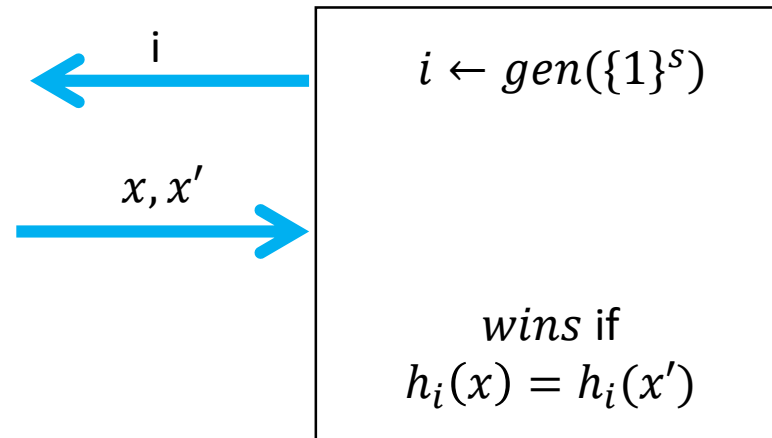


# Cryptographic hash functions

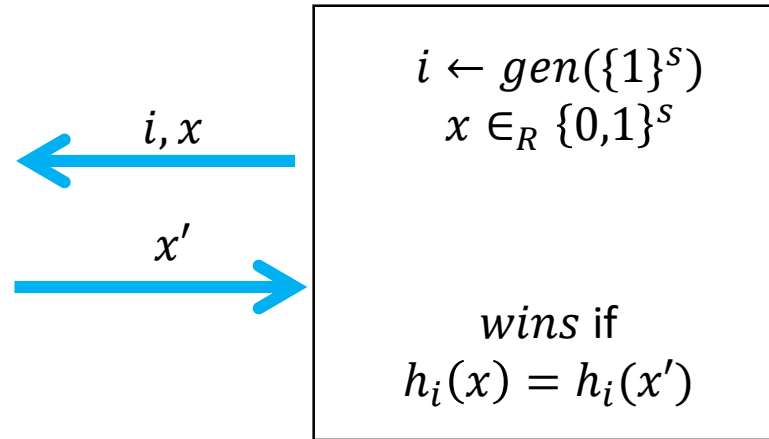
# Definition of hash function

- $H := \{h_1, \dots, h_m\}$ 
  - $h_i : \{0,1\}^n \rightarrow \{0,1\}^{\ell(n)}$
- Compression :  $\ell(n) < n$
- $Gen(1^s)$  picks an index between 1 and  $m$ .

