Preimage attack on TCS_SHA-3

Research Paper Review

Group 3

Ishita Gupta   Sri Rajitha
Rohith Krishna   Yashraj Varma
Aditya Kedar Tata   Akshat Mehra
Roopchand Parise   Parikshit Powani

Madras School of Economics

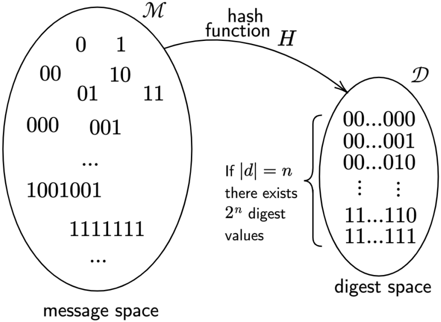
20 April 2021

Outline

1. Introduction to Hash Functions
2. Construction of TCS_SHA-3
3. Compression Function in TCS_SHA-3
4. Poor Diffusion & Differential input characteristic
5. Algorithm for Preimage attack

Hash Functions

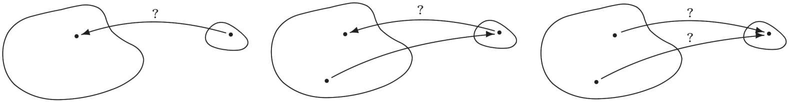
- A cryptographic hash function takes a message M of arbitrary length and outputs a bit string h of fixed length.
- Hash functions are used in:
  - Password Protection, Message Authentication, ...
  - Digital Signatures, Public Key Cryptography, ...



Properties of good hash functions

A cryptographic hash function must satisfy certain conditions:

- **Preimage Resistance** : Given the hash h of an unknown message, it should be computationally infeasible to find preimage M such that $H(M) = h$.
- **Second Preimage Resistance** : Given a particular message M and its hash $H(M)$, it should be computationally infeasible to find another distinct message M' having the same hash, $H(M) = H(M')$, such that $M \neq M'$.
- **Collision Resistance**: It is computationally infeasible to find any M and $M' \in \mathcal{M}$, such that $H(M) = H(M')$, and $M \neq M'$.



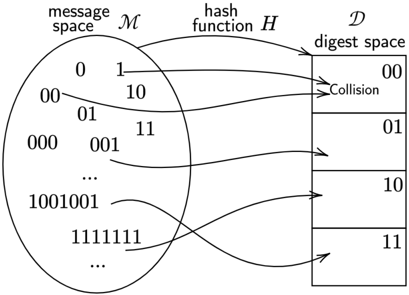
(a) Preimage (b) Second Preimage (c) Collision

Preimage and Second Preimage Resistance

- A preimage refers to a message $M \in \mathcal{M}$ that maps to a known hash $h \in \mathcal{D}$. The basic underlying assumption here is that there exists at least one message that hashes to the given value.
- One way of finding preimages is through a *exhaustive search*, wherein we hash random messages until the given hash value is reached. For a *n-bit* hash value, the number of random messages that must be tried is at least 2^n.
- A second preimage is a message that hashes to the same value as a randomly selected first preimage. The basic assumption here is that the attacker possesses the hash value of the first preimage.
- Thus second preimage resistance points out that given a randomly selected preimage, it is computationally infeasible to find another message that has an identical hash value.

Collision Resistance

- A collision refers to the case when a pair of distinct messages $M, M' \in \mathcal{M}$ have the same hash h.
- This follows from the pigeon-hole principle that if p items are put in q containers with $p > q$, then there must be at least one container with more than 1 item (thus collision occurs).
- Since the message space is larger than digest space, in fact $\mathcal{D} \subseteq \mathcal{M}$, collision is bound to occur.
- The challenge for the designer is to ensure that the adversary would find it infeasible to find a collision.



Outline

1. Introduction to Hash Functions
2. Construction of TCS_SHA-3
3. Compression Function in TCS_SHA-3
4. Poor Diffusion & Differential input characteristic
5. Algorithm for Preimage attack

Introduction to TCS_SHA-3

- TCS_SHA-3 is a family of four cryptographic hash functions proposed by Vijayarangan of the Tata Consultancy Services (TCS).
- It is a product of the E-Security group of the TCS Innovation Labs, Hyderabad, India and is covered by a US patent (US 2009/0262925).
- It produces a digest size of 224, 256, 384 and 512 bits. .
- TCS SHA-3-d denotes the member that produces d-bit digests.
- The design of TCS_SHA-3 deviates from the general model such that the standard compression function is replaced by a bijective function. The function uses a linear feedback shift register (LFSR) and a T-function.
- The design goals of TCS_SHA-3 include preventing hash collisions and providing a secure hash function.

