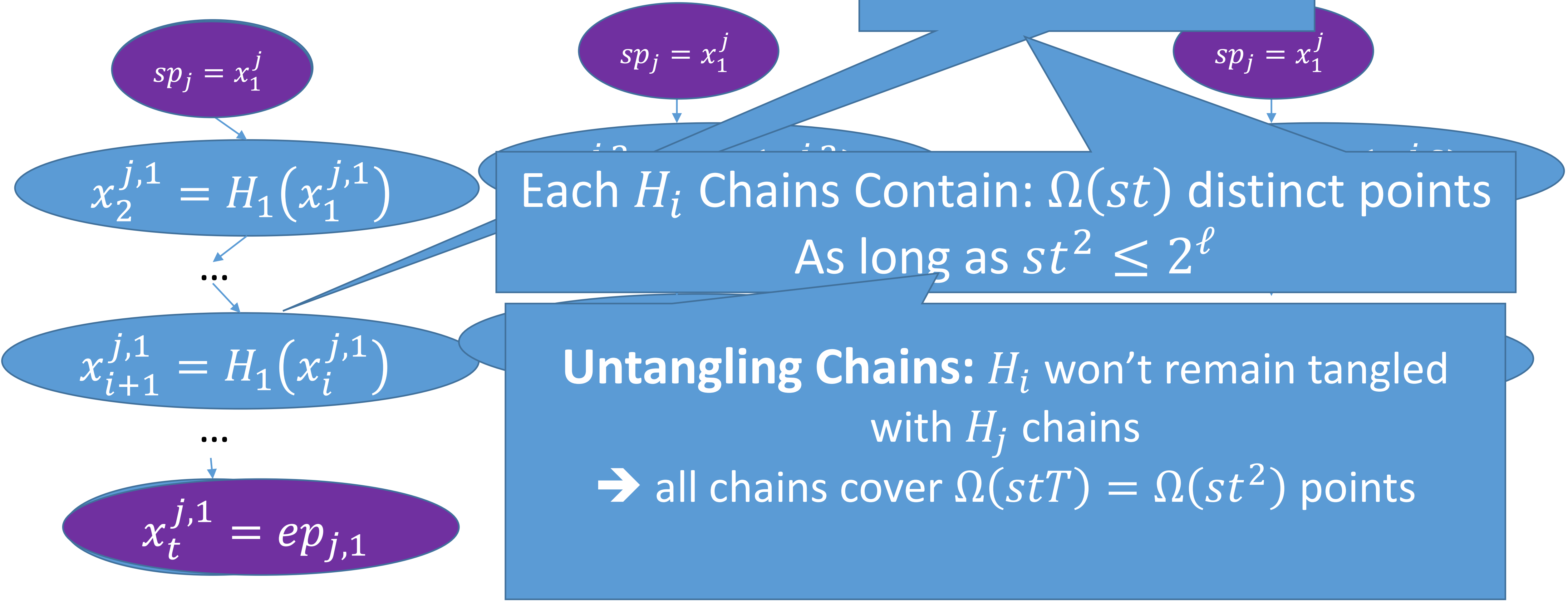
…

$x_t^{j,T} = ep_{j,T}$

Targeted Collision Attacks

- Precomputation (st^2 steps, $2st$ mem

$$H_i(x) = H\left(F_{K_i}(x)\right)$$



$sp_j = x_1^j$
$sp_j = x_1^j$
$sp_j = x_1^j$
$x_2^{j,1} = H_1\left(x_1^{j,1}\right)$
…
$x_{i+1}^{j,1} = H_1\left(x_i^{j,1}\right)$
…
$x_t^{j,1} = ep_{j,1}$

