3 HASH FUNCTIONS

CRYPTO 101: Building Blocks

cryptography101.ca

V3 outline

- V3a: Fundamental concepts
- V3b: Relationships between PR, 2PR, CR
- V3c: Generic attacks
- V3d: Iterated hash functions
- V3e: SHA-256

V3a
Fundamental concepts

HASH FUNCTIONS

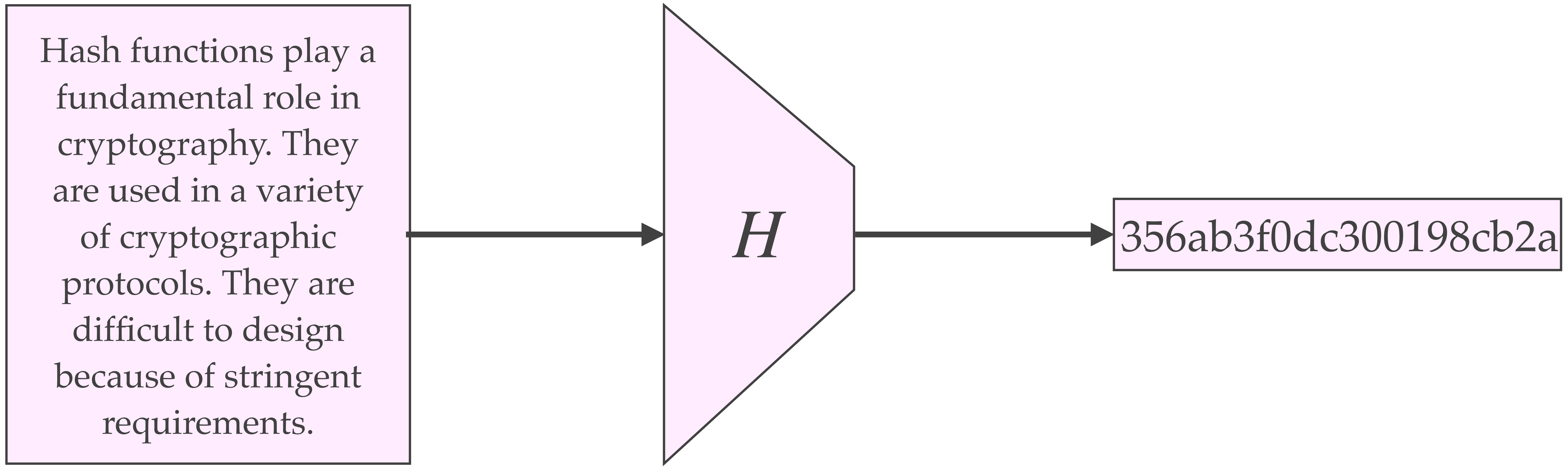
CRYPTO 101: Building Blocks

cryptography101.ca

Definitions and terminology

- Hash functions play a fundamental role in cryptography
- They are used in a variety of cryptographic primitives and protocols.
- They are very difficult to design because of stringent security and performance requirements.
- The most commonly used hash functions are:
  - SHA-1
  - SHA-2 family: SHA-224, SHA-256, SHA-384, SHA-512
  - SHA-3 family

What is a hash function?


Hash functions play a fundamental role in cryptography. They are used in a variety of cryptographic protocols. They are difficult to design because of stringent requirements.
H
356ab3f0dc300198cb2a

See:
www.xorbin.com/tools/md5-hash-calculator (MD5)
www.xorbin.com/tools/sha1-hash-calculator (SHA-1)
www.xorbin.com/tools/sha256-hash-calculator (SHA-256)

Example: SHA-256

SHA-256 : $\{0,1\}^* \longrightarrow \{0,1\}^{256}$

SHA-256("Hello there") =

```
0x4e47826698bb4630fb4451010062fadbf85d61427cbdfaed7ad0f23f239bed89
```

SHA-256("Hello There") =

```
0xabf5dacd019d2229174f1daa9e62852554ab1b955fe6ae6bbbb214bab611f6f5
```

Definition of a hash function

A hash function is a mapping H such that:

1. H maps binary messages of arbitrary lengths $\leq L$ to outputs of a fixed length n:
   $H : \{0,1\}^{\leq L} \rightarrow \{0,1\}^n$. ($L$ is usually large, e.g., $L = 2^{64}$, whereas n is small, e.g. $n = 256$.)
2. $H(x)$ can be efficiently computed for all $x \in \{0,1\}^{\leq L}$.

- H is called an n-bit hash function. $H(x)$ is called the hash or message digest of x.
- Notes:
  - The description of a hash function is public; there are no secret keys.
  - For simplicity, we will usually write $\{0,1\}^*$ instead of $\{0,1\}^{\leq L}$.
  - More generally, a hash function is an efficiently computable function from a set S to a set T.

Toy hash function

x	$H(x)$	x	$H(x)$	x	$H(x)$	x	$H(x)$
0	00	1	01				
00	11	01	01	10	01	11	00
000	00	001	10	010	11	011	11
100	11	101	01	110	01	111	10
0000	00	0001	11	0010	11	0011	00
0100	01	0101	10	0110	10	0111	01
1000	11	1001	01	1010	00	1011	01
1100	10	1101	00	1110	00	1111	11

$H : \{0,1\}^{\leq 4} \longrightarrow \{0,1\}^2$

- (00,1000) is a **collision**.
- 1001 is a **preimage** of 01.
- 10 is a **second preimage** of 1011.

Some applications of hash functions

- Hash functions are used in all kinds of applications, including some that they were not designed for.
- One reason for this widespread use of hash functions is speed.

Preimage resistance (PR)

Definition: A hash function $H : \{0,1\}^* \longrightarrow \{0,1\}^n$ is preimage resistant if, given a hash value $y \in_R \{0,1\}^n$, it is computationally infeasible to find (with non-negligible success probability) *any* $x \in \{0,1\}^*$ with $H(x) = y$. (x is called *a* preimage of y.)

Password protection on a multi-user computer system:

- The server stores [userid, H(password)] in a password file.
- If an attacker obtains a copy of the password file, she does not learn any passwords.
- This application requires preimage resistance.

2nd preimage resistance (2PR)

Definition: A hash function $H : \{0,1\}^* \longrightarrow \{0,1\}^n$ is 2nd preimage resistant if, given $x \in_R \{0,1\}^*$, it is computationally infeasible to find (with non-negligible success probability) *any* $x' \in \{0,1\}^*$ with $x' \neq x$ and $H(x') = H(x)$.

Modification Detection Codes (MDCs):

- To ensure that a message m is not modified by unauthorized means, one computes $H(m)$ and protects $H(m)$ from unauthorized modification.
- This is useful in malware protection.
- This application requires 2nd preimage resistance.

Collision resistance (CR)

Definition: A hash function $H : \{0,1\}^* \longrightarrow \{0,1\}^n$ is collision resistant if it is computationally infeasible to find (with non-negligible success probability) $x, x' \in \{0,1\}^*$ with $x' \neq x$ and $H(x') = H(x)$. Such a pair (x, x') is called a collision for H.

Message digests for digital signature schemes:

- For reasons of efficiency, instead of signing a (long) message x, the (much shorter) message digest $h = H(x)$ is signed.
- This application requires preimage-resistance, 2nd preimage resistance, and collision resistance.
- To see why collision resistance is required, suppose that the legitimate signer Alice can find a collision (x_1, x_2) for H. Alice can sign x_1 and later claimed to have signed x_2.

Some other applications of hash functions

1. Message Authentication Codes: HMAC.
2. Pseudorandom bit generation:
   Distilling random bits $s = H(x_1, x_2, \ldots, x_t)$ from several "pseudorandom" sources $x_1, x_2, \ldots, x_t$.
3. Key derivation functions (KDF):
   Deriving a cryptographic key from a secret.
4. Proof-of-work in cryptocurrencies (Bitcoin).
5. Quantum-safe signature schemes.

V3b
Relationships between PR, 2PR and CR

HASH FUNCTIONS

CRYPTO 101: Building Blocks

cryptography101.ca

Typical cryptographic requirements

Definition: A hash function $H : \{0,1\}^* \longrightarrow \{0,1\}^n$ is preimage resistant if, given a hash value $y \in_R \{0,1\}^n$, it is computationally infeasible to find (with non-negligible success probability) *any* $x \in \{0,1\}^*$ with $H(x) = y$.

Definition: A hash function $H : \{0,1\}^* \longrightarrow \{0,1\}^n$ is 2nd preimage resistant if, given $x \in_R \{0,1\}^*$, it is computationally infeasible to find (with non-negligible success probability) *any* $x' \in \{0,1\}^*$ with $x' \neq x$ and $H(x') = H(x)$.