Contributions of the paper

The paper establishes the inability of TCS_SHA-3 to meet the design goals of 'preventing hash collisions' and 'providing a secure hash function' by demonstrating the following attacks:

1. A second pre-image attack that requires negligible time and negligible memory for nearly guaranteed success.
2. A first pre-image attack on the TCS_SHA-3-d that requires $O(2^{27}.d)$ time and negligible memory.
3. A second pre-image attack that also requires negligible time and negligible memory for nearly guaranteed success on a strengthened variant of TCS_SHA-3.

Notations

- $\mathcal{M}$: Message Space
- $\mathcal{D}$: Digest Space
- M: An arbitrary length bit string, $M \in \mathcal{M}$
- H: A hash function
- h: A fixed length bit string called hash value or digest.
- d: Length of a block. $d \in \{224, 256, 384, 512\}$
- k: Number of partitioned message blocks $= \lceil |M|/d \rceil$
- F: A bijective function
- M^*: Partitioned message blocks after padding.
- c: a $d-$bit constant.

Notations

- LSB: Least significant bit
- MSB: Most significant bit
- $\Gamma_i(\omega)$: i^{th} 32-bit word ($i = 0$ denotes the least significant word of $d-$bit ω
- $|x|$: length of x in bits
- $x_{(i)}$: i^{th} bit ($i = 0$ denotes the LSB) of x
- $x||y$: concatenation of two 32-bit words, x and y
- $\oplus$: XOR (exclusive OR)
- α_i: 32-bit words - blocks of the original message M
- z_l^j: The output at the end of the l^{th} step of the j^{th} round.

Specification of TCS_SHA-3

- The **digest length**, for TCS_SHA-3 are $d \in \{224, 256, 384, 512\}$ as specified earlier.
- The **padding rule** for TCS_SHA-3 can be defined as follows: for any $k \geq 1$,

$$M_k \to M_k^* := \begin{cases} M_k \oplus IV & \text{if } |M_k| < d, \\ M_k & \text{if } |M_k| = d \end{cases}$$

- Here, $k = \lceil \frac{|M|}{d} \rceil$ where k is the **number of partitioned blocks** and the ceiling function denotes that if M is not a multiple of d then we will choose the next integer following M/d.
- The **IV rule** for TCS_SHA-3 is,

$$IV = 1||\{0\}^{d-1}$$

Example, if $d = 224$, the IV Rule is $1\{0\}^{223}$ which can be written as,

$$1\,\underbrace{000\ldots00}_{223 \text{ zeroes}}$$

Thus, forming a 224 bit string.

Construction of TCS_SHA-3

Each round in TCS_SHA-3 has k steps where k is the number of blocks in the message (after padding).

- **Round 1:**
  - Step 1: $k \geq 1$ Choose a constant c of length d' where $\{d' \leq d\}$. Perform $c \oplus M_1$ which becomes the input for the bijective function F.

    $$c \oplus M_1^* \xrightarrow{\text{input}} F \implies F(c \oplus M_1^*) \text{ is the output.}$$

  - Step $2 - k$: For each step i, $2 \leq i \leq k-1$, use recursion $z_i^1 = F(z_{i-1}^1 \oplus M_i)$. Thus, for step k, we will have,

    $$z_k^1 = F(z_{k-1}^1 \oplus M_k^*)$$

- **Round 2:** The number of steps in Round 2 is s which may or may not be equal to k.
  - Step 1: We XOR c, a $d-$ bit constant with the output of the previous round z_k^1. Thus, after applying the bijective function F on this input, the final output for this step will be,

    $$F(c \oplus z_k^1)$$

- Step $2 - s$: For each step i, $2 \leq i \leq s$, we have,

$$z_i^2 = F(z_{i-1}^2 \oplus z_k^1)$$

  z_l^2 denotes the output of step l, $1 \leq l \leq s$, of round 2 and z_{i-1}^2 is the output for the previous step within round 2.

