Journal of Harbin Engineering University
ISSN: 1006-7043

Vol 45 No. 7
July 2024

Impossible Differential Cryptanalysis on QuadriAES and SemiAES

Siba Zazo[*] , Baseem Barhoum[*]
[*] Department of Software Engineering and Information Systems, Faculty of Informatics Engineering, Tishreen University, Lattakia, Syria

Abstract:
Introduction: Impossible differential cryptanalysis is a powerful method in cryptanalysis that leverages differentials that cannot occur to eliminate incorrect key guesses, effectively reducing the key space. While this technique has proven effective, its application to AES typically requires substantial memory, up to 128 GB, making it impractical for many aspiring cryptanalysts. To address this challenge, this paper introduces two new derivations of the AES algorithm, named QuadriAES and SemiAES, which operate on smaller plaintext blocks of 32 bits and 64 bits, respectively.
Objectives: The primary objective of this study is to employ impossible differential cryptanalysis on these new AES derivations, which feature significantly reduced memory requirements. Specifically, the study aims to introduce QuadriAES and SemiAES, apply impossible differential cryptanalysis to these derivations, optimize the attack process to limit memory usage to 6 GB, and evaluate the success rates and efficiency of the proposed methods.
Methods: In this study, we applied impossible differential cryptanalysis to a 5-round version of QuadriAES-32 and SemiAES-64. Our approach encompassed several significant optimizations. These included implementing a concatenated pairs algorithm to minimize writing operations, which was necessitated by limited memory, leveraging multiple output differences to increase the number of desired pairs, and utilizing multi-threading to enhance the efficiency of the attack execution. The attack process entailed generating random plaintext pairs, encrypting them with a randomly selected initial key, collecting the desired pairs, and subsequently leveraging them to eliminate incorrect keys and reduce the key space.
Results: The successful execution of impossible differential cryptanalysis on QuadriAES-32 and SemiAES-64 has demonstrated the feasibility of conducting attacks using only 6 GB of RAM, compared to other implementations on different AES variants that required up to 128 GB of memory. For QuadriAES, the attack involved 1000 concatenated pairs, 10 threads, two output differences, two first-round output differences, and 8 chunks of 5 million plaintext pairs each, resulting in a total of 40 million plaintext pairs. The attack took approximately 3 hours to complete. Similarly, the attack on SemiAES utilized 1000 concatenated pairs and 10 threads, along with four output differences, four first-round output differences, and 11 chunks of 25 million plaintext pairs each, resulting in a total of 275 million plaintext pairs. This attack took nearly 29 hours to execute.
Conclusions: This study implements impossible differential cryptanalysis on two newly proposed smaller derivations of the AES algorithm, namely QuadriAES and SemiAES. To overcome the memory requirements, we used the concatenated pairs algorithm and an incremental implementation of the attack. Additionally, we examined the effect of using multiple output differences and first-round output differences on the success rate. We also suggested multi-threading to enhance execution efficiency. This research not only expands the understanding of AES security but also provides practical insights for aspiring cryptanalysts aiming to employ advanced cryptanalytic techniques on widely used encryption standards. Future work will focus on extending these methods to other AES derivations and eventually to the standard AES algorithm.
Keywords: Impossible-differential cryptanalysis, impossible characteristic, AES.

1. Introduction

The Advanced Encryption Standard (AES) is a 128-bit block cipher that was selected by NIST as a standard encryption system in 2001 due to its resilience against common attacks, such as linear and differential cryptanalysis[1]. Since then, extensive research has been conducted to study its security against other attacks, including impossible differential cryptanalysis. This type of cryptanalysis was first introduced by Biham et al. in 1999 to attack the Skipjack cipher [2]. In 2001, Biham and Keller used it to attack a 5-round version of AES-128 [3], which required 4TB of memory as mentioned in [6]. Then, in 2002, Cheon et al. extended this attack to target a 6-round version of

AES-128 [4]. In 2004, R.C.W Phan exploited the key schedule algorithm and extended the attack to a 7-round version of AES-192 and AES-256[5].

Since then, numerous studies have aimed to enhance these attacks by introducing new impossible differentials or by reducing the memory and time requirements. For instance, a study by Kakarla et al. in 2017 on a 5-round AES-128 [6] managed to reduce the memory requirement of Biham and Keller's attack from 4 TB to 128.5 GB.

Another study by Debranjan Pal et al. in 2019 [7] utilized a parallel implementation to attack AES-128, dividing the execution among one master node and 8 worker nodes, with each requiring 96.5 GB, and successfully retrieved the full key in only 6.5 minutes. Then, in 2022, Debranjan Pal et al. [8] employed a similar approach to retrieve the full key of a 6-round AES-192 and AES-256 in 12.5 minutes.

While these attacks are effective in discovering the key, their significant memory requirements pose a challenge for aspiring cryptanalysts. This challenge was initially anticipated by R.C.W Phan in 2002 when he introduced a smaller version of the AES algorithm called the MiniAES which works with a 16-bit block [9], and subsequently implemented the impossible differential attack on a 5-round version of it in 2003 [10].

In this paper, we aim to expand on R.C.W. Phan's work by introducing two additional versions of the AES algorithm, namely QuadriAES and SemiAES, which operate with 32-bit and 64-bit blocks, respectively, and subject them to impossible differential cryptanalysis using only 6 GB of memory. To overcome the memory limitation, we propose several methods. The concatenated pairs algorithm significantly reduces the writing operations required, as we cannot execute the entire attack at once due to memory constraints and must perform each step individually, writing the results to files. Additionally, we suggest an incremental attack method that divides the overall attack, involving a large chosen plaintext chunk that exceeds current memory capacity, into multiple individual attacks on smaller chosen plaintext chunks, and then combining their results. We also recommend multi-threading to enhance execution efficiency. Furthermore, we examined the effect of using multiple output differences derived from the same input difference of the utilized characteristic on the success rate, as this increases the chances of finding desired pairs. Similarly, we examined the effect of employing multiple first-round output differences on the success rate by increasing the likelihood of eliminating incorrect keys.