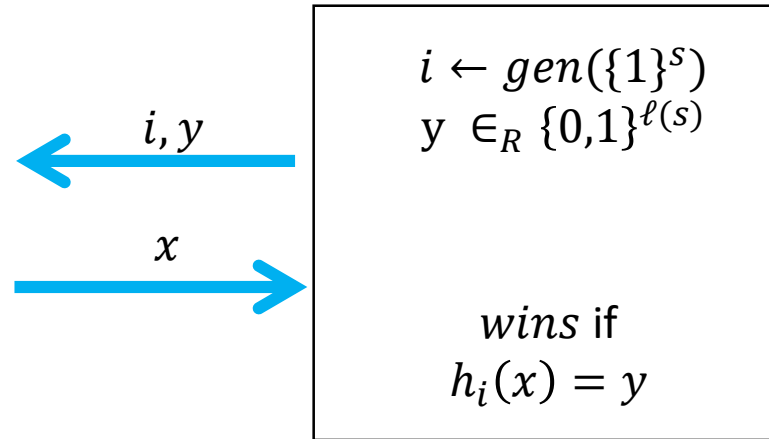
# Collision resistance



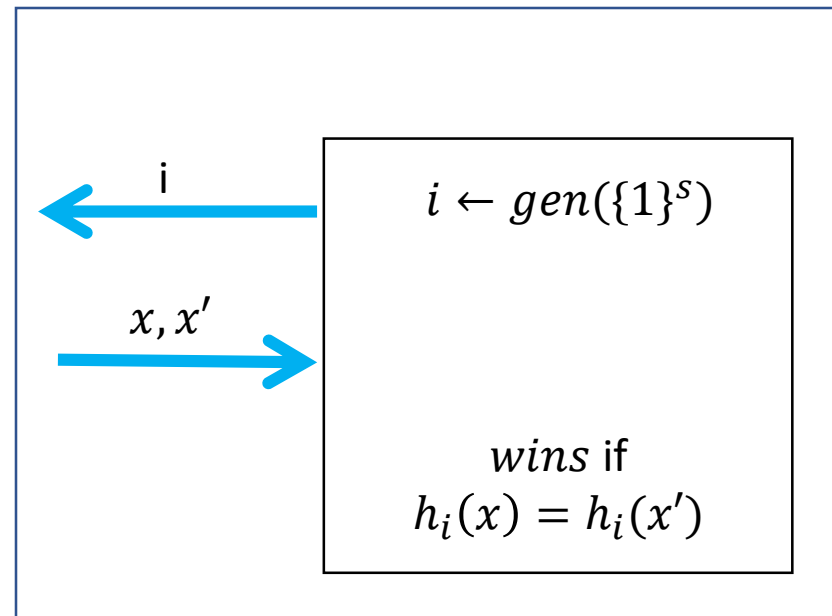
# Second-collision resistant



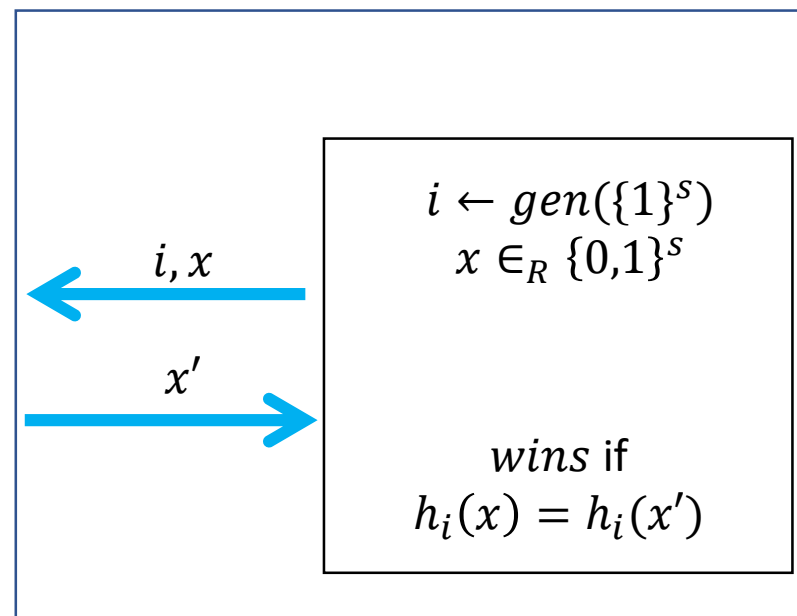
# Preimage resistance



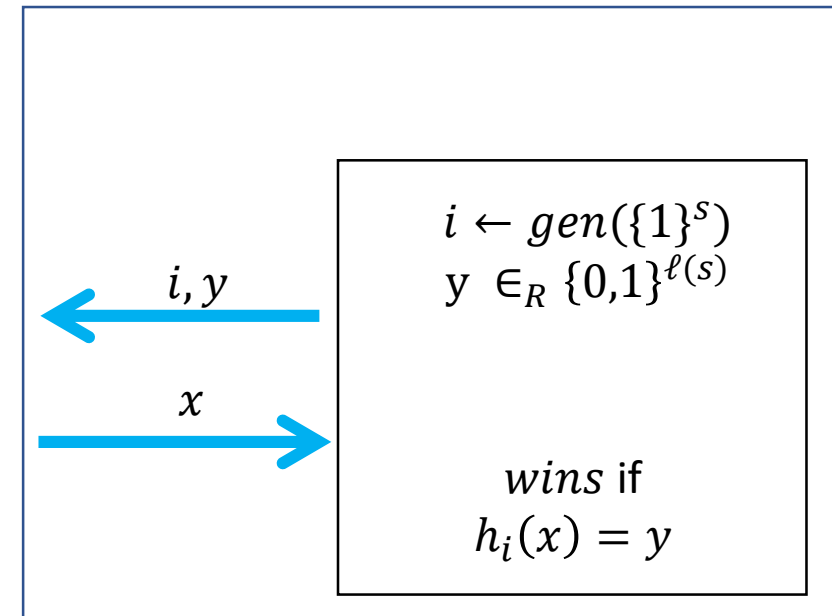
# Comparison between different security



*collision resistant*



*target – collision resistance*



*preimage resistant*

# Birthday attack

- What is the probability that in a class of 23 people there are at least two people with the same birthday.
  - Answer: more than 50% chance
- General question: Suppose you sample  $m$  values from  $n$  values, what is the probability that there exists at least two values that are the same.
  - Answer: if  $m = \sqrt{n}$  then probability is about one-half