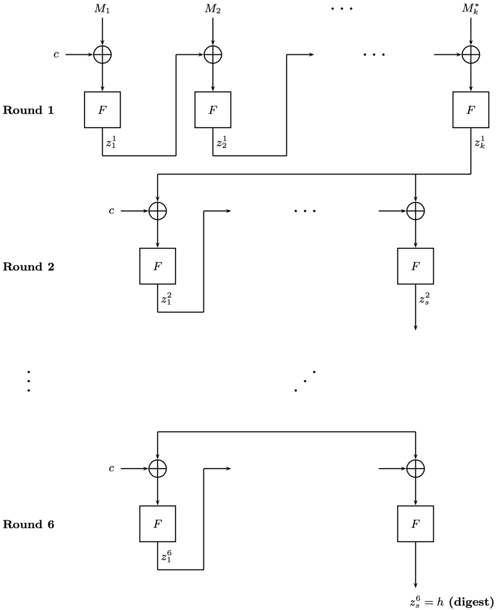
- **Round 3-6:** The number of steps in these rounds is again s. Similar to round 2, the output of the previous round is XORed with the output of the previous step.
  - Step 1: Constant c is XORed with the output of previous round z_s^{j-1}. This forms the input to the bijective function F. A $d-$bit string, $F(c \oplus z_s^{j-1})$ is generated as output.
  - Step $2 - s$: These steps are given by the following recursion,

$$z_i^j = F(z_{i-1}^j \oplus z_s^{j-1}), \qquad z_1^j = F(c \oplus z_s^{j-1})$$

  z_l^j denotes the output of step l of round j.

- The $d-$bit digest h is simply the final output z_s^6.

Construction of TCS_SHA-3



Outline

1. Introduction to Hash Functions
2. Construction of TCS_SHA-3
3. Compression Function in TCS_SHA-3
4. Poor Diffusion & Differential input characteristic
5. Algorithm for Preimage attack

Compression Function F

- The compression funtion in TCS_SHA-3 is bijective.
  $F : \{0,1\}^d \to \{0,1\}^d$ i.e. it has the same input and output space.

$$F(\alpha) = \lambda \text{ with } |\alpha| = |\lambda| = d$$

- We require a d-bit input α and must ensure a d-bit output λ. Algorithm follows these 5 steps :
  1. **Partition**: $\alpha \to \alpha_1 || \alpha_2 || ... || \alpha_{d/32}$ such that $|\alpha_i| = 32$ for all $1 \leq i \leq d/32$ where α is a d bit input.
  2. **Shuffling**: $\alpha_i \to \beta_i,$ for all $1 \leq i \leq d/32$, such that

$$\beta_{i(j)} = \begin{cases} \alpha_{i(j/2)} & \text{if } 2|j \\ \alpha_{i(16+(j-1)/2)} & \text{otherwise;} \end{cases}$$

  3. **T-function** : $\beta_i \to \gamma_i$ where $\gamma_i = 2\beta_i^2 + \beta_i \mod 2^{32}$, for all $1 \leq i \leq d/32$
  4. **LFSR**: $\gamma_i \to \lambda_i$, for all $1 \leq i \leq 32$, such that $|\lambda_i| = 32$
  5. **Concatenate**: $\lambda := \lambda_1 || \lambda_2 || ... || \lambda_{d/32}$

Partition Function

- The given message is divided into a sequence of equal sized words and the total number of words is m.
- The size of each word is chosen to be 32 bits, where $m = \{7, 8, 12, 16\}$ depending upon the digest length, $d = \{224, 256, 384, 512\}$ bits.
- And $\alpha \rightarrow \alpha_1 || \alpha_2 || \dots || \alpha_{d/32}$ such that $|\alpha_i| = 32$ for all $1 \leq i \leq d/32$ where α is a d-bit input.

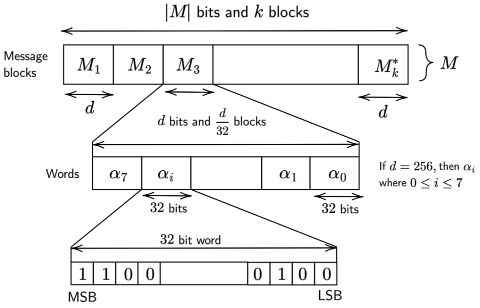


Figure: Partitioning the Message

Shuffle Function

- Shuffling of bits helps to improve the diffusion of the input across different parts of system.
- Shuffle function S, shuffles the bits in the 32-bit words i.e., within a partition of the message.
- The shuffling procedure used is an *outer perfect shuffle*, which means the outer (end) bits remain in the outer positions.
- $\alpha_i \to \beta_i,$ for all $1 \leq i \leq d/32$, such that

$$\beta_{i(j)} = \begin{cases} \alpha_{i\left(\frac{j}{2}\right)} & \text{if } 2|j \\ \alpha_{i\left(16+\frac{j-1}{2}\right)} & \text{otherwise} \end{cases}$$

