Cryptographic Hash Functions

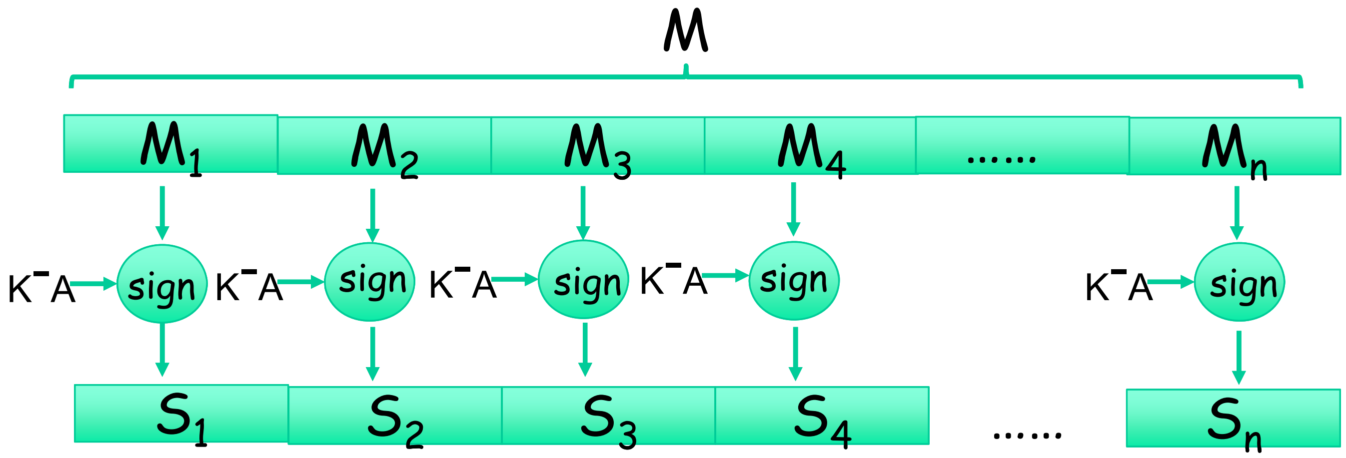
Contents

Hash function

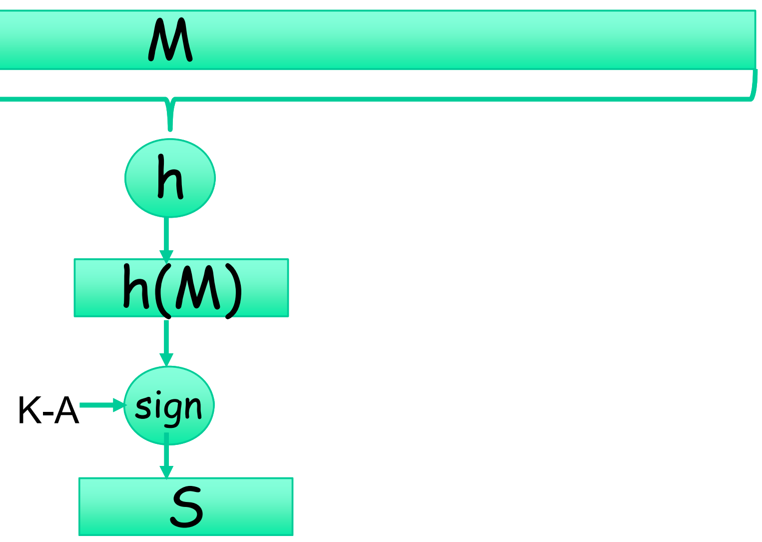
- Hash function is important tool in crypto for the following applications:
  - Digital signature
  - Message authentication code(MAC)
  - Key derivation
  - (crypto) random number generation, etc
- Notice:
  - Hash is not for encryption/decryption.
  - No keys
- Main motivation is for digital signature

Hash Function Motivation

- Suppose Alice signs *M*
  - Alice sends *M* and $S = sign(M)_{K^-_{Alice}}$ to Bob
  - Bob verifies that $M = verify(S)_{K^+_{Alice}}$
- If *M* is big, $sign(M)_{K\text{-}Alice}$ costly to ***compute*** & ***send***



- Instead, suppose Alice signs h(M), where h(M) is much smaller than M
  - Alice sends M and S = $\text{sign}(h(M))_{K\text{-}Alice}$ to Bob
  - Bob verifies that h(M) = $\text{verify}(S)_{K+Alice}$

