Introduction to SHA-2

ECE 4156/6156 Hardware-Oriented Security and Trust

Spring 2024

Assoc. Prof. Vincent John Mooney III

Georgia Institute of Technology

Reading Assignment

- Please read Chapters 3 and 5 of the course textbook by Katz and Lindell

Notation from Katz and Lindell

- $\{X\}$ is a set of elements of type X
- m is a message in plaintext
  - m is composed of smaller blocks m_i suitable for individual encryption steps
  - $m = \{m_i\}$
- c_i is ciphertext corresponding to message block m_i
- c is ciphertext corresponding to message m
- Enc_k is encryption with key k
  - $c \leftarrow Enc_k(m)$
- Dec_k is decryption with key k
  - $m \leftarrow Dec_k(c)$
- $<a,b>$ is a concatenation of a followed by b

One-Way Hash Functions (Keyless *H*)

- Given a message *m* of arbitrary length, a hash function *H* generates a fixed-length output *h*
  - $h = H(m)$
- A hash function is one-way if it satisfies the following
  1. Given *m* and *H*, it is easy to compute *h*
  2. Given *h* and *H*, it is hard to compute *m*
  3. Given *m* and *H*, it is hard to compute *m'* such that $H(m') = H(m)$
  4. Given *H*, it is hard to compute m_1 and m_2 such that $H(m_1) = H(m_2)$
- A one-way hash function can be used to provide a "fingerprint" of *m*
  - Note that properties iii and iv above make it hard for an adversary to change the message but not the one-way hash value
  - Property iv above is also known as *collision resistance*

A More Formal Hash Function Definition

- Formalities
  - A hash function *H* maps a domain into a smaller range
    - Let x be an input to *H*, e.g., if the domain is the set of possible messages, x = *m*
  - If the hash function uses a key, let key *s* of size *n* bits be generated by $\mathcal{Gen}$
    - Recall that some bitstrings may have to be omitted from the set of possible keys, e.g., DES has a small set of known weak keys which should be avoided
  - A keyed hash function take inputs *s* and x in order to produce output *h*
    - $H^s(x) \stackrel{\text{def}}{=} H(s, x)$
    - Note that many times the adversary trying to defeat a hash function possesses the key; hence, in order to emphasize the fact that the typical attack surface includes scenarios where the adversary has possession of the key, a superscript is used for *s*, i.e., H^s, instead of a subscript, i.e., H_s
  - Let the number of bits in the domain be ℓ' where ℓ' > *n*
- Definition 5.1 from Katz and Lindell
  - A hash function π is a pair of polynomial-time algorithms $\mathcal{Gen}$ and *H* such that $\mathcal{Gen}$ outputs a key *s* and *H* takes as input x of size ℓ' bits and key *s* to produce output *h* of size *n* bits

Collision Experiment on Hash Functions

- Note that as defined a hash function π maps a larger number of bits (ℓ') into a smaller number of bits (n)
  - Therefore it is impossible to always generate a unique h
  - H may also be called or referred to as a compression function
    - Note that the above bullet point informally uses H to refer to two algorithms $\mathcal{Gen}$ and H
- Collision-finding experiment
  - $\mathcal{Gen}$ outputs a key s
  - Adversary A is given s
  - A finds a collision if A can find x and x' such that $H^s(x) = H^s(x')$
  - If it is infeasible for A to find a collision, we say that H^s is *collision resistant*

Weaker Notions of Security

- *Target-collision resistance*
  - Given *s* and a uniformly random x, it is infeasible for an adversary to find x' such that $H^s(x) = H^s(x')$
  - Note1: this is also referred to in the literature as *second preimage resistance*
  - Note2: collision resistance (see previous page) implies *target-collision resistance*, i.e., second preimage resistance
- *Preimage resistance*
  - Given *s* and a uniformly random y, it is infeasible for an adversary to find x such that $H^s(x) = y$
  - Note that second preimage resistance (i.e., target-collision resistance) already implies preimage resistance

Why do collisions matter?

