Cryptographic Hash Functions

G.P. Biswas

Prof. /CSE, IIT, Dhanbad

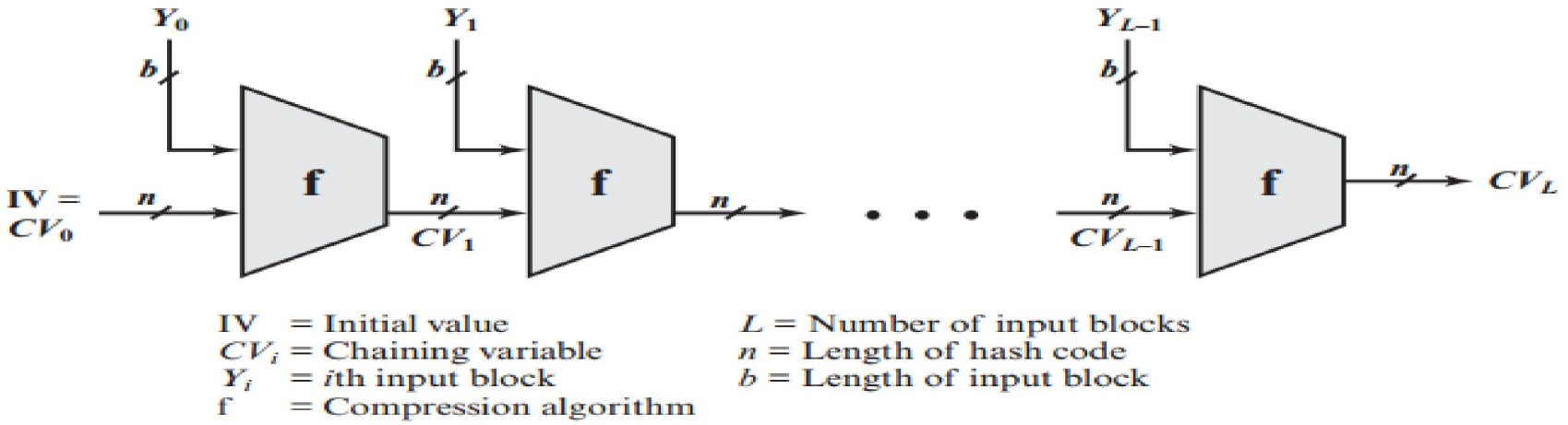
Outline

- Cryptographic hash function
- Important properties of hash function
- Security analysis of hash function
- Overview of different families of hash functions

Introduction

- A cryptographic hash function takes a message of arbitrary length and creates a message digest of fixed length (Ex. MD5 produces 128-bit hash value)
- For a particular message, the message digest, or hash value, can be seen as the fingerprint of a message, i.e., a unique representation of a message.
- All cryptographic hash functions need to create a fixed-size digest out of a variable-size message.
- The best way to create such function is using iteration, and used a necessary number of times.
- The fixed-size input function is referred to as a compression function.

General Structure of a Secure Hash Code (Digest)