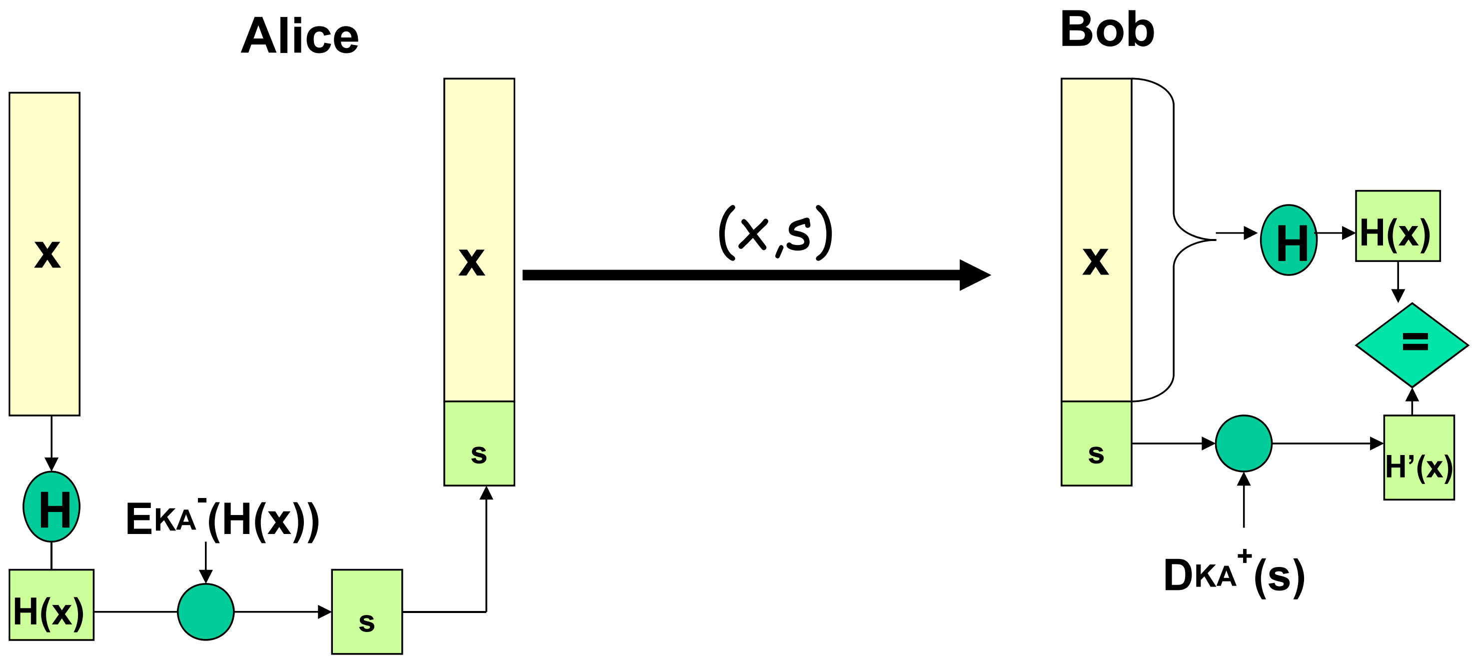


Hash Function Motivation

- So, Alice signs h(M)
  - Alice computes S=sign(h(M)) $_{K^-_{Alice}}$
  - Alice then sends (M, S) to Bob
  - Bob verifies that h(M) = verify(S) $_{K^+_{Alice}}$
- What properties must h(M) satisfy?
  - Suppose Trudy finds M’ so that h(M) = h(M’)
  - Then Trudy can replace (M, S) with (M’, S)
- Does Bob detect this tampering?
  - No, since h(M’) = h(M) = verify(S) $_{K^+_{Alice}}$

Hash function for public key digital signature

- Hash function provides the fast way of generating the digital signature using public key crypto.



Crypto Hash Function

- Crypto hash function h(x) must provide
  - **Arbitrary length** of input message
  - **Compression** : output length is small and fixed
  - **Efficiency** : h(x) is easy to compute for any x
  - **One-way** : given a value y it is infeasible to find an x such that h(x) = y
  - **Weak collision resistance** : given x and h(x), infeasible to find $y \neq x$ such that h(y) = h(x)
  - **Strong collision resistance** : infeasible to find ***any*** x and y, with $x \neq y$ such that h(x) = h(y)
- Lots of collisions exist, but should be hard to find any collision

Pre-Birthday Problem

- Suppose N people in a room
- How large must N be before the probability someone has the same birthday as me is $\geq 1/2$?
  - Solve: $1/2 = 1 - (364/365)^N$ for N
  - We find $N = 253$

Birthday Problem

- How many people must be in a room before probability is $\geq 1/2$ that any two (or more) have the same birthday?
  - Prob(same birthday) = $1 - 365/365 \cdot 364/365 \cdot \cdot \cdot (365-N+1)/365$
  - Set equal to $1/2$ and solve: **$N = 23$** $\approx \sqrt{365}$
- Surprising? A paradox?
- Maybe not: it should be about $\sqrt{(365)}$ since we compare all **pairs** x and y
  - And there are 365 possible birthdays

Strong collision resistance and Birthdays

- If h(x) is N bits, 2^N different hash values are possible
- So, if you hash about 2^N random values, then you expect to find a collision since $\mathrm{sqrt}(2^N) = 2^{N/2}$
- **Implication:** secure N bit symmetric key requires 2^{N-1} work to "break" while secure N bit hash function requires $2^{N/2}$ work to "break" assuming exhaustive search attacks.