- Consider a legal document that is transmitted
- Suppose that the recipient has the expected hash
  - More on how encrypted documents and hash values are transmitted later…
- If collisions can be found in a reasonable time, the adversary could alter the legal document in such a way as to be favorable to the adversary and result in the same has value
  - Keep in mind that a typical file with human readable text values may be altered in many minor ways without changing the text, e.g., adding extra whitespaces or commas

The Original Widely Used One-way Hash: MD5

- Authored by Ronald Rivest, Professor of Electrical Engineering and Computer Science at MIT
  - Co-author of the asymmetric RSA cryptographic algorithm in 1977
  - Invented MD5 in 1991
    - MD stands for "Message Digest" and "5" is for Version 5
    - The "digest" is the hash value, i.e., a long message is consumed or "digested"
- Example use
  - Send the hash value first, i.e., the sender sends h first
  - Then send the message M
    - Note: the message should be encrypted! We will make our examples more and more realistic as we explain additional methods and terminology
  - The recipient can then calculate $h = H(M)$ and compare with the initial hash value
- Note that no key is used, i.e., the MD5 one-way hash is keyless

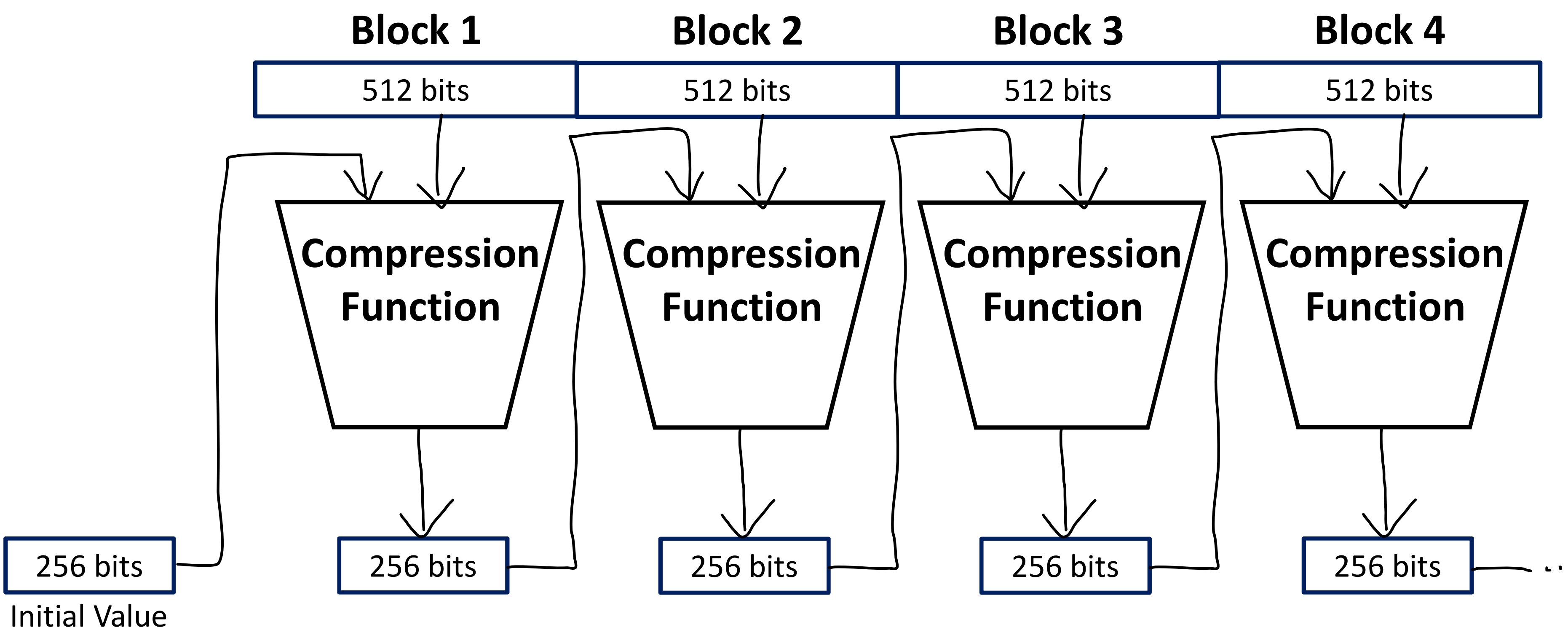
SHA-1

- In 1993, Den Boer and Bosselaers gave an early, but limited, result of finding a collision in MD5, although it was not generally applicable
- In 1995 NIST announced the release of a “secure hash algorithm” version 1, i.e., SHA-1
- By 1996 more attacks on MD5 were announced
- By early 2001 both MD5 and SHA-1 were both considered to be in danger of becoming broken, and so SHA-2 was announced by NIST
- Note that today both MD5 and SHA-1 are considered to be broken, i.e., an adversary with sufficient compute power can find collisions

SHA-2

- First published in 2001 with public comments accepted
- First complete version published in August 2002
  - Digest or hash sizes of 256, 384 or 512
- In 2004, a version of SHA-2 supporting a hash size of 224 was released to provide backward compatibility