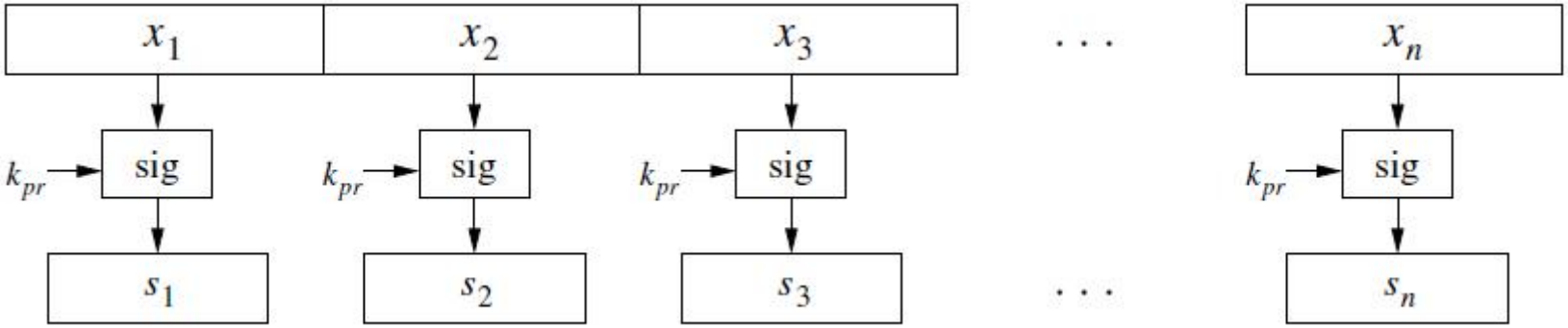


The hash algorithm involves repeated use of a compression function, f, that takes two inputs (an n-bit input from the previous step, called the chaining variable (CV), and a b-bit block) and produces an n-bit output. At the start of hashing, the chaining variable has an initial value (IV) that is specified as part of the algorithm. The final value
of the chaining variable is the hash value (CV-L). Often, b>n; hence the term compression.

$$CV_0 = IV = \text{initial } n\text{-bit value}$$
$$CV_i = \mathrm{f}(CV_{i-1}, Y_{i-1}) \quad 1 \leq i \leq L$$
$$\mathrm{H}(M) = CV_L$$

where the input to the hash function is a message M consisting of the blocks $Y_0, Y_1, \ldots, Y_{L-1}$.

Motivation: Signing Long Messages

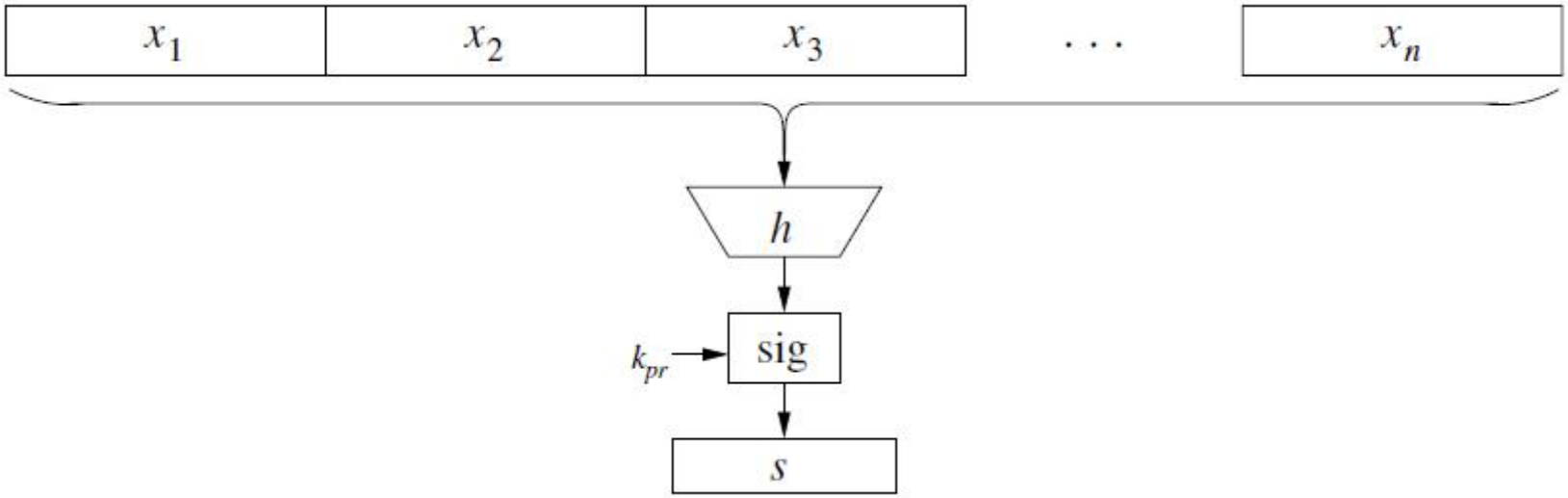


Problem 1: High Computational Load

Problem 2: Message Overhead

Problem 3: Security Limitations

Signing of long messages with a hash function



Security Requirements of Hash Functions

- The following table list the security requirements of cryptographic hash function:

Table 11.1 Requirements for a Cryptographic Hash Function H

Requirement	Description
Variable input size	H can be applied to a block of data of any size.
Fixed output size	H produces a fixed-length output.
Efficiency	$H(x)$ is relatively easy to compute for any given x, making both hardware and software implementations practical.
Preimage resistant (one-way property)	For any given hash value h, it is computationally infeasible to find y such that $H(y) = h$.
Second preimage resistant (weak collision resistant)	For any given block x, it is computationally infeasible to find $y \neq x$ with $H(y) = H(x)$.
Collision resistant (strong collision resistant)	It is computationally infeasible to find any pair (x, y) with $x \neq y$, such that $H(x) = H(y)$.
Pseudorandomness	Output of H meets standard tests for pseudorandomness.

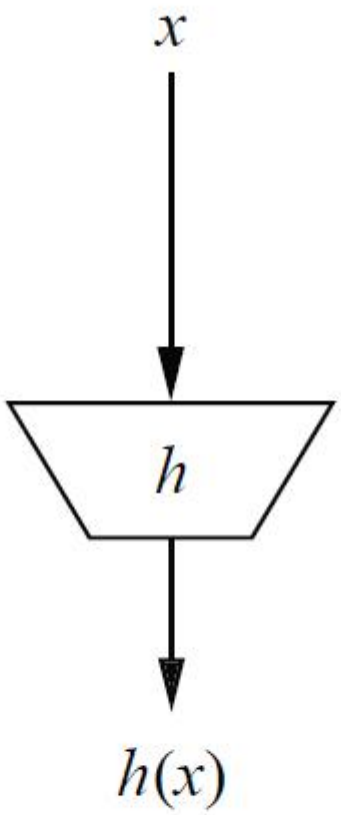
Security Requirements of Hash Function

There are three central properties which hash functions need to possess in order to be secure:

- Pre-image resistance (or one-way-ness)
- Second pre-image resistance (or weak collision resistance)
- Collision resistance (or strong collision resistance)

Pre-image Resistance

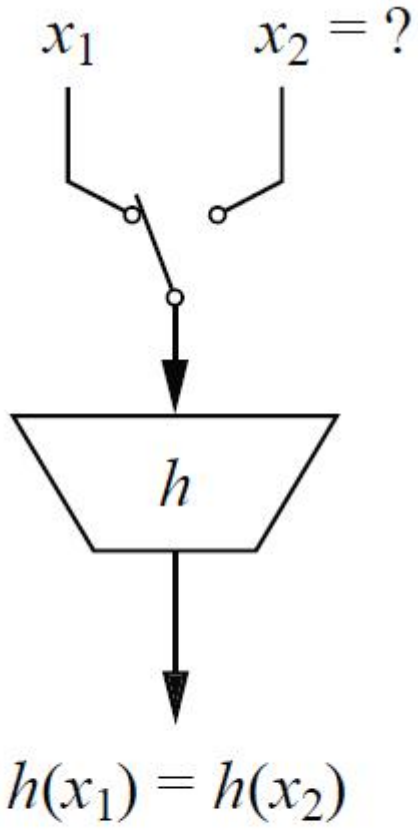
- Hash functions need to be *one-way*
- *Given a hash output z, it must be computationally* infeasible to find an input message *x such that z = h(x), i.e.,* $x = h^{-1}(z)$



preimage resistance

Second Pre-image (or Weak Collison) Resistance

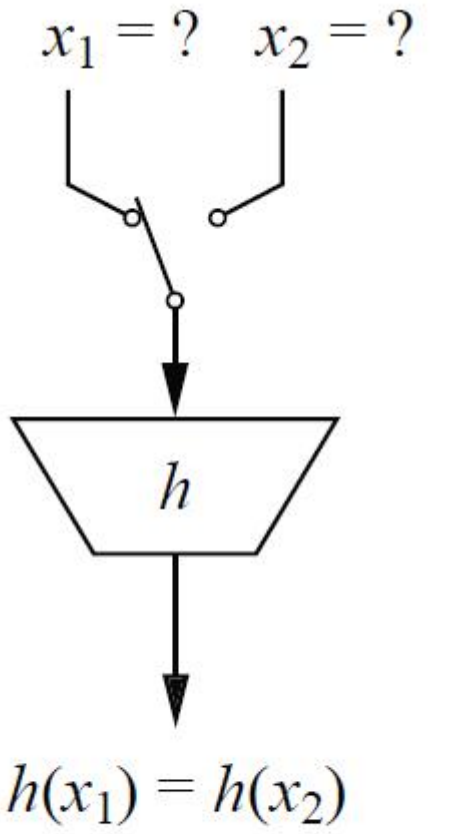
- It is essential that two different messages do not hash to the same value.
- It should be computationally infeasible to find a message *x2* for a given message *x1* such that *x1 ≠ x2 and h(x1) = h(x2).*