# of hash values needed for a collision

Prob for at least one collision	Hash output length				
	128 bits	160 bits	256 bits	284 bits	512 bits
0.5	2^{55}	2^{81}	2^{129}	2^{193}	2^{237}
0.9	2^{57}	2^{82}	2^{130}	2^{194}	2^{258}

Secure hash function design

- There are many non-crypto hash functions (eg, Cyclic Redundancy Check). But they are not secure.
- Desired property: **avalanche effect**
  - Change to 1 bit of input should affect about half of output bits
- Crypto hash functions consist of some number of rounds
- Want security and speed
  - Avalanche effect after few rounds
  - But simple rounds
- Analogous to design of block cipher

Hash function algorithms

- block cipher design
- Dedicated Hash function
  - MD4 family
    - MD5
    - SHA-1
    - SHA-2
  - SHA-3
  - others

MD4 Family hash functions

algorithm		Output (bits)	Input (bits)	# of rounds	Collision found
MD5		128	512	64	yes
SHA-1		160	512	80	Not yet
SHA-2	SHA-224	234	512	64	no
	SHA-256	256	512	64	no
	SHA-384	384	1024	80	no
	SHA-512	512	1024	80	no

(source: Understanging Cryptography)

Popular hash function algorithms

- SHA-1
  - Developed by NIST and published in 1993
  - Input: max. length of less than 2^{64}bits
    - Input is processed in 512 bits blocks.
  - Output: 160 bits hash code
- MD5
  - RFC 1321
  - Input: arbitrary length, output: 128 bits
- RIPEMD-160
  - Developed by European RACE Integrity Primitives Evaluation (RIPE) project
  - Input: arbitrary length, output: 160 bits

How secure is SHA-1?

- SHA-1 does not provide collision resistance any more: requires only 2^{69} operations to find a hash collision(2005)
- How long would it take to find collision?
  - $2^{69} / (2^{20} * 2^{20}) = 2^{29}$ seconds
  - 1 year has approximately 2^{25} seconds
  - $2^{29} / 2^{25} \sim 16$ years

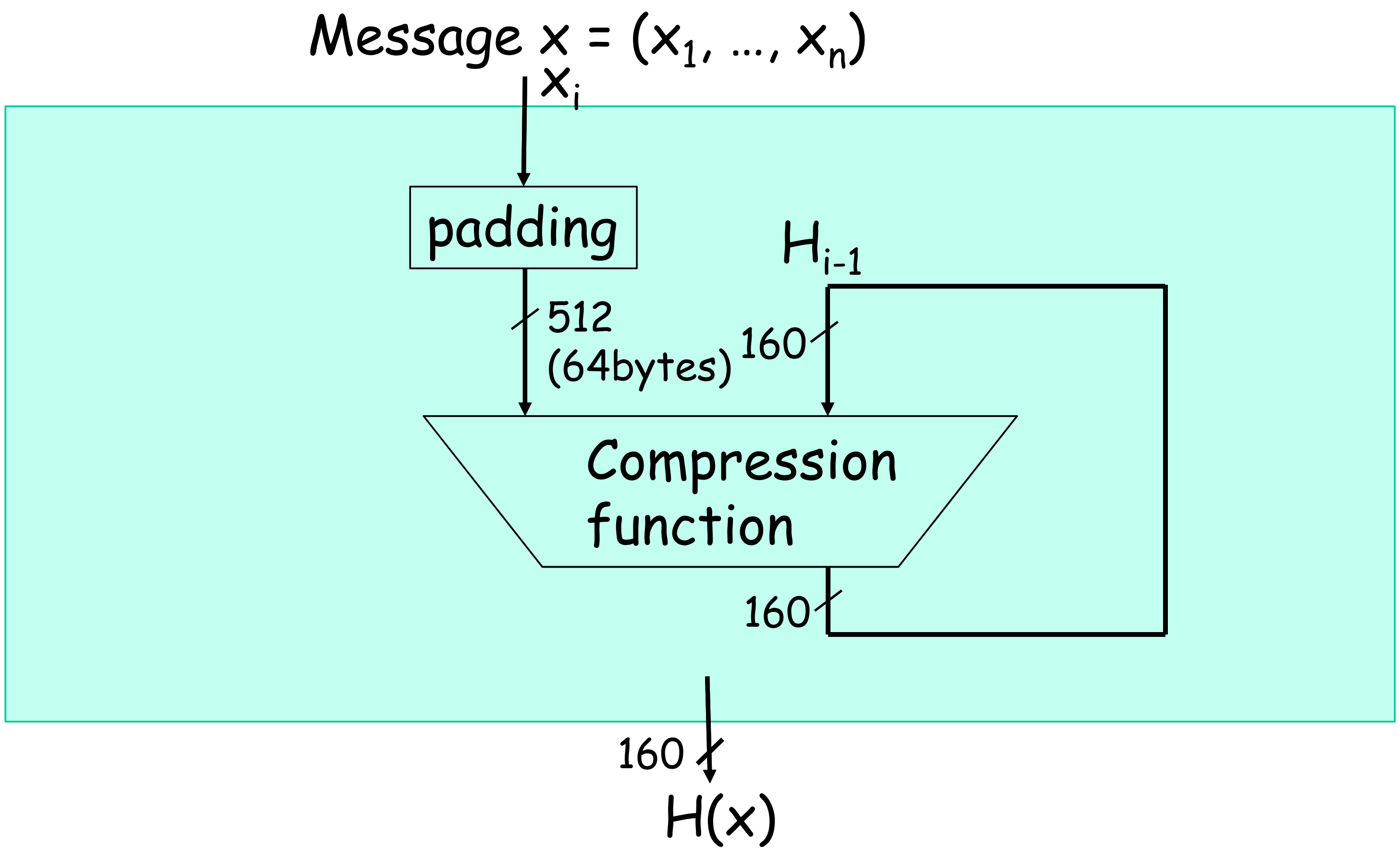
Earlier this week, three Chinese cryptographers showed that SHA-1 is not collision-free. That is, they developed an algorithm for finding collisions faster than brute force.

