Study and Analysis of BLAKE2 Hash Function

Ajay Choudhury (18018)
Supervisor: Dr. Shashank Singh
Department of Electrical Engineering and Computer Science(EECS)
Indian Institute of Science Education and Research Bhopal, Madhya Pradesh, India

This project aims to study and analyse the security of the `BLAKE2` hash function, which is a tweaked version of the `BLAKE`, a `SHA3` competition finalist. We plan to implement the original version of `BLAKE-256` and `BLAKE2s-256` hash functions in C/C++ to analyse the working mechanism and identify the non-linear and redundant components in terms of complexity and security of hash functions. In the `BLAKE2`, designers managed to speed up the process to reach a hashing speed similar to the MD5 hash function. For this, they had to modify the original design. In this work, we plan to study and analyse these modifications concerning the security guarantee of the hash function.

1 Introduction

Hash functions are mathematical functions that take binary strings of arbitrary length as input and output strings of some fixed length. Mathematically, a hash function can be represented as:

$$h : \{0,1\}^{\infty} \to \{0,1\}^{n}\,, \tag{1}$$

where n is the number of bits the hash function produces as output, for example, the `MD5` [1] hash function produces 128-bit output, so for the `MD5` hash function, $n = 128$. In layman's terms, hashing is a process of scrambling and often compressing a piece of information/data beyond recognition. As it is evident from the Eq.(1) that the hash functions are many-to-one functions, i.e., there are many inputs which hash to the same output, theoretically. But when for a hash function, it is computationally hard to get the two elements which map to the same element, such hash functions are termed Cryptographic Hash Functions. This property of the hash function is called "Collision Resistant". A cryptographic hash function can be practically treated as an irreversible function with respect to the computational resources available. From now on, by hash function, we always mean a cryptographic hash function. The output of a hash function is usually called a hash or digest of the input value.

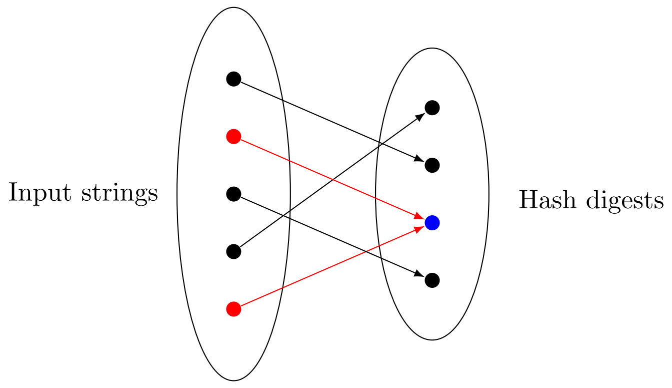


Figure 1: Many-to-one mapping of inputs to finite hash digests

1.1 Features of a cryptographic hash function

Cryptographic hash functions satisfy the following properties.

1. **Collision Resistant:** It is computationally hard to find two inputs (messages) with the same hash value. Let h be a hash function; then, it is hard to find two messages x_1 and x_2 such that $h(x_1) = h(x_2) = y$.

2. **Preimage Resistant:** It is computationally hard to find a message with a given hash value. Suppose h is a hash function, and $y \in \{0,1\}^n$, then, it should be hard to find an input x such that $h(x) = y$.

3. **Second Preimage Resistant:** Given one message, it should be hard to find another message with the same hash value. Let x_1 be any input string; it should be computationally hard to find another input string $x_2 \neq x_1$ such that $h(x_1) = h(x_2)$.

As the input to a hash function is a bit string of arbitrary length, it could be practically anything. In practice, designing functions of the infinite domain is impractical, hence usually, a compression function is designed which maps bit strings of length s into bit strings of length n for $s > n$ and then chains this in some way to produce a function on an infinite domain.

2 Description of BLAKE-256 Hash Function

The BLAKE [2] is one of the five finalists of the NIST hash function competition aimed to standardise the SHA-3 in 2012. It is a family of four hash functions: BLAKE-224, BLAKE-256, BLAKE-384, and BLAKE-512. The BLAKE-256 is the 32-bit version, and BLAKE-512 is the 64-bit version of BLAKE hash function. Other versions are derived using different initial values, padding, and truncated output. We provide the design and implementation details of BLAKE-256 hash function in the following. We use the notations given below for describing the design of BLAKE-256.

(i) '$\leftarrow$' denotes variable assignment;
(ii) '$+$' denotes addition in $\mathbb{Z}_{2^{32}}$ or in $\mathbb{Z}_{2^{32}}$ (modular addition);
(iii) '$\oplus$' denotes addition in $\mathbb{Z}_2^{32}$ or in $\mathbb{Z}_2^{64}$ (bitwise exclusive or);
(iv) '$\lll r$' denotes rotation of r bits towards the most significant bit;
(v) '$\ggg r$' denotes rotation of r bits towards the least significant bit;
(vi) '$\langle l \rangle_k$' denotes encoding of the integer l over k bits;

The BLAKE-256 hash function operates on 32-bit message blocks and produces a 32-byte hash value. The 8 constant initialization vectors in BLAKE-256 are assigned as follows:

$$\begin{array}{llll} \mathtt{IV}_0 = \mathtt{0x6A09E667}; & \mathtt{IV}_1 = \mathtt{0xBB67AE85}; & \mathtt{IV}_2 = \mathtt{0x3C6EF372}; & \mathtt{IV}_3 = \mathtt{0xA54FF53A} \\ \mathtt{IV}_4 = \mathtt{0x510E527F}; & \mathtt{IV}_5 = \mathtt{0x9B05688C}; & \mathtt{IV}_6 = \mathtt{0x1F83D9AB}; & \mathtt{IV}_7 = \mathtt{0x5BE0CD19} \end{array}$$

The BLAKE-256 also uses the following 16 word constants, defined as:

$$\begin{array}{llll} \mathtt{c}_0 = \mathtt{0x243F6A88}; & \mathtt{c}_1 = \mathtt{0x85A308D3}; & \mathtt{c}_2 = \mathtt{0x13198A2E}; & \mathtt{c}_3 = \mathtt{0x03707344} \\ \mathtt{c}_4 = \mathtt{0xA4093822}; & \mathtt{c}_5 = \mathtt{0x299F31D0}; & \mathtt{c}_6 = \mathtt{0x082EFA98}; & \mathtt{c}_7 = \mathtt{0xEC4E6C89} \\ \mathtt{c}_8 = \mathtt{0x452821E6}; & \mathtt{c}_9 = \mathtt{0x38D01377}; & \mathtt{c}_{10} = \mathtt{0xBE5466CF}; & \mathtt{c}_{11} = \mathtt{0x34E90C6C} \\ \mathtt{c}_{12} = \mathtt{0xC0AC29B7}; & \mathtt{c}_{13} = \mathtt{0xC97C50DD}; & \mathtt{c}_{14} = \mathtt{0x3F84D5B5}; & \mathtt{c}_{15} = \mathtt{0xB5470917} \end{array}$$