second preimage resistance

Collision Resistance (or Strong Collision)

- It is computationally infeasible to find any pair (x1, x2) such that *x1≠x2 and h(x1) = h(x2).*



collision resistance

Properties of Hash Functions

1. **Arbitrary message size** $h(x)$ can be applied to messages x of any size.
2. **Fixed output length** $h(x)$ produces a hash value z of fixed length.
3. **Efficiency** $h(x)$ is relatively easy to compute.
4. **Preimage resistance** For a given output z, it is impossible to find any input x such that $h(x) = z$, i.e, $h(x)$ is one-way.
5. **Second preimage resistance** Given x_1, and thus $h(x_1)$, it is computationally infeasible to find any x_2 such that $h(x_1) = h(x_2)$.
6. **Collision resistance** It is computationally infeasible to find any pairs $x_1 \neq x_2$ such that $h(x_1) = h(x_2)$.

Relation among Hash function properties

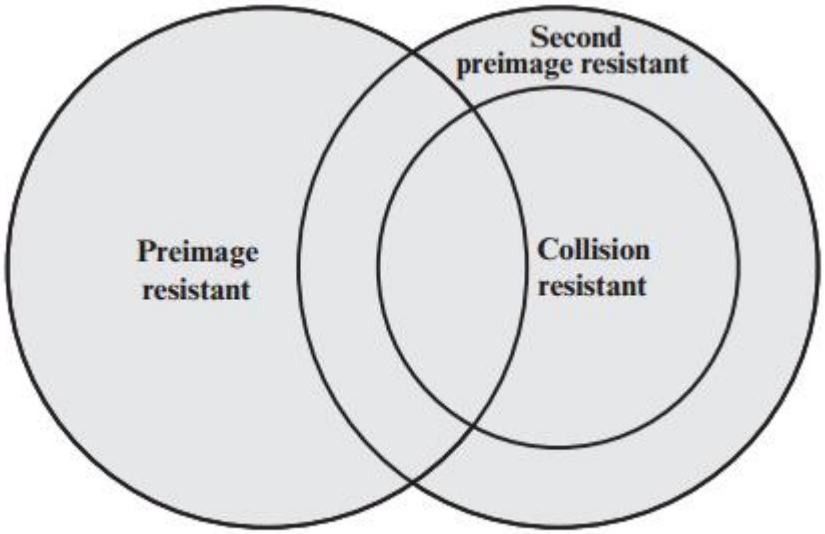


Figure 11.6 Relationship Among Hash Function Properties

Birthday Attack

- Due to the pigeonhole principle, collisions always exist, where pigeonhole principle states that if *n* items are put into m containers (n> m), then at least one container must contain more than one item.
- The question is how difficult it is to find them.
- Our first guess is probably that this is as difficult as finding second pre-images, i.e., if the hash function has an output length of 80 bits, we have to check about 2^{80} messages. However, it turns out that an attacker needs only about 2^{40} messages due to the *birthday attack*.

Birthday Paradox

Problem Statement: How many people are needed at a party such that there is a reasonable chance (probability more than 0.5) at least two people have the same birthday?