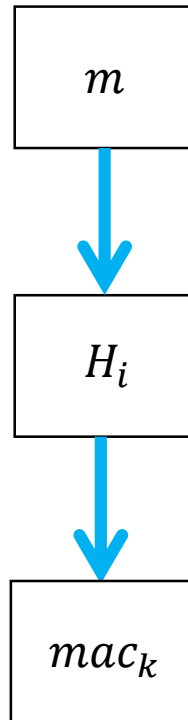
# Proof of the birthday attack

- If you store  $\sqrt{n}$  output with a given input the probability that a new input gets sent to a previously visited output is  $1/\sqrt{n}$
- The probability that  $\sqrt{n}$  elements all get mapped to fresh output is

$$\left(1 - \frac{1}{\sqrt{n}}\right)^{\sqrt{n}} \geq 1/e$$



# Mac using hash functions



# HMAC

- Global constants  $v_1, v_2$
- Gen
  - $s \leftarrow \text{Gen}(1^s)$
  - $k \in_R \{0,1\}^s$
- Auth(m)
  - $w_1 \leftarrow (k \oplus v_2, m)$
  - $w_2 \leftarrow H^s(w_1)$
  - $w_3 \leftarrow (k \oplus v_1, w_2)$
  - $t \leftarrow H(w_3)$

# Random oracle heuristic

- Assume that a hash function acts as a random function
- Allows us to prove security for efficient schemes
- Unsound but
  - Only for contrived example
  - Never broken for practical schemes

# Applications of hash functions

- Virus fingerprinting
- Deduplication
- Password hashing
- File changes/integrity

# Virus fingerprinting

- Hash the virus using the hash function
- To lookup a virus, simply look at the output of the function and see if it maps to a known virus
- False positives imply collision

# Deduplication

- Avoid storing the same thing in memory many times
- Uses hash function to index values so that we don't need to copy the same thing many times.

# File changes / integrity

- To keep track of changes, we keep a list of hash for every function
- The output of the hash function can be much shorter than the size of the files

# Proper way to hash passwords

- Naïve way to hash passwords
  - $h \leftarrow H(pwd)$
  - Same password hashed to same value
- Correct way to hash a password
  - $r \in_R \{0,1\}^n$
  - $h \leftarrow H(r, pwd)$
  - $(r, h)$



# Bad way to hash passwords (xkcd)

HACKERS RECENTLY LEAKED **153 MILLION** ADOBE USER EMAILS, ENCRYPTED PASSWORDS, AND PASSWORD HINTS. ADOBE ENCRYPTED THE PASSWORDS IMPROPERLY, MISUSING BLOCK-MODE 3DES. THE RESULT IS SOMETHING WONDERFUL:

USER	PASSWORD	HINT	
4e18acc1ab27a2d6		WEATHER VANE SWORD	<input type="text"/>
4e18acc1ab27a2d6			<input type="text"/>
4e18acc1ab27a2d6	a0a2876c7b1ea1fca	NAME 1	<input type="text"/>
8babb6299e06eb6d		DUH	
8babb6299e06eb6d	a0a2876c7b1ea1fca		<input type="text"/>
8babb6299e06eb6d	85e9da81a8a78adc	57	
4e18acc1ab27a2d6		FAVORITE OF 12 APOSTLES	
1ab29ae86da6e5ca	7a246a0a2876eb1e	WITH YOUR OWN HAND YOU HAVE DONE ALL THIS	
a1f9b2b6299e7a2b	ecodec1e6ab797397	SEXY EARLOBES	<input type="text"/>
a1f9b2b6299e7a2b	617ab027727ad85	BEST TOS EPISODE	<input type="text"/>
39738b7adb068af7	617ab027727ad85	SUGARLAND	
1ab29ae86da6e5ca		NAME + JERSEY #	
877ab7889d3862b1		ALPHA	<input type="text"/>
877ab7889d3862b1			<input type="text"/>
877ab7889d3862b1			<input type="text"/>
877ab7889d3862b1		OBVIOUS	<input type="text"/>
877ab7889d3862b1		MICHAEL JACKSON	
38a7c9279codeb44	9dca1d79d4dec6d5		
38a7c9279codeb44	9dca1d79d4dec6d5	HE DID THE MASH, HE DID THE PURLOINED	<input type="text"/>
38a7c9279codeb44			<input type="text"/>
a8ae5245c7b7af7a	9dca1d79d4dec6d5	EARL LATER-3 POKEMON	

THE GREATEST CROSSWORD PUZZLE  
IN THE HISTORY OF THE WORLD

# Hash tree

- Goal
  - Client has constant overhead
  - Server holds client's database  $x_1, \dots, x_n$
  - Client requests  $x_i$
  - Server sends  $x'_i$
  - How can client verify that  $x'_i = x_i$  with only logarithmic overhead

# Hash tree

- $Leaf_i = Hash(File_i)$
- Non-leaf node
  - $Hash(Hash(node.left) || Hash(node.right))$
- Client only need to store the root
- To prove that a given file is correct, the server only needs to send the client hashes of nodes that follow the path from the given leaf to the root node.