Let m be an input/message to the hash function with a bit length $l < 2^{64}$. The input m is first padded so that the length of the padded message becomes congruent to 447 (mod 512). The padding is done by appending a bit 1 followed by a sufficient number of 0 bits. The number of bits padded till now should at least be one bit and at most be 447 bits. Then a bit 1 is added again, followed by a 64-bit unsigned big-endian representation of l (the length of the message). All of the padded bits combined with the original message ensure that the length of the padded message is a multiple of 512. The padding in BLAKE can be denoted by the following notation.

$$\mathtt{m} \leftarrow \mathtt{m} \| 1000 \cdots 0001 \langle l \rangle_{64} \tag{2}$$

The next step is to split the padded message into blocks of 512 bits, each of which is further divided into 16 words, namely $\mathtt{m}^0, \ldots, \mathtt{m}^{\mathtt{N}-1}$. A 512-bit (16 words) initial state $\mathtt{v}_0, \ldots, \mathtt{v}_{15}$ is initialized and represented as a 4×4 array. It is initialized with the variables $h_0, \cdots, h_7$ also called the chain values (initially, $\mathtt{h}_0 = \mathtt{IV}_0, \cdots, \mathtt{h}_7 = \mathtt{IV}_7$), user-chosen salt values denoted by $\mathtt{s}_0, \cdots, \mathtt{s}_3$, counter variables $\mathtt{t}_0$ and $\mathtt{t}_1$ that count the numbers of bits hashed so far by the compression function and constants $\mathtt{c}_0, \cdots, \mathtt{c}_7$ such that different inputs produce different initial states. The initialisation of states can be represented as:

$$\begin{pmatrix} v_0 & v_1 & v_2 & v_3 \\ v_4 & v_5 & v_6 & v_7 \\ v_8 & v_9 & v_{10} & v_{11} \\ v_{12} & v_{13} & v_{14} & v_{15} \end{pmatrix} \leftarrow \begin{pmatrix} h_0 & h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 & h_7 \\ s_0 \oplus c_0 & s_1 \oplus c_1 & s_2 \oplus c_2 & s_3 \oplus c_3 \\ t_0 \oplus c_4 & t_0 \oplus c_5 & t_1 \oplus c_6 & t_1 \oplus c_7 \end{pmatrix} \tag{3}$$

Here, the salt s is user-defined and set to the null value when no salt is required or provided (i.e. $s_0 = s_1 = s_2 = s_3 = 0$). The initial state, message blocks, and other variables are then passed on to the round function. The round function iterates 14 times on the state v and uses a single core function G. The core function G operates on a 4-word column or diagonal (a, b, c, d), which is combined with two words of message m and two constant words c. The core function is called eight times per full round, i.e., a total of 112 times in 14 rounds. The operation is depicted in the Figure 2.

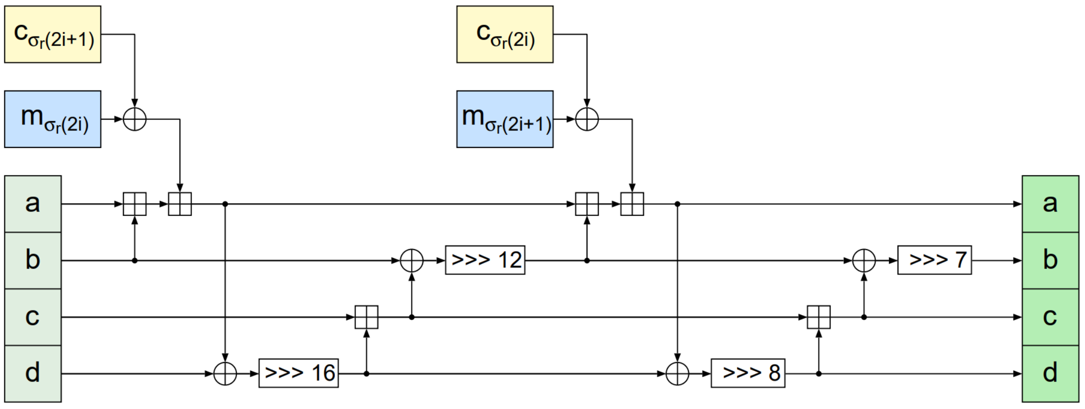


Figure 2: The core function G_i. (This image is taken from SHA3 Proposal BLAKE)

The first four calls $G_0, \cdots, G_3$ is called the column step (shown in Figure 3), and it can be shown as

$$G_0(v_0, v_4, v_8, v_{12}) \quad G_1(v_1, v_5, v_9, v_{13}) \quad G_2(v_2, v_6, v_{10}, v_{14}) \quad G_3(v_3, v_7, v_{11}, v_{15}),$$

Similarly, the last four calls $G_4, \ldots, G_7$ operate on diagonals and is called a diagonal step; it computes:

$$G_4(v_0, v_5, v_{10}, v_{15}) \quad G_5(v_1, v_6, v_{11}, v_{12}) \quad G_6(v_2, v_7, v_8, v_{13}) \quad G_7(v_3, v_4, v_9, v_{14}).$$

where, i varies from 0 to 7 and at round r, the function $G_i(a, b, c, d)$ sets:

```
1: $a \leftarrow a + b + (m_{\sigma_r(2i)} \oplus c_{\sigma_r(2i+1)})$      5: $a \leftarrow a + b + (m_{\sigma_r(2i+1)} \oplus c_{\sigma_r(2i)})$
2: $d \leftarrow (d \oplus a) \ggg 16$                                    6: $d \leftarrow (d \oplus a) \ggg 8$
3: $c \leftarrow c + d$                                                  7: $c \leftarrow c + d$
4: $b \leftarrow (b \oplus c) \ggg 12$                                    8: $b \leftarrow (b \oplus c) \ggg 7$
```

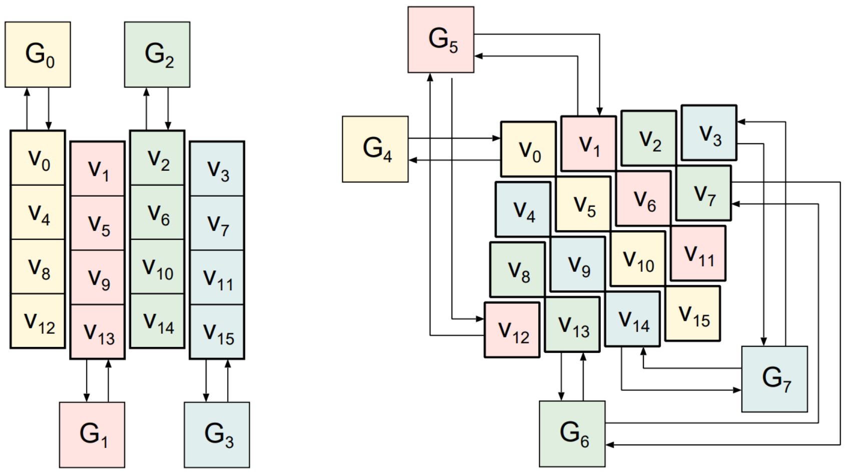


Figure 3: Column step and diagonal step used in the round function. (This image is taken from SHA3 Proposal BLAKE)

At round r >9, the permutation used is $\sigma_{r \pmod{10}}$ e.g., in the 12th round, $r = 11$ and the permutation used is $\sigma_{11 \pmod{10}} = \sigma_1$. Here, σ refers to the 10×16 permutation table (shown in Table 1) used by BLAKE to select message blocks and word constants for a round function, e.g. $m_{\sigma_{r(2i+1)}}$ refers to $\sigma[r][2i+1] = \sigma[3][15] = 4$ (considering $r = 13$ and $i = 7$).