The structure of this paper is organized as follows: Section 2 provides an overview of the AES algorithm. Section 3 discusses the principles and techniques of impossible differential cryptanalysis. Section 4 introduces the proposed QuadriAES, including its design and specifications. Section 5 describes the implementation of impossible differential cryptanalysis on a 5-round version of QuadriAES-32. Section 6 introduces the proposed SemiAES, outlining its design and specifications. Section 7 details the impossible differential cryptanalysis on a 5-round version of SemiAES-64. Section 8 presents the results, including the effectiveness of the proposed optimization methods and the outcomes of the attacks on QuadriAES-32 and SemiAES-64. Finally, Section 9 provides a discussion, analyzing the findings and recommendations for future research.

2. AES overview

The Advanced Encryption Standard (AES) is a 128-bit block cipher that supports key lengths of 128, 192, and 256 bits with 10, 12, and 14 rounds, respectively.

AES organizes its plaintext into a 4x4 state matrix, with each cell in the matrix being 8 bits long. All operations in AES are performed using the Galois field, specifically GF(2^8).

The primary components of AES include:

1- ***Key Addition:*** This entails performing an XOR operation between the round key and the state matrix.

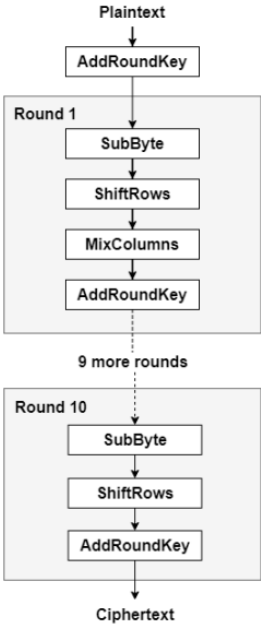
2- ***SubByte:*** This non-linear component applies the SBox to each state matrix cell.

3- ***ShiftRow:*** This involves a left cyclic shift performed on each row of the state matrix with different offsets based on the row number.

4- ***MixColumns:*** This component involves a matrix multiplication that transforms each column of the state matrix using the following equation:

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \times \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} \qquad (2.1)$$

Each of the previous components exists in a round function that repeat itself 10 times for AES-128. The entire encryption process is illustrated in Fig(1).



Fig(1)- AES encryption

3. Impossible Differential Cryptanalysis

Impossible differential cryptanalysis is a chosen-plaintext attack that involves collecting pairs of plaintexts P_1 , P_2 and their corresponding ciphertexts C_1 , C_2 . Each pair has an input difference ΔP and an output difference ΔC , which are computed as follows:

$$\Delta \mathrm{P} = P_1 \oplus P_2 \qquad (3.1)$$

$$\Delta \mathrm{C} = C_1 \oplus C_2 \qquad (3.2)$$

This attack relies on an impossible differential characteristic that traverses all rounds of the targeted cipher, creating a contradiction in the middle that should not occur. If the attacker finds a pair that follows this impossible characteristic, he can conclude that the key used to encrypt this pair is incorrect and discard it from the key space. This either leaves only the right key if the attack has a 100% success rate or reduces the key space, decreasing the complexity of the brute force attack.

The attack consists of three main steps:

a. ***Collecting the chosen plaintext pairs:*** The attacker gathers plaintext pairs with an input difference equal to that of the used characteristic. The size of this collection depends on the attacked cipher and the used characteristic.
b. ***Finding the desired pairs:*** The attacker filters the chosen plaintext pairs to find the desired pairs, which are the plaintext pairs with an output difference equal to that of the characteristic
c. ***Discarding the wrong keys:*** The attacker uses the desired pairs from the previous step and encrypts their plaintexts with all keys from the key space. When the attacker finds a pair that complies with the impossible characteristic, he can conclude that the key used to encrypt it is wrong and discard it from the key space.

4. QuadriAES

In this section, we present a derivation of AES called QuadriAES, which utilizes a smaller 32-bit block size, a quadrant of the block size of AES. QuadriAES supports three key lengths: 32, 48, and 64 bits, with 10, 12, and 14 rounds, respectively.

For organizing the plaintext, we propose a 2x2 state matrix as shown in Fig(2), with each cell consisting of 8 bits, referred to as a nibble, and using GF(2^8) as in the standard AES.

P_1	P_3
P_2	P_4

Fig(2)- the state matrix of QuadriAES

We take advantage of the fact that the first three encryption components of AES (key addition, SubByte, and ShiftRows) are independent of the block size, allowing us to use them as defined in the standard AES. As MixColumns is dependent on the state matrix dimensions, we propose to use a similar matrix as the one used in the mini- version of AES proposed by R.C.W Phan[9], which is shown in equation (4.1).

$$\begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 03 & 02 \\ 02 & 03 \end{bmatrix} \times \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \qquad (4.1)$$

The key scheduling algorithm computes the words of the round keys, with a suggested word size of 16 bits, resulting in 32, 48, and 64-bit keys consisting of 2, 3, and 4 words, respectively.

Denoting N_k for the number of words in each key, we can compute each word of the 32 and 48-bit round keys as follows:

$$w_i = \begin{cases} w_{i-1} \oplus w_{i-N_k} & ; i \bmod N_k \neq 0 \\ t_i \oplus w_{i-N_k} & ; i \bmod N_k = 0 \end{cases} \qquad (4.2)$$

Where $t_i = sub\big(Rotword(w_{i-1})\big) \oplus Rcon_{\,i/N_k}$ (4.3)

For the 64-bit round keys, each word is computed as follows:

$$w_i = \begin{cases} sub(w_{i-1}) \oplus w_{i-N_k} & ; i \bmod N_k \neq 0 \\ t_i \oplus w_{i-N_k} & ; i \bmod N_k = 0 \\ w_{i-1} \oplus w_{i-N_k} & ; elsewhere \end{cases} \qquad (4.4)$$

5. Impossible differential cryptanalysis of 5-round QuadriAES-32

In this section, we will analyze the vulnerability of a reduced version of the proposed QuadriAES with a 32-bit key using an impossible differential characteristic.