- The first lab in this course uses the 256 bit version of SHA-2 also known as SHA-256

SHA-256 Calculation on 2048 Bits



SHA-256 Initial Value

- The initial 256 bits used in SHA-2 were calculated by taking the fractional parts of the square roots of the first eight prime numbers
  - Least significant four bytes = 0x6a09e667
  - 0xbb67ae85
  - 0x3c6ef372
  - 0xa54ff53a
  - 0x510e527f
  - 0x9b05688c
  - 0x1f83d9ab
  - Most significant four bytes = 0x5be0cd19
- The initial value never changes (for compatibility with the standard)

Message Padding

- The message to be hashed by SHA-256 needs to be padded to reach a size of a multiple of 512

For Each 512-bit Block

- M_0= least significant 32 bits
- M_1= next to least significant 32 bits
- …
- M_{15}= most significant 32 bits of the block

SHA-256 Compression Function

- Six logical functions are used
- Each function operates on 32-bit words to be easy to implement in sw
- $Ch(x, y, z) = (x\ OR\ y)\ XOR\ (not(x)\ OR\ z)$
- $Maj(x, y, z) = (x\ OR\ y)\ XOR\ (x\ OR\ z)\ XOR\ (y\ OR\ z)$
- $\sum_0(x) = S^2(x)\ XOR\ S^{13}(x)\ XOR\ S^{22}(x)$
  - where $S^n(x)$ means *rotate x left by n bits*
- $\sum_1(x) = S^6(x)\ XOR\ S^{11}(x)\ XOR\ S^{25}(x)$
- $\sigma_0(x) = S^7(x)\ XOR\ S^{18}(x)\ XOR\ R^3(x)$
  - where $R^n(x)$ means *rotate x right by n bits*
- $\sigma_1(x) = S^{17}(x)\ XOR\ S^{19}(x)\ XOR\ R^{10}(x)$

SHA-256 Compression Function (continued)

- Let there be four blocks so we have M^1, M^2, M^3, M^4
  - So M^1 is broken up into M_0^1, M_1^1,…,M_{15}^1
- For each block *i*, expanded message blocks W_0, W_1,…,W_{63} are computed as follows
  - $W_0 = M_0^i, W_1 = M_1^i, \ldots, W_{15} = M_{15}^i$
  - For j = 16 to 63

    {

    $W_j = \sigma_1\left(W_{j-2}\right) + W_{j-7} + \sigma_0\left(W_{j-15}\right) + W_{j-16}$ (mod 2^{32});

    }

SHA-256 Compression Function (continued)

- Let there be four blocks so we have M^1, M^2, M^3, M^4
  - So M^1 is broken up into M_0^1, M_1^1,…,M_{15}^1
- For each block *i*, expanded message blocks W_0, W_1,…,W_{63} are computed as follows
  - $W_0 = M_0^i, W_1 = M_1^i, \dots, W_{15} = M_{15}^i$
  - For j = 16 to 63