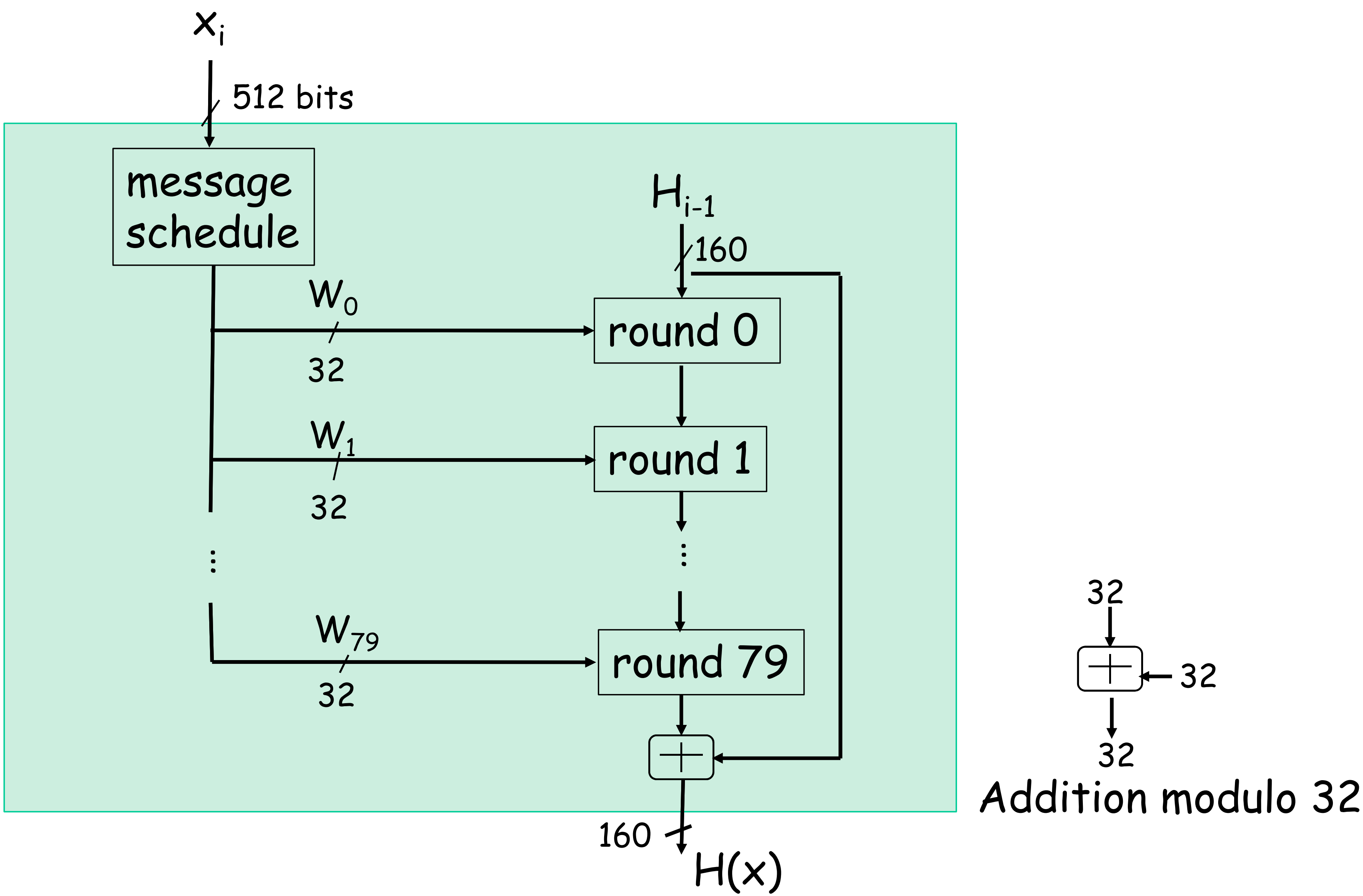
SHA-1 produces a 160-bit hash. That is, every message hashes down to a 160-bit number. Given that there are an infinite number of messages that hash to each possible value, there are an infinite number of possible collisions. But because the number of possible hashes is so large, the odds of finding one by chance is negligibly small (one in 2^{80}, to be exact). If you hashed 2^{80} random messages, you'd find one pair that hashed to the same value. That's the "brute force" way of finding collisions, and it depends solely on the length of the hash value. "Breaking" the hash function means being able to find collisions faster than that. And that's what the Chinese did.