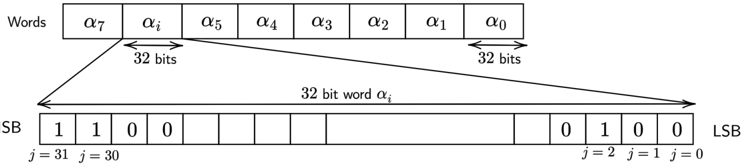


Figure: Word wise shuffle

Shuffle Function

- Suppose $j = 4$, then

$$j \bmod 2 = 0 \implies \beta_{3(4)} = \alpha_{3(4/2)} = \alpha_{3(2)}$$

- Suppose $j = 7$, then

$$j \bmod 2 \neq 0 \implies \beta_{3(7)} = \alpha_{3(16+(7-1)/2)} = \alpha_{3(19)}$$

- The following table shows where each bit is mapped to:

α_i	0	1	2	3	4	5	6	7
β_i	0	16	1	17	2	18	3	19
α_i	8	9	10	11	12	13	14	15
β_i	4	20	5	21	6	22	7	23
α_i	16	17	18	19	20	21	22	23
β_i	8	24	9	25	10	26	11	27
α_i	24	25	26	27	28	29	30	31
β_i	12	28	13	29	14	30	15	31

Table: Shuffle Function mapping of bit-positions.

T Function

- T function is a bijective mapping that updates every bit of the state.
- Each bit of the state is updated by a linear combination of the same bit and a function of a subset of its less significant bits.
- If every single less significant bit is included in the update of every bit in the state, such a T-function is referred to as *triangular*.
- The output of shuffle function is passed into the T-function which performs an invertible mapping containing all the 2^n possible states on a single cycle for any word size n.
- T function: $\beta_i \to \gamma_i$ where

$$\gamma_i = 2\beta_i^2 + \beta_i \text{ mod } 2^{32}, \quad \forall \;\; 1 \leq i \leq d/32$$

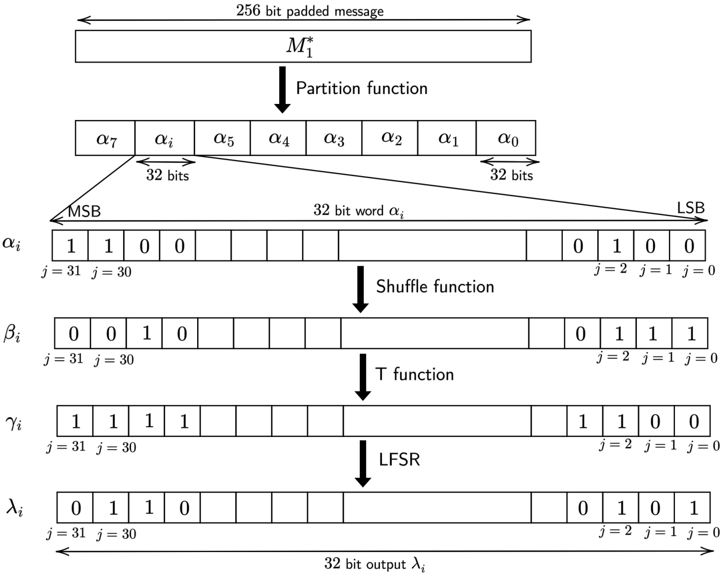
LFSR function

- The Linear Feedback Shift Register uses the connection polynomial,

$$\begin{aligned} f(x) &= x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} \\ &\quad + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + x + 1 \end{aligned}$$

- LFSR is an irreducible polynomial of degree 32 with period $2^{32} - 1$.
- Each time the 32-bit input is executed for different number of rounds. Thus, even if the input bits are identical, the output will appear random.
- After applying the T-function, the LFSR function is applied on the output of the T-function γ_i.
- LFSR function: $\gamma_i \rightarrow \lambda_i$, for all $1 \leq i \leq 32$, such that $|\lambda_i| = 32$
- Finally we concatenate the $\lambda_i's$ to obtain a d-bit output λ where $\lambda = \lambda_1||\lambda_2||...||\lambda_{d/32}$

Schematic of Bijective function F



Outline

1. Introduction to Hash Functions
2. Construction of TCS_SHA-3
3. Compression Function in TCS_SHA-3
4. Poor Diffusion & Differential input characteristic
5. Algorithm for Preimage attack

Diffusion in Block-Cipher based Hash Functions

- Diffusion in a block cipher means that flipping 1 bit of the plaintext, should ideally flip the bits of the ciphertext with probability 1/2. (Shannon)
- In other words a single bit flip in the plaintext should result in approximately half the ciphertext bits flipped.
- When it is exactly half, it is called the Strict Avalanche Criterion (as named by Feistel).
- MD5 has good diffusion properties. A single change in input string results in significantly different digests.

Message	Hash Value
0000	4a7d1ed414474e4033ac29ccb8653d9b
0001	25bbdcd06c32d477f7fa1c3e4a91b032
0010	fc1198178c3594bfdda3ca2996eb65cb
0011	ae2bac2e4b4da805d01b2952d7e35ba4

Table: MD5 hash values in hexadecimal

First Preimage attack on TCS_SHA-3-256

- Consider the TCS_SHA-3 variant with $d = 256$ and $k = 1$ (single block), that is message of size $|M| \leq d$.
- Since hash functions are often used to store user passwords, this assumption is fairly practical.
- It can be shown that TCS_SHA-3-256 has poor diffusion property.
- That is a flip of 1-bit of message does not result in approximately 128 bits of digest flipped.
- Thus weakness is exploited by and the result of a differential characteristic of the input on the hash is explored.
- It is found that ideal worst case complexity of $O(2^{256})$ is reduced drastically to $O(2^{35})$. Likewise for digest size $d = 512$, the worst case complexity is $O(2^{36})$.

Exploiting Poor Diffusion

- For $k = 1$ (single block input), the bijective function F first partitions the message into $d/32 = 256/32 = 8$ words, each of size 32-bits.
- For each 32-bit word α_i input to function F we get a pseudo-random 32-bit output λ_i.
- The output λ_i depends only on α_i and not on any other $\alpha_u, u \neq i$.
- Thus the last 32-bits of the hash depends only on the least significant 32-bit word of M_1^*.

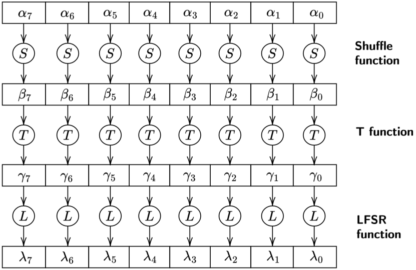


Figure: Diffusion in a 32-bit word is confined to that word alone.

Differential Characteristic of input

- Instead of exhaustively searching the 256-bit message space with 2^{256} elements for matching hash value, one can now search the 32-bit word space with 2^{32} elements for matching corresponding word in the hash.
- In general, the complexity for d-bit hash variant reduces from $O(2^d)$ to $O(2^{32})$, for a specified 32-bit word of the hash and for $k = 1$.
- Let $\Gamma_i(M_1^*)$ denote an arbitrary 32-bit word in the message M_1^*.
- Under exhaustive search, we start with 32-bit word $000\ldots000$ producing hash h_i^a and $000\ldots001$ producing hash h_i^b respectively.
- Difference in the input $\Delta\Gamma_i(M_1^*)$ is now 1-bit. Say this produces a corresponding difference in hash of $\Delta h_i = h_i^b - h_i^a$.
- From the poor diffusion property, exhaustive search of 32-bit word is expected to uncover the portion of M_1^*, α_i corresponding to hash portion h_i.

Differential Characteristic of input

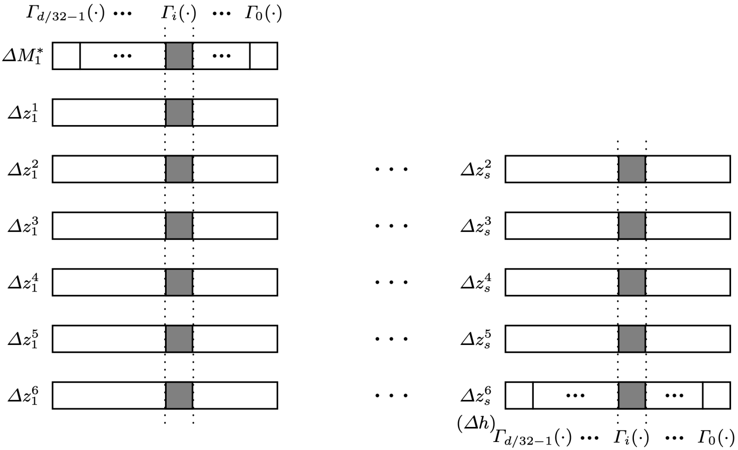


Figure: Differences in input $\Delta\Gamma_i(.)$ propagates only through 6 rounds and is confined to the chosen word i alone.

Outline

1. Introduction to Hash Functions
2. Construction of TCS_SHA-3
3. Compression Function in TCS_SHA-3
4. Poor Diffusion & Differential input characteristic
5. Algorithm for Preimage attack

Algorithm for M_1 recovery

Algorithm 2 Recovering M_1 from h when $k = 1$

Require: Whether $|M_1| = d$ or $|M_1| < d$
Ensure: d-bit output M_1

for $i = 0 \to d/32 - 1$ **do**
  for $j = \{0_2\}^{32} \to \{1_2\}^{32}$ **do**
    $\ell \leftarrow \{0_2\}^{32(d/32-i-1)} \| j \| \{0_2\}^{32i}$;
    Compute $\hbar :=$ TCS_SHA-3-$d(\ell)$;
    if $\mathit{\Gamma}_i(\hbar) = \mathit{\Gamma}_i(h)$ **then**
      $\mathit{\Gamma}_i(M_1^*) \leftarrow j$;
      break;
    else
      $j \leftarrow j + 1$;
  $i \leftarrow i + 1$;
Compute $M_1^* = \mathit{\Gamma}_{d/32-1}(M_1^*) \| \mathit{\Gamma}_{d/32-2}(M_1^*) \| \ldots \| \mathit{\Gamma}_0(M_1^*)$;
if $|M_1| < d$ **then**
  Output $M_1 = M_1^* \oplus$ IV;
else
  Output $M_1 = M^*$;

Algorithm for M_1 recovery

We fix $k = 1$, that is a single message block. We also require if $|M| = d$ or $|M| < d$, in order to reverse the IV rule.

Line 1. Loop i across each of 8 words for a 256 bit block. This takes time 8 units. Or in the general case $O(d/32)$.

Line 2. Loop j across each of the 2^{32} elements for the chosen 32-bit word.

Line 3. Compute 256-bit string l by prefixing and suffixing required number of zeros in all positions other than the chosen word.

$$l \leftarrow \{0^{32(\frac{d}{32}-1-i)}\}||j||\{0\}^{32i}$$

								$j \in \{0,1\}^{32}$
$i = 0$	00..00	00..00	00..00	00..00	00..00	00..00	00..00	j
$i = 1$	00..00	00..00	00..00	00..00	00..00	00..00	j	00..00
				$\vdots$				
$i = 6$	00..00	j	00..00	00..00	00..00	00..00	00..00	00..00
$i = 7$	j	00..00	00..00	00..00	00..00	00..00	00..00	00..00

Algorithm for M_1 recovery

Line 4 : Compute the hash $\hbar$ produced by l.

Line 5^{+}: If the corresponding 32-bit word of the hash $\Gamma_i(h)$ matches with the computed hash $\Gamma_i(\hbar)$, then assign j (32-bit word) to the corresponding part of the message $\Gamma_i(M_1^*)$. Else increment and go to the next 32-bit element (in loop 2).

Line 10: Increment i to the next word. Exit loop once search over all 8 words are complete.

Line 11: Concatenate all $\Gamma_i(M_1^*)$ to obtain 256-bit M_1^*.

$$M_1^* = \Gamma_{d/32-1}(M_1^*)||\Gamma_{d/32-2}(M_1^*)||\ldots||\Gamma_1(M_1^*)||\Gamma_0(M_1^*)$$

Line 12: If $|M| < d$, then the original message would have been padded using the IV rule. To reverse this, we XOR the obtained M_1^* with IV, that is

$$M_1 = \begin{cases} M_1^* \oplus IV & \text{if } |M_1| < d \\ M_1^* & \text{if } |M_1| = d \end{cases}$$

How many searches?