So

$$P(\text{no collision among 2 people}) = \left(1 - \frac{1}{365}\right)$$

If a third person joins the party, he or she can collide with both of the people already there, hence:

$$P(\text{no collision among 3 people}) = \left(1 - \frac{1}{365}\right) \cdot \left(1 - \frac{2}{365}\right)$$

Consequently, the probability for t people having no birthday collision is given by:

$$P(\text{no collision among } t \text{ people}) = \left(1 - \frac{1}{365}\right) \cdot \left(1 - \frac{2}{365}\right) \cdots \left(1 - \frac{t-1}{365}\right)$$

Contd…

For $t = 366$ people we will have a collision with probability 1 since a year has only 365 days. We return now to our initial question: how many people are needed to have a 50% chance of two colliding birthdays? Surprisingly—following from the equations above—it only requires 23 people to obtain a probability of about 0.5 for a birthday collision since:

$$\begin{aligned} P(\text{at least one collision}) &= 1 - P(\text{no collision}) \\ &= 1 - \left(1 - \frac{1}{365}\right) \cdots \left(1 - \frac{23-1}{365}\right) \\ &= 0.507 \approx 50\%. \end{aligned}$$

73 people with more than 99%

Note that for 40 people the probability is about 90%. Due to the surprising outcome it is often referred to as the birthday paradox.