They can find collisions in SHA-1 in 2^{69} calculations, about 2,000 times faster than brute force. Right now, that is just on the far edge of feasibility with current technology. Two comparable massive computations illustrate that point.

(Feb. 15, 2005. Bruce Schneier)

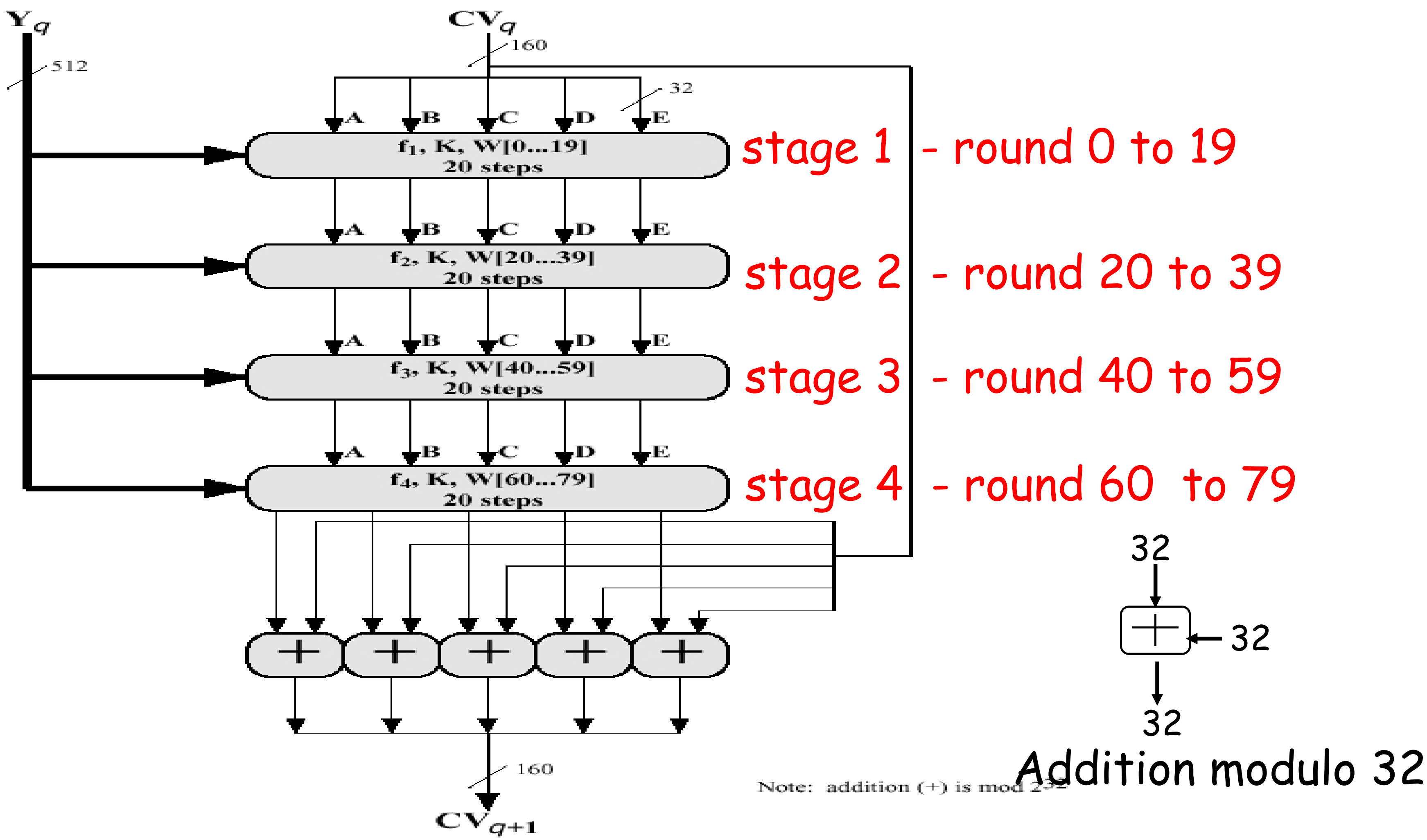
SHA-1 conceptual model



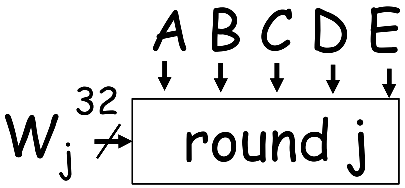


x_i
512 bits
message schedule
H_{i-1}
160
W_0
32
round 0
W_1
32
round 1
W_{79}
32
round 79
160
H(x)
32
32
32
Addition modulo 32

