Cryptographic Hash Functions

Message Integrity

- Cryptographic hash function is a mechanism which provides message integrity.
- The cryptographic hash function creates a compressed image of the message.
- The output from the cryptographic hash function is called message digest or hash.
- In order to check the integrity of the message the message digest should be kept integral.

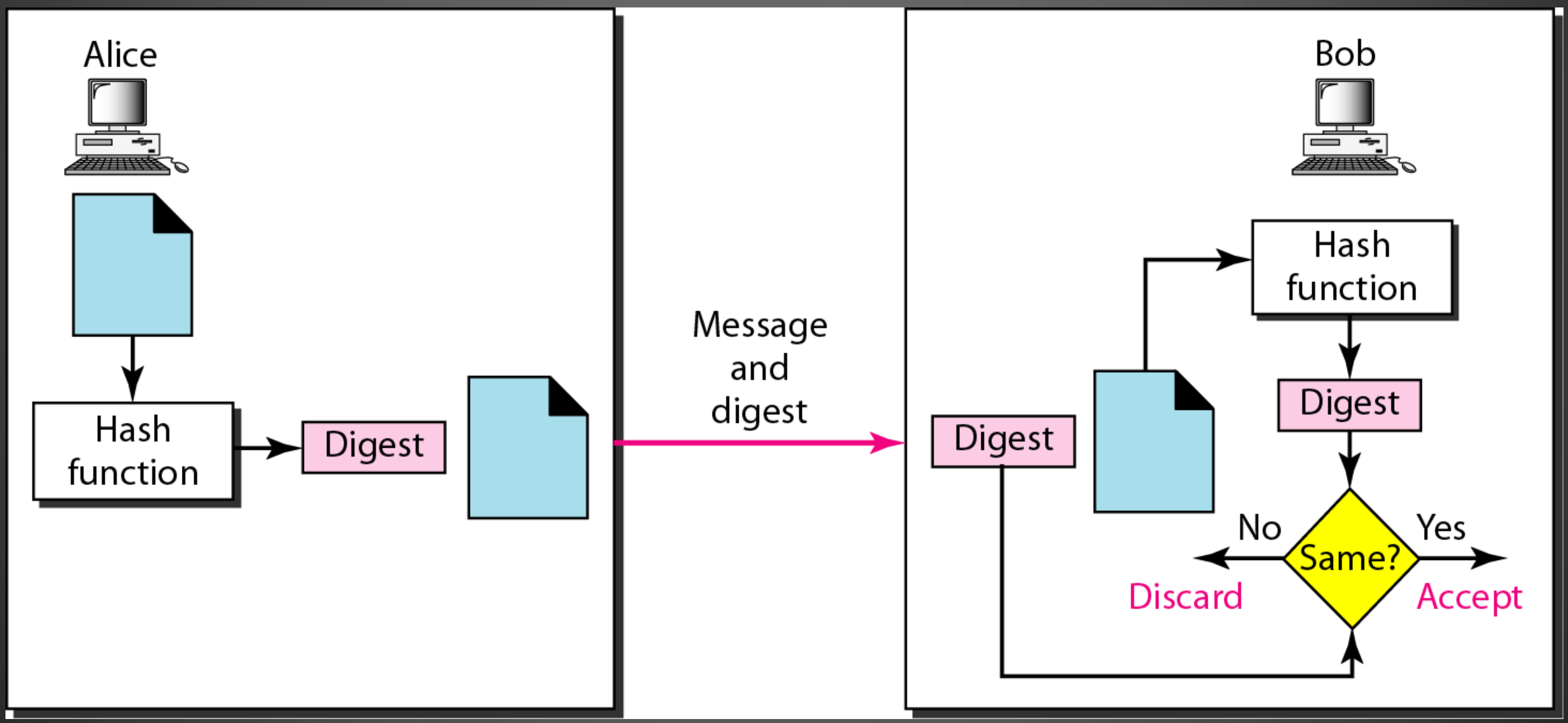
- Cryptographic hash function accepts a variable size message, M, as input & produces a fixed size output referred to as a hash code, H(M).

- A hash code , is a function of the input message.

- The hash code is a function of all the bits of the message.

- A change to any bits in the message results in the change to the hash code.

Fig: Message Integrity



- A cryptographic hash function should satisfy three criteria : preimage resistance, second preimage resistance, and collision resistance.