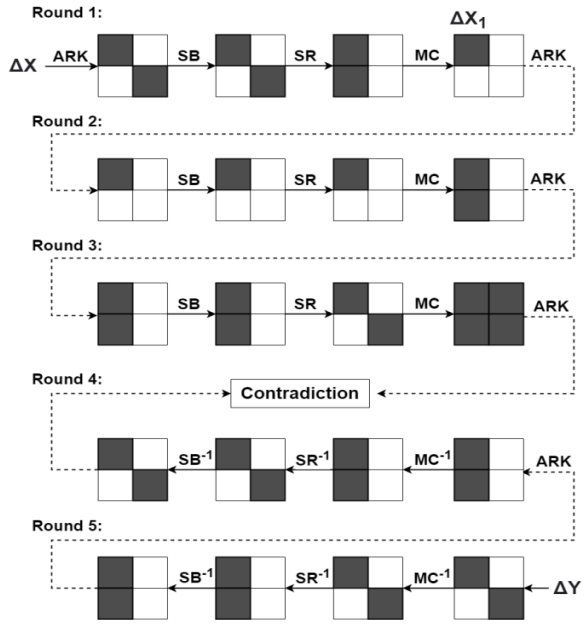
This characteristic is similar to the one used in attacking miniAES[3] as shown in Fig(3).

The attack consists of the following steps:

a. ***Collection of chosen plaintext pairs:*** Gather 2^x plaintexts pairs (P_1, P_2) that differ only in nibbles (1,4) and are identical in the other nibbles, where 2^x is the size of the chunk utilized in the attack.

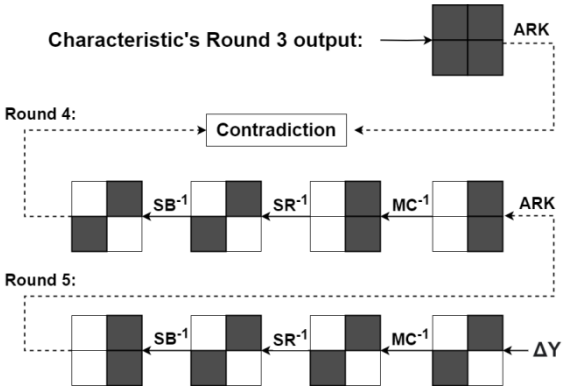
b. ***Identification of the desired pairs:***

Derive the corresponding ciphertexts (C_1, C_2) for the previously chosen plaintext pairs and select only the pairs in which their ciphertexts differ solely in nibbles (1,4) and are identical in the other nibbles.



Fig(3)- 5-round QuadriAES impossible characteristic

It is expected that such ciphertexts will be obtained from the 2^x plaintext pairs with a probability that is equal to $2^{-\#activeNibble\ *\ nibbleSize} = 2^{-16}$. To increase this probability, we will employ an additional characteristic that is equivalent to the previous characteristic in the first part – preceding the contradiction- and has a second part as follows:



Fig(4)- the second part of the additional characteristic

Consequently, we will have two possibilities for ΔY, each with a 2^{-16} probability, which will make the overall probability equal to $2 * 2^{-16} = 2^{-15}$, and we will obtain $2^x * 2^{-15} = 2^{x-15}$ desired pairs.

c. ***Discarding the incorrect keys:***

Each desired pair will be utilized to eliminate as many incorrect keys as possible by following these steps for each desired pair:

For all $2^{\#nibbleUsed\ *\ nibbleSize} = 2^{16}$ possible keys corresponding to an active nibble of K_0, compute (X, X`), which are the outputs of the first round of encryption for the plaintext pairs. This involves the following steps: Adding Round Key K_0, SubByte, ShiftRow, and MixColumn.

Then, calculate $X \oplus X`$. If it equals the ΔX_1 of the impossible characteristic, then the key used in the encryption is incorrect and can be discarded from the key space. This can happen with a probability of 2^{-8} because we have only one active nibble in ΔX_1, which corresponds to the following characteristic:

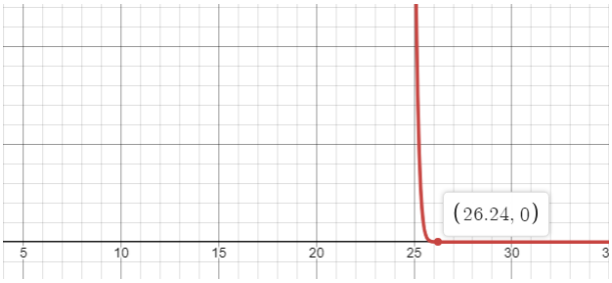
Characteristic (1): if we have one active column, then after applying MixColumn we would have only one active nibble in that column.

Since the state matrix is of size 2x2, we have two possibilities for the active nibble to be in the first row or the second row, which makes the probability equal to $2x\ 2^{-8} = 2^{-7}$.

By analysing all 2^{x-15} desired pairs, we can discard all incorrect keys when the following equation holds:

$$2^{16}(1 - 2^{-7})^{2^{x-15}} = 0 \qquad (5.1)$$

By plotting the previous function as in Fig(5), we see that $2^{26.24}$ is the minimum chunk size for the initial plaintext pairs that will result in a 100% success rate.



Fig(5) - the initial plaintext chunk equation of QuadriAES

6. SemiAES

In this section, we propose an another derivation of AES, called SemiAES, which utilizes a 64-bit block size, half of the standard AES block size. SemiAES operates with three key lengths: 64-bit, 96-bit, and 128-bit, with 10, 12, and 14 rounds, respectively.