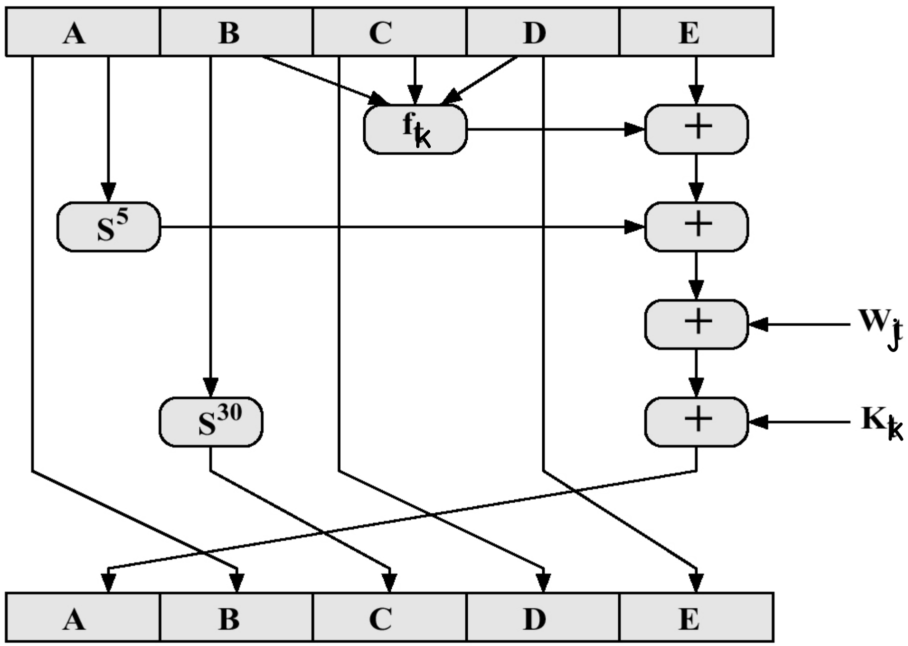
There are 4 stages, each stage has 20 rounds(steps).



Each round has 5 X 32 bits input (A,B,C,D,E) & W_j.

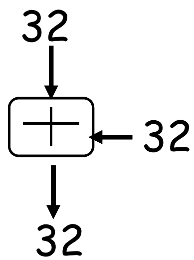


stage k, round j



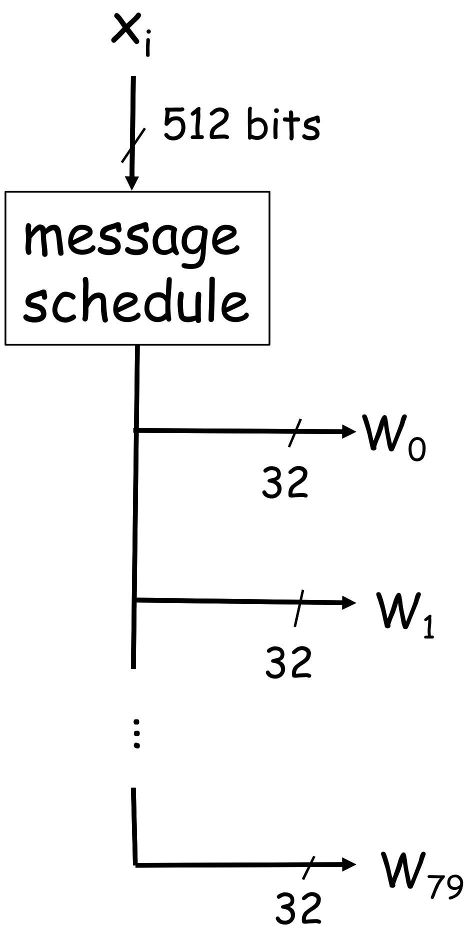
f_k: functions f_1, f_2, f_3,f_4
K_k: round constants K_1,K_2,K_3,K_4