- **Pre image resistance (one way property)**

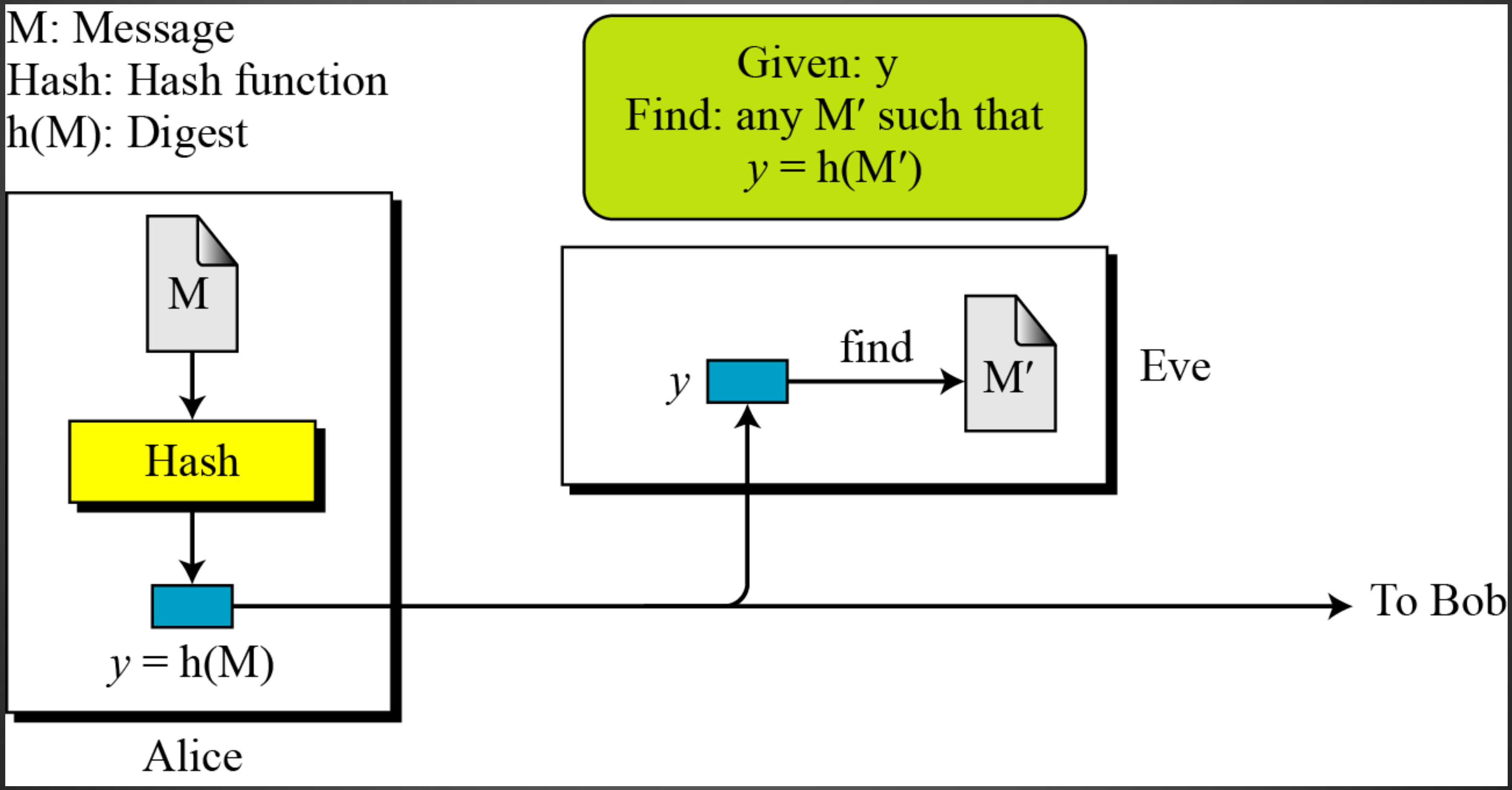
- Given y = h (M) it must be extremely difficult for the eve to find any message M’ such that y= h (M’)

- If the hash is not pre image resistant ,Eve can intercept the digest h (M) and create a message M'.
- Eve can then send M' to Bob and can fool Bob.

- If the hash function is pre image resistant it ensures that a message cannot be easily forged.