For organizing the plaintext block, we propose using a 4x4 state matrix, as shown in Fig(6), with each cell being 4 bits in length, referred to as a nibble, and using the finite field GF(2^4) from the miniAES algorithm[9].

P_1	P_5	P_9	P_{13}
P_2	P_6	P_{10}	P_{14}
P_3	P_7	P_{11}	P_{15}
P_4	P_8	P_{12}	P_{16}

Fig(6)- the state matrix of SemiAES

Due to the 4-bit nibble size, the SBox of miniAES is used instead of the SBox of the standard AES. The AddRoundKey and shiftRows components remain identical to those in the standard AES, while for the MixColumn component, we propose utilizing a specific matrix for mixing columns, as shown in the following equation:

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \times \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} \qquad (6.1)$$

The key scheduling algorithm computes the words of the round keys, with a suggested word size of 16 bits, resulting in 4, 6, and 8 words for 64-bit, 96-bit, and 128-bit keys, respectively. Each word can be computed using the equations (4.2), (4.3), and (4.4) in section 4.

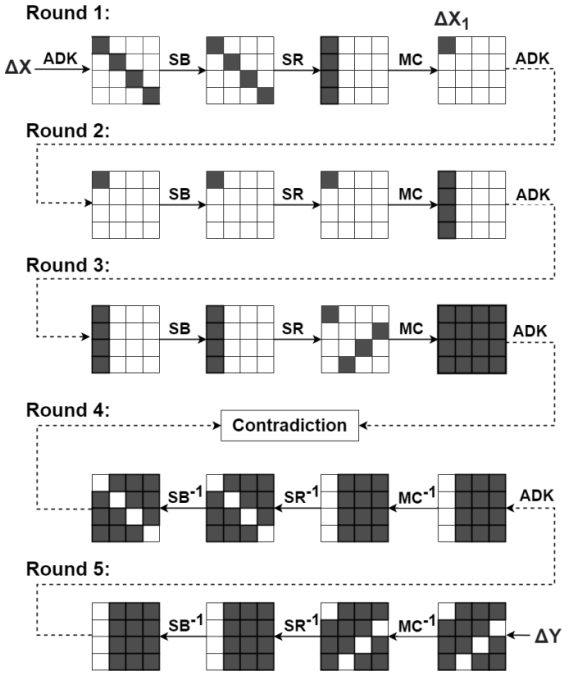
7. Impossible differential cryptanalysis of 5-round SemiAES-64

In this section, we will analyse the attack on a reduced 5-round version of the proposed SemiAES with a 64-bit key using an impossible differential characteristic similar to the one used to attack the standard AES[6] as illustrated in Fig(7).

The attack consists of the following steps:

a. *Collection of chosen plaintext pairs:*

Gather 2^x plaintext pairs (P_1, P_2) that differ only in nibbles (1,6,11,16) and are equal in the other nibbles, where 2^x is the size of the chunk used in the attack.

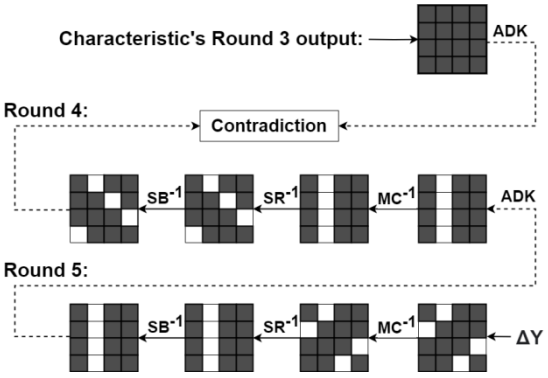


Fig(7)- 5-round SemiAES impossible characteristic

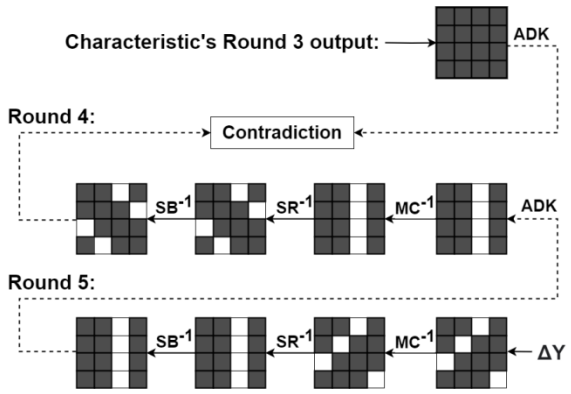
b. *Identification of the desired pairs:*

Derive the corresponding ciphertexts (C_1, C_2) for the previously chosen plaintext pairs and select only the pairs in which their ciphertexts differ in all nibbles but nibbles (1,8,11,14). It is expected that such ciphertexts will be obtained from the 2^x plaintext pairs with a probability that is equal to $2^{-\#activeNibble * nibbleSize} = 2^{-48}$.

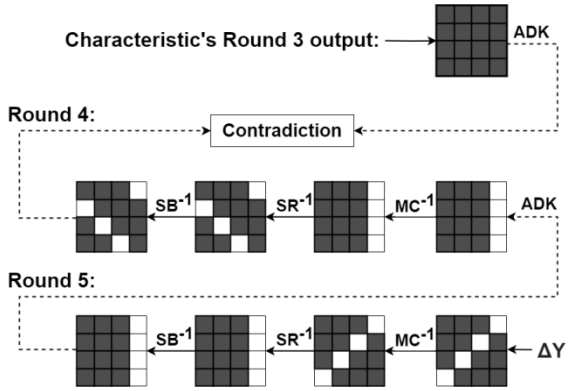
To enhance this probability, we will utilize three additional characteristics that are equivalent to the previous characteristic in the first part –prior to the contradiction- and their second part is as follows:



Fig(8)- the second part of 1st additional characteristic



Fig(9)- the second part of 2nd additional characteristic



Fig(10)- the second part of 3rd additional characteristic

Consequently, we will have four possibilities for delta ΔY, each with a probability of 2^{-48}, which will make the overall probability be equal to $4 * 2^{-48} = 2^{-46}$. Therefore, we will obtain $2^{x} * 2^{-46} = 2^{x-46}$ desired pairs.

c. Discarding the incorrect keys:

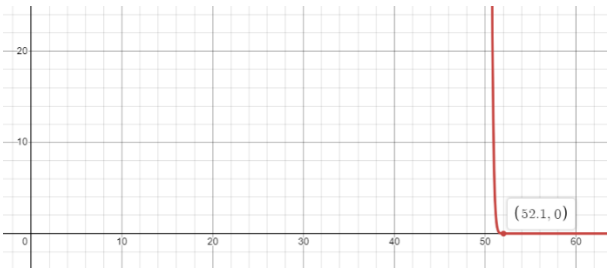
Utilize each desired pair to eliminate as many wrong keys as possible by following these steps for each desired pair:

For all possible keys $2^{\#nibbleUsed * nibbleSize} = 2^{16}$ that correspond to an active nibble of K_0, compute (X, X`) which are the outputs of the first round of encryption as in section 5.c.

Subsequently, compute $X \oplus X`$, if it is equal to the ΔX_1 of the impossible characteristic, then the key used in the encryption is an incorrect key and can be eliminated from the key space. This can occur with a probability 2^{-4}, because we have only one active nibble in ΔX_1. By following characteristic [1] found in section 5.c, there are four possibilities for this active nibble which makes the probability to be equal to $4x\ 2^{-4} = 2^{-2}$.

By analysing all 2^{x-46} desired pairs, we can eliminate all incorrect keys when the following equation holds:

$$2^{16}(1 - 2^{-2})^{2^{x-46}} = 0 \qquad (7.1)$$



Fig(11)- the initial plaintext chunk equation of SemiAES

Upon plotting the previous equation as in Fig(11), it is evident that $2^{52.1}$ is the minimum chunk size for the initial plaintext pairs that will result in a 100% success rate.

8. Results

In this section, we will delve into the implementation of each step of the attack and the various methods used to enhance them. Subsequently, we will explore the execution results of the attack on a 5-round version of QuadriAES-32 and SemiAES-64.

The attack comprises three primary steps:

1. Collection of chosen plaintext pairs.
2. Identification of the desired pairs.
3. Discarding the incorrect keys.

The chosen plaintext pairs chunk necessary for the attack is $2^{26.24}$ for QuadriAES and $2^{52.1}$ for SemiAES (refer to section 5.c, 7.c). Given our limited resources of 6 GB of RAM, we are unable to retain these chunks in memory to perform all three steps. Consequently, we will execute each step separately and then store the results in files to proceed further.

a. Collection of chosen plaintext pairs results:

A straightforward algorithm for this step would be as follows:

Algorithm 1 Collecting the chosen plaintexts

```
procedure COLLECTCHOSENPLAINTEXTSS(ΔX, chunkSize)
    chosenPlaintexts ← {}
    while chosenPlaintexts.size() < chunkSize do
        choose a random plaintext pair p1, p2 where p1 ⊕ p2 = ΔX
        pair ← ""
        if p1 < p2 then
            pair ← p1 + p2
        else
            pair ← p2 + p1
        end if
        chosenPlaintexts ← chosenPlaintexts ∪ {pair}
    end while
    return chosenPlaintexts
end procedure
```

Employing Algorithm (1), random pairs of plaintexts (P, P`) will be selected, when their difference $\Delta P$ corresponds to the $\Delta X$ in the characteristics mentioned in sections 5 and 7. We will then concatenate P and P` ensuring that the smaller plaintext comes first, and add them to the chosen plaintext set. This method of unified concatenation

ensures that we do not add both P, P` and P`, P to the set as they are essentially the same pair.

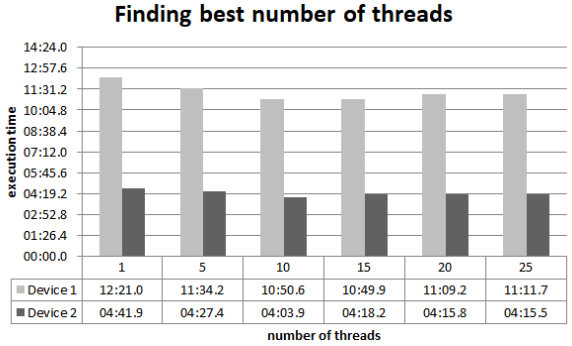
We will continue this generation process until the chosen plaintext set reaches the desired chunk size, and then write this set to a file to perform the second step of the attack

To expedite this process, we recommend leveraging threads and dividing the desired chunk size across them. Given the hardware-dependent nature of the thread count, our proposed approach will be assessed using two distinct devices, as described by their specifications in the following table:

Table 1: Devices Specifications

Specification	Device #01	Device #02
CPU	Intel Xeon E5-2620 v4 (2 processors)	Intel Core i5-2450M
Clock Speed	2.10 GHz	2.50 GHz
RAM	6 GB	8 GB
Physical Cores	8 (Threads: 16)	2 (Threads: 4)

We conducted an experiment to find the best thread count for each device by generating a chosen plaintext set for QuadriAES, containing 300,000 plaintext pairs, while utilizing the thread counts of {1, 5, 10, 15, 25}. A graphical representation of the results of the two devices is enlisted in Fig(12), illustrating the impact of each thread count on execution time:



Fig(12)- Finding the best thread count for each device

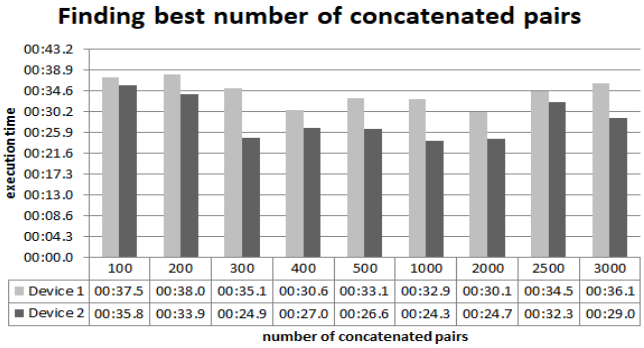
Fig(12) indicates that the optimal thread count is 15 for device 1 and 10 for device 2.