Round	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	14	10	4	8	9	15	13	6	1	12	0	2	11	7	5	3
3	11	8	12	0	5	2	15	13	10	14	3	6	7	1	9	4
4	7	9	3	1	13	12	11	14	2	6	5	10	4	0	15	8
5	9	0	5	7	2	4	10	15	14	1	11	12	6	8	3	13
6	2	12	6	10	0	11	8	3	4	13	7	5	15	14	1	9
7	12	5	1	15	14	13	4	10	0	7	6	3	9	2	8	11
8	13	11	7	14	12	1	3	9	5	0	15	4	8	6	2	10
9	6	15	14	9	11	3	0	8	12	2	13	7	1	4	10	5
10	10	2	8	4	7	6	1	5	15	11	9	14	3	12	13	0

Table 1: Permutations used for choosing the message block and constants in BLAKE and BLAKE2

Finally, after all the rounds are over, the new chain values $\mathtt{h}'_0, \ldots, \mathtt{h}'_7$ are obtained from the modified state variables $\mathtt{v}_0, \ldots, \mathtt{v}_{15}$ along with salts $\mathtt{s}_0, \ldots, \mathtt{s}_3$ and the initial chain values $\mathtt{h}_0, \ldots, \mathtt{h}_7$ and represented as final hash value:

$$\begin{array}{ll}
\mathtt{h}'_0 \leftarrow \mathtt{h}_0 \oplus \mathtt{s}_0 \oplus \mathtt{v}_0 \oplus \mathtt{v}_8 & \mathtt{h}'_4 \leftarrow \mathtt{h}_4 \oplus \mathtt{s}_0 \oplus \mathtt{v}_4 \oplus \mathtt{v}_{12} \\
\mathtt{h}'_1 \leftarrow \mathtt{h}_1 \oplus \mathtt{s}_1 \oplus \mathtt{v}_1 \oplus \mathtt{v}_9 & \mathtt{h}'_5 \leftarrow \mathtt{h}_5 \oplus \mathtt{s}_1 \oplus \mathtt{v}_5 \oplus \mathtt{v}_{13} \\
\mathtt{h}'_2 \leftarrow \mathtt{h}_2 \oplus \mathtt{s}_2 \oplus \mathtt{v}_2 \oplus \mathtt{v}_{10} & \mathtt{h}'_6 \leftarrow \mathtt{h}_6 \oplus \mathtt{s}_2 \oplus \mathtt{v}_6 \oplus \mathtt{v}_{14} \\
\mathtt{h}'_3 \leftarrow \mathtt{h}_3 \oplus \mathtt{s}_3 \oplus \mathtt{v}_3 \oplus \mathtt{v}_{11} & \mathtt{h}'_7 \leftarrow \mathtt{h}_7 \oplus \mathtt{s}_3 \oplus \mathtt{v}_7 \oplus \mathtt{v}_{15}
\end{array}$$

The final hash value is represented in the big-endian format as $\mathtt{h}'_0 \| \mathtt{h}'_1 \| \ldots \| \mathtt{h}'_7$.

3 Description of BLAKE2s-256 Hash Function

The BLAKE2 [3] is a modified family of hash functions based on the BLAKE. The main objective of the new BLAKE2 is to provide several parameters for use in applications without the need for additional constructions and constants, and modes and also to increase the speed of hashing process to reach a level close to that of MD5. The designers have managed to achieve this goal by slightly modifying the original BLAKE hash function; they have modified the initial setup of the compression function, changed the rotation constants to be optimal for software performance, excluded constants from the round functions, made it little-endian, etc.

As a successor of the BLAKE family, the BLAKE2 hash functions share many similarities with the original design of BLAKE. However, differences occur at every level: internal permutation, compression function, and hash function construction. Here, a brief specification of BLAKE2 and a highlight of the differences with BLAKE is drawn. The notations used here are the following:

(a) '$\leftarrow$' denotes variable assignment;
(b) '$+$' denotes addition in $\mathbb{Z}_{2^{32}}$ or in $\mathbb{Z}_{2^{32}}$ (modular addition);
(c) '$-$' denotes subtraction in $\mathbb{Z}_{2^{32}}$ or in $\mathbb{Z}_{2^{32}}$ (modular subtraction);
(d) '$\oplus$' denotes addition in $\mathbb{Z}_2^{32}$ or in $\mathbb{Z}_2^{64}$ (bitwise exclusive or);
(e) '$\lll r$' denotes rotation of r bits towards the most significant bit;
(f) '$\ggg r$' denotes rotation of r bits towards the least significant bit;
(g) if not specified otherwise, numbers written in typewriter font are in base 16, e.g. `f` is the number 15.

The internal state of the compression function is composed of 16 words (each word is 32 bits) for BLAKE2s. The compression function takes as input an 8-word chaining value $h_0, \ldots, h_7$, eight constant initialization vectors $IV_0, ..., IV_7$, a 2-word counter t_0 and t_1 that counts the number of bytes hashed so far, and two finalization flags f_0 and f_1. The flag f_0 is set to `ff...ff` when the current message block is the last, and to `00...00` otherwise. The eight constant initialization vectors are:

$$\begin{array}{llll}
\mathtt{IV}_0 = \mathtt{0x6A09E667}; & \mathtt{IV}_1 = \mathtt{0xBB67AE85}; & \mathtt{IV}_2 = \mathtt{0x3C6EF372}; & \mathtt{IV}_3 = \mathtt{0xA54FF53A} \\
\mathtt{IV}_4 = \mathtt{0x510E527F}; & \mathtt{IV}_5 = \mathtt{0x9B05688C}; & \mathtt{IV}_6 = \mathtt{0x1F83D9AB}; & \mathtt{IV}_7 = \mathtt{0x5BE0CD19}
\end{array}$$

BLAKE2s uses a 32-byte 'parameter block' that encodes various parameters of the BLAKE2 hash function. General parameters are the digest length, the optional key length, an optional salt, and a personalization string. Here, values spanning more than one byte are written in little-endian. Additional parameters are defined for tree hashing(not discussed). This parameter block in BLAKE2 is one of the major differences with BLAKE. Some of the optional features of BLAKE (e.g. the salt) have been moved from the compression function to the parameter block in BLAKE2. An overview of the parameter block for BLAKE2s is shown in Table 2 below.

Offset	**0**	**1**	**2**	**3**
0	Digest length	Key length	Fanout	Depth
4	Leaf length			
8	Node Offset			
12	Node offset (cont.)		Node depth	Inner length
16 **20**	Salt			
24 **28**	Personalization			

Table 2: BLAKE2s parameter block structure (offsets in bytes).

Example of a parameter block for BLAKE2s: Here, no key, salt and personalization is used, so they are set to NULL. BLAKE2s hashes data sequentially; thus, for the sequential mode, the tree parameters are set to the values specified beforehand, like fanout and maximal depth are set to 01, leaf maximal length is set to 00000000, node offset is set to 00000000 00000000, node depth and inner hash length are set to 00. The parameter block for this instance of BLAKE2s is thus:

```
20000101 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

Initially, the chained values of the context $h_0...h_7$ are initialised with the constant values of $IV_0...IV_7$ and the first 4 bytes of the parameter block are XORed with h_0 (as the rest are null bytes) in little-endian format. The input to the compression function is initialized as:

$$\begin{pmatrix} v_0 & v_1 & v_2 & v_3 \\ v_4 & v_5 & v_6 & v_7 \\ v_8 & v_9 & v_{10} & v_{11} \\ v_{12} & v_{13} & v_{14} & v_{15} \end{pmatrix} \leftarrow \begin{pmatrix} h_0 & h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 & h_7 \\ IV_0 & IV_1 & IV_2 & IV_3 \\ t_0 \oplus IV_4 & t_1 \oplus IV_5 & f_0 \oplus IV_6 & f_1 \oplus IV_7 \end{pmatrix} \tag{4}$$

The original message is padded with null bytes if and only if necessary to make the message a multiple of the block length (512 bits for BLAKE2s). The padded message is then passed to the **compression function** G. The initial state is then processed by ten rounds(reduced from 14 in original BLAKE-256) of a column and diagonal(shown in Figure 3) application of the G function for BLAKE2s. The G function takes four state words (a, b, c, d) and two message words m_i and m_j as input. The message blocks to be used are defined by a position index i of the function: at round r, m_i is given by $\sigma_{r \bmod 10}(2i)$ and m_j by $\sigma_{r \bmod 10}(2i+1)$, where $\sigma_{r \bmod 10}$ is one of the ten permutations given in Table 1 The G function of BLAKE2s, $\mathtt{G}(a, b, c, d)$ is defined as:

$$\begin{array}{ll} 1\colon a \leftarrow a + b + m_i & 5\colon a \leftarrow a + b + m_j \\ 2\colon d \leftarrow (d \oplus a) \ggg 16 & 6\colon d \leftarrow (d \oplus a) \ggg 8 \\ 3\colon c \leftarrow c + d & 7\colon c \leftarrow c + d \\ 4\colon b \leftarrow (b \oplus c) \ggg 12 & 8\colon b \leftarrow (b \oplus c) \ggg 7 \end{array}$$

The differences between the G functions of BLAKE2s and BLAKE-256 are the omission of an '$\oplus$' addition between the message words and round constants in steps 1 and 5 of BLAKE2s. A column step of in BLAKE2s can be defined as:

$$\mathtt{G}_0(v_0, v_4, v_8, v_{12}) \quad \mathtt{G}_1(v_1, v_5, v_9, v_{13}) \quad \mathtt{G}_2(v_2, v_6, v_{10}, v_{14}) \quad \mathtt{G}_3(v_3, v_7, v_{11}, v_{15})$$

and a diagonal step computes:

$$\mathtt{G}_4(v_0, v_5, v_{10}, v_{15}) \quad \mathtt{G}_5(v_1, v_6, v_{11}, v_{12}) \quad \mathtt{G}_6(v_2, v_7, v_8, v_{13}) \quad \mathtt{G}_7(v_3, v_4, v_9, v_{14})$$

Finally, the output of the compression function $h_0, \ldots, h_7$ combines the input chaining value and the final state $v_0, \ldots, v_{15}$ by computing:

$$\begin{array}{ll} h'_0 \leftarrow h_0 \oplus v_0 \oplus v_8 & h'_4 \leftarrow h_4 \oplus v_4 \oplus v_{12} \\ h'_1 \leftarrow h_1 \oplus v_1 \oplus v_9 & h'_5 \leftarrow h_5 \oplus v_5 \oplus v_{13} \\ h'_2 \leftarrow h_2 \oplus v_2 \oplus v_{10} & h'_6 \leftarrow h_6 \oplus v_6 \oplus v_{14} \\ h'_3 \leftarrow h_3 \oplus v_3 \oplus v_{11} & h'_7 \leftarrow h_7 \oplus v_7 \oplus v_{15} \end{array}$$

The only difference between BLAKE2 and BLAKE in the final step of deriving the hash digest is the omission of the optional salt value s. A brief overview of the complete process of obtaining hash digest in BLAKE2s is:

1. A parameter block(given in Table 2) is XORed with the initialization vectors used in the compression function. The modified vectors are used as the first input chaining value to the compression function.
2. The message is padded with null bytes if and only if necessary to make it a multiple of a block length (i.e. 512 bits for BLAKE2s).
3. The compression function is run with blocks of the padded message as input; after processing all the blocks, the final output is converted into the little-endian format and taken as the hash value.

4 Differential Cryptanalysis

Cryptanalysis is analysing the security standards and vulnerabilities of any encryption or cryptosystem and transforming or decoding messages from non-interpretable to interpretable formats without access to the key used in the process. It is a mathematical attempt at breaking the algorithm or decryption utilizing knowledge about the encryption scheme that is already available publicly. It can use encrypted messages (ciphertext) and original messages given as input (plaintext) to conduct the analysis.

Differential cryptanalysis may be used to decrypt both block and stream ciphers, as well as cryptographic hash functions. Broadly, it studies how changes in input impact the difference in output. It focuses on finding the non-random behaviour of the algorithm and analyses the transformations to retrieve the hidden key (in the case of encryption).

Here, a study of the working mechanism of the DES block cipher and a known-plaintext differential attack on the DES block cipher reduced to 6 rounds are discussed. Later on, similar concepts will be utilized to mount differential attacks on BLAKE2s hash function.

4.1 Description of DES block cipher

Data Encryption Standard(DES)[4] is based on the Feistel construction and uses 16 consecutive rounds of Feistel. Initial and final permutations are performed before and after applying the Feistel rounds to jumble up the bits; the final permutation here is the inverse of the initial permutation. The initial and final permutations of DES are depicted in Table 3. It says that the first bit of the permuted input is 58^{th} bit of the original input, the second bit in the table is the 50^{th}, the third bit in the table is the 52^{nd} and so on.

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

(a)

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

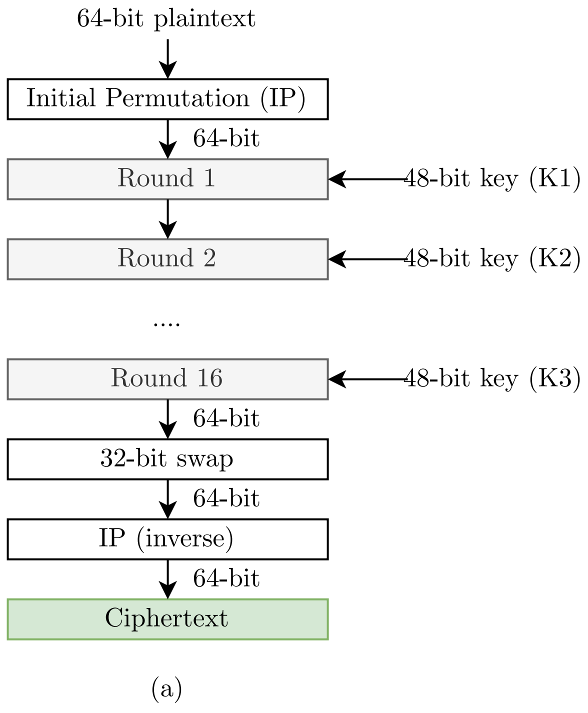
(b)

Table 3: (a)Initial permutation in DES and (b)Final permutation in DES

Initially, the input plaintext is passed through the initial permutation, and the 64-bit permuted input is then split into two 32-bit halves $L_0 = 1$ to 32 bits and $R_0 = 33$ to 64 bits. The key scheduling algorithm generates 16 round keys for each round of DES. Here, K_i depicts the round key for round i. At round i, the DES algorithm computes:

$$L_i = R_i - 1 \text{ and } R_i = L_{i-1} \oplus f(R_{i-1}, K_i) \tag{5}$$

where f is the round function (shown in Figure 4(b)) keyed by K_i. After round 16, the concatenation of R_{16} and L_{16} is passed through a 32-bit swap where L_{16} and R_{16} are swapped. Then the final permutation(inverse of the initial permutation) is applied to the 64-bit output to get the generated ciphertext. A pictorial overview of the DES algorithm is shown in Figure 4(a).



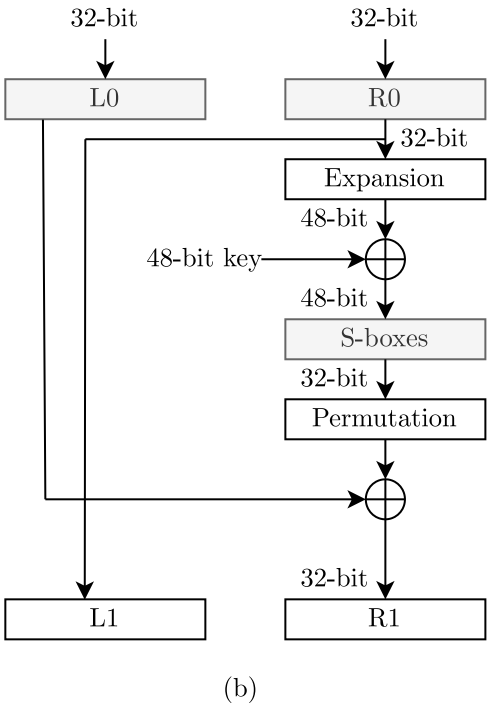


Figure 4: (a)Overview of DES and (b)Round function of DES

DES actually accepts a 64-bit key, but when the key is passed to the key scheduling algorithm of DES, it converts it into a 56-bit key, and further 48-bit keys are generated for each round of DES function. The round function of DES can be represented mathematically as:

$$f(x, k) = P \circ S(E(x) \oplus k) \tag{6}$$

where E is a linear expansion of input from 32 to 48 bits by duplicating some of them(shown in Table 5), S a non-linear transform of 6-bit inputs to 4 bits called S-box(eight such S-boxes are given in Tables 6 to 13) and P is a bit permutation in round function on 32 bits(shown in Table 4).

16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
19	13	30	6	22	11	4	25

Table 4: Permutation of the round function in DES

32	1	2	3	4	5	4	5	6	7	8	9	8	9	10	11
12	13	12	13	14	15	16	17	16	17	18	19	20	21	20	21
22	23	24	25	24	25	26	27	28	29	28	29	30	31	32	1

Table 5: Expansion permutation of the round function in DES

After expansion permutation, the 48-bit block is XORed with the 48-bit round key and then split into eight sub-blocks of 6 bits each, each of the sub-blocks is passed to the respective S-box. The first sub-block is passed to the first S-box S1 and the second sub-block to the S-box S2, and so on. Each S-box gives a 4-bit output. Hence a total of 32 bits are obtained from all eight S-boxes, and the outputs from S-boxes are then concatenated back into a 32-bit word, starting with S1 and ending with S8.

When computing S_i on a set of six bits $b_1b_2b_3b_4b_5b_6$, the number obtained from b_1b_6 (i.e., $2b_1 + b_6$) is used to determine the row number from 0 to 3 and the number obtained from $b_2b_3b_4b_5$ is used to determine the column number from 0 to 15. The output value of the S-box is then interpreted as a bitstring $c_1c_2c_3c_4$ found at the position $S_i[2b_1 + b_6][b_2b_3b_4b_5]$ corresponding to the number $8c_1 + 4c_2 + 2c_3 + c_4$ in decimal. The output bits of S1 are numbered from 1 to 4, while the output bits of S8 are numbered from 29 to 32. The lookup tables for all eight S-boxes are given in Tables 6 to 13.

14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

Table 6: S-box S1

15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

Table 7: S-box S2

10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

Table 8: S-box S3

7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

Table 9: S-box S4

2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

Table 10: S-box S5

12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

Table 11: S-box S6

4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

Table 12: S-box S7

13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Table 13: S-box S8

4.2 Differential cryptanalysis of DES block cipher

In brief, differential cryptanalysis [5] can be defined as a method which analyses the impact of differences in plaintext pairs (input) on the differences of the resultant ciphertext pairs (output). These differences can be analysed to assign probabilities to the possible keys generated and to find the most probable key according to the assigned probabilities.

Differential Cryptanalysis requires many pairs of plaintexts and corresponding ciphertexts with a particular difference. As if the attacker has access to the encryption machine and can generate many input-output pairs or plaintext-ciphertext pairs but cannot see the hidden key used in the machine. For `DES` and `DES`-like cryptosystems, the difference between chosen plaintext is usually a fixed XORed value of the two plaintexts.

4.2.1 Notations Used

Here are the notations used in the differential analysis:

- 'n_x' : Number n represented in hexadecimal form (i.e., $A_x = 10$);
- 'X, X'' : For a pair of messages, X_1 and X_2, X' is defined to be $X' = X_1 \oplus X_2$;
- 'P' : Input plaintext
- 'T' : Ciphertext;
- '$P(X)$' : The permutation in DES round function (Table 4);
- '$E(X)$' : The expansion permutation in DES round function (Table 5);
- '$IP(X)$' : The initial permutation (Table 3(a));
- '(L, R)' : The left and right halves of the plaintext P (after the initial permutation);
- '(l, r)' : The left and right halves of the ciphertext T respectively (before the final permutation);
- '$a, ..., j$' : The 32-bit inputs to the F-box in different rounds;
- '$A, ..., J$' : The 32-bit outputs of the F-box in different rounds.

4.2.2 Characteristics

Considering a pair of messages(or plaintexts) as an input, the XOR value of the messages, the XOR of its ciphertexts, the XORs of the inputs of each round and the XORs of the outputs of each round form an n-round characteristic.

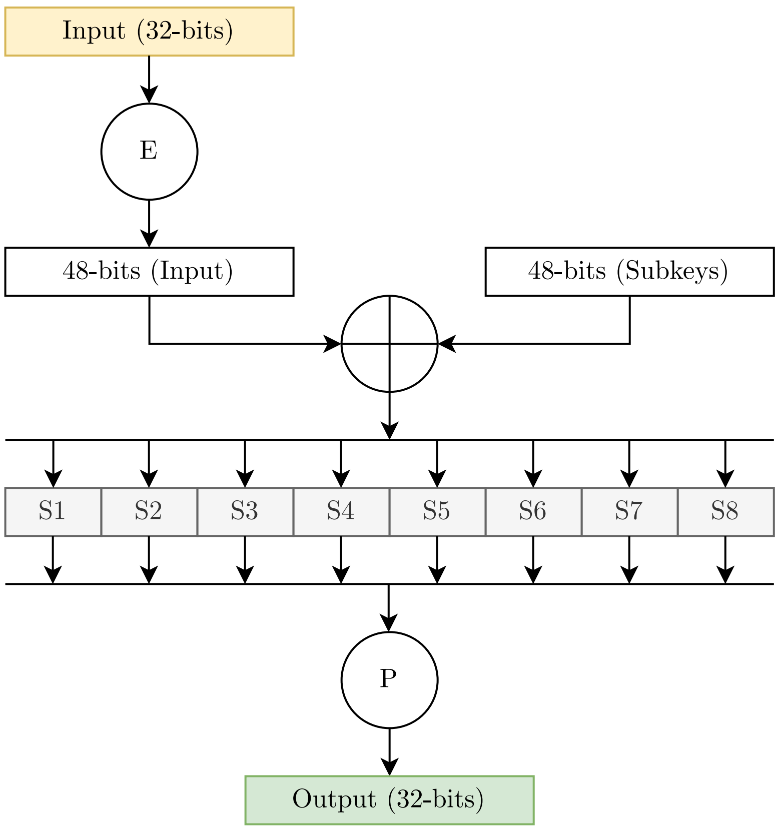


Figure 5: Design of F-box used in the characteristics of DES

Each characteristic has a probability, which refers to the probability that a randomly chosen plaintext pair with the predetermined plaintext XOR values has the round and ciphertext XORs already provided in the given characteristic. The concept of probability comes from the fact that different message pairs with the same plaintext XOR value may lead to different output XORs. E.g., if we consider one round of `DES` with the given characteristic as:

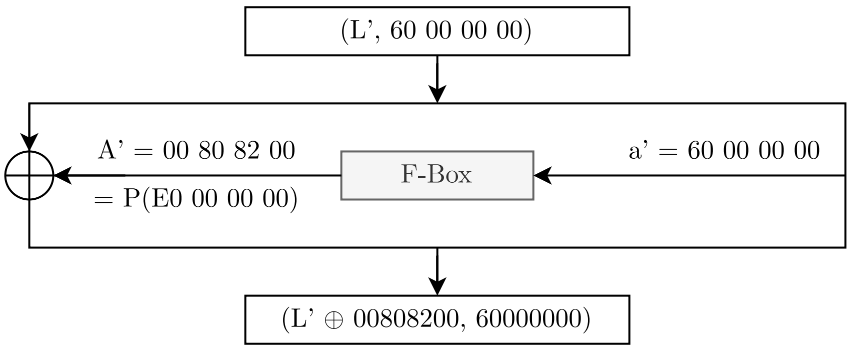


Figure 6: One round characteristic of `DES`