Preimage Attack

Given: y = h(M) **Find: M′ such that y = h(M′)**



- **Second Preimage resistance (weak collision resistance)**

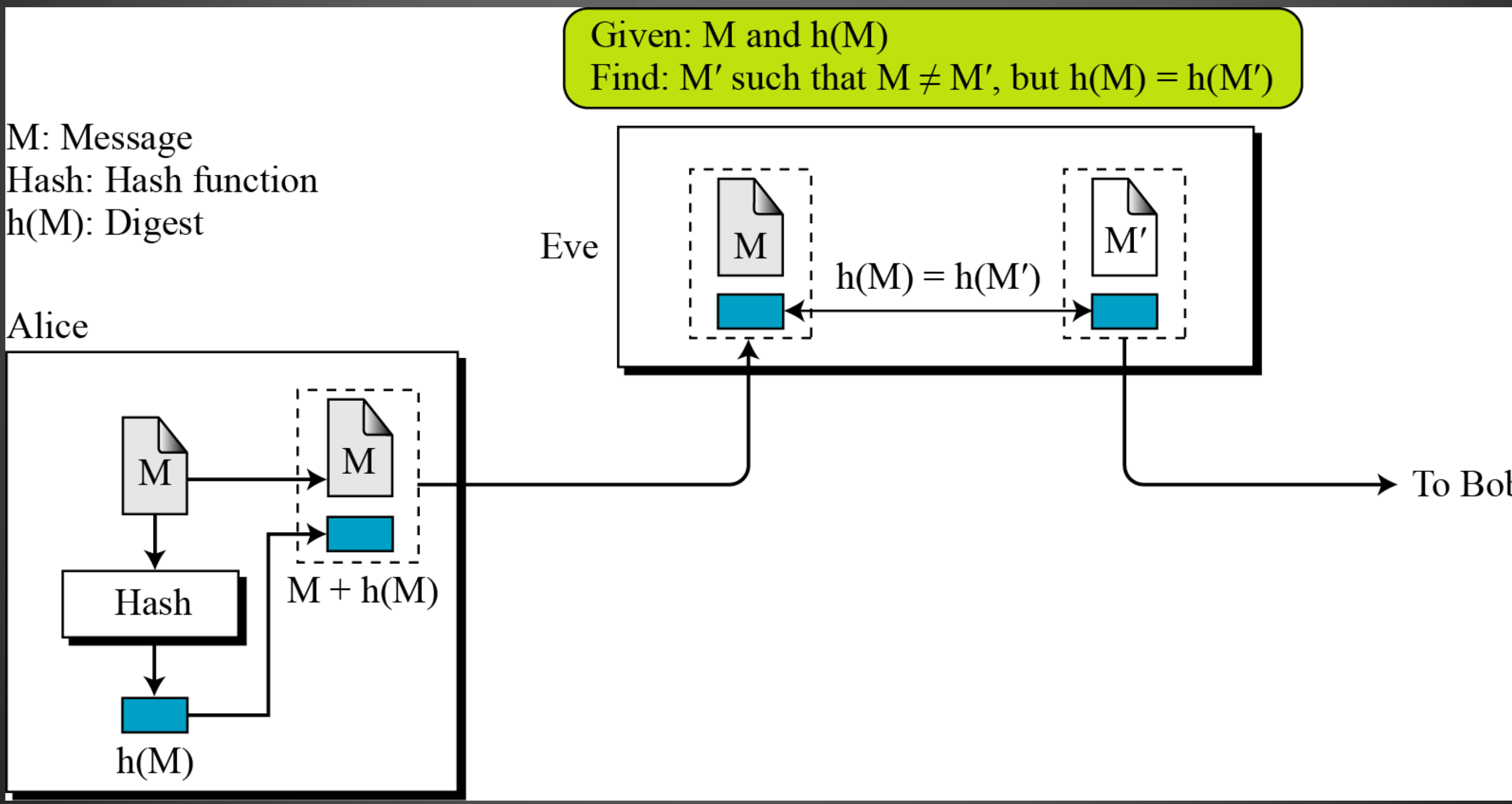
- Given a specific message and digest it should be computationally infeasible to create another message with the same digest.

- If eve intercepts a message M and it's digest h (M) , and she creates another message M' ≠ M but h(M) = h (M').

- Eve sends the M' and h (M') to bob.

Second Preimage Attack

Given: M and h(M) **Find: M′ ≠ M such that h(M) = h(M′)**



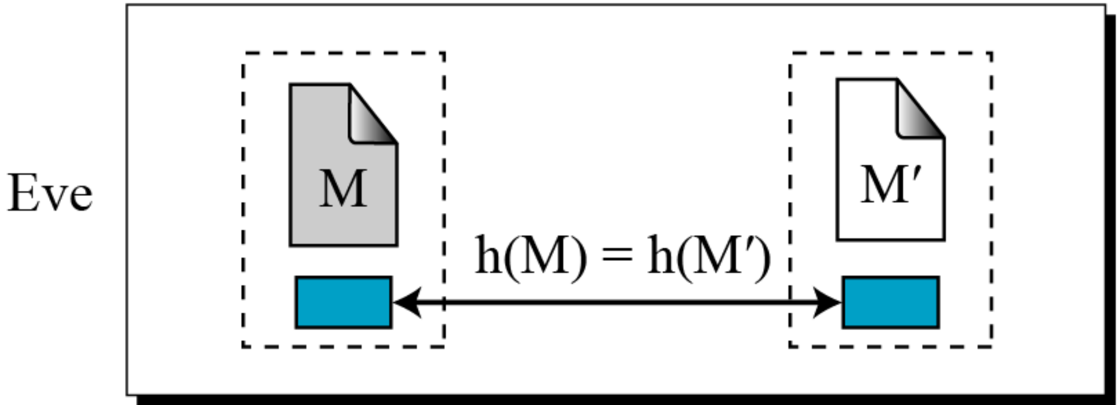
- **Collision resistance (Strong collision resistance)**

- Eve is trying to find two messages that hash to the same digest.

Collision Attack	
Given: none	**Find: $M' \neq M$ such that $h(M) = h(M')$**

M: Message
Hash: Hash function
h(M): Digest

Find: M and M′ such that M ≠ M′, but h(M) = h(M′)



- Preimage attack

- Create all the possible combinations of message.
- Calculate the hash value of each of the message.
- Check it against the given hash value.
- The difficulty of preimage attack is proportional to 2^n .

- Second preimage attack

- Create all the possible combinations of the message and calculate it’s hash value.
- Check it against the given hash value.
- The difficulty of second preimage attack is proportional to 2^n .

- Collision attack

Algorithm 11.3 *Collision attack*

```
Collision_Attack
{
   for (i = 1 to k )
   {
        create (M[i])
        D[i] ← h (M[i])                               // D [i] is a list of created digests
        for (j = 1 to i − 1)
        {
           if (D[i] = D[j]) return (M[i] and M[j])
        }
   }
   return failure
}
```

- The probability of success depends on the size of list k.