Each H_i Chains Contain: $\Omega(st)$ distinct points

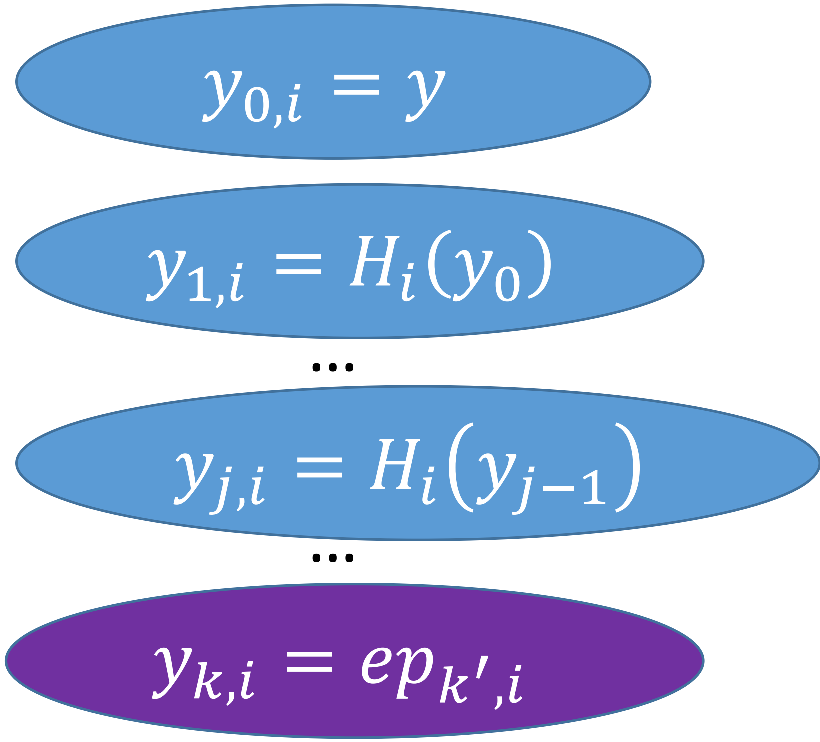
As long as $st^2 \leq 2^\ell$

Untangling Chains: H_i won't remain tangled with H_j chains

➔ all chains cover $\Omega(stT) = \Omega(st^2)$ points

Post-Processing

```
Input: y
For each $i \leq T$  // Compute T chains of length t
    $y' \coloneqq y$         // Start each chain at $y$
    For each $j \leq t$
        $y' := H_i(y')$    // $y' = y_{j,i}$
        For each k’ such that $y' = ep_{k',i}$
            $w' \coloneqq sp_{k'}$   // recompute $H_i$ chain at $sp_{k'}$
            For each $j' \leq t$
                If $y == H_i(w')$ return $F_{K_i}(\boldsymbol{w}')$ else $w' := H_i(w')$
```



$y_{0,i} = y$

$y_{1,i} = H_i(y_0)$

…

$y_{j,i} = H_i(y_{j-1})$

…

$y_{k,i} = ep_{k',i}$

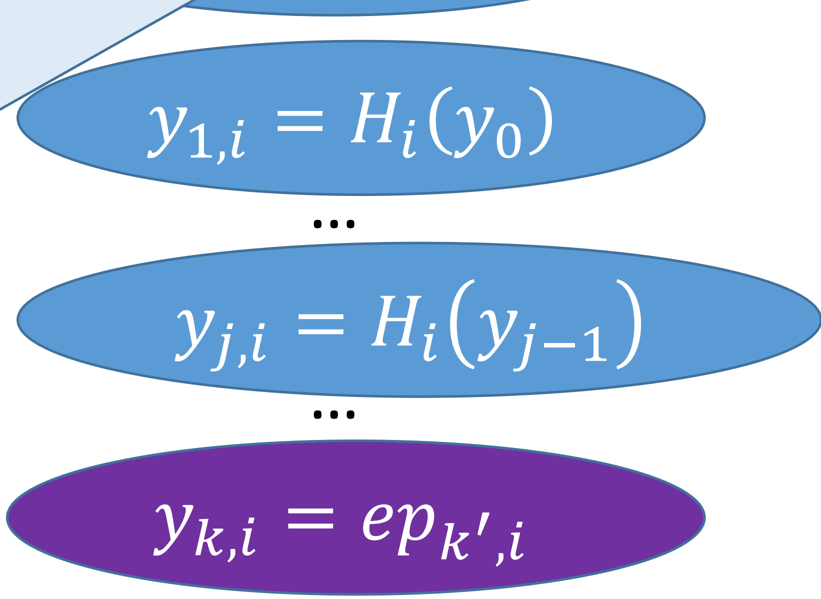
Post-Processing

Observation 1: If y is on any of the chains i.e., $y = x_k^{j,i}$ for some $i \leq T, j \leq t, k \leq s$

→ We will hit the endpoint $y' = ep_{k',i}$

→ We will find a pre-image of y

```
Input: y
For each $i \leq T$   // Compute T chains of length t
    $y' \coloneqq y$        // Start each chain at $y$
    For each $j \leq t$
        $y' := H_i(y')$    // $y' = y_{j,i}$
        For each k' such that $y' = ep_{k',i}$
            $w' \coloneqq sp_{k'}$   // recompute $H_i$ chain at $sp_{k'}$
            For each $j' \leq t$
                If $y == H_i(w')$ return $F_{K_i}(\mathbf{w'})$ else $w' := H_i(w')$
```