Definition: A hash function $H : \{0,1\}^* \longrightarrow \{0,1\}^n$ is collision resistant if it is computationally infeasible to find (with non-negligible success probability) $x, x' \in \{0,1\}^*$ with $x' \neq x$ and $H(x') = H(x)$.

Breaking PR, 2PR, CR

Breaking PR:
Given: $y \in_R \{0,1\}^n$.
Required: $x \in \{0,1\}^*$ with $H(x) = y$.

$$H : \{0,1\}^* \longrightarrow \{0,1\}^n$$

Breaking 2PR:
Given: $x \in_R \{0,1\}^*$.
Required: $x' \in \{0,1\}^*$ with $x' \neq x$ and $H(x') = H(x)$.

Breaking CR:
Given: $-$.
Required: $x, x' \in \{0,1\}^*$ with $x' \neq x$ and $H(x') = H(x)$.

Claim 1: If H is CR, then H is 2PR

Proof: Suppose that $H : \{0,1\}^* \longrightarrow \{0,1\}^n$ is not 2PR.

We'll show that H is not CR.

Select $x \in_R \{0,1\}^*$. Since H is not 2PR, we can efficiently find $x' \in \{0,1\}^*$, $x' \neq x$, with $H(x') = H(x)$.

Thus, (x, x') is a collision for H that we have efficiently found, showing that H is not CR. $\square$

$$\begin{tikzcd}
 & \text{CR} \arrow[dr] & \\
\text{PR} & & \text{2PR}
\end{tikzcd}$$

Note: The proof established the *contrapositive* statement.

Claim 2: CR does not guarantee PR

Proof: Suppose that $H : \{0,1\}^* \longrightarrow \{0,1\}^n$ is CR.

Consider the hash function $\overline{H} : \{0,1\}^* \longrightarrow \{0,1\}^{n+1}$ defined by

$$\overline{H}(x) = \begin{cases} 0\|H(x), & \text{if } x \notin \{0,1\}^n \\ 1\|x, & \text{if } x \in \{0,1\}^n. \end{cases}$$

Then $\overline{H}$ is CR (since H is).

And, $\overline{H}$ is not PR since preimages can be efficiently found for at least half of all $y \in \{0,1\}^{n+1}$, namely the hash values that begin with 1. $\square$

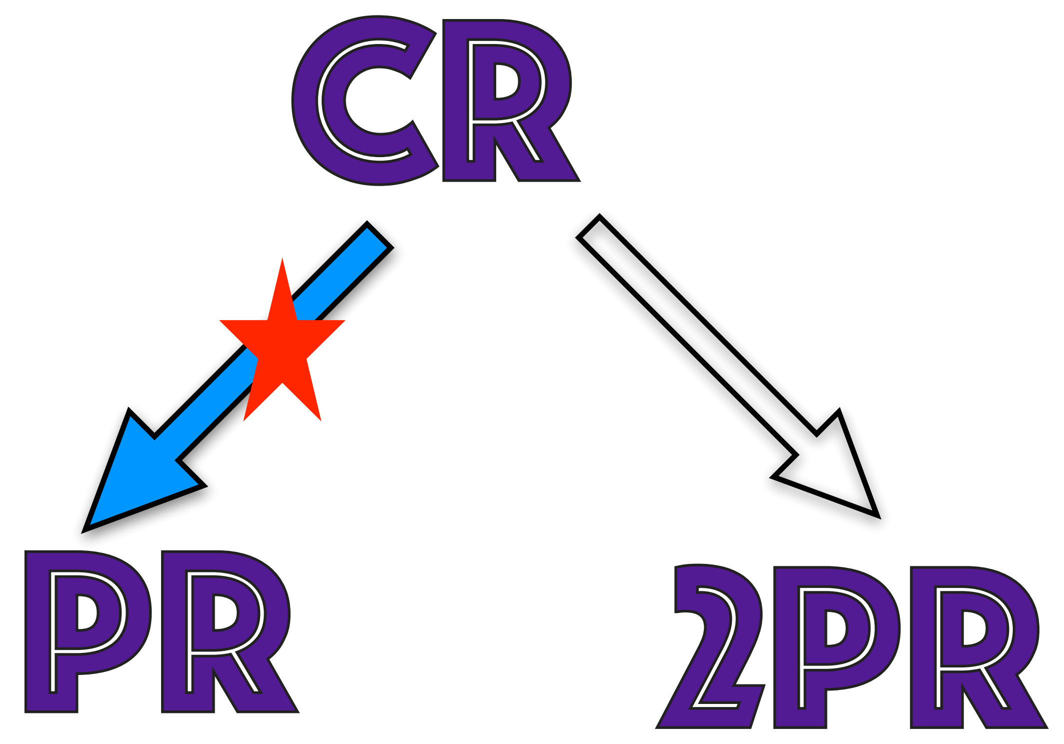
Note: The hash function $\overline{H}$ is rather contrived. For *somewhat uniform* hash functions, i.e., hash function for which all hash values have roughly the same number of preimages, CR does indeed guarantee PR.

Claim 2*: Suppose H is somewhat uniform. If H is CR, then H is PR.

Proof: Suppose that $H : \{0,1\}^* \longrightarrow \{0,1\}^n$ is not PR.

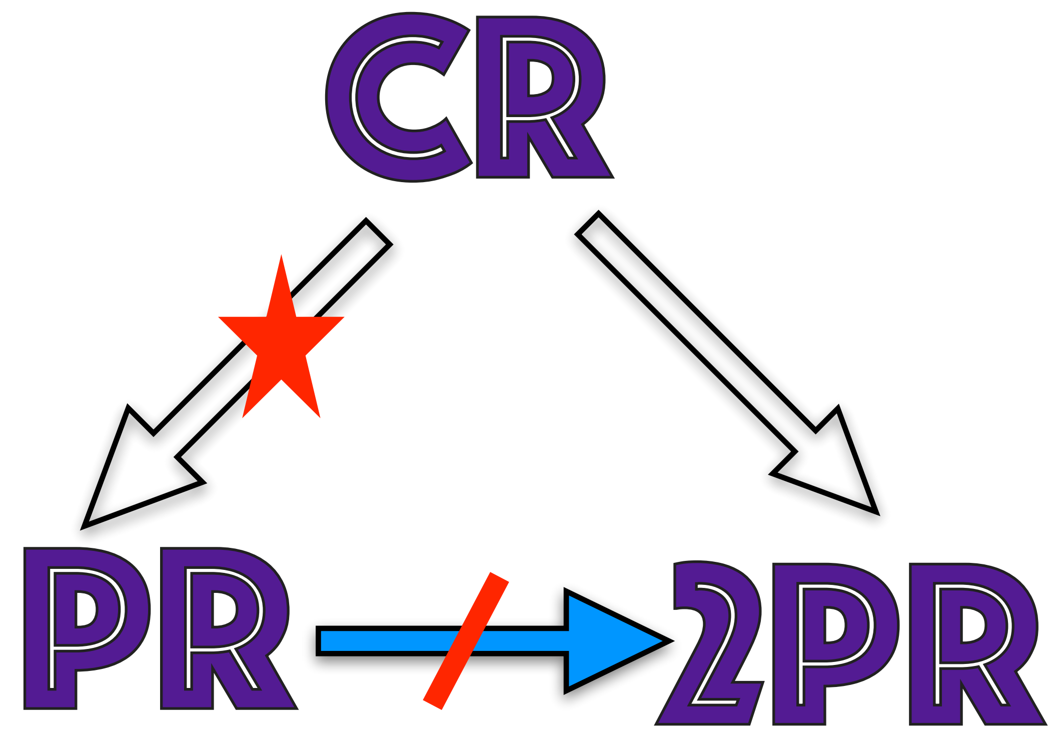
We'll show that H is not CR.

Select $x \in_R \{0,1\}^*$ and compute $y = H(x)$. Since H is not PR, we can efficiently find $x' \in \{0,1\}^*$ with $H(x') = y$. Since H is somewhat uniform, we expect that y has many preimages, and thus $x' \neq x$ with very high probability. Thus, (x, x') is a collision for H that we have efficiently found, so H is not CR. $\square$



Note: *For the remainder of the course we'll assume that hash functions are somewhat uniform.*

Claim 3: PR does not guarantee 2PR



Proof: Suppose that $H : \{0,1\}^* \longrightarrow \{0,1\}^n$ is PR.