- To find the probability third birthday problem is used.
- “ What is the minimum number , k , students in a class room such that it is likely that at least two students have the same birthday ? “

- The probability of success is proportional to $1 – e^{-k\,(k-1)/\,2N}$

- If eve needs to be 50 percent successful the size of k should be proportional to $1.18 \times N^{1/2}$ or $k \approx 1.18 \times 2^{n/2}$

- The difficulty of collision attack is proportional to $2^{n/2}$.

Secure Hash Algorithm (SHA)

- Developed by the National Institute of Standards & Technology (NIST) & published as a standard in 1993.

- A revised version, SHA-1, was issued in 1995.

- SHA-1 produces a hash value of 160 bits.

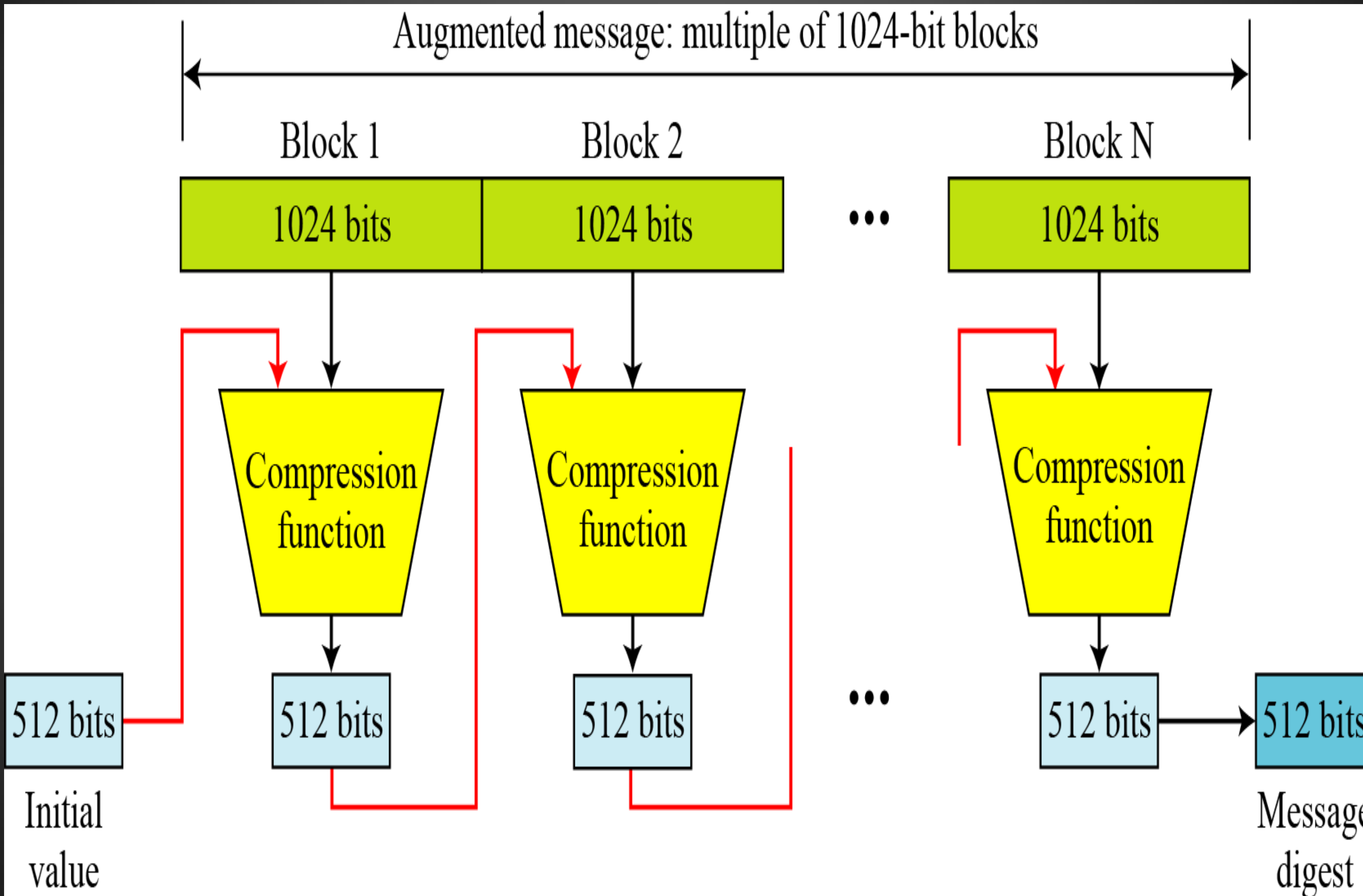
- Three new versions, SHA-256, SHA-384 & SHA-512 produces hash value of lengths 256, 384 & 512 bits.

SHA - 512

- Input - A message with a maximum length of less than 2^{128} bits.

- Output – A 512-bit message digest.