$y_{1,i} = H_i(y_0)$

…

$y_{j,i} = H_i(y_{j-1})$

…

$y_{k,i} = ep_{k',i}$

Post-Processing

```
Input: y
For each $i \leq T$   // Compute T chains of length t
    $y' := y$          // Start each chain at $y$
    For each $j \leq t$
        $y' := H_i(y')$     // $y' = y_{j,i}$
        For each k' such that $y' = ep_{k',i}$
            $w' := sp_{k'}$   // recompute $H_i$ chain at $sp_{k'}$
            For each $j' \leq t$
                If $y == H_i(w')$ return $F_{K_i}(w')$ else $w' := H_i(w')$
```

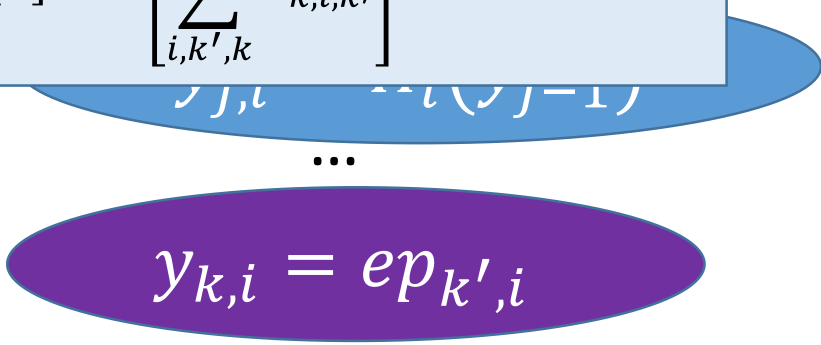
Observation 2: If $y' = ep_{k',i}$ when y is not on the H_i chain starting at $sp_{k'}$ **then we waste t steps checking this chain.**

Let $Z_{k,i,k'}$ be an indicator random variable for the event that $y_{k,i} = ep_{k',i}$ even though y is not on the chain

$$\mathbf{E}[Z_{k,i,k'}] = \mathbf{Pr}[y_{k,i} = ep_{k',i}] \approx 2^{-\ell}$$

Let Z be total number of false positives

$$\mathbf{E}[Z] = \mathbf{E}\left[\sum_{i,k',k} Z_{k,i,k'}\right] \approx stT2^{-\ell}$$


$y_{j,i} = H_i(y_{j-1})$

…

$y_{k,i} = ep_{k',i}$

Post-Processing

```
Input: y
For each $i \leq T$  // Compute T chains of length t
  $y' := y$        // Start each chain at $y$
  For each $j \leq t$
    $y' := H_i(y')$    // $y' = y_{j,i}$
    For each k' such that $y' = ep_{k',i}$
      $w' := sp_{k'}$   // recompute $H_i$ chain at $sp_{k'}$
      For each $j' \leq t$
        If $y == H_i(w')$ return $F_{K_i}(\mathbf{w'})$ else $w' := H_i(w')$
```

Let Z be total number of false positives

$$\mathbf{E}[Z] = \mathbf{E}\left[\sum_{i,k',k} Z_{k,i,k'}\right] \approx stT2^{-\ell}$$

Total Running Time: $O(Tt + Zt)$

If $stT \approx 2^{\ell}$ **and** $T = O(t)$ **then total running time is** $O(t^2)$