Moreover, the extended execution time may be attributed to the process of individually writing each plaintext pair to the file. This necessitates 300 thousand file write operations, with each operation writing only 4 characters due to the 4-character block size of QuadriAES, resulting in high inefficiency. To mitigate this, we recommend implementing Algorithm (2), which involves concatenating multiple pairs before appending them to the file.

Algorithm 2 Writing plaintext set in a file using concatenated pairs method

```
procedure WRITEPLAINTEXTSET(plaintextSet, concatenatedPairsNum)
    create a plaintext file
    p1 ← ""
    p2 ← ""
    for pair in plaintextSet do
        append the pair's first half to p1
        append the pair's second half to p2
        blockNum ← blockNum + 1
        if blockNum=concatenatedPairsNum then
            append p1 and p2 to plaintext file
            p1 ← ""
            p2 ← ""
        end if
    end for
    if plaintextSet.Size()%concatenatedPairsNum==0 then
        append p1 and p2 to plaintext file
    end if
end procedure
```

For instance, by concatenating 100 plaintexts together and then performing the write operation, we can reduce the required number of writing operations by a factor of 100. The optimal number of concatenated pairs may vary depending on the hardware. Therefore, we conducted an experiment by generating a chosen plaintext set for QuadriAES of size 300 thousand and employing different numbers of concatenated pairs {100, 200, 300, 400, 500, 1000, 2000, 2500, 3000} on the two devices specified in Table 1. Additionally, we utilized 10 threads, which is considered the optimal number of threads for the two devices. The impact of varying the number of concatenated pairs on the execution time is demonstrated in the following figure:



Fig(13) - Finding the best number of concatenated pairs

From Fig(13) we can observe that the ideal number of threads is 2000 for device 1 and 1000 for device 2.

b. Identification of the desired pairs results:

A straightforward and efficient algorithm for identifying the desired pairs is presented in Algorithm 3. This involves encrypting all selected plaintexts from the initial step using a randomly chosen initial key. For each pair $\{(P_1,C_1),(P_2,C_2)\}$, we compute $C_1 \oplus C_2$. If the result equals ΔY of the employed characteristic, the pair is added to the list of desired pairs.

Algorithm 3 Encrypting desired pairs

```
procedure ENCRYPTDESIREDPAIRS(ΔY, chosenPlaintexts)
    choose a random initialKey
    desiredPairs ← []
    for plaintextPair in chosenPlaintexts do
        p1 ← first half of plaintextPair
        p2 ← second half of plaintextPair
        c1 ← encrypt p1 using initialKey
        c2 ← encrypt p2 using initialKey
        Δc ← c1 ⊕ c2
        if Δc == ΔY then
            pair ← {(p1, c1), (p2, c2)}
            desiredPairs.add(pair)
        end if
    end for
    return desiredPairs
end procedure
```

Once all desired pairs have been identified from the set of chosen plaintexts, we will save them to files using the concatenated pairs approach discussed in the previous section. To expedite this process further, we suggest employing multi-threading, dividing the chosen plaintext set among threads, rather than executing the process sequentially.

In Section 5.b, we discussed the utilization of two different ΔY values in the QuadriAES characteristic and its effect on the probability of finding desired pairs. We will test this hypothesis with five sets of chosen plaintexts, each comprising 3 million plaintexts, to evaluate the effect of using two ΔY values on the number of desired pairs and the corresponding success rate, as illustrated in Table 2.

Table 2: QuadriAES with different number of used output difference

Chunk name	Number of used output difference			
	1		2	
	#desired pairs	success rate	#desired pairs	success rate
chunk 1	43	15.62%	93	30.74%
chunk 2	35	12.80%	82	27.59%
chunk 3	34	12.29%	81	27.52%
chunk 4	41	15.02%	100	32.74%
chunk 5	56	**19.48%**	104	**33.29%**

From Table 2, it is evident that using two ΔY values increases the success rate from a maximum of 19.48% (with a single ΔY) to a maximum of 33.29%.

Furthermore, in Section 7.b, we discussed the effect of using four different ΔY values in the SemiAES characteristic on the success rate. To assess this, we will experiment with five sets of chosen plaintexts, each consisting of 25 million plaintexts, to determine the effect of using four ΔY values on the desired pairs and the resultant success rate, as shown in Table 3.

Table 3 demonstrates that employing four ΔY values boosts the success rate from a maximum of 6.48% (with a single ΔY) to a maximum of 20.28%.

Table 3: SemiAES with different number of used output difference

Chunk name	Number of used output difference							
	1		2		3		4	
	#desired pairs	success rate	#desired pairs	success rate	#desired pairs	success rate	#desired pairs	success rate
chunk 1	181	**6.48%**	361	11.79%	518	16.33%	673	19.41%
chunk 2	164	4.23%	299	6.94%	480	15.39%	655	**20.28%**
chunk 3	165	4.49%	346	9.19%	520	13.95%	715	19.09%
chunk 4	177	5.76%	343	10.27%	516	14.90%	684	17.74%
chunk 5	182	6.18%	336	10.20%	519	14.05%	701	17.16%

c. Discarding the incorrect keys results:

A straightforward algorithm for this step is outlined in Algorithm 4. In this algorithm, we iterate through all possible keys for the active nibbles of ΔX. For each key, we loop through the desired pairs identified in the previous step and encrypt only the first round for the plaintexts of each pair. If the computed ΔP_1 matches ΔX_1 of the impossible characteristic, the key is identified as incorrect and discarded from the key space.