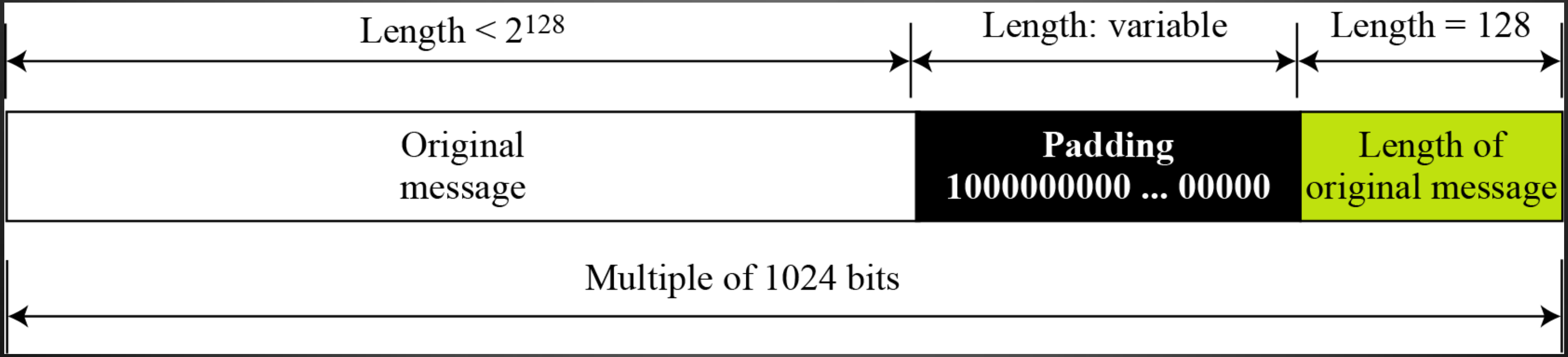
- The input is processed in 1024-bit blocks.


Augmented message: multiple of 1024-bit blocks
Block 1
Block 2
Block N
1024 bits
1024 bits
1024 bits
Compression function
Compression function
Compression function
512 bits
512 bits
512 bits
512 bits
512 bits
Initial value
Message digest

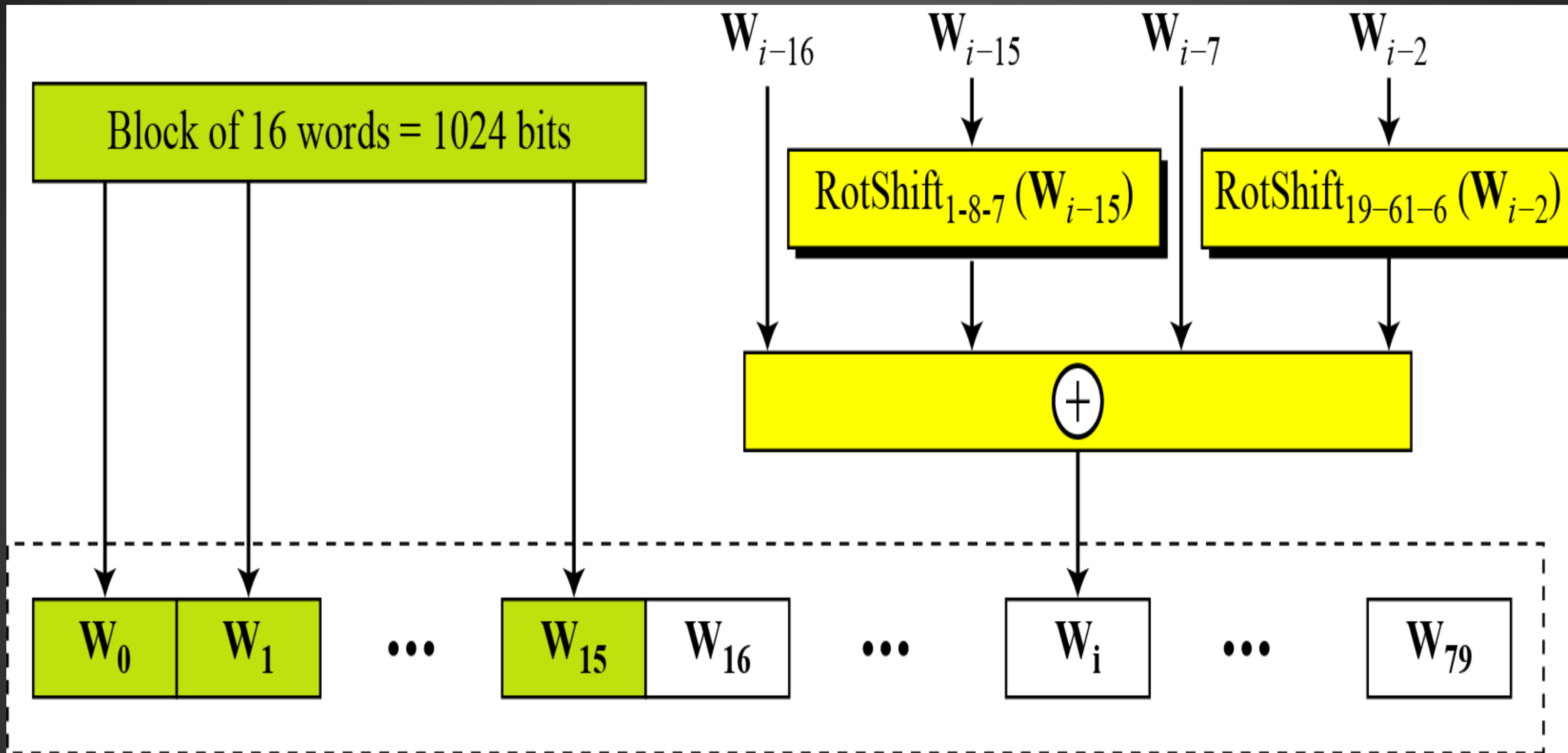
- The processing consists of the following steps:
  i. Append padding bits
     - The message is padded so that its length is a multiple of 1024 .
     - $| M | + | P | + 128 = 0 \bmod 1024$
     - Padding consists of a single 1-bit followed by the necessary no.of 0-bits.

ii. Append length

- A block of 128 bits is appended to the message.
- This contains the length of the original message.