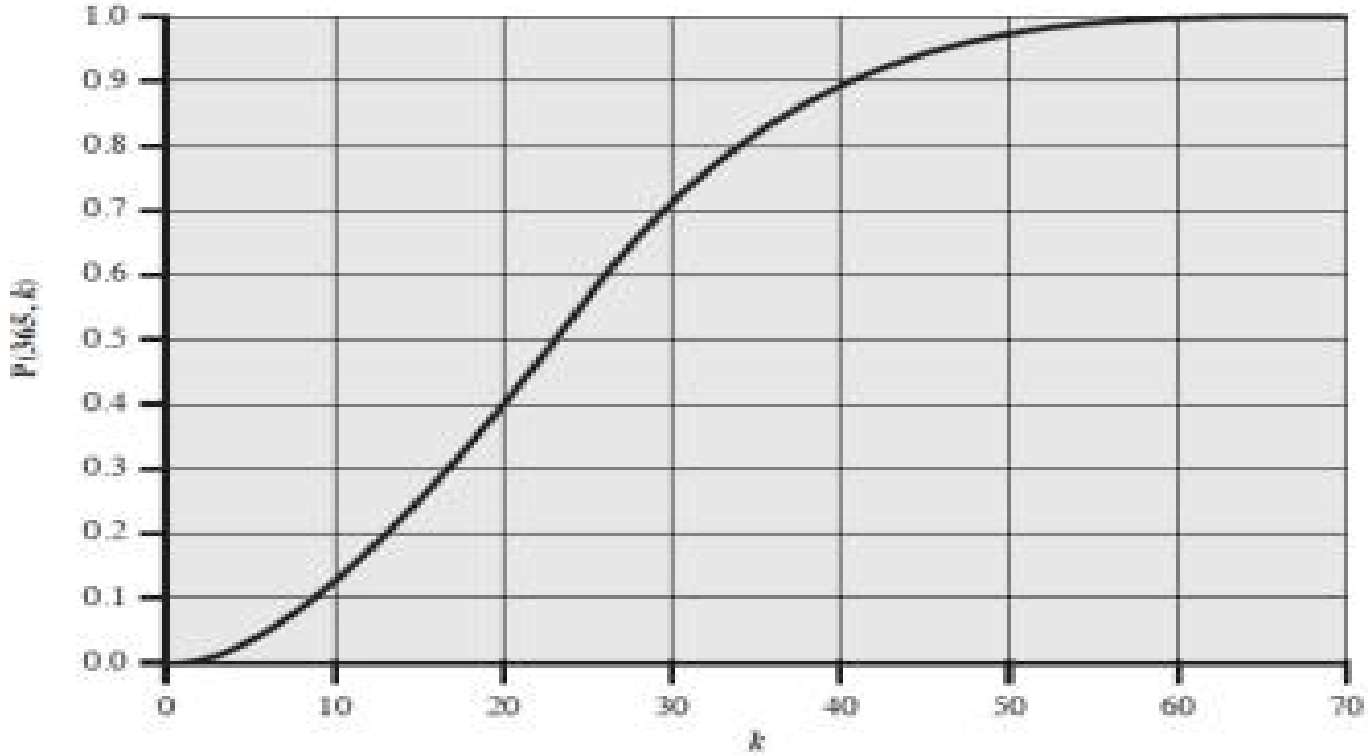


Figure 11.13 The Birthday Paradox

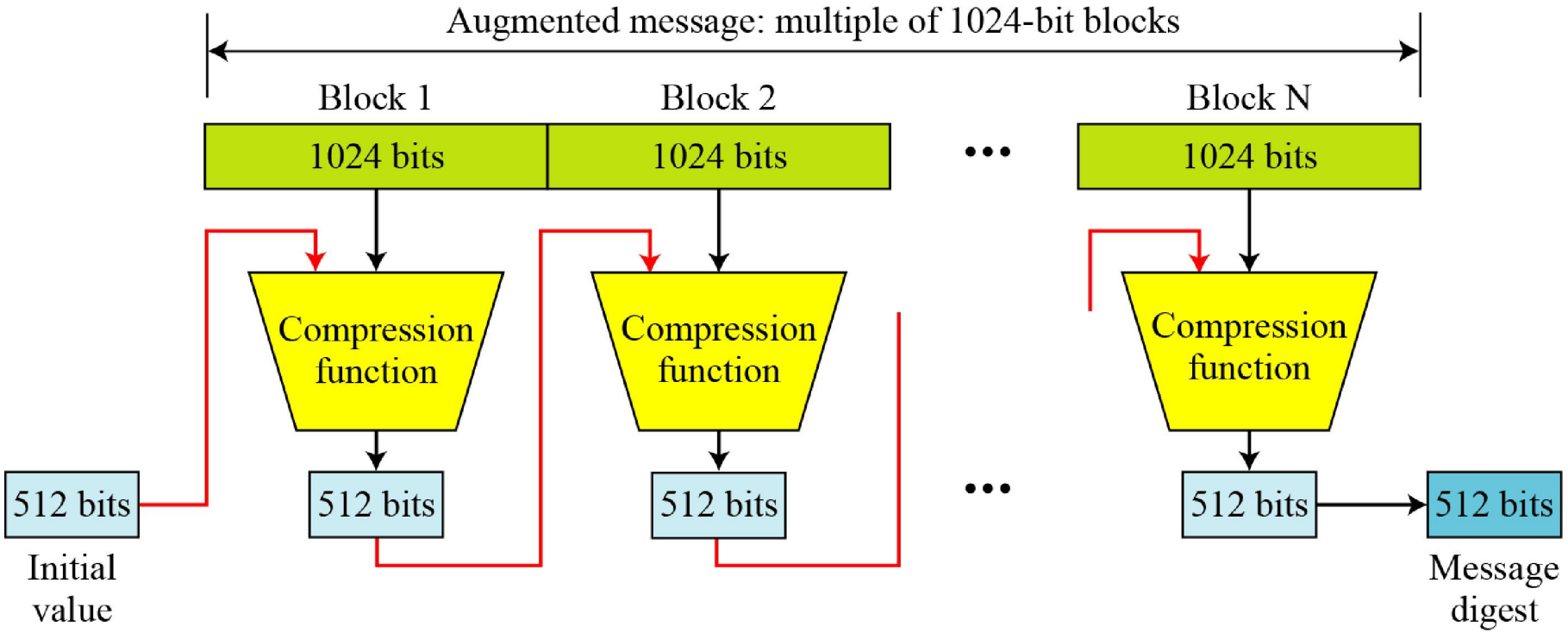
Secure Hash Algorithm (SHA)