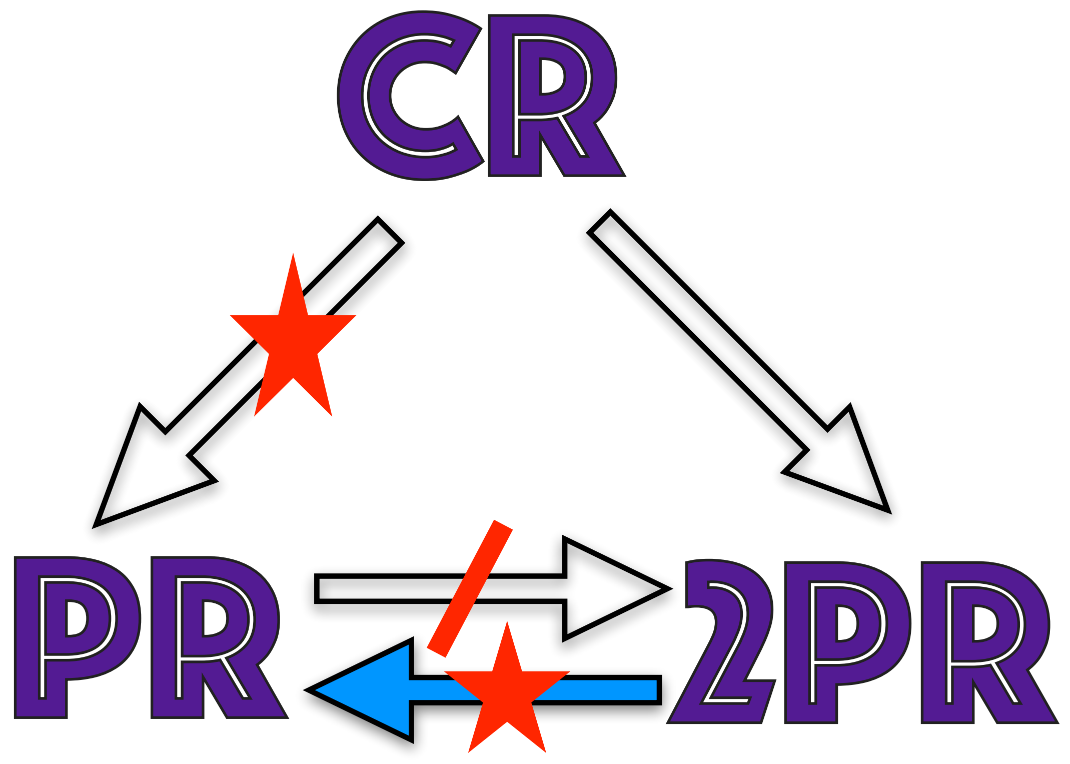
Define $\overline{H} : \{0,1\}^* \longrightarrow \{0,1\}^n$ by

$\overline{H}(x_1, x_2, \ldots, x_t) = H(0, x_2, \ldots, x_t)$ for all $(x_1, x_2, \ldots, x_t) \in \{0,1\}^*$.

Then $\overline{H}$ is PR [Why?].

However, $\overline{H}$ is not 2PR [Why?]. $\square$

Claim 4: Suppose H is somewhat uniform. If H is 2PR, then H is PR.



Proof: Suppose that $H : \{0,1\}^* \longrightarrow \{0,1\}^n$ is not PR.

We'll show that H is not 2PR.

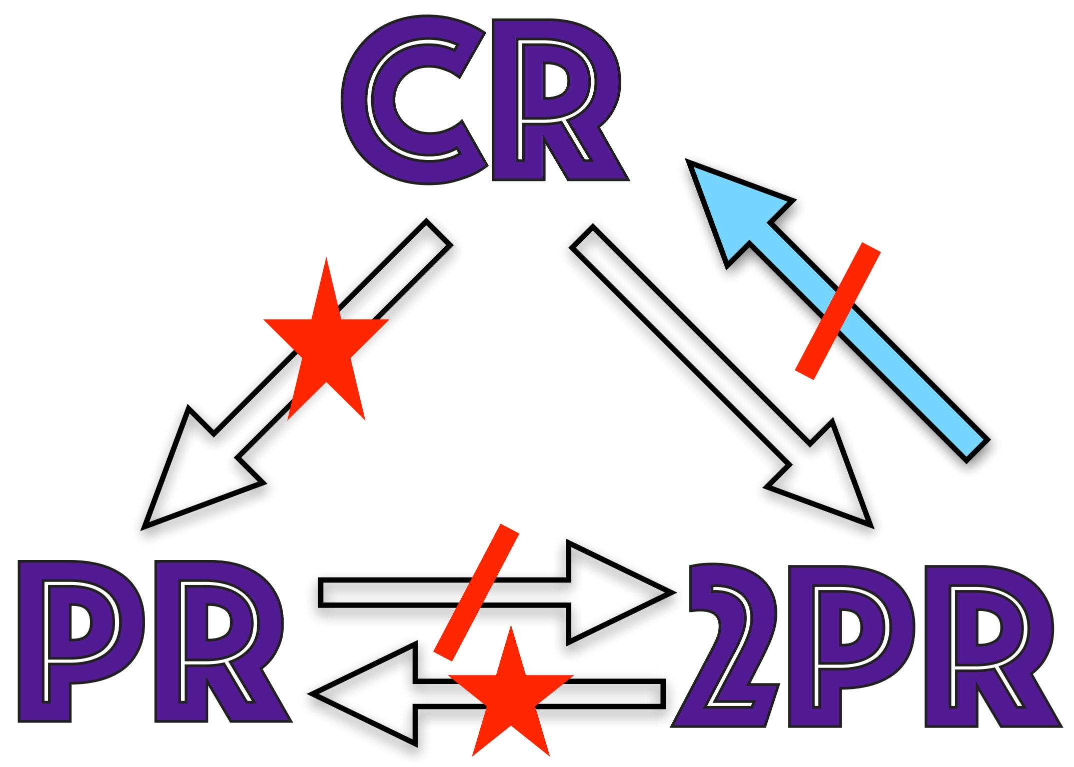
So, suppose we are given $x \in_R \{0,1\}^*$. We compute $y = H(x)$.

Since H is not PR, we can efficiently find $x' \in \{0,1\}^*$ with $H(x') = y$.

Since H is somewhat uniform, we expect that $x' \neq x$ with very high probability. Hence, x' is a second preimage of x that we have efficiently found.

Thus H is not 2PR. $\square$

Claim 5: 2PR does not guarantee CR



Proof: Suppose that $H : \{0,1\}^* \longrightarrow \{0,1\}^n$ is 2PR.

Consider $\overline{H} : \{0,1\}^* \longrightarrow \{0,1\}^n$ defined by $\overline{H}(x) = H(x)$ if $x \neq 1$, and $\overline{H}(1) = H(0)$.

- Then $\overline{H}$ is not CR, since $(0,1)$ is a collision for $\overline{H}$.

- Suppose now that $\overline{H} : \{0,1\}^* \longrightarrow \{0,1\}^n$ is not 2PR. We'll show that H is not 2PR.

So, we are given $x \in_R \{0,1\}^*$. Since $\overline{H}$ is not 2PR, we can efficiently find $x' \in \{0,1\}^*$, $x' \neq x$, with $\overline{H}(x') = \overline{H}(x)$. With probability essentially 1, we can assume that $x \neq 0,1$. Hence, $\overline{H}(x) = H(x)$.

Now, if $x' \neq 1$, then $H(x') = \overline{H}(x') = \overline{H}(x) = H(x)$.

And, if $x' = 1$, then $\overline{H}(x') = \overline{H}(1) = H(0) = H(x)$.

In either case, we have efficiently found a second preimage for x w.r.t. H .

Hence, H is not 2PR, a contradiction. Thus, $\overline{H}$ is 2PR. $\square$

Relationships between PR, 2PR, CR

Let $H : \{0,1\}^* \longrightarrow \{0,1\}^n$ be a hash function.

$$\begin{tikzcd}
 & \mathbf{CR} \arrow[dl, shift left, "2^*", "\bigstar" marking] \arrow[dr, shift left, "1"] & \\
\mathbf{PR} \arrow[ur, shift left, "/" marking] \arrow[rr, shift left, "3", "/" marking] & & \mathbf{2PR} \arrow[ul, shift left, "5"', "/" marking] \arrow[ll, shift left, "4", "\bigstar" marking]
\end{tikzcd}$$

★ for somewhat uniform hash functions

V3c
Generic attacks

HASH FUNCTIONS

CRYPTO 101: Building Blocks

Generic attacks

A generic attack on hash functions $H : \{0,1\}^* \longrightarrow \{0,1\}^n$ does not exploit any properties that the specific hash function might have.

- In the analysis of a generic attack, we view H as a random function in the sense that for each $x \in \{0,1\}^*$, the hash value $y = H(x)$ was defined by selecting $y \in_R \{0,1\}^n$.
- From a security point of view, a random function is an ideal hash function. However, random functions are not suitable for practical applications because they cannot be compactly described.

Generic attack for finding preimages

- **Attack**: Given $y \in_R \{0,1\}^n$, repeatedly select arbitrary $x \in \{0,1\}^*$ until $H(x) = y$.
- **Analysis**: The expected number of hash operations is 2^n.

- This generic attack is infeasible if $n \geq 128$.
- Note: It has been proven that this generic attack for finding preimages is optimal, i.e., no faster generic attack exists. Of course, for a specific hash function, there might exist a faster preimage finding algorithm.

Generic attack for finding collisions

- **Attack**: Select arbitrary $x \in \{0,1\}^*$ and store $(H(x), x)$ in a table sorted by first entry. Repeat until a collision is found.
- **Analysis**: By the birthday paradox, the expected number of hash operations is $\sqrt{\pi 2^n/2} \approx \sqrt{2^n}$.

- This generic attack is infeasible if $n \geq 256$.
- Note: It has been proven that this generic attack for finding collisions is optimal, i.e., no faster generic attack exists.
- Expected space required: $\sqrt{\pi 2^n/2} \approx \sqrt{2^n}$.
- **Example**: If $n = 128$, the expected running time is 2^{64} (feasible), whereas the expected space required is 5×10^8 Tbytes (infeasible).

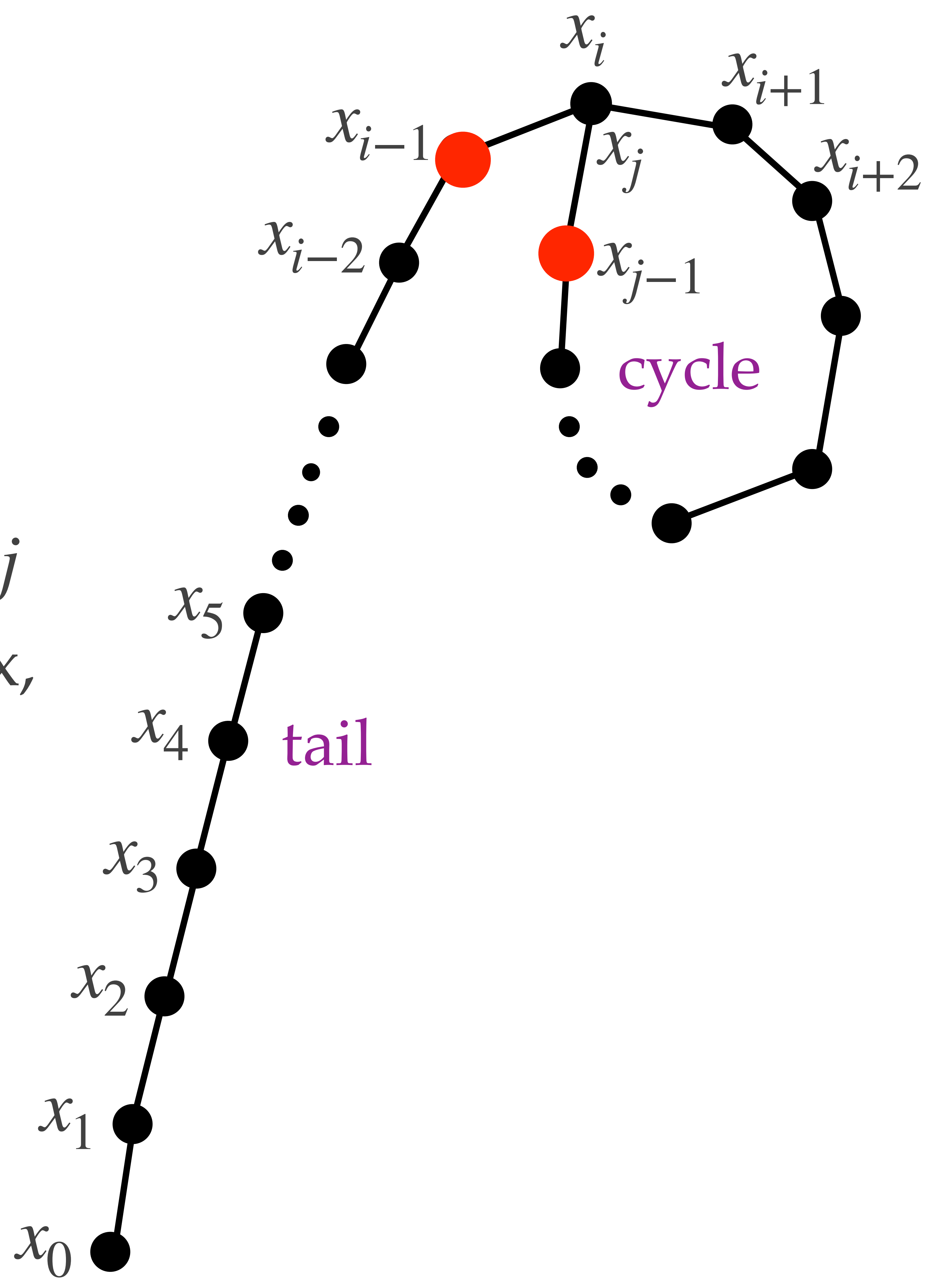
VW parallel collision search

- VW: van Oorschot & Wiener (1993)
- Expected number of hash operations: $\approx \sqrt{2^n}$.
- Expected space required: negligible.
- Easy to parallelize — m-fold speedup with m processors.
- The VW collision-finding algorithm can easily be modified to find "meaningful" collisions. (See Optional Readings at cryptography101.ca.)
- **Conclusion**: If collision resistance is desired, then use an n-bit hash function with $n \geq 256$.

Parallel collision search (VW method)

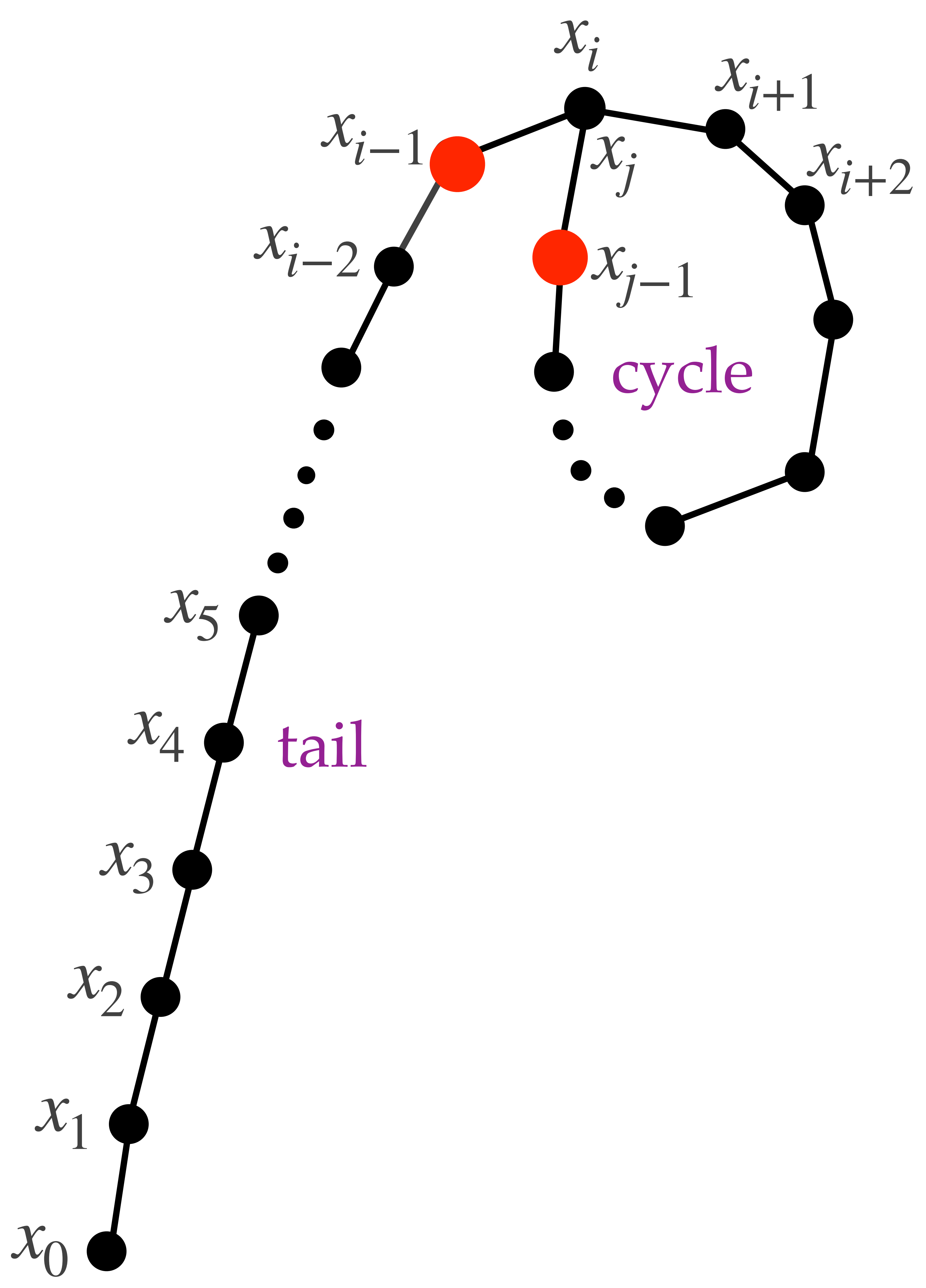
- **Problem**: Find a collision for $H : \{0,1\}^* \longrightarrow \{0,1\}^n$.
- **Assumption**: H is a random function.
- **Notation**: Let $N = 2^n$.
  Define a sequence $\{x_i\}_{i \geq 0}$ by $x_0 \in_R \{0,1\}^n$, $x_i = H(x_{i-1})$ for $i \geq 1$.

  Let j be the smallest index for which $x_j = x_i$ for some $i < j$; such a j must exist. Then $x_{j+\ell} = x_{i+\ell}$ for all $\ell \geq 1$. By the birthday paradox, $E[j] \approx \sqrt{\pi N/2} \approx \sqrt{N}$. In fact, $E[i] \approx \frac{1}{2}\sqrt{N}$ and $E[j-i] \approx \frac{1}{2}\sqrt{N}$.
- Now, $i \neq 0$ with overwhelming probability, in which event (x_{i-1}, x_{j-1}) is a collision for H.
- **Question**: How to find (x_{i-1}, x_{j-1}) without using much storage?

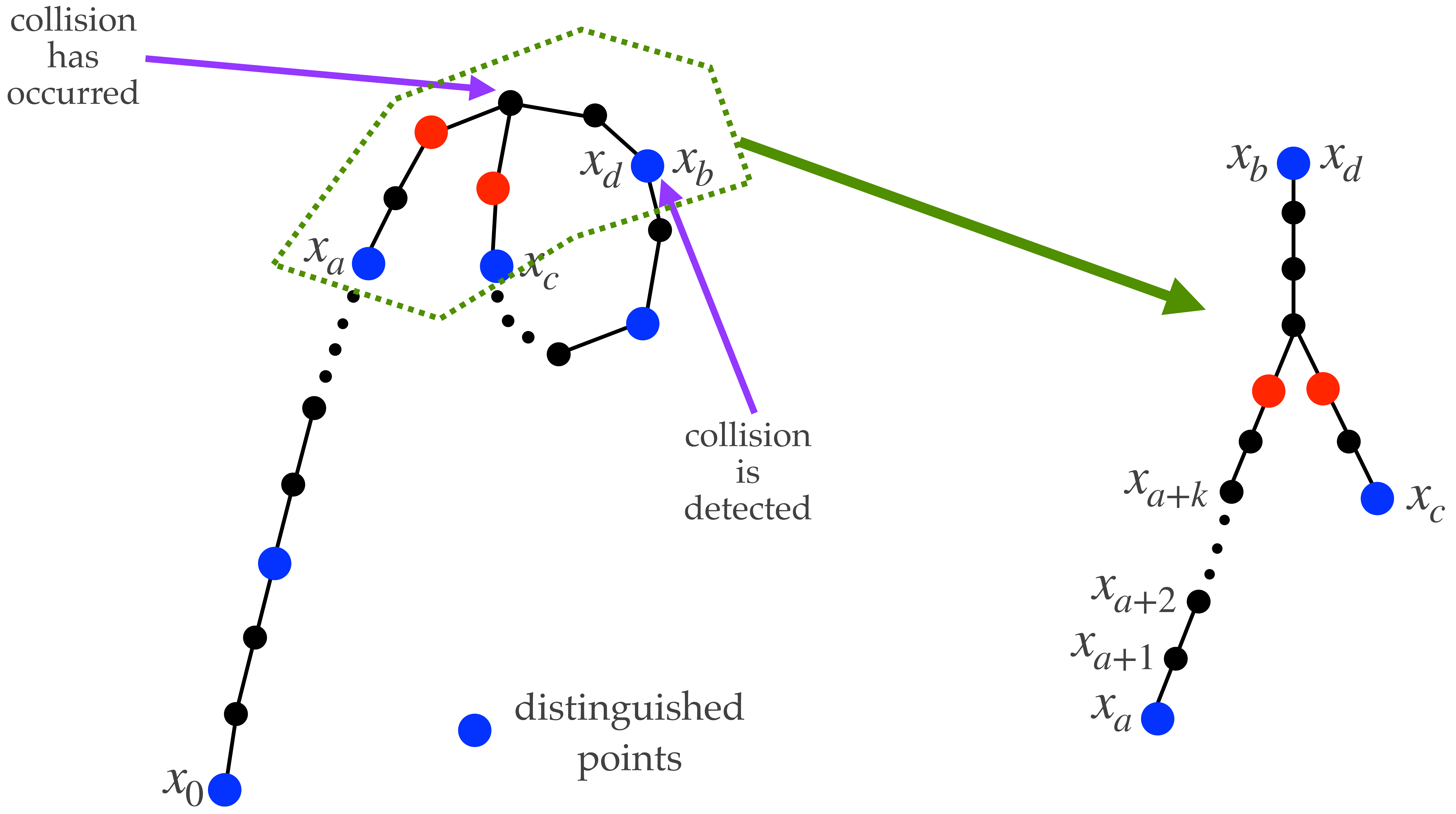


Distinguished points

- **Answer**: Only store distinguished points.
- Distinguished points: Select an easily-testable distinguishing property for elements of $\{0,1\}^n$, e.g. leading 32 bits are all 0.
  Let θ be the proportion of elements of $\{0,1\}^n$ that are distinguished.
- VW method: *Compute the sequence* $x_0, x_1, x_2, x_3, \ldots$ *and only store the points that are distinguished.*



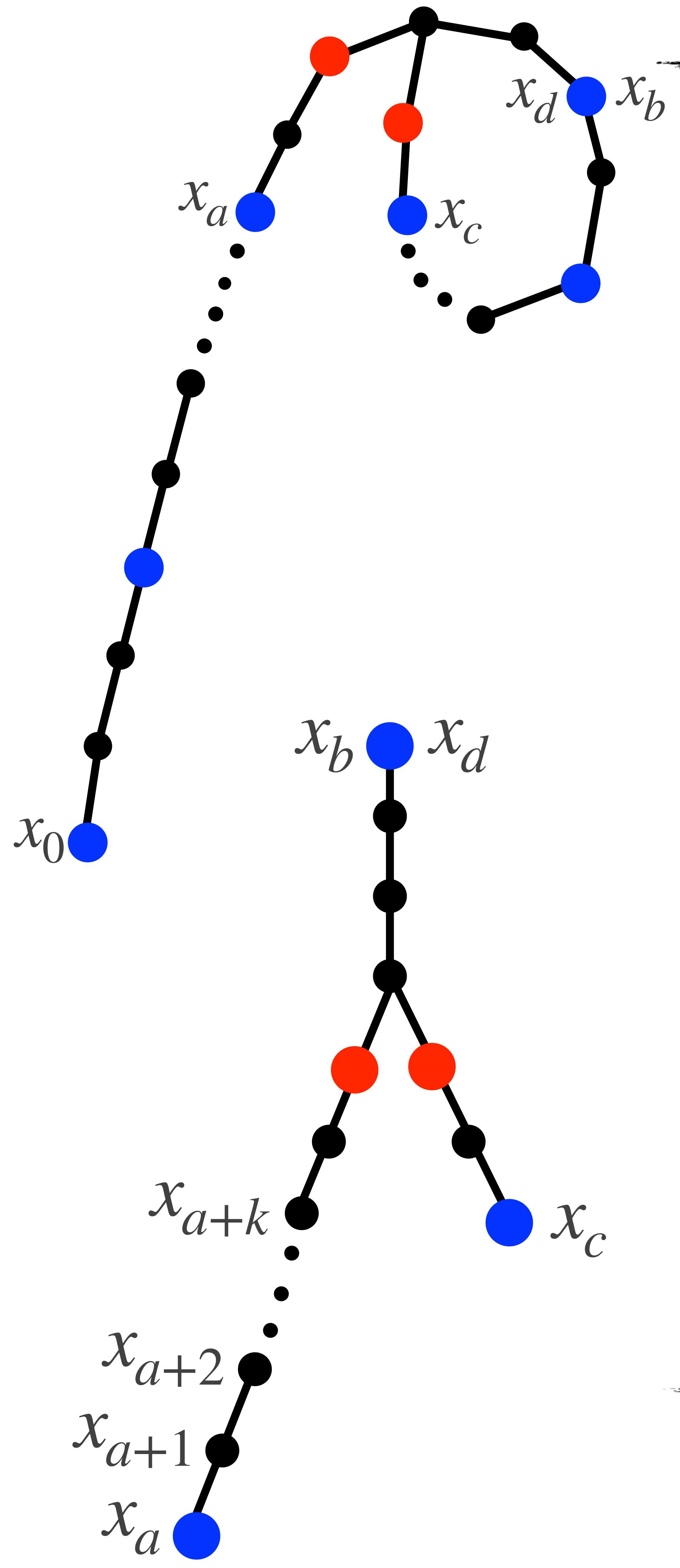
VW collision finding



VW collision finding

Stage 1: Detecting a collision

```
1. Select $x_0 \in_R \{0,1\}^n$.
2. Store $(x_0, 0, -)$ in a sorted table.
3. LP $\leftarrow x_0$.    (LP= last point stored)
4. For $d = 1,2,3,\ldots$ do:
    a. Compute $x_d = H(x_{d-1})$ .
    b. If $x_d$ is distinguished then
        i. If $x_d$ is already in the table, say $x_d = x_b$
           where $b < d$, then go to Stage 2.
        ii. Store $(x_d, d, \text{LP})$ in the table.
        iii. LP $\leftarrow x_d$.
```



Stage 2: Finding a collision

```
1. Set $\ell_1 \leftarrow b - a$,   $\ell_2 \leftarrow d - c$.
2. Suppose $\ell_1 \geq \ell_2$, and set
   $k \leftarrow \ell_1 - \ell_2$.
3. Compute $x_{a+1}, x_{a+2}, \ldots, x_{a+k}$ .
4. For $m = 1,2,3,\ldots$ do:
        a) Compute $(x_{a+k+m}, x_{c+m})$.
5. Until $x_{a+k+m} = x_{c+m}$.
6. The collision is $(x_{a+k+m-1}, x_{c+m-1})$.
```

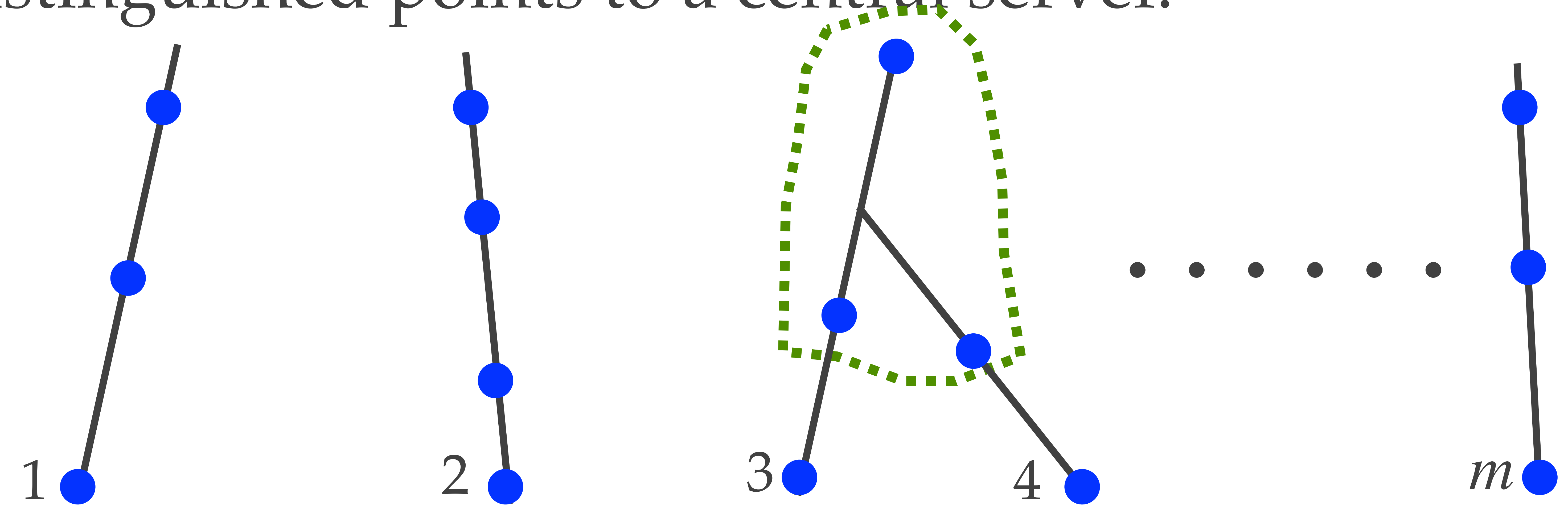
VW analysis

- Stage 1: Expected number of H-evaluations is:
$$\sqrt{\pi N/2} + \frac{1}{\theta} \approx \sqrt{N} + \frac{1}{\theta}.$$
- Stage 2: Expected number of H-evaluations is $\leq \frac{3}{\theta}$ (see optional readings).
- Overall expected running time: $\sqrt{N} + \frac{4}{\theta}$.
- Expected storage: $\approx 3n\theta\sqrt{N}$ bits (each table entry has bitlength $3n$).

- **Example**: Consider $n = 128$. Take $\theta = 1/2^{32}$. Then the expected run time of VW collision search is 2^{64} H-evaluations (feasible), and the expected storage is 192 Gbytes (negligible).

Parallelizing VW collision search

- Run independent copies of VW on each of m processors
- Report distinguished points to a central server.



Analysis

- Expected time $\approx \dfrac{1}{m}\sqrt{N} + \dfrac{4}{\theta}$.
- Expected storage $\approx 3n\theta\sqrt{N}$ bits.

Notes

1. Factor-m speedup.
2. No communications between processors.
3. Occasional communications with the central server.

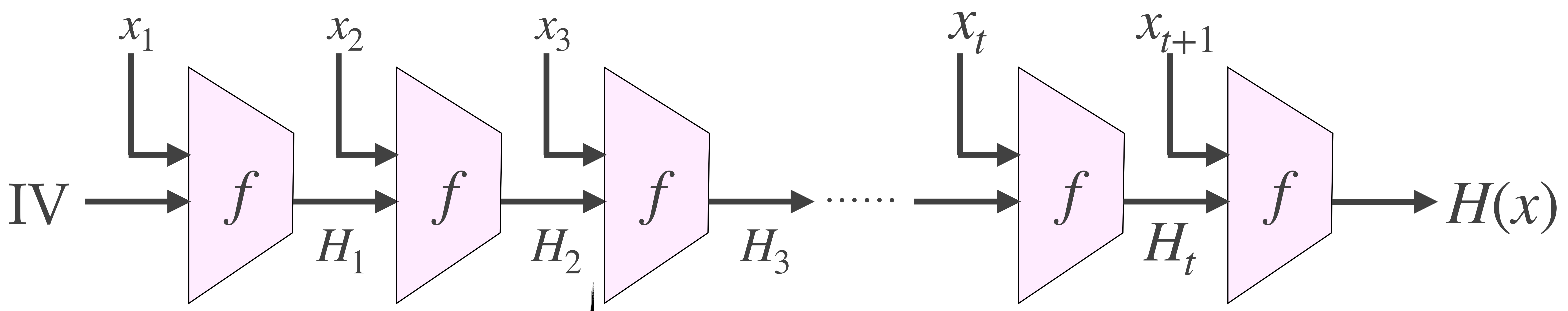
V3d
Iterated hash functions

HASH FUNCTIONS

CRYPTO 101: Building Blocks

cryptography101.ca

Iterated hash functions (Merkle's meta method)



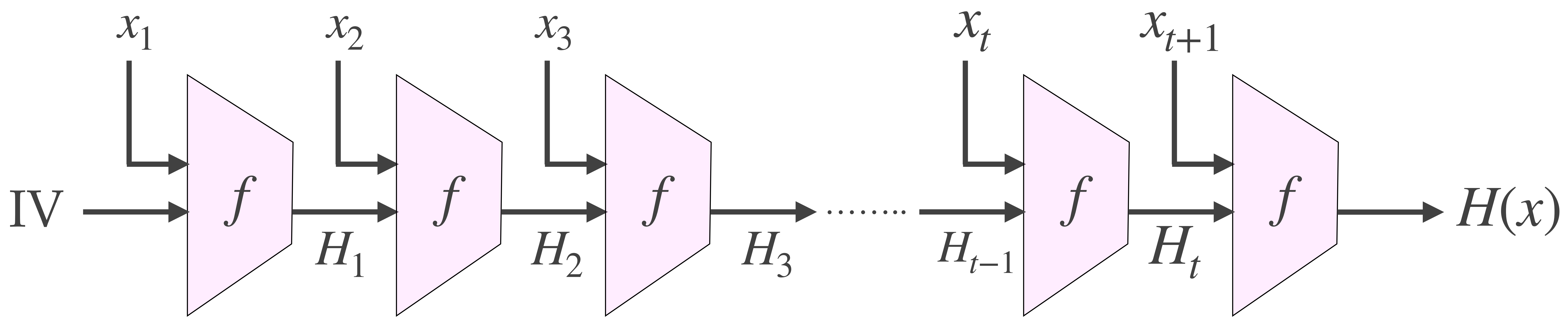
Components:

- Fixed initializing value $IV \in \{0,1\}^n$.
- Efficiently-computable compression function $f : \{0,1\}^{n+r} \to \{0,1\}^n$.

To compute $H(x)$ where x has bitlength $b < 2^r$ do:

1. Break up x into r-bit blocks, $\bar{x} = x_1, x_2, \ldots, x_t$, padding the last block with 0 bits as necessary.
2. Define x_{t+1}, the length-block, to hold the right-justified binary representation of b.
3. Define $H_0 = IV$.
4. Compute $H_i = f(H_{i-1}, x_i)$ for $i = 1,2,\ldots,t+1$. (The H_i's are called chaining variables.)
5. Define $H(x) = H_{t+1}$.

Collision resistance of iterated hash functions



Theorem (Merkle): If the compression function f is collision resistant, then the iterated hash function H is also collision resistant.

Merkle's theorem reduces the problem of designing collision-resistant hash functions to that of designing collision-resistant compression functions.

Provable security

A major theme in cryptographic research is to formulate precise security definitions and assumptions, and then prove that a cryptographic protocol is secure.

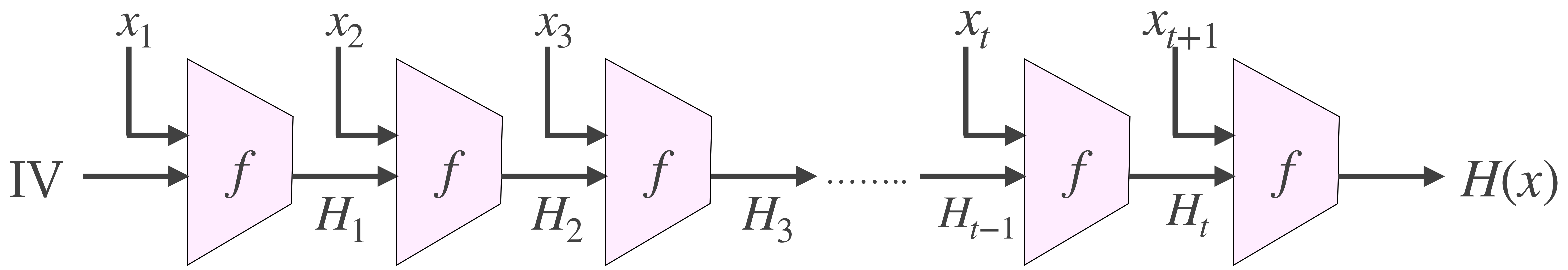
A proof of security is certainly desirable since it rules out the possibility of attacks being discovered in the future.

However, it isn't always easy to assess the practical security assurances (if any) that a security proof provides.

Optional reading: anotherlook.ca

- The assumptions might be unrealistic, or false, or circular.
- The security proof might be *fallacious*.
- The security model might not account for certain kinds of realistic attacks.
- The security proof might be *asymptotic*.
- The security proof might have a large *tightness gap*.

Proof of Merkle's Theorem (f is CR $\Rightarrow H$ is CR)



- Suppose that H is not CR. We'll show that f is not CR.
- Since H is not CR, we can efficiently find messages $x, x' \in \{0,1\}^*$, with $x \neq x'$ and $H(x) = H(x')$.
- Let $\overline{x} = x_1, x_2, \ldots, x_t, \quad b = \text{bitlength}(x), \quad x_{t+1} = \text{length block}$.
- Let $\overline{x'} = x'_1, x'_2, \ldots, x'_{t'}, \quad b' = \text{bitlength}(x'), \quad x'_{t'+1} = \text{length block}$.

Proof of Merkle's Theorem (2)

- We efficiently compute:

$$\boxed{\begin{aligned} H_0 &= IV \\ H_1 &= f(H_0, x_1) \\ H_2 &= f(H_1, x_2) \\ &\vdots \\ H_{t-1} &= f(H_{t-2}, x_{t-1}) \\ H_t &= f(H_{t-1}, x_t) \\ H(x) = \boxed{H_{t+1}} &= f(H_t, x_{t+1}) \end{aligned}} \qquad \boxed{\begin{aligned} H_0 &= IV \\ H'_1 &= f(H_0, x'_1) \\ H'_2 &= f(H'_1, x'_2) \\ &\vdots \\ H'_{t'-1} &= f(H'_{t'-2}, x'_{t'-1}) \\ H'_{t'} &= f(H'_{t'-1}, x'_{t'}) \\ H(x') = \boxed{H'_{t'+1}} &= f(H'_{t'}, x'_{t'+1}) \end{aligned}}$$

- Since $H(x) = H(x')$, we have $H_{t+1} = H'_{t'+1}$.

Proof of Merkle's Theorem (3)

- Case 1: Now, if $b \neq b'$, then $x_{t+1} \neq x'_{t'+1}$. Thus, $(H_t, x_{t+1}),\ (H'_{t'}, x'_{t'+1})$ is a collision for f that we have efficiently found.
- Case 2: Suppose next that $b = b'$. Then $t = t'$ and $x_{t+1} = x'_{t+1}$
  - Let i be the largest index, $0 \leq i \leq t$, for which $(H_i, x_{i+1}) \neq (H'_i, x'_{i+1})$. Such an i must exist since $x \neq x'$.
  - Then $H_{i+1} = f(H_i, x_{i+1}) = f(H'_i, x'_{i+1}) = H'_{i+1}$, so $(H_i, x_{i+1}), (H'_i, x'_{i+1})$ is a collision for f that we have efficiently found.
- Thus, f is not collision resistant. $\square$

MDx-family of hash functions

- MDx is a family of iterated hash functions.
- MD4 was proposed by Ron Rivest in 1990.
- MD4 has 128-bit outputs.
- Professor Xiaoyun Wang et al. (2004) found collisions for MD4 **by hand.**
- Leurent (2008) discovered an algorithm for finding MD4 preimages in 2^{102} operations.

MD5 hash function

- MD5 is a strengthened version of MD4.
- Designed by Ron Rivest in 1991.
- MD5 has 128-bit outputs.
- Wang and Yu (2004) found MD5 collisions in 2^{39} operations.
- MD5 collisions can now be found in 2^{24} operations, which takes a few seconds on a laptop computer.
- Sasaki & Aoki (2009) discovered a method for finding MD5 preimages in $2^{123.4}$ steps.

MD5 hash function (2)

Summary: MD5 should not be used if collision resistance is required, but is probably okay as a preimage-resistant hash function.

- MD5 is still used today.
- 2006: MD5 was implemented more than 850 times in Microsoft Windows source code.
- 2014: Microsoft issues a patch that restricts the use of MD5 in certificates in Windows: tinyurl.com/MicrosoftMD5.

Flame malware

- Discovered in 2012, Flame malware was a highly sophisticated espionage tool.
- Targeted computers in Iran and the Middle East.
- Contains a forged Microsoft certificate for Windows code signing.
- Forged certificate used a new "zero-day MD5 chosen-prefix" collision attack.
- Microsoft no longer allows the use of MD5 for code signing.

WIKIPEDIA
The Free Encyclopedia

Flame (malware)

SHA-1

- Secure Hash Algorithm (SHA) was designed by NSA and published by NIST in 1993 (FIPS 180).
- 160-bit iterated hash function, based on MD4.
- Slightly modified to SHA-1 (FIPS 180-1) in 1994 in order to fix an undisclosed security weakness.
  - Wang et al. (2005) found collisions for SHA in 2^{39} operations.
- Wang et al. (2005) discovered a collision-finding algorithm for SHA-1 that takes 2^{63} operations.
  - The first SHA-1 collision was found on February 23, 2017.
- No preimage or 2nd preimage attacks that are faster than the generic attacks are known for SHA-1.