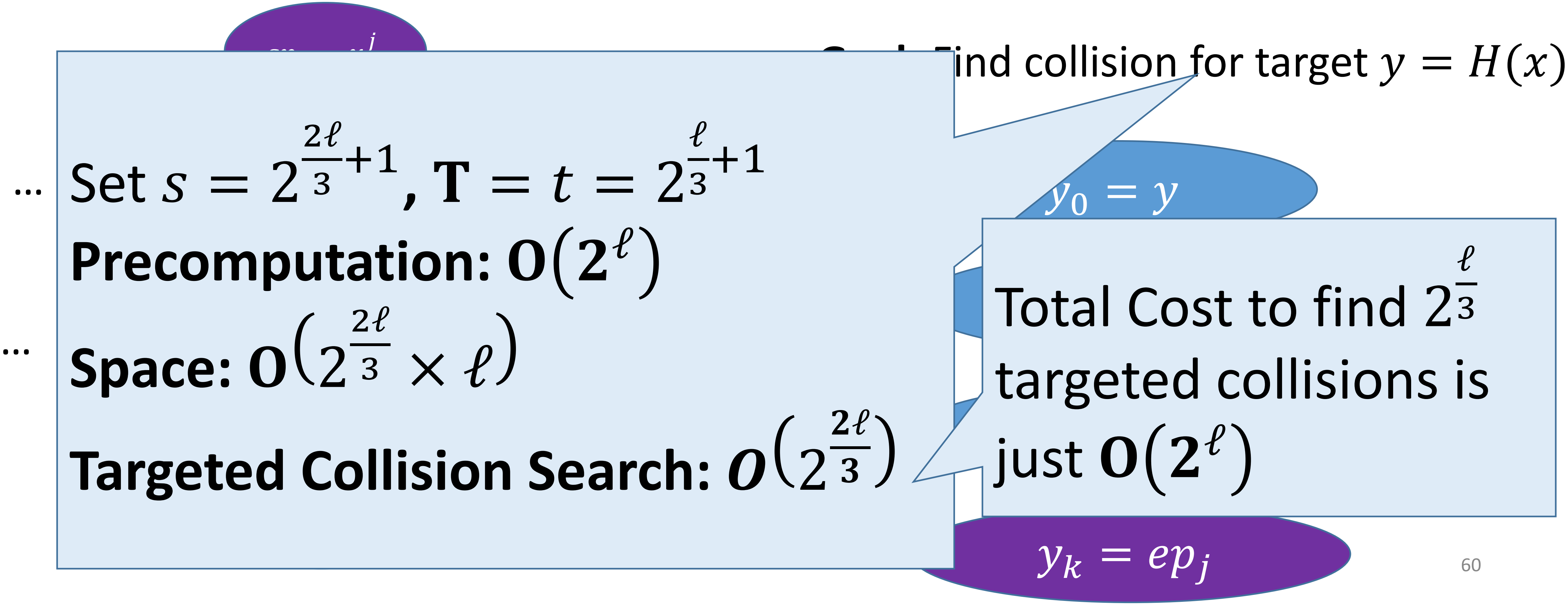
$y_{j,i} = H_i(y_{j-1})$

...

$y_{k,i} = ep_{k',i}$

Targeted Collision Attacks

- Precomputation ($tT \times s$ steps, $2sT \times \ell$ memory)



Set $s = 2^{\frac{2\ell}{3}+1}$, $\mathbf{T} = t = 2^{\frac{\ell}{3}+1}$

Precomputation: $\mathbf{O}\left(\mathbf{2}^{\ell}\right)$

Space: $\mathbf{O}\left(2^{\frac{2\ell}{3}} \times \ell\right)$

Targeted Collision Search: $\boldsymbol{O}\left(2^{\frac{2\ell}{3}}\right)$

Total Cost to find $2^{\frac{\ell}{3}}$ targeted collisions is just $\mathbf{O}\left(\mathbf{2}^{\ell}\right)$

Applications

- Key-Recovery Attacks on Block Cipher $E: \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$
  - Pre-Computation: $O(|\mathcal{K}|)$
  - **Crack** $2^{\frac{n}{3}}$ **secret keys in total time** $\boldsymbol{O}(|\mathcal{K}|)$ with space $s = \boldsymbol{O}\left(2^{\frac{2n}{3}}\right)$
  - Run prior attack with "hash function" $\mathrm{H}: \{0,1\}^n \to \{0,1\}^n$
    - $\mathrm{H}(K) = E_K(r)$ for some random (fixed) $r \in \{0,1\}^n$
- Password Cracking
  - Attacker is given $H'(x_1)$,…, $H'(x_k)$ for passwords $x_1, \ldots, x_k \in \mathcal{PWDs}$ with $|\mathcal{PWDs}| \ll |\mathcal{K}|$
  - **Goal:** Recover passwords $x_1, \ldots, x_k$
  - Can crack **all** $k = |\mathcal{PWDs}|^{1/3}$ passwords in total time $\boldsymbol{O}(|\mathcal{PWDs}|)$ with space $s = \boldsymbol{O}\left(|\mathcal{PWDs}|^{2/3}\right)$
  - Domain Challenge: $\mathrm{H}': |\mathcal{PWDs}| \to \{0,1\}^n$ with $|\mathcal{PWDs}| \ll 2^n$
    - Define (pseudo)random mapping $\mu: \{0,1\}^n \to \mathcal{PWDs}$
    - Run prior attack with "hash function" $\mathrm{H}: \mathcal{PWDs} \to \mathcal{PWDs}$ as $\mathrm{H}(x) = \mu\left(\mathrm{H}'(x)\right)$

Week 5: Topic 3:
Random Oracle Model +
Hashing Applications

When Collision Resistance Isn’t Enough

- **Example**: Message Commitment
  - Alice sends Bob: $c = H^s(r \parallel m)$ (e.g., predicted winner of NCAA Tournament)
  - Alice can later reveal message (e.g., after the tournament is over)
    - Just send r and m (note: r has fixed length)
    - Why can Alice not change her message?
      - Collision Resistance ➔ Alice can’t find r’ and m’ s.t. $c = H^s(r' \parallel m')$
  - In the meantime Bob shouldn’t learn *anything* about m