    {

    $W_j = [\sigma_1(W_{j-2}) + W_{j-7} + \sigma_0(W_{j-15}) + W_{j-16}]$ (mod 2^{32});

    }

Main Loop Initialization

- Let *N* be the number of blocks (e.g., earlier *N* = 4)
- Put the initial values in registers as follows for the first block (Block 1):
  - $H_1^{(0)}$ = 0x6a09e667
  - $H_2^{(0)}$ = 0xbb67ae85
  - $H_3^{(0)}$ = 0x3c6ef372
  - $H_4^{(0)}$ = 0xa54ff53a
  - $H_5^{(0)}$ = 0x510e527f
  - $H_6^{(0)}$ = 0x9b05688c
  - $H_7^{(0)}$ = 0x1f83d9ab
  - $H_8^{(0)}$ = 0x5be0cd19
- For Blocks 2 and higher, use the previous 256 bit hash result for $H_1^{(0)}, \ldots, H_8^{(0)}$

Constants K_0, …, K_{63}

- 64 constants K_0, …, K_{63} are defined based on the fractional parts of the cube roots of the first 64 prime numbers
- K_0 = 0x428a2f98
- K_1 = 0x71374491
- …
- K_{63} = 0xc67178f2

Main Loop

For i = 1 to N

{

$a = H_1^{(i-1)}; b = H_2^{(i-1)}; c = H_3^{(i-1)}; d = H_4^{(i-1)}; e = H_5^{(i-1)}; f = H_6^{(i-1)}; g = H_7^{(i-1)}; h = H_8^{(i-1)};$

For j = 0 to 63

{

Compute $Ch(e, f, g)$, $Maj(a, b, c)$, $\sum_0(a)$, $\sum_1(e)$ and W_j;

$T_1 = h + \sum_1(e) + Ch(e, f, g) + W_j + K_j \pmod{2^{32}}$;

$T_2 = h + \sum_1(e) + Maj(a, b, c) \pmod{2^{32}}$;

$h = g; g = f; f = e; e = d + T_1 \pmod{2^{32}}; d = c;$

$c = b; b = a; a = T_1 + T_2 \pmod{2^{32}}$;

}

$H_1^{(i)} = a + H_1^{(i-1)} \pmod{2^{32}}$; $H_2^{(i)} = b + H_2^{(i-1)} \pmod{2^{32}}$; …; $H_8^{(i)} = h + H_8^{(i-1)} \pmod{2^{32}}$;

}

Main Loop

For i = 1 to N

{

$a = H_1^{(i-1)}; b = H_2^{(i-1)}; c = H_3^{(i-1)}; d = H_4^{(i-1)}; e = H_5^{(i-1)}; f = H_6^{(i-1)}; g = H_7^{(i-1)}; h = H_8^{(i-1)};$

For j = 0 to 63

{

Compute $Ch(e, f, g)$, $Maj(a, b, c)$, $\sum_0(a)$, $\sum_1(e)$ and W_j;