Algorithm 4 Discarding the wrong keys form the key space

```
procedure DISCARDWRONGKEYS(desiredPairs, ΔX, ΔX1)
    generate keySpace according to active nibbles in ΔX
    discardedKeys ← {}
    for key in keySpace do
        for pair in desiredPairs do
            get p1, p2 from pair
            encrypt first round for p1, p2 and compute ΔP1
            if ΔP1 == ΔX1 then
                discardedKeys ← discardedKeys ∪ {key}
                break
            end if
        end for
    end for
    RemainingKeys ← KeySpace \ discardedKeys
    return RemainingKeys
end procedure
```

Upon completing the iteration through all possible keys, the remaining keys in the key space are returned. The success rate is then calculated as follows:

$$\text{success rate} = 100 * \frac{2^{activeBits} - remaining\ keys\ num}{2^{activeBits} - 1}$$

To expedite this process, we suggest employing multi-threading, dividing the key space among the threads.

In Section 5.c, we discussed the utilization of two different ΔX_1 values in the QuadriAES characteristic to enhance the probability of discarding incorrect keys. To test this hypothesis, we will experiment with five lists of desired pairs, each derived from a chunk of 5 million chosen plaintexts. The impact of using two ΔX_1 values on the success rate and execution time is shown in Table 4.

Table 4: QuadriAES with different number of first round output difference

Chunk name	Number of used first round output difference			
	1		2	
	success rate	execution time	success rate	execution time
chunk 1	44.58%	11:02.1	69.19%	10:14.3
chunk 2	44.41%	10:38.7	69.19%	10:12.5
chunk 3	**46.02%**	10:45.2	70.44%	10:14.2
chunk 4	45.90%	10:48.8	**70.57%**	09:54.9
chunk 5	42.08%	10:05.4	66.56%	08:45.8

From Table 4, it is evident that using two ΔX_1 values increases the success rate from a maximum of 46.02% (with a single ΔX_1) to a maximum of 70.57%. Additionally, the execution time decreases even though no changes were made to Algorithm 4. This is because identifying incorrect keys more quickly allows us to exit the desired pairs loop sooner, avoiding unnecessary checks.

In Section 7.c, we also discussed using four different ΔX_1 values in the SemiAES characteristic. We will conduct experiments with five lists of desired pairs, each derived from a chunk of 25 million chosen plaintexts, to evaluate the effect of using four ΔX_1 values on the success rate and execution time, as shown in Table 5.

Table 5: SemiAES with different number of first round output difference

Chunk name	Number of used first round output difference							
	1		2		3		4	
	success rate	execution time	success rate	execution time	success rate	execution time	success rate	execution time
chunk 1	19.41%	27:11.2	33.08%	26:28.2	44.29%	09:49.2	56.14%	17:57.2
chunk 2	**20.28%**	50:59.5	33.83%	16:40.6	46.10%	06:22.7	56.36%	05:36.0
chunk 3	19.09%	39:27.2	34.60%	20:06.6	47.75%	11:58.2	**57.88%**	08:48.5
chunk 4	17.74%	33:32.0	31.38%	17:08.3	42.40%	11:11.1	52.60%	10:58.4
chunk 5	17.16%	32:23.5	33.41%	19:59.0	44.91%	11:41.0	55.84%	03:44.4

From Table 5, we see that using four ΔX_1 values increases the success rate from a maximum of 20.28% (with a single ΔX_1) to a maximum of 57.88%. Furthermore, the execution time is significantly reduced from approximately 51 minutes to just about 9 minutes.

d. Performing the full attack on 5-round version of QuadriAES-32:

The number of chosen plaintext pairs required to achieve a 100% success rate is $2^{26.24}$ (refer to section 5.c). Given the constraint of 6 GB of RAM, as specified for device 1 in Table 1, we are unable to retain a chunk of this size in memory.

Therefore, we suggest implementing the attack incrementally. Instead of attempting to process all chosen plaintext pairs at once, we divide them into smaller chunks, each containing 5 million pairs. Each chunk, treated as an independent attack on the cipher, is executed separately. These smaller chunks are encrypted to generate the required pairs, which are then used to eliminate incorrect keys. However, since all plaintext pairs share the same ΔX and are encrypted with the same randomly selected initial key, the results from each chunk can be combined. By doing this, we can effectively simulate an attack on a much larger chunk. This incremental approach allows us to manage memory constraints while still achieving the desired results. Algorithm 5 is used to merge the results of these smaller chunks into a comprehensive attack.

Algorithm 5 merges the desired pairs and the remaining keys from each chunk. Additionally, we incrementally merge these chunks to observe the effect of each additional chunk on the overall results. This involves iterating from 1 chunk to the total number of chunks, computing the possible combinations of each size. When the total number of chunks reaches 40, the number of possible combinations can become extremely large, potentially in the millions. To address this, we provide an option to generate all possible combinations or to set a limit, obtaining only the initial combinations up to the specified limit.

For each computed combination of size k, we merge the results of the chunks whose indexes are included in the combination.