- **Problem**: Let (Gen,H’) be collision resistant then so is (Gen,H)

$$H^s(x_1, \dots, x_d) = H'^s(x_1, \dots, x_d) \parallel x_d$$

When Collision Resistance Isn’t Enough

- **Problem**: Let (Gen,H’) be collision resistant then so is (Gen,H)

$$H^s(x_1,\dots,x_d) = H'^s(x_1,\dots,x_d) \parallel x_d$$

Note: An H^s collision trivially yields a H'^s collision

- (Gen,H) definitely does not hide all information about input $(x_1,\dots,x_d)$

- **Conclusion**: Collision resistance is not sufficient for message commitment

The Tension

- **Example**: Message Commitment
  - Alice sends Bob: $\mathrm{H}^s(r \parallel m)$ (e.g., predicted winners of NCAA Final Four)
  - Alice can later reveal message (e.g., after the Final Four is decided)
  - In the meantime Bob shouldn’t learn anything about m

This is still a reasonable approach in practice!

- No attacks when instantiated with any reasonable candidate (e.g., SHA3)
- Cryptographic hash functions seem to provide “something” beyond collision resistance, but how do we model this capability?

Random Oracle Model

- Model hash function H as a truly random function
- Algorithms can only interact with H as an oracle
  - **Query**: x
  - **Response**: H(x)
- If we submit the same query you see the same response
- If x has not been queried, then the value of H(x) is uniform

- **Real World:** H instantiated as cryptographic hash function (e.g., SHA3) of fixed length (no Merkle-Damgård)

Back to Message Commitment

- **Example**: Message Commitment
  - Alice sends Bob: $\mathrm{H}(r \parallel m)$ (e.g., predicted winners of NCAA Final Four)
  - Alice can later reveal message (e.g., after the Final Four is decided)
    - Just send r and m (note: r has fixed length)
    - Why can Alice not change her message?
  - In the meantime Bob shouldn't learn *anything* about m

- **Random Oracle Model**: Above message commitment scheme is secure (Alice cannot change m + Bob learns nothing about m)
- Security Definition + Proof later…

Random Oracle Model: Pros

- It is easier to prove security in Random Oracle Model

- Suppose we are simulating attacker A in a reduction
    - **Extractability**: When A queries H at x we **see this query** and learn x (and can easily find H(x))
    - **Programmability**: We can set the value of H(x) to a value of our choice
        - As long as the value is correctly distribute i.e., close to uniform
- Both **Extractability** and **Programmability** are useful tools for a security reduction!

Random Oracle Model: Pros

- It is easier to prove security in Random Oracle Model

- Provably secure constructions in random oracle model are often much more efficient (compared to provably secure construction is “standard model”

- Sometimes we only know how to design provably secure protocol in random oracle model

Random Oracle Model: Cons

- Lack of formal justification
- Why should security guarantees translate when we instantiate random oracle with a real cryptographic hash function?

- We can construct (contrived) examples of protocols which are
  - Secure in random oracle model…
  - But broken in the real world

Random Oracle Model: Justification

“A proof of security in the random-oracle model is significantly better than no proof at all.”

- **Evidence of sound design** (any weakness involves the hash function used to instantiate the random oracle)
- **Empirical Evidence for Security**

> “there have been no successful real-world attacks on schemes proven secure in the random oracle model”