$T_1 = [h + \sum_1(e) + Ch(e, f, g) + W_j + K_j] (\text{mod } 2^{32});$

$T_2 = [h + \sum_1(e) + Maj(a, b, c)] (\text{mod } 2^{32});$

$h = g; g = f; f = e; e = [d + T_1] (\text{mod } 2^{32}); d = c;$

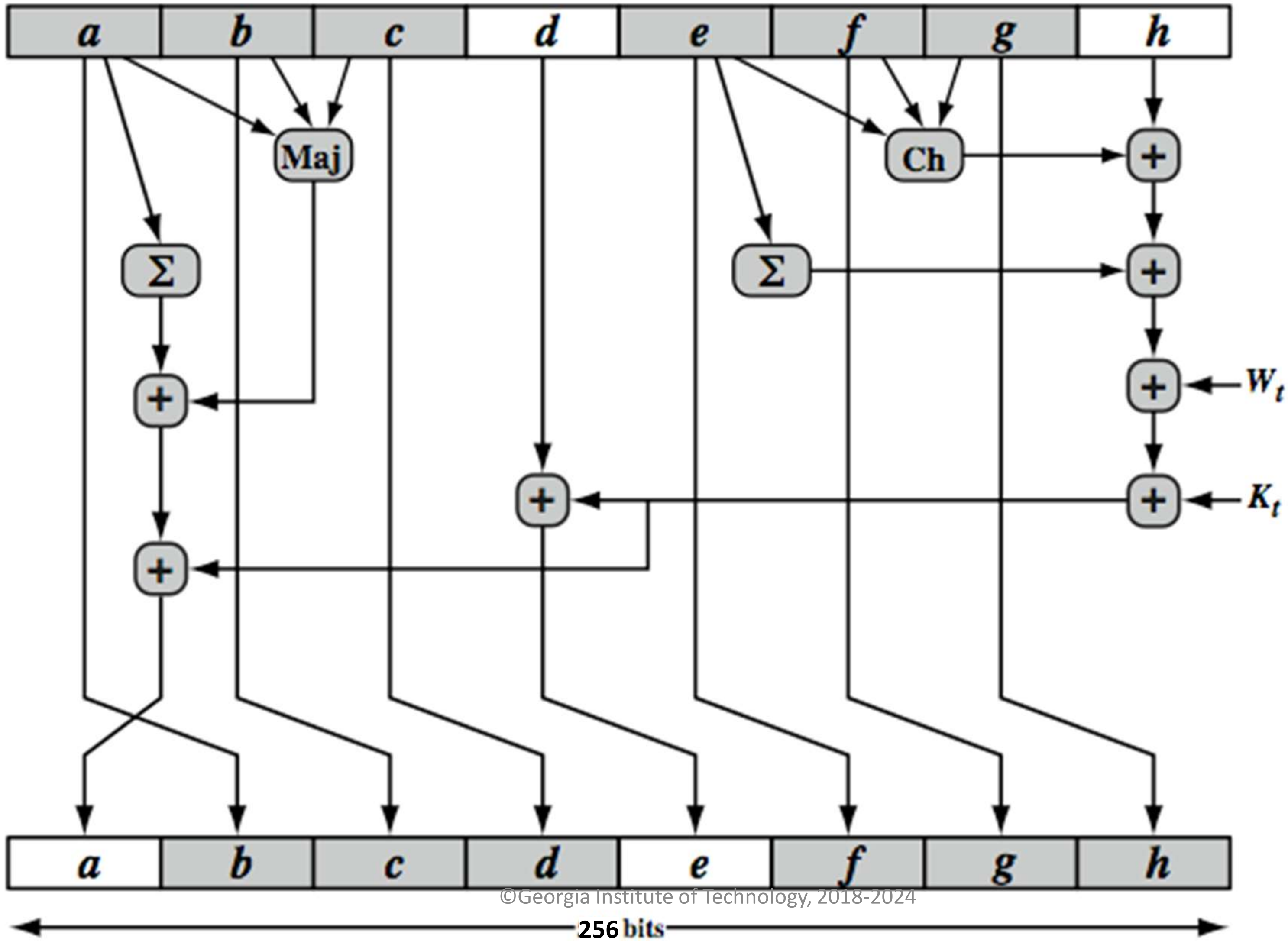
$c = b; b = a; a = [T_1 + T_2] (\text{mod } 2^{32});$

}

$H_1^{(i)} = a + H_1^{(i-1)} (\text{mod } 2^{32})$; $H_2^{(i)} = b + H_2^{(i-1)} (\text{mod } 2^{32})$; …; $H_8^{(i)} = h + H_8^{(i-1)} (\text{mod } 2^{32})$;

}

Figure for the inner loop (j = 0 to 63)



Final Result

- The final result is $H_1^{(N)}, H_2^{(N)}, \ldots, H_8^{(N)}$
- These eight 32-bit values constitute the 256-bit hash result