Consider the following scenario of a 8-bit message (to be split into multiple words.) The corresponding hash h is also 8 bits long (say).

M	00000000
h	10000110

Start with 00000000 and search all 2^8 possible elements in $\mathcal{M}$ such that $H(M) = h$.

256 searches!

M	0000	0000
h	1000	0110

Now, diffusion is confined to the chosen 4-bit space.
Start with 0000 and search all 2^4 elements in $\mathcal{M}$.
Repeat for second word $\implies$ $2^4 \times 2$

32 searches!

M	00	00	00	00
h	10	00	01	10

Now, diffusion is confined to the chosen 2-bit space.
Start with 00 and search all 2^2 elements in $\mathcal{M}$.
Repeat for other 3 words $\implies$ $2^2 \times 4$

16 searches!

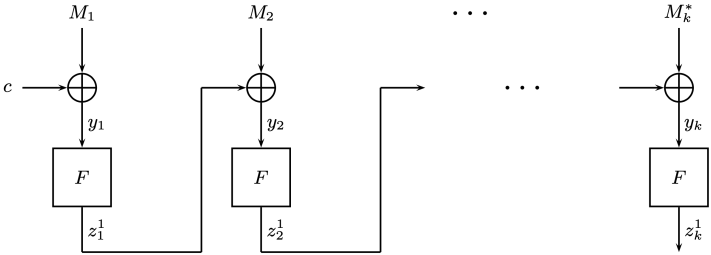
Reduction in Computational Complexity

- As seen previously, had there been good diffusion of bits across all 256-bit length of the block size, recovery would take $O(2^{256})$ time. And in general it would take $O(2^d)$ time.
- The reduction in complexity stems from the fact that diffusion is limited to 32-bit words.
- The time taken to run the two loops is as follows. The first loop runs over $d/32$ values. The second loop runs over 2^{32} elements. The total run time complexity is therefore:

$$O\left(\frac{d}{32} \times 2^{32}\right) = O\left(\frac{d}{2^5} \times 2^{32}\right) = O\Big(d \cdot 2^{27}\Big)$$

- For $d = 512$, it takes $O(2^{36})$ time and for $d = 256$, it takes $O(2^{35})$ time.

Second pre-image attack



- Let the message $M = M_1||M_2||M_3||....||M_k^*$ where $k \geq 2$.
- Let another message be $M' = M_1'||M_2'||M_3||....||M_k^*$ where M_1 and M_2 are not equal to M_1' and M_2', but the successive blocks are identical.
- If $y_2' = y_2$, we see that the outputs are identical, i.e. $h = h'$

What does $y = y'$ entail?

- The implication of $y = y'$ is as follows. We know that $y_2 = F(M_1 \oplus c) \oplus M_2$ and $y_2' = F(M_1' \oplus c) \oplus M_2'$
- The condition $y_2' = y_2$ therefore implies:

$$F(M_1 \oplus c) \oplus M_2 = F(M_1' \oplus c) \oplus M_2'$$

- If we assume the following forms of M_2' and M_1', that $y_2 = y_2'$ follows.

$$\begin{aligned} M_2' &= F(M_1' \oplus c) \oplus M_1' \\ M_1' &= F(M_1 \oplus c) \oplus M_2 \end{aligned}$$

- From the RHS of the $y_2 = y_2'$ condition we have:

$$\begin{aligned} F(M_1' \oplus c) \oplus M_2' &= \underbrace{F(M_1' \oplus c) \oplus \Big[F(M_1' \oplus c)}\oplus M_1'\Big] \\ &= 0 \oplus M_1' = M_1' \\ &= F(M_1 \oplus c) \oplus M_2 \end{aligned}$$

References

1. Sekar, Gautham, and Soumyadeep Bhattacharya. "Practical (Second) Preimage Attacks on TCS SHA-3." (2013).
2. Preneel, Bart. Analysis and design of cryptographic hash functions. Diss. Katholieke Universiteit te Leuven, 1993.
3. Thomsen, Søren Steffen, and Lars Ramkilde Knudsen. Cryptographic hash functions. Diss. PhD thesis, Technical University of Denmark, 2005.
4. Klimov, Alexander, and Adi Shamir. "Cryptographic applications of T functions." International Workshop on Selected Areas in Cryptography. Springer, Berlin, Heidelberg, 2003.