Hash Function Application: Fingerprinting

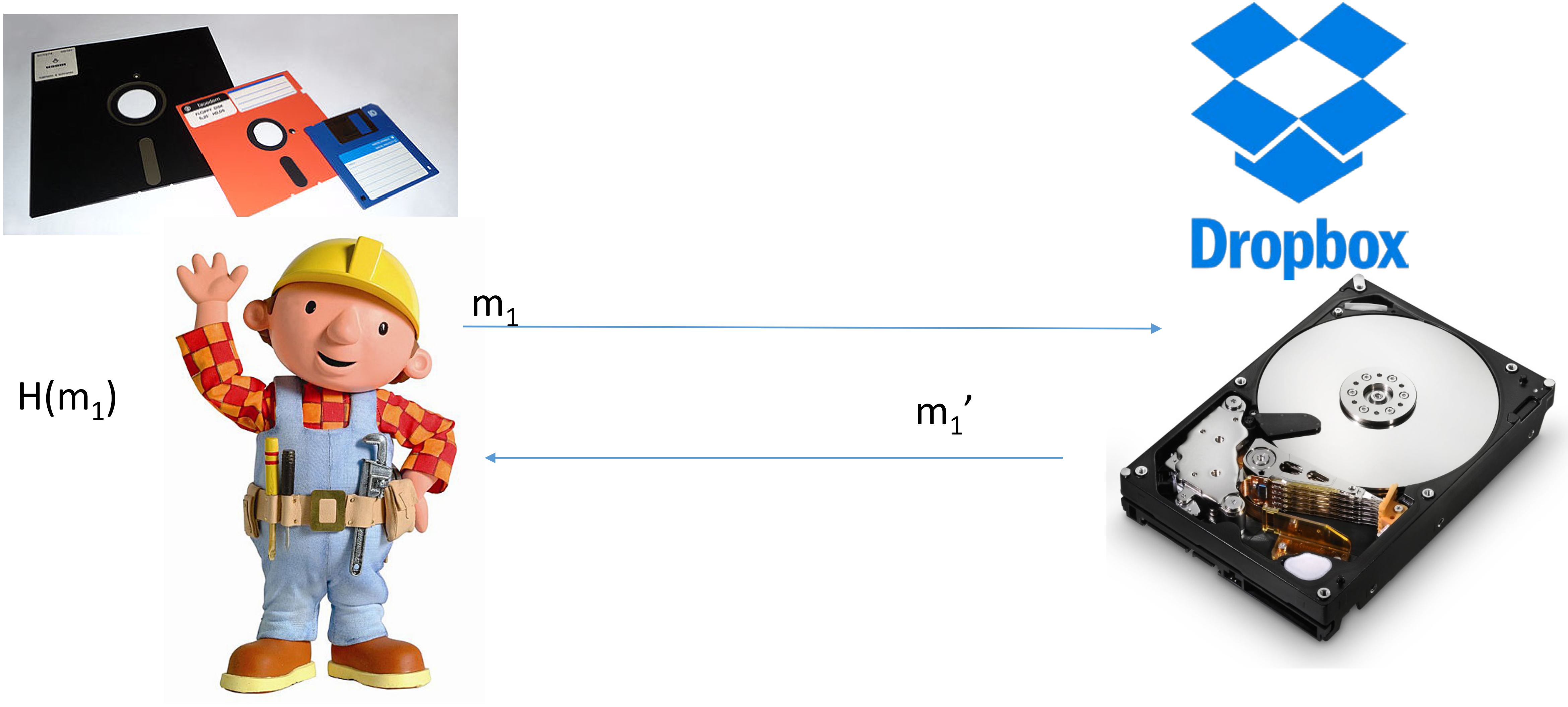
- The hash h(x) of a file x is a unique identifier for the file
  - Collision Resistance → No need to worry about another file y with H(y)=H(y)

- Application 1: Virus Fingerprinting

- Application 2: P2P File Sharing

- Application 3: Data deduplication

Tamper Resistant Storage



m_1

$H(m_1)$

m_1'

Tamper Resistant Storage

File Index	Hash
1	H(m_1)
2	H(m_2)
3	H(m_3)

Disadvantage: Too many hashes to store

Dropbox

m_1, m_2, m_3

Send file 1

m_1'

Tamper Resistant Storage

Disadvantage: Need all files to compute hash m_1,m_2,m_3

$H(m_1,m_2,m_3)$

m_1,m_2,m_3

Send file 1

m_1'