- iii Array initialization
  - Array is of 80 words
  - Each block is split into 16 words of 64 bits.
  - This 16 words occupy the first 16 words of the array.
  - The remaining can be computed from the first 16 words.
  - $W_i = W_{i-16}$ xor $\text{RotShift}_{1-8-7}(W_{i-15})$ xor W_{i-7} xor $\text{RotShift}_{19-61-6}(W_{i-2})$



$\text{RotShift}_{l\text{-}m\text{-}n}(x)$: $\text{RotR}_l(x) \oplus \text{RotR}_m(x) \oplus \text{ShL}_n(x)$

$\text{RotR}_i(x)$: Right-rotation of the argument x by i bits

$\text{ShL}_i(x)$: Shift-left of the argument x by i bits and padding the left by 0's.

iv Initialize hash buffer

- A 512-bit buffer is used to hold intermediate & final results of the hash function.

- The buffer is represented as eight 64-bit words which are initialized with the following values:

  A = 6A09E667F3BCC908

  B = BB67AE8584CAA73B

  C = 3C6EF372FE94F82B

D= A54FF53A5F1D36F1

E = 510E527FADE682D1

F= 9B05688C2B3E6C1F

G = 1F83D9ABFB41BD6B

H= 5BE0CDI9137E2179

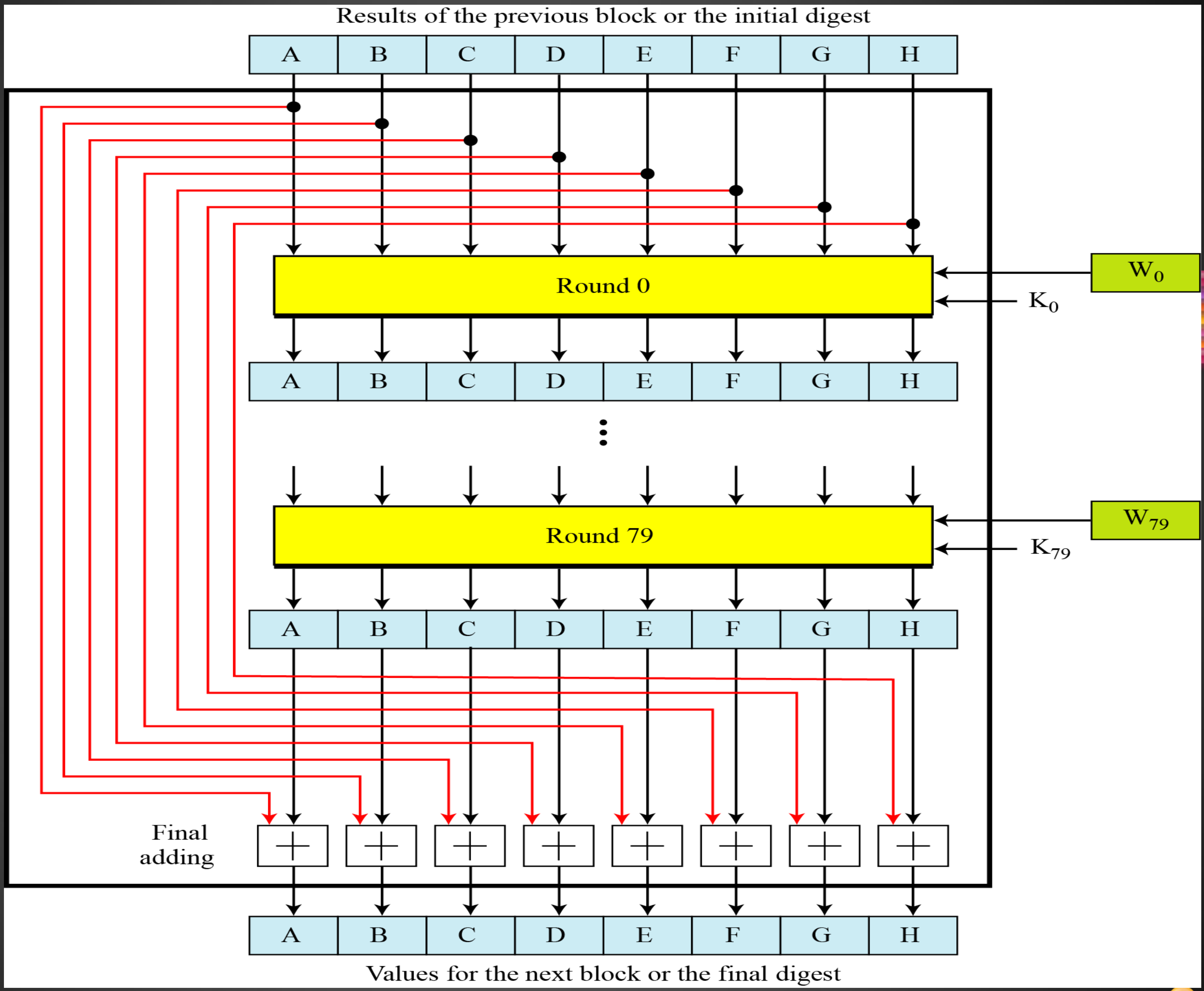
v. Process message in 1024-bit blocks

(Compression Function)