SHA-2 family

- In 2001, NSA proposed variable output-length versions of SHA-1.
- Output lengths are 224 bits (SHA-224 and SHA-512/224), 256 bit (SHA-256 and SHA-512/256), 384 bits (SHA-384), and 512 bits (SHA-512).
- 2024: No weaknesses in any of these hash functions have been found.
- Note: The security levels of these hash functions against VW collision finding attacks are the same as the security levels of Triple-DES, AES-128, AES-192, and AES-256 against exhaustive key search attacks.
- The SHA-2 hash functions are standardized in FIPS 180-2.

Summary: Collision resistance of iterated hash functions

Hash function $H : \{0,1\}^* \longrightarrow \{0,1\}^n$	n	Security level against generic attack VW attack (in bits)	Security level after Prof. Wang's attacks (in bits)
MD4 (1990)	128	64	4 (2004)
MD5 (1991)	128	64	39 (2005) —> 24
SHA (1993)	160	80	39 (2005)
SHA-1 (1994)	160	80	63 (2005)
SHA-224	224	112	112
SHA-256	256	128	128
SHA-384	384	192	192
SHA-512	512	256	256

SHA-3 family

- The SHA-2 design is similar to SHA-1, and thus there were lingering concerns that the SHA-1 weaknesses could eventually extend to SHA-2.
- SHA-3: NIST hash function competition.
  - 2008: 64 candidates submitted from around the world.
  - 2012: Keecak was selected as the winner.
- Keecak uses the "sponge construction" and not the Merkle iterated hash design.
- SHA-3 is being used in practice, but is not (yet) as widely deployed as SHA-2.

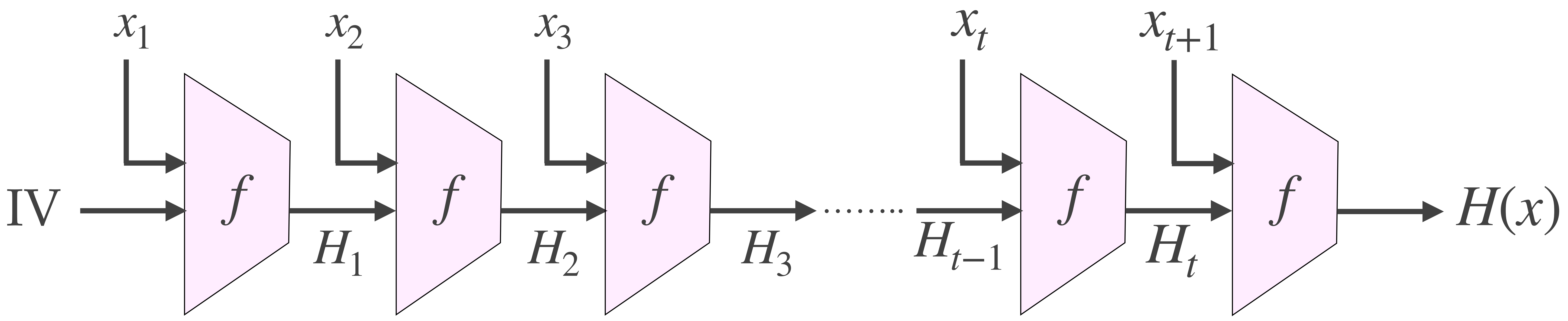
V3e
SHA-256

HASH FUNCTIONS

CRYPTO 101: Building Blocks

cryptography101.ca

Description of SHA-256



- Iterated hash function (Merkle's meta method).
- $n = 256$, $r = 512$.
- Compression function is $f : \{0,1\}^{256+512} \longrightarrow \{0,1\}^{256}$.
- Input: bit string x of arbitrary bitlength $b \geq 0$.
- Output: 256-bit hash value $H(x)$ of x.

SHA-256 notation

A, B, C, D, E, F, G, H are 32-bit words

$+$	addition modulo 2^{32}
$\overline{A}$	bitwise complement
$A \gg s$	shift A right by s positions
$A \hookrightarrow s$	rotate A right by s positions
AB	bitwise AND of A, B
$A \oplus B$	bitwise exclusive-OR

$f(A, B, C)$	$AB \oplus \overline{A}C$
$g(A, B, C)$	$AB \oplus AC \oplus BC$
$r_1(A)$	$(A \hookrightarrow 2) \oplus (A \hookrightarrow 13) \oplus (A \hookrightarrow 22)$
$r_2(A)$	$(A \hookrightarrow 6) \oplus (A \hookrightarrow 11) \oplus (A \hookrightarrow 25)$
$r_3(A)$	$(A \hookrightarrow 7) \oplus (A \hookrightarrow 18) \oplus (A \gg 3)$
$r_4(A)$	$(A \hookrightarrow 17) \oplus (A \hookrightarrow 19) \oplus (A \gg 10)$

SHA-256 constants

- 32-bit initial chaining values (IVs): These words were obtained by taking the first 32 bits of the fractional parts of the square roots of the first 8 prime numbers.

$h_1 =$ `0x6a09e667`	$h_2 =$ `0xbb67ae85`	$h_3 =$ `0x3c6ef372`	$h_4 =$ `0xa54ff53a`
$h_5 =$ `0x510e527f`	$h_6 =$ `0x9b05688c`	$h_7 =$ `0x1f83d9ab`	$h_8 =$ `0x5be0cd19`

- Per-round integer additive constants: These words were obtained by taking the first 32 bits of the fractional parts of the cube roots of the first 64 prime numbers.

$y_0 =$ `0x428a2f98`	$y_1 =$ `0x71374491`	$y_2 =$ `0xb5c0fbcf`	$y_3 =$ `0xe9b5dba5`
....	....	$y_{62} =$ `0xbef9a3f7`	$y_{63} =$ `0xc67178f2`

SHA-256 preprocessing

1. Pad x with 1, followed by as few 0's as possible so that the bitlength is 64 less than a multiple of 512.
2. Append the 64-bit binary representation of $b \bmod 2^{64}$.
3. The formatted input is $x_0, x_1, \ldots, x_{16m-1}$, where each x_i is a 32-bit word.
4. Initialize the words of the chaining variable:
   $(H_1, H_2, \ldots, H_7, H_8) \leftarrow (h_1, h_2, \ldots, h_7, h_8)$.

SHA-256 processing

For each i from 0 to $m-1$ do the following:

- Copy the ith block of sixteen 32-bit words into temporary storage:
  $X_j \leftarrow x_{16i+j}, \quad 0 \le j \le 15$.
- Expand the 16-word block into a 64-word block:
  For j from 16 to 63 do: $X_j \leftarrow r_4(X_{j-2}) + X_{j-7} + r_3(X_{j-15}) + X_{j-16}$.
- Initialize working variables: $(A, B, \ldots, G, H) \leftarrow (H_1, H_2, \ldots, H_7, H_8)$.
- For j from 0 to 63 do:
  - $T_1 \leftarrow H + r_2(E) + f(E, F, G) + y_j + X_j \qquad T_2 \leftarrow r_1(A) + g(A, B, C)$.
  - $H \leftarrow G,\ G \leftarrow F,\ F \leftarrow E,\ E \leftarrow D + T_1,\ D \leftarrow C,\ C \leftarrow B,\ B \leftarrow A,\ A \leftarrow T_1 + T_2$.
- Update chaining variable: $(H_1, H_2, \ldots, H_7, H_8) \leftarrow (H_1 + A,\ H_2 + B,\ \ldots,\ H_7 + G,\ H_8 + H)$.

Output: SHA-256$(x) = H_1 \| H_2 \| H_3 \| H_4 \| H_5 \| H_6 \| H_7 \| H_8$.

Performance

Speed benchmarks† from 2018 on an Intel Xeon CPU (E3-1220 V2) at 3.10 GHz in 64-bit mode.

†Relative speeds will likely be very different on other processors.

Source: www.bearssl.org/speed.html

Algorithm	block length (bits)	key length (bits)	digest length (bits)	speed (Mbytes/sec)
ChaCha20	—	256	—	323
Triple-DES	64	168	—	21
AES-128	128	128	—	170
AES-128-NI	128	128	—	2426
AES-256	128	256	—	129
AES-256-NI	128	256	—	1830
MD5	512	—	128	517
SHA-1	512	—	160	331
SHA-256	512	—	256	212
SHA-512	1024	—	512	332