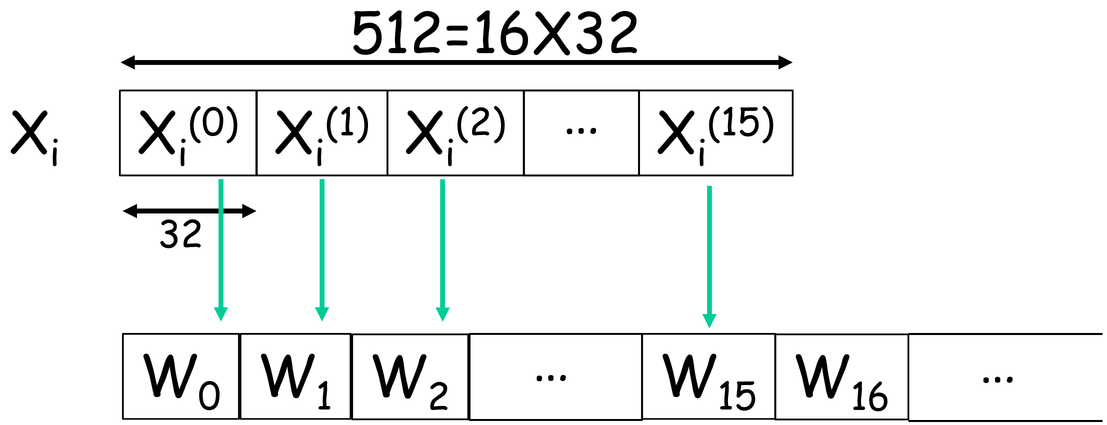
S^n: left shift n bits



Addition modulo 32

Message schedule





$$W_j = W_{j-16} \oplus W_{j-14} \oplus W_{j-8} \oplus W_{j-3}$$

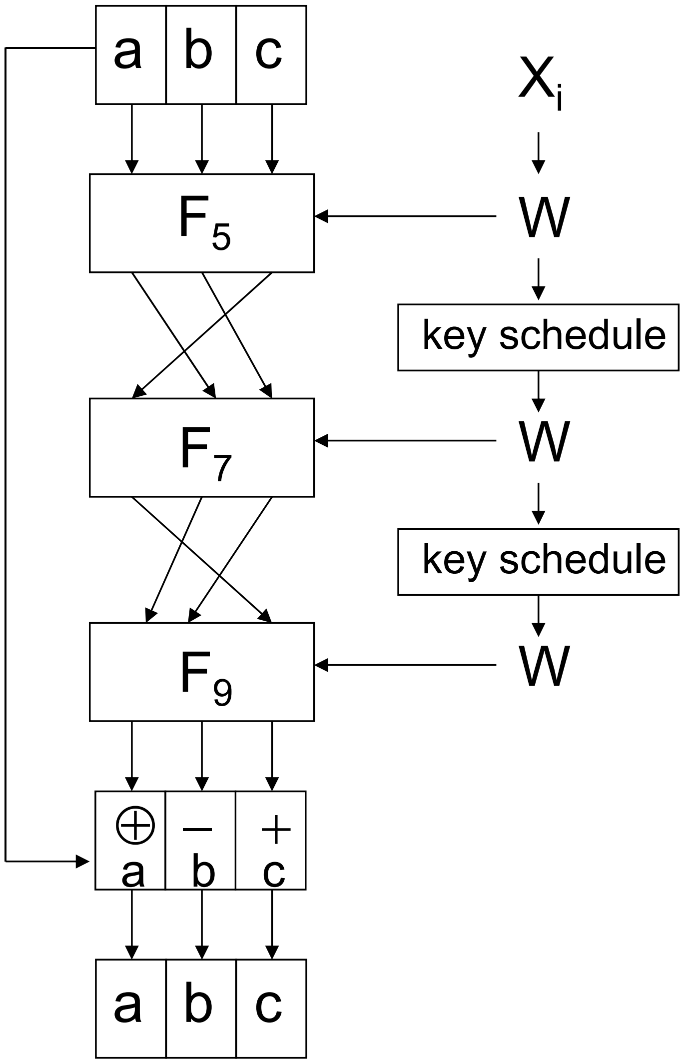
$$16 \leq j \leq 79$$

Tiger Hash

- “Fast and strong”
- Designed by Ross Anderson and Eli Biham — leading cryptographers
- Design criteria
  - Secure
  - Optimized for **64-bit** processors
  - Easy replacement for MD5 or SHA-1

Tiger Hash

- Like MD5/SHA-1, input divided into 512 bit blocks (padded)
- Unlike MD5/SHA-1, output is **192 bits** (three 64-bit words)
  - Truncate output if replacing MD5 or SHA-1
- Intermediate rounds are all 192 bits
- 4 S-boxes, each maps 8 bits to 64 bits
- A "key schedule" is used

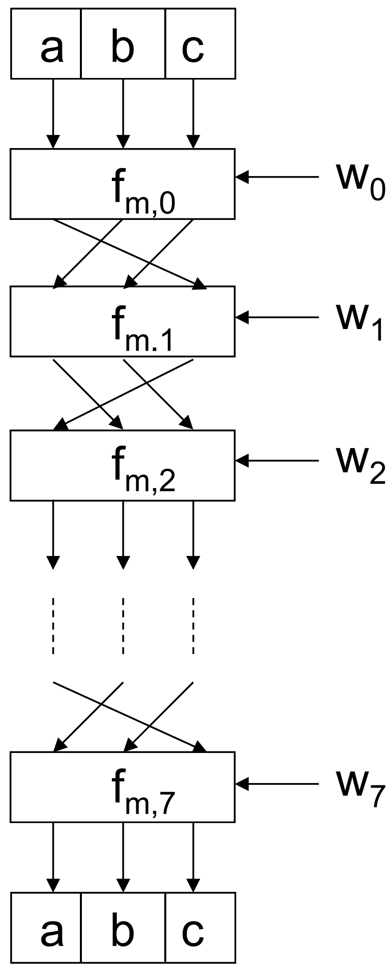


Tiger Outer Round

- Input is X
  - $X = (X_0, X_1, \ldots, X_{n-1})$
  - X is padded
  - Each X_i is 512 bits
- There are n iterations of diagram at left
  - One for each input block
- Initial (a,b,c) constants
- Final (a,b,c) is hash
- Looks like block cipher!