- The processing of each block of data in SHA – 512 consists of 80 rounds.
- Each round takes as input a 512-bit buffer value ABCDEFGH & updates the contents of the buffer.
- Each round makes use of one word W_i & a 64-bit constant K_i.

- At the beginning of the processing the values of the eight words ABCDEFGH are stored in temporary variables.

- At the end of the processing the output of the 79 th round is added to the values that are stored in temporary variables in the beginning.


Results of the previous block or the initial digest
A B C D E F G H
Round 0
W_0
K_0
A B C D E F G H
Round 79
W_{79}
K_{79}
A B C D E F G H
Final adding
A B C D E F G H
Values for the next block or the final digest

v. Output

- After all N 1024-bit blocks have been processed, the output from the N^{th} stage is the 512-bit message digest.

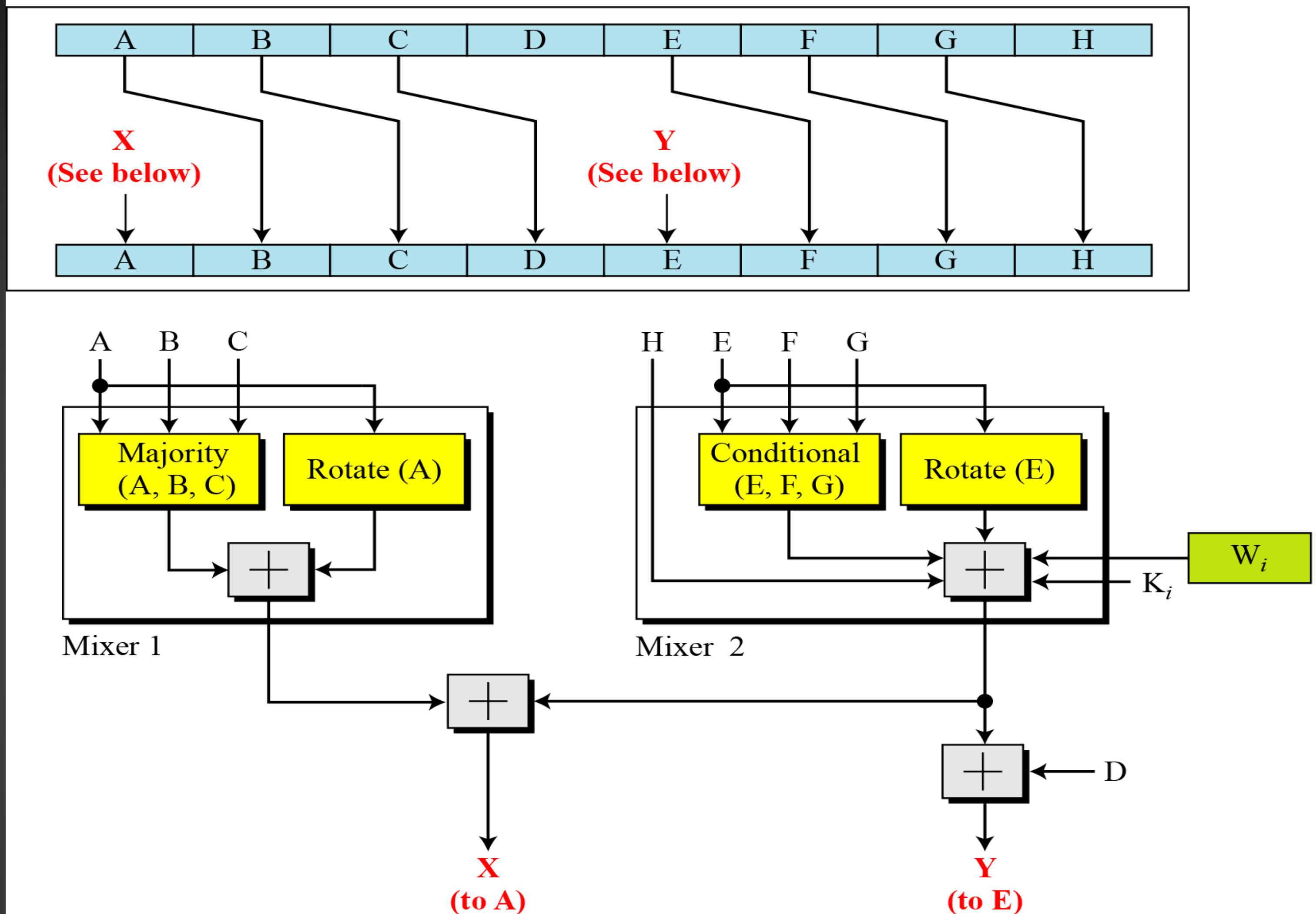
Structure of each round

- In each round the eight new values for the buffers are created from the values of buffers in the previous round.