Algorithm 5 Joining the discard wrong keys results

```
procedure JOINRESULTS(ΔX, remainingKeysLists, desiredPairsLists,
                      , generateAll, limitVal)
    for k=1 to desiredPair.size() do
        combinatations ← {}
        if generateAll then
            combinations ← choose all chunk combinations of size k
        else
            combinations ← choose chunk combinations of size k while
                           not exceeding limitVal for a single combination
        end if
        for combination in combinations do
            x ← combination[1]
            desiredPairs ← desiredPairsLists[x]
            remainingKeys ← remainingKeysLists[x]
            for y=2 to combintaion.size() do
                x ← combination[y]
                D ← desiredPairsLists[x]
                desiredPairs ← desiredPairs ∪ D
                R ← remainingKeysLists[x]
                remainingKeys ← remainingKeys ∩ R
            end for
        end for
    end for
    write the number of desired pairs and remaining keys to a statistics file
end procedure
```

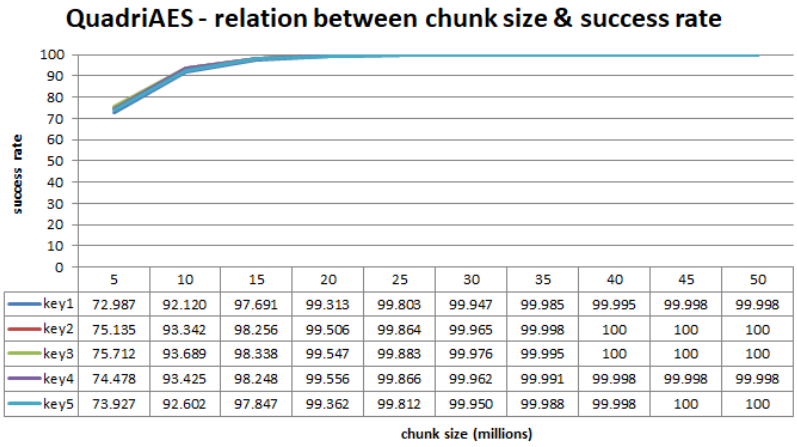
The desired pairs are merged by computing the union of each chunk's desired pairs list. This ensures that the attack corresponding to chunk 1 and chunk 2 is represented by the combined desired pairs of chunks 1 and 2, avoiding repetition by adding them to a set of desired pairs.

The remaining keys are merged by computing the intersection of each chunk's remaining key set. If any chunk within the combination discards a key, that key is also discarded from the remaining key set of the combined chunks.

Utilizing all methods for improving each step of the attack, as discussed in Sections 7.a, 7.b, and 7.c, we attack QuadriAES using 1000 concatenated pairs, 10 threads, two ΔY, and two ΔX_1 . We employ 10 chunks of size 5 million, encrypt them with 5 randomly selected initial keys, and combine their results to achieve the success rates shown in Fig(14).

From Fig(14), it is evident that three out of five keys are revealed using only 9 chunks, with two of them being revealed using only 8 chunks. Thus, a 100% success rate can be achieved with a chunk size of 8x5 million chosen plaintext pairs.

The average execution time to complete the attack for each set of 10 chunks encrypted by one of these five keys is shown in Table 6. The average overall time to complete an attack with 10 chunks is 3 hours and 50 minutes, while the average time to complete the attack for the 8 chunks required to achieve 100% success for keys 2 and 3 is 3 hours and 4 minutes.



	5	10	15	20	25	30	35	40	45	50
key1	72.987	92.120	97.691	99.313	99.803	99.947	99.985	99.995	99.998	99.998
key2	75.135	93.342	98.256	99.506	99.864	99.965	99.998	100	100	100
key3	75.712	93.689	98.338	99.547	99.883	99.976	99.995	100	100	100
key4	74.478	93.425	98.248	99.556	99.866	99.962	99.991	99.998	99.998	99.998
key5	73.927	92.602	97.847	99.362	99.812	99.950	99.988	99.998	100	100

Fig(14)- QuadriAES

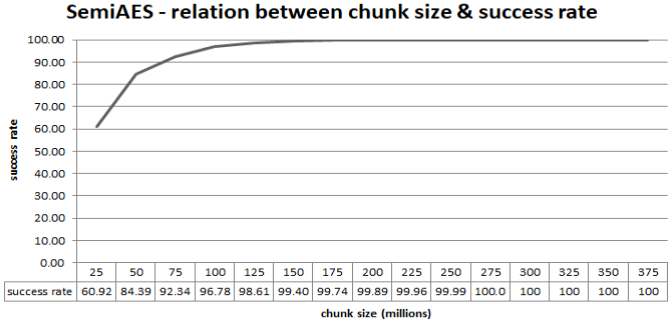
Table 6: QuadriAES execution time

step	all 10 chunks' execution time	each chunk's average execution time
collecting chosen plaintexts	0:11:24.810	0:01:08.481
encrypting desired pairs	2:07:20.759	0:12:44.076
discarding wrong keys	1:31:46.265	0:09:10.626
total	**3:50:31.834**	**0:23:03.183**

e. Performing the full attack on a 5-round version of SemiAES-64:

Employing the methodologies outlined in the preceding section, we will execute an attack on SemiAES using 1000 concatenated pairs, 10 threads, four ΔY, and four ΔX_1. This involves utilizing 15 chunks, each containing 25 million plaintext pairs. These chunks are encrypted using a single randomly selected initial key, and their results are combined to derive the subsequent success rates, as illustrated in Fig(15).

From Fig(15), it is discernible that a 100% success rate can be attained with only 11 chunks, necessitating a chosen plaintext chunk of size 11*25=275 million pairs.



	25	50	75	100	125	150	175	200	225	250	275	300	325	350	375
success rate	60.92	84.39	92.34	96.78	98.61	99.40	99.74	99.89	99.96	99.99	100.0	100	100	100	100

Fig(15)- SemiAES results

The corresponding execution times for all 15 chunks are presented in Table 7, indicating an overall execution time of 39 hours and 25 minutes. For the subset of 11 chunks, required to achieve a 100% success rate, the execution time is 28 hours and 54 minutes.

Table 7: SemiAES execution time

step	all 15 chunks' execution time	each chunk's average execution time
collecting chosen plaintexts	3:52:41.040	0:15:30.736
encrypting desired pairs	19:54:58.211	1:19:39.881
discarding wrong keys	15:37:46.408	1:02:31.094
total	**39:25:25.659**	**2:37:41.711**

9. