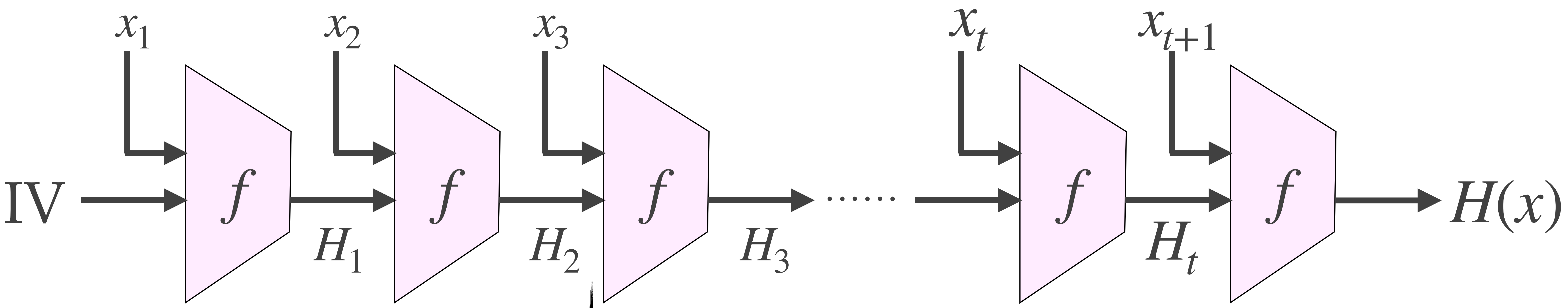
V3d
Iterated hash functions

HASH FUNCTIONS

CRYPTO 101: Building Blocks

cryptography101.ca

Iterated hash functions (Merkle's meta method)



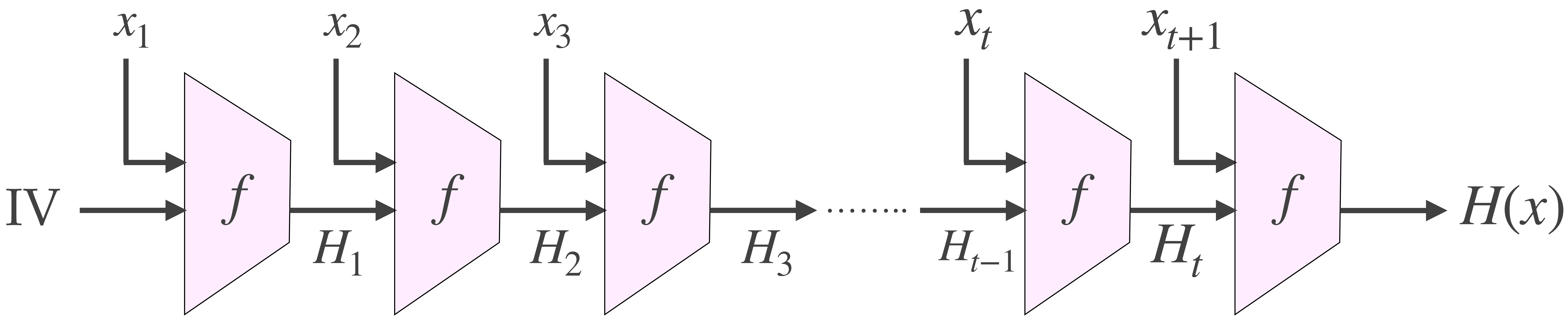
Components:

- 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.

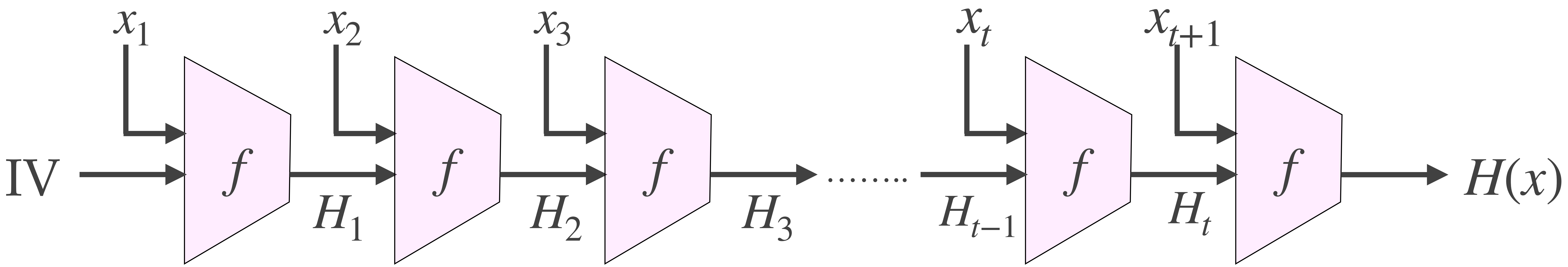
A proof of security is certainly desirable since it rules out the possibility of attacks being discovered in the future.

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 $\overline{x} = x_1, x_2, \ldots, x_t$, $\quad b =$ bitlength(x), $\quad x_{t+1} =$ length block.
- Let $\overline{x'} = x'_1, x'_2, \ldots, x'_{t'}$, $\quad b' =$ bitlength(x'), $\quad x'_{t'+1} =$ length block.

Proof of Merkle's Theorem (2)

- We efficiently compute:

$$\begin{aligned} H_0 &= IV \\ H_1 &= f(H_0, x_1) \\ H_2 &= f(H_1, x_2) \\ &\vdots \\ H_{t-1} &= f(H_{t-2}, x_{t-1}) \\ H_t &= f(H_{t-1}, x_t) \\ H(x) = \boxed{H_{t+1}} &= f(H_t, x_{t+1}) \end{aligned}$$

$$\begin{aligned} H_0 &= IV \\ H'_1 &= f(H_0, x'_1) \\ H'_2 &= f(H'_1, x'_2) \\ &\vdots \\ H'_{t'-1} &= f(H'_{t'-2}, x'_{t'-1}) \\ H'_{t'} &= f(H'_{t'-1}, x'_{t'}) \\ H(x') = \boxed{H'_{t'+1}} &= f(H'_{t'}, x'_{t'+1}) \end{aligned}$$

- 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.

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.

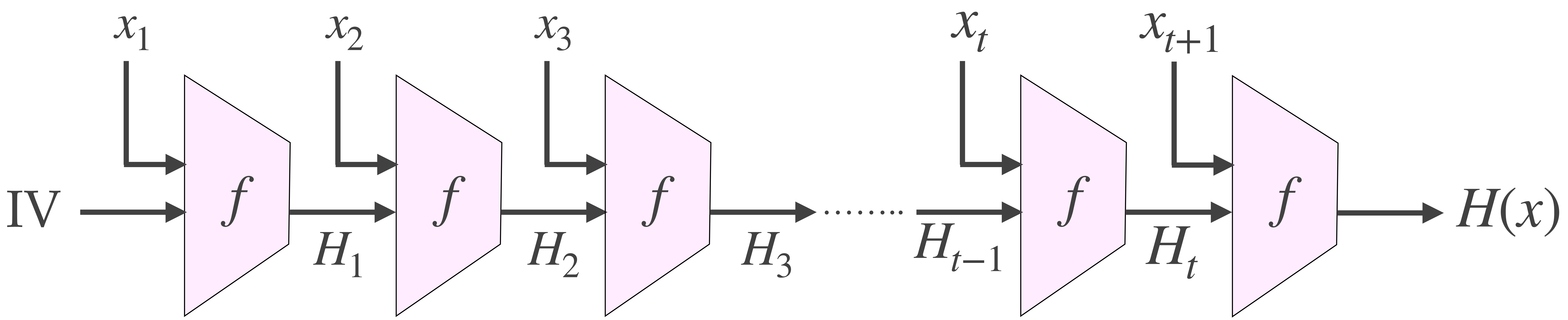
V3e
SHA-256

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.

$$h_1 = \texttt{0x6a09e667} \quad h_2 = \texttt{0xbb67ae85} \quad h_3 = \texttt{0x3c6ef372} \quad h_4 = \texttt{0xa54ff53a}$$
$$h_5 = \texttt{0x510e527f} \quad h_6 = \texttt{0x9b05688c} \quad h_7 = \texttt{0x1f83d9ab} \quad h_8 = \texttt{0x5be0cd19}$$

- 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.

$$y_0 = \texttt{0x428a2f98} \quad y_1 = \texttt{0x71374491} \quad y_2 = \texttt{0xb5c0fbcf} \quad y_3 = \texttt{0xe9b5dba5}$$
$$\ldots\ldots \quad \ldots\ldots \quad y_{62} = \texttt{0xbef9a3f7} \quad y_{63} = \texttt{0xc67178f2}$$

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: $(H_1, H_2, \ldots, H_7, H_8) \leftarrow (H_1 + A,\ H_2 + B,\ \ldots,\ H_7 + G,\ H_8 + H)$.

<u>Output</u>: 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