Tiger Inner Rounds

- Each F_m consists of precisely **8 rounds**
- 512 bit input W to F_m
  - $W=(w_0,w_1,\ldots,w_7)$
  - W is one of the input blocks X_i
- All lines are 64 bits
- The $f_{m,i}$ depend on the S-boxes (next slide)



Tiger Hash: One Round

- Each $f_{m,i}$ is a function of a,b,c,w_i and m
  - o Input values of a,b,c from previous round
  - o And w_i is 64-bit block of 512 bit W
  - o Subscript m is multiplier
  - o And $c = (c_0,c_1,\ldots,c_7)$
- Output of $f_{m,i}$ is
  - o $c = c \oplus w_i$
  - o $a = a - (S_0[c_0] \oplus S_1[c_2] \oplus S_2[c_4] \oplus S_3[c_6])$
  - o $b = b + (S_3[c_1] \oplus S_2[c_3] \oplus S_1[c_5] \oplus S_0[c_7])$
  - o $b = b * m$
- Each S_i is **S-box**: 8 bits mapped to 64 bits

Tiger Hash Key Schedule

- Input is X
  - o $X=(x_0,x_1,\ldots,x_7)$
- Small change in X will produce large change in key schedule output

$$
\begin{aligned}
x_0 &= x_0 - (x_7 \oplus \text{0xA5A5A5A5A5A5A5A5}) \\
x_1 &= x_1 \oplus x_0 \\
x_2 &= x_2 + x_1 \\
x_3 &= x_3 - (x_2 \oplus ((\sim x_1) \ll 19)) \\
x_4 &= x_4 \oplus x_3 \\
x_5 &= x_5 + x_4 \\
x_6 &= x_6 - (x_5 \oplus ((\sim x_4) \gg 23)) \\
x_7 &= x_7 \oplus x_6 \\
x_0 &= x_0 + x_7 \\
x_1 &= x_1 - (x_0 \oplus ((\sim x_7) \ll 19)) \\
x_2 &= x_2 \oplus x_1 \\
x_3 &= x_3 + x_2 \\
x_4 &= x_4 - (x_3 \oplus ((\sim x_2) \gg 23)) \\
x_5 &= x_5 \oplus x_4 \\
x_6 &= x_6 + x_5 \\
x_7 &= x_7 - (x_6 \oplus \text{0x0123456789ABCDEF})
\end{aligned}
$$

Tiger Hash Summary (1)

- Hash and intermediate values are 192 bits
- 24 (inner) rounds
  - **S-boxes:** Claimed that each input bit affects a, b and c after 3 rounds
  - **Key schedule:** Small change in message affects many bits of intermediate hash values
  - **Multiply:** Designed to ensure that input to S-box in one round mixed into many S-boxes in next
- S-boxes, key schedule and multiply together designed to ensure strong **avalanche** effect

Tiger Hash Summary (2)

- Uses lots of ideas from block ciphers
  - S-boxes
  - Multiple rounds
  - Mixed mode arithmetic
- At a higher level, Tiger employs
  - Confusion
  - Diffusion

Hash Uses

- Authentication (HMAC)
- Message integrity (HMAC)
- Message fingerprint
- Efficient digital signature
- Almost anything you can do with symmetric crypto
- Also, many, many clever/surprising uses…

Hash function use: Online Bids

- Suppose Alice, Bob and Charlie are bidders
- Alice plans to bid A, Bob B and Charlie C
- They don't trust that bids will stay secret
- A possible solution?
  - Alice, Bob, Charlie submit **hashes** h(A), h(B), h(C)
  - All hashes received and posted online
  - Then bids A, B, and C submitted and revealed
- Hashes don't reveal bids (one way)
- Can't change bid after hash sent (collision)
- But there is a flaw here…

Hash function use: Spam Reduction

- Spam reduction
- Before accept email, want proof that sender spent effort to create email
  - Here, effort == CPU cycles
- Goal is to limit the amount of email that can be sent
  - This approach will not eliminate spam
  - Instead, make spam more costly to send

Spam Reduction

- Let M = email message
  R = value to be determined
  T = current time
- Sender must find **R** so that
  h(M,**R**,T) = (00…0,X), where
  N initial bits of hash value are **all zero**
- Sender then sends (M,**R**,T)
- Recipient accepts email, provided that…
  h(M,**R**,T) begins with N zeros

Spam Reduction

- Sender: h(M,R,T) begins with N zeros
- Recipient: verify that h(M,R,T) begins with N zeros
- **Work for sender:** about **2^N hashes**
- **Work for recipient:** always **1 hash**
- Sender's work increases exponentially in N
- Small work for recipient regardless of N
- Choose N so that…
  - Work acceptable for normal email users
  - Work is too high for spammers