```
a' = 0110 0000 0000 0000 0000 0000 0000 0000
E(a') = 001100 000000 000000 000000 000000 000000 000000 000000
```

Now, for S2 to S8, as input XOR is 0, output XOR will also be 0. This is derived from the difference distribution table for each S-box. The difference distribution table for an S-box gives the number of input pairs with a chosen input XOR producing a chosen output XOR value. Here, for S1, the input XOR is `001100`; the possible output XORs and corresponding possible input pairs are mentioned in Table 14. From Table 14, we can see that the probability of getting '`E`' is 7/32 (i.e., 14/64 with input XOR as '`C`' and output XOR as '`E`'). Finally, when Permutation P is applied to the output of the S-box, it gives $A' = (00\ 80\ 82\ 00)_x$. So, the 1-round characteristic derived is $(L' \oplus 00\ 80\ 82\ 00,\ 60\ 00\ 00\ 00)_x$ with probability 14/64.

Output XOR ($S1'_O$)	**Possible Input Pairs ($S1_I$)**
3	(10,1C), (14,18), (24,28), (31,3D)
5	(00,0C), (15,19), (16,1A)
6	(07,0B), (20,2C), (33,3F)
9	(05,09), (11,1D), (35,39)
10	(22,2E), (30,3C), (34,38)
11	(23,2F), (27,2B)
12	(02,0E), (25,29), (32,3E)
13	(01,0D), (12,1E), (36,3A)
E	(03,0F), (06,0A), (13,1F), (17,1B), (21,2D), (26,2A), (37,3B)
F	(04,08)

Table 14: Possible output pairs corresponding to each output XOR for S1

4.2.3 Cryptanalysis and Key recovery from 6-round DES

For key recovery from 6-round DES, two 3-round characteristics are used, both with probability 1/16 and the key occurring most often is taken into consideration. Both characteristics help find 30 bits of round 6 key each, but 3 of the S-boxes are common among them, so only 42 bits are derived using these characteristics. The remaining bits can be generated using brute force and the key scheduling algorithm, which is publicly available.

Suppose the attacker has access to the DES encryption machine and can generate any number of plaintext-ciphertext pairs, this kind of approach is termed a chosen-plaintext attack. Here, the key is hidden and is not available to one performing the attack; suppose the key is:

```
Key:1110111100110011011101101101111000110100010101111111000100010011
```

The first characteristic used is:

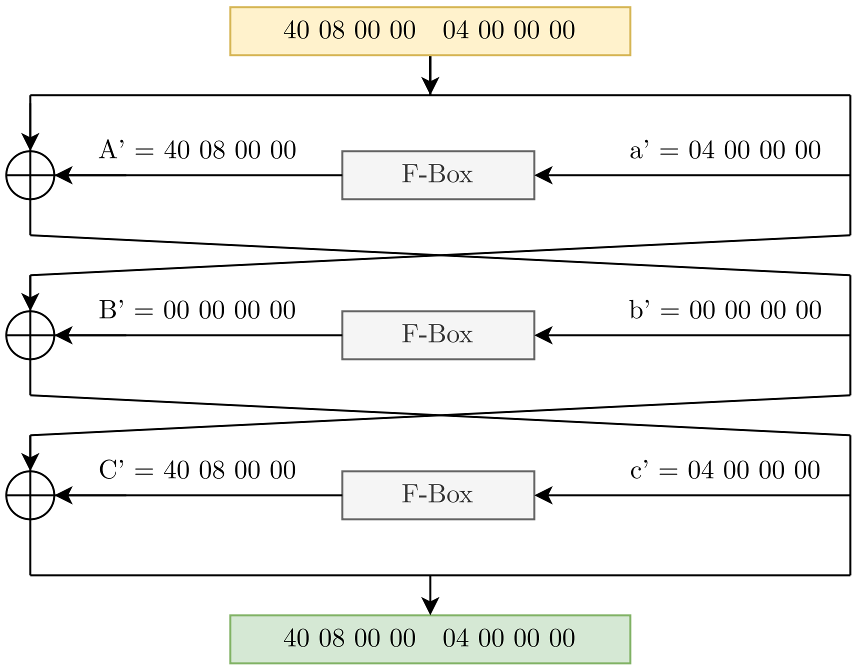


Figure 7: 3-round characteristic of DES

First, a plaintext pair is chosen such that after the Initial Permutation(IP), the input XOR becomes (40 08 00 00 04 00 00 00)$_x$.

$$\begin{aligned}&\texttt{P}_1\texttt{=1101 0101 0010 0010 1101 0011 0111 0010 1110 0001 1011 1100 0011 0010 0010 1010}\\&\texttt{P}_2\texttt{=1101 0101 0010 0010 0101 0011 0110 0010 1110 0001 1011 1100 0111 0010 0010 1010}\end{aligned}$$

After applying the initial permutation on the plaintext taken:

$$\begin{aligned}&\texttt{IP(P}_1\texttt{)=0001 1101 0110 1101 0010 0001 0001 0101 0011 0101 1111 1010 1010 0000 1100 1110}\\&\texttt{IP(P}_2\texttt{)=0101 1101 0110 0101 0010 0001 0001 0101 0011 0001 1111 1010 1010 0000 1100 1110}\end{aligned}$$

The input XOR given in the characteristic is obtained by $\texttt{IP(P}_1\texttt{)} \oplus \texttt{IP(P}_2\texttt{)}$. In binary, the input XOR is:

```
Input XOR=0100 0000 0000 1000 0000 0000 0000 0000 0000 0100 0000 0000 0000 0000 0000 0000
```

The input to the fourth round(d) is (40 08 00 00)$_x$ and after the expansion permutation:

```
     d′ = 0100 0000 0000 1000 0000 0000 0000 0000
E(d′) = 001000 000000 000001 010000 000000 000000 000000 000000
```

It can be noticed here that five of the S-boxes (S2, S5, S6, S7, S8) in the fourth round have zero input XORs, and hence their output XORs are also zero(obtained from difference distribution table). Considering these 5 S-boxes, the output XOR for the sixth round can be obtained as $\texttt{F}' = \texttt{c}' \oplus \texttt{l}'$, as:

From characteristics,

$$\begin{aligned}\texttt{l}' = \texttt{F}' \oplus \texttt{e}' &\Rightarrow \texttt{F}' = \texttt{l}' \oplus \texttt{e}'\\ &\Rightarrow \texttt{F}' = \texttt{l}' \oplus \texttt{D}' \oplus \texttt{c}'\end{aligned}$$

Thus, $\texttt{F}' = \texttt{c}' \oplus \texttt{l}'$ (as for these five S-boxes $\texttt{D}' = 0$).

Now, the inverse of the final permutation, i.e. the initial permutation, can be applied on the ciphertext pair to obtain the input to the S-box of the sixth round of DES as:

$$\begin{aligned}\texttt{T}_1 &= \texttt{0101 0100 1100 0110 0110 0101 1101 0111 1001 0001 1101 0111 0111 0110 0100 1110}\\ \texttt{T}_2 &= \texttt{1110 0100 0011 0100 0011 1111 1000 0010 0101 0110 0111 0011 0101 0000 1101 0010}\\ \texttt{IP(T}_1\texttt{)} &= \texttt{1110 1111 0111 1001 1110 1111 0011 1100 0011 1010 0100 0100 1000 0000 1110 1010}\\ \texttt{IP(T}_2\texttt{)} &= \texttt{1111 0001 1111 0110 0001 0111 0010 0100 1000 1001 0010 0111 0000 0100 1011 1100}\end{aligned}$$

Hence, the input to the sixth round can be obtained from the last 32-bits of IP(T_1) and IP(T_2) respectively as:

$$f_1 = 0011\ 1010\ 0100\ 0100\ 1000\ 0000\ 1110\ 1010$$
$$f_2 = 1000\ 1001\ 0010\ 0111\ 0000\ 0100\ 1011\ 1100$$

The input to S-boxes of the sixth round can be derived after passing f_1 and f_2 through the expansion function E() and obtained inputs are:

$$E(f_1) = 000111\ 110100\ 001000\ 001001\ 010000\ 000001\ 011101\ 010100$$
$$E(f_2) = 010001\ 010010\ 100100\ 001110\ 100000\ 001001\ 010111\ 111001$$

Each input pair considered here may not suggest the correct key as it is probabilistic in nature. We iterate over all $64(= 2^6)$ values of a key for each of the corresponding five S-boxes for many input pairs and increase the count of the key for which the above f values result in the XOR given by $F' = c' \oplus l'$ and consider the key which occurs most often as the most probable key.

$$c' = 0000\ 0100\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000$$
$$l' = 1011\ 0011\ 0110\ 0011\ 1000\ 0100\ 0101\ 0110$$
$$F' = 1011\ 0111\ 0110\ 0011\ 1000\ 0100\ 0101\ 0110$$
$$\text{Output of S-box} = 1100\ 1101\ 0100\ 0100\ 0100\ 1101\ 0001\ 1011 \text{ (obtained by } P_{inverse}(F'))$$

Considering the **S2 S-box**, for the chosen input pair, the count of keys '011011', '011101', '111011', and '111101' is increased and moving on to the validation, suppose for key '011011',

$$E(f_1) \oplus 011011 = 110100 \oplus 011011 = 101111$$
$$E(f_1) \oplus 011011 = 010010 \oplus 011011 = 001001$$
$$\text{Now, Output XOR of S-box(S2)} = S2(101111) \oplus S2(001001) = 0010 \oplus 1111 = 1101$$

Considering 250 such plaintext-ciphertext pairs, the following probable key bits corresponding to the five S-boxes are obtained:

```
S2:  111101
S5:  011010
S6:  101100
S7:  111011
S8:  010011
```

Similar to the process above, the second characteristic is used to obtain the probable keys corresponding to S-boxes S1, S2, S4, S5, and S6. The second characteristic is represented diagrammatically below in Figure 8.

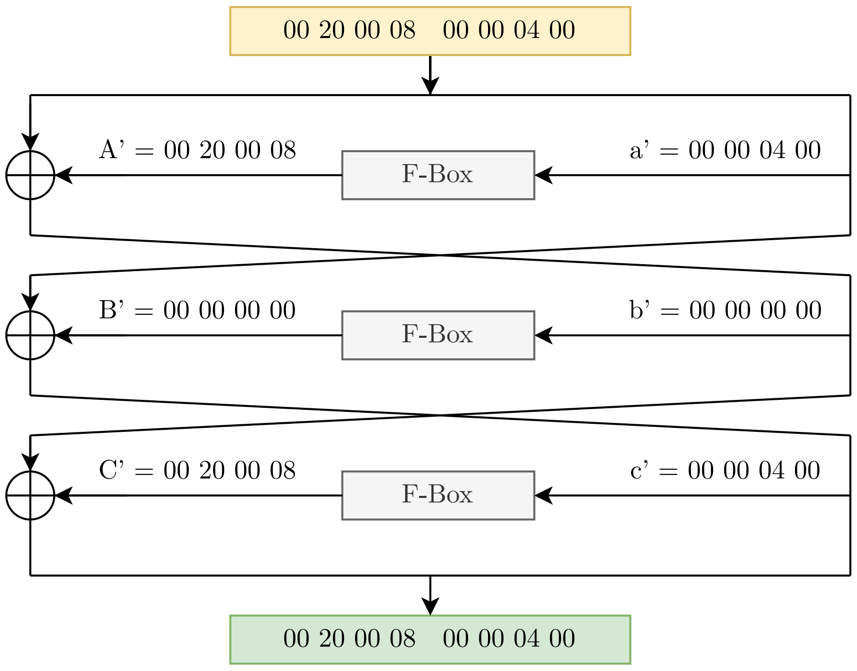


Figure 8: Another 3-round characteristic of DES

The probable keys obtained using the second characteristic for the S-boxes S1, S2, S4, S5 and S6 are:

```
S1:  110010
S2:  111101
S4:  100110
S5:  011010
S6:  101100
```

Here, the need for analysing more inputs can be figured out by comparing the probable keys for the common S-boxes from both characteristics. Once they are the same after analysing enough inputs, they can be termed as the most probable keys. Combining all the bits obtained, 42 bits from the 56-bit key are retrieved. The positions of the bits retrieved and the remaining 14 bits can be determined using a brute-force approach and the publicly available key scheduling algorithm.

5 Results and Discussion

A known-plaintext differential attack is mounted on the Data Encryption Standard (`DES`) block cipher reduced to 6 rounds after the complete attack with 250 different input-output or plaintext-ciphertext pairs, 42 bits out of the 56 bits of the key has been successfully retrieved using the cryptanalysis. Along with 250 plaintext-ciphertext pairs, two 3-round characteristics of `DES` are also used. The probable keys retrieved for S-boxes S1, S2, S4, S5, S6, S7 and S8 from the cryptanalysis are:

```
S1:  110010
S2:  111101
S4:  100110
S5:  011010
S6:  101100
S7:  111011
S8:  010011
```

The obtained bits can be arranged to get the 48-bit round key at round 6 as:

```
110010 111101 xxxxxx 100110 011010 101100 111011 010011
```

The key guessed after reversing the key scheduling algorithm is:

```
x11011xx x011001x 0xx1011x x10x111x x01xx10x 01x1011x 11x1000x x001x01x
```

Now, brute-force or exhaustive search approach can generate 2^{14} different 56-bit keys, out of which one is the correct key. The correct key can be verified by checking against some plaintext-ciphertext pairs. Hence, the key retrieved after the brute-force approach:

56-bit key : `1110111x0011001x0111011x1101111x0011010x0101011x1111000x0001001x`

64-bit key : `1110111100110011011101101101111000110100010101111111000100010011`(Original key)

Only 56 bits out of the original 64-bit key is helpful as in the key scheduling algorithm, every 8th bit is dropped, and the rest 56-bit key is processed further.

6 Future Work

Similar to the application of known-plaintext differential cryptanalysis on the `DES` block cipher, forms of differential and other attacks are planned to be mounted on the hash function `BLAKE2s`. The primary aim of attacks on hash functions is to test the robustness of the hash function against one or more of the three core properties of hash functions mentioned in Section 1.1. We further aim to study and modify some existing attacks on the `BLAKE2s`[6][7] hash function and make them efficient, improving the complexity of the attack or extend the attack to further rounds of the hash function.

References

[1] Ronald L. Rivest. The MD5 Message-Digest Algorithm. RFC 1321, April 1992.

[2] Jean-Philippe Aumasson, Willi Meier, Raphael C.-W. Phan, and Luca Henzen. *Specification of BLAKE*, pages 37–44. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[3] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, and Christian Winnerlein. Blake2: simpler, smaller, fast as md5. Cryptology ePrint Archive, Paper 2013/322, 2013. `https://eprint.iacr.org/2013/322`.

[4] A. Joux. *Algorithmic Cryptanalysis*. Chapman & Hall/CRC Cryptography and Network Security Series. Taylor & Francis, 2009.

[5] Eli Biham and Adi Shamir. *Differential Cryptanalysis of DES Variants*, pages 33–77. Springer New York, New York, NY, 1993.

[6] Jian Guo, Pierre Karpman, Ivica Nikolic, Lei Wang, and Shuang Wu. Analysis of blake2. Cryptology ePrint Archive, Paper 2013/467, 2013. `https://eprint.iacr.org/2013/467`.

[7] Bozhan Su, Wenling Wu, Shuang Wu, and Le Dong. Near-collisions on the reduced-round compression functions of skein and blake. Cryptology ePrint Archive, Paper 2010/355, 2010. `https://eprint.iacr.org/2010/355`.