Dropbox

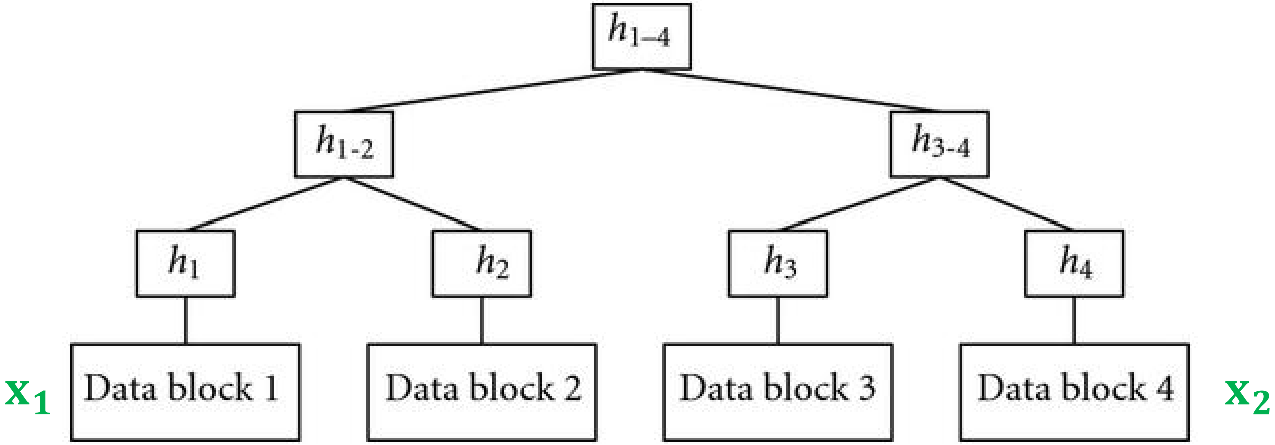
Merkle Trees

$$\mathbf{MT^{s}}(x) := \mathrm{h}^{s}(x)$$

$$\mathbf{MT^{s}}\left(\mathbf{x}_1, \ldots, \mathbf{x}_{2^i}\right) :=$$

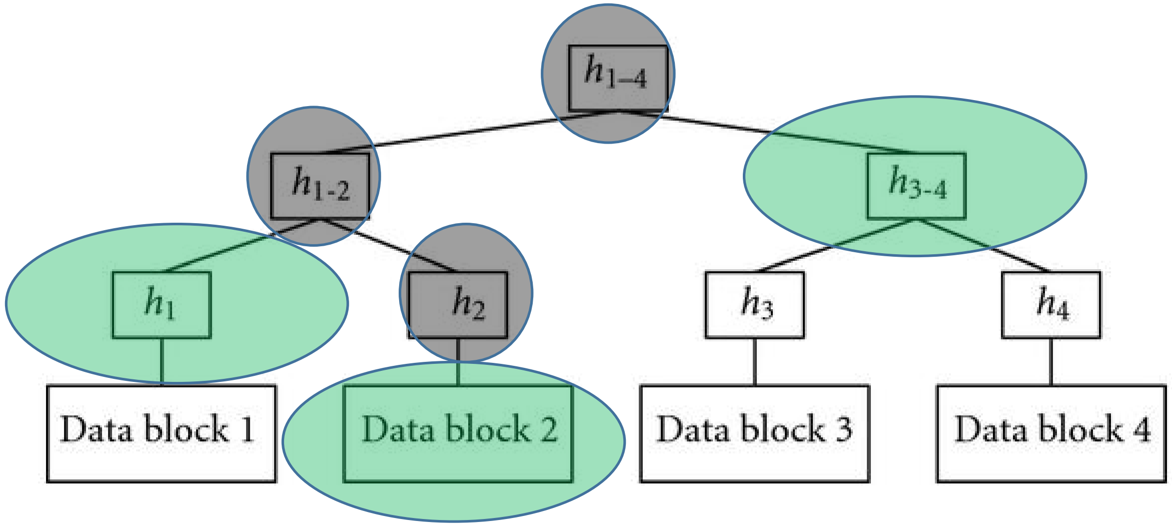
$$\mathrm{h}^{s}\left(\mathbf{MT^{s}}\left(\mathbf{x}_1, \ldots, \mathbf{x}_{2^{i-1}}\right), \mathbf{MT^{s}}\left(\mathbf{x}_{2^{i-1}+1}, \ldots, \mathbf{x}_{2^i}\right)\right)$$

Theorem: Let (Gen, h^s) be a collision resistant hash function then $\mathbf{MT^{s}}$ is collision resistant.



Merkle Trees

- **Proof of Correctness for data block 2**



- **Verify that root matches**
- **Proof consists of just log(n) hashes**
  - **Verifier only needs to permanently store only one hash value**

Tamper Resistant Storage

Root: H_{1-4}

m_1,m_2,m_3,m_4

Send file 2

m_2',h_1,h_{3-4}

Dropbox

Commitment Schemes

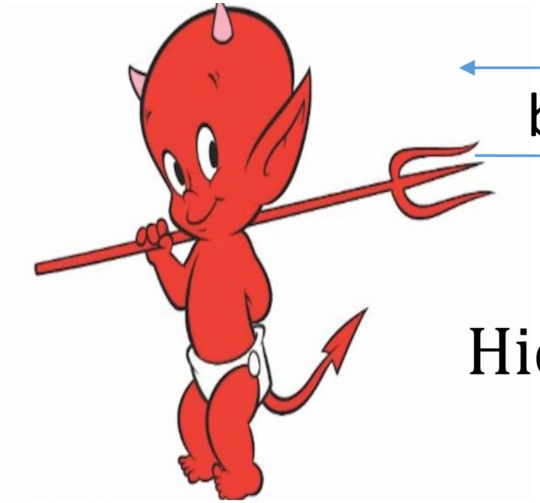
- Alice wants to commit a message m to Bob
  - And possibly reveal it later at a time of her choosing
- Properties
  - Hiding: commitment reveals nothing about m to Bob
  - Binding: it is infeasible for Alice to alter message

Commitment Hiding $\left(\text{Hiding}_{A,Com}(n)\right)$

m$_0$,m$_1$

commit(r,m$_b$)

b’



$$\text{Hiding}_{A,Com}(n) = \begin{cases} 1 & \text{if } b = b' \\ 0 & otherwise \end{cases}$$