SHA Versions

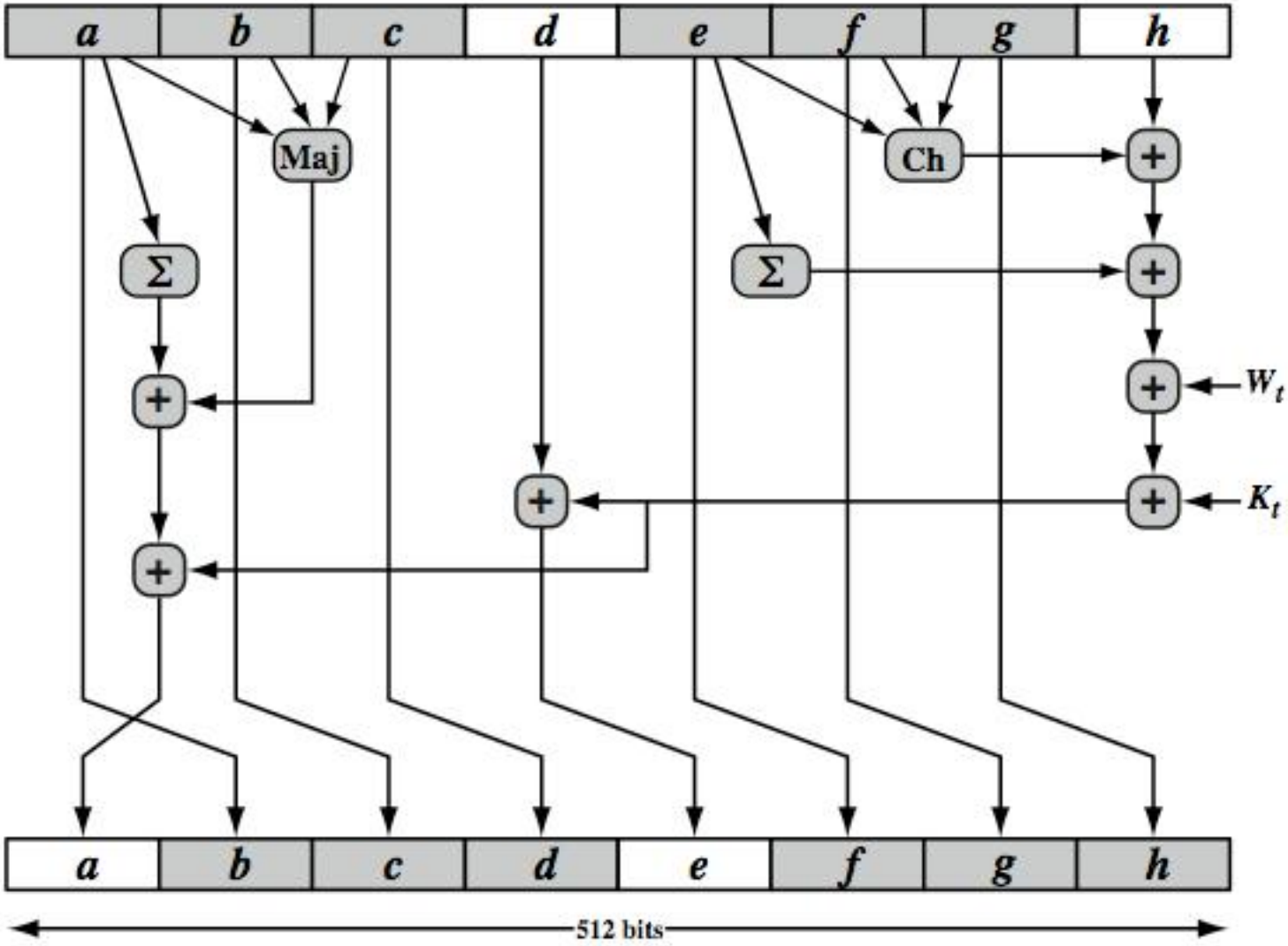
Table 11.3 Comparison of SHA Parameters

	SHA-1	SHA-224	SHA-256	SHA-384	SHA-512
Message Digest Size	160	224	256	384	512
Message Size	$< 2^{64}$	$< 2^{64}$	$< 2^{64}$	$< 2^{128}$	$< 2^{128}$
Block Size	512	512	512	1024	1024
Word Size	32	32	32	64	64
Number of Steps	80	64	64	80	80

SHA-512 (Secure Hash Algorithm)



H-Function



Step 1: Append padding bits, consists of a single 1-bit followed by the necessary number of 0-bits, so that its length is congruent to 896 modulo 1024

Step 2: Append length as an (big-endian) unsigned 128-bit integer

Step 3: Initialize hash buffer to a set of 64-bit integer constants

Step 4: Process the message in 1024-bit (128-word) blocks, which forms the heart of the algorithm. Each round takes as input the 512-bit buffer value H_i, and updates the contents of that buffer.

Step 5: Output the final state value as the resulting hash

Thank You