B <- A ,C <- B ,D <- C ,F <- E ,G <- F ,H <- G

- A and E receive their values from some complex functions.

Round



Majority (x, y, z)

$(x \text{ AND } y) \oplus (\mathbf{y} \text{ AND } z) \oplus (z \text{ AND } x)$

Rotate (x)

$\text{RotR}_{28}(x) \oplus \text{RotR}_{34}(x) \oplus \text{RotR}_{39}(x)$

Conditional (x, y, z)

$(x \text{ AND } y) \oplus (\text{NOT } x \text{ AND } z)$

$\boxed{+}$ addition modulo 2^{64}

$\text{RotR}_i(x)$: Right-rotation of the argument x by i bits

Majority Function

$$(A_j \text{ AND } B_j) \oplus (B_j \text{ AND } C_j) \oplus (C_j \text{ AND } A_j)$$

Conditional Function

$$(E_j \text{ AND } F_j) \oplus (\text{NOT } E_j \text{ AND } G_j)$$

Rotate Functions

$$\text{Rotate (A): } \text{RotR}_{28}(A) \oplus \text{RotR}_{34}(A) \oplus \text{RotR}_{29}(A)$$

$$\text{Rotate (E): } \text{RotR}_{28}(E) \oplus \text{RotR}_{34}(E) \oplus \text{RotR}_{29}(E)$$

- The majority function is a bitwise function.
- It takes the corresponding three bits in the buffers A, B and C.
- The resulting bit is the majority of the three bits.

- Ex : 0111, 1010 and 1110

- The first bits are 0,1,1 ; the majority is 1
- Second bits are 1,0,1 ; majority is 1
- Third bits are 1,1,1 ; majority is 1
- Fourth bits are 1,0,0 ; majority is 0
- The final result will be 1110

- The conditional function is also a bitwise function.
- It takes the corresponding three bits in the buffers.
- The resulting bit logic is “ If E_j then F_j , else G_j “

- Example : 1001,1010,1111

- The first bits are 1,1,1 ; $E_1 = 1$, so result is $F_1 = 1$
- Second bits are 0,0,1 ; $E_2 = 0$, so result is $G_2 = 1$
- Third bits are 0,0,1 ; $E_3 = 0$, so result is $G_3 = 1$
- Fourth bits are 1,0,1 ; $E_4 = 1$, so result is $F_4 = 0$

- The final result is 1110.

Eighty constants used for eighty rounds in SHA-512

428A2F98D728AE22	7137449123EF65CD	B5C0FBCFEC4D3B2F	E9B5DBA58189DBBC
3956C25BF348B538	59F111F1B605D019	923F82A4AF194F9B	AB1C5ED5DA6D8118
D807AA98A3030242	12835B0145706FBE	243185BE4EE4B28C	550C7DC3D5FFB4E2
72BE5D74F27B896F	80DEB1FE3B1696B1	9BDC06A725C71235	C19BF174CF692694
E49B69C19EF14AD2	EFBE4786384F25E3	0FC19DC68B8CD5B5	240CA1CC77AC9C65
2DE92C6F592B0275	4A7484AA6EA6E483	5CB0A9DCBD41FBD4	76F988DA831153B5
983E5152EE66DFAB	A831C66D2DB43210	B00327C898FB213F	BF597FC7BEEF0EE4
C6E00BF33DA88FC2	D5A79147930AA725	06CA6351E003826F	142929670A0E6E70
27B70A8546D22FFC	2E1B21385C26C926	4D2C6DFC5AC42AED	53380D139D95B3DF
650A73548BAF63DE	766A0ABB3C77B2A8	81C2C92E47EDAEE6	92722C851482353B
A2BFE8A14CF10364	A81A664BBC423001	C24B8B70D0F89791	C76C51A30654BE30
D192E819D6EF5218	D69906245565A910	F40E35855771202A	106AA07032BBD1B8
19A4C116B8D2D0C8	1E376C085141AB53	2748774CDF8EEB99	34B0BCB5E19B48A8
391C0CB3C5C95A63	4ED8AA4AE3418ACB	5B9CCA4F7763E373	682E6FF3D6B2B8A3
748F82EE5DEFB2FC	78A5636F43172F60	84C87814A1F0AB72	8CC702081A6439EC
90BEFFFA23631E28	A4506CEBDE82BDE9	BEF9A3F7B2C67915	C67178F2E372532B
CA273ECEEA26619C	D186B8C721C0C207	EADA7DD6CDE0EB1E	F57D4F7FEE6ED178
06F067AA72176FBA	0A637DC5A2C898A6	113F9804BEF90DAE	1B710B35131C471B
28DB77F523047D84	32CAAB7B40C72493	3C9EBE0A15C9BEBC	431D67C49C100D4C
4CC5D4BECB3E42B6	4597F299CFC657E2	5FCB6FAB3AD6FAEC	6C44198C4A475817

- SHA 512 is resistant to all attacks.

- More testing and research are needed to confirm this claim.