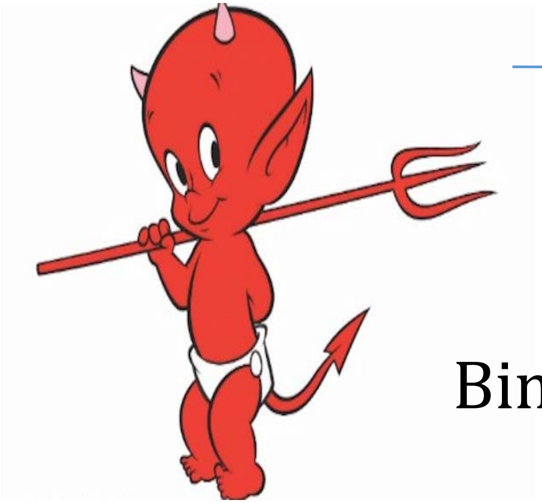
r = Gen(.)

Bit b

$$\forall PPT\ A\ \exists\mu\ (\text{negligible})\ \text{s.t}$$

$$\Pr\left[\text{Hiding}_{A,Com}(n) = 1\right] \le \frac{1}{2} + \mu(n)$$

Commitment Binding ($\text{Binding}_{A,Com}(n)$)



r_0, r_1, m_0, m_1 →

$$\text{Binding}_{A,Com}(n) = \begin{cases} 1 & \text{if } \textbf{commit}(\mathbf{r_0},\mathbf{m_0}) \textbf{= commit}(\mathbf{r_1},\mathbf{m_1}) \\ 0 & otherwise \end{cases}$$

$$\forall PPT\ A\ \exists \mu \text{ (negligible) s.t}$$
$$\Pr\left[\text{Binding}_{A,Com}(n) = 1\right] \le \mu(n)$$

Secure Commitment Scheme

- **Definition:** A secure commitment scheme is **hiding** and **binding**
- **Hiding**

$$\forall PPT\ A\ \exists\mu\ (\text{negligible})\ \text{s.t}$$
$$\Pr\left[\text{Hiding}_{A,Com}(n) = 1\right] \leq \frac{1}{2} + \mu(n)$$

- **Binding**

$$\forall PPT\ A\ \exists\mu\ (\text{negligible})\ \text{s.t}$$
$$\Pr\left[\text{Binding}_{A,Com}(n) = 1\right] \leq \mu(n)$$

Commitment Scheme in Random Oracle Model

- **Commit**(r,m):=H(m|r)
- **Reveal**(c):= (m,r)

Theorem: In the random oracle model this is a secure commitment scheme.

Proof Intuition: Let BAD event that attacker queries $\mathrm{H}(r \parallel m')$ for any message m’ on any of q queries

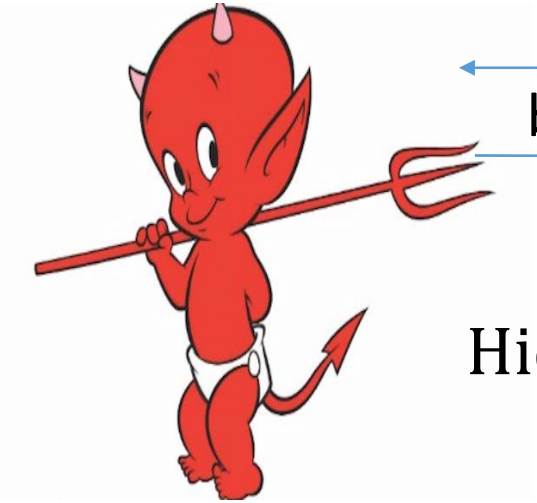
- As long as the event BAD never occurs Bob learns nothing about m (in an information theoretic sense)
- If r is a random n-bit string then $\Pr[\mathrm{BAD}] \leq \frac{\mathrm{q}}{2^{\mathrm{n}}}$

Commitment Hiding $\left(\mathrm{Hiding}_{A,Com}(n)\right)$

m$_0$,m$_1$ →

← **H(r,m$_b$)**

b' →



$$\mathrm{Hiding}_{A,Com}(n) = \begin{cases} 1 & if b = b' \\ 0 & otherwise \end{cases}$$

r = Gen(1^n)

Bit b

$$\forall \cancel{PPT}\ A\ making\ at\ most\ q(n)\ queries$$

$$\Pr\left[\mathrm{Hiding}_{A,Com}(n) = 1\right] \le \frac{1}{2} + \frac{q(n)}{2^n}$$

Other Applications

- Password Hashing

- Key Derivation

Next Week

- Stream Ciphers
- Block Ciphers
- Feistel Networks
- DES, 3DES
- Read Katz and Lindell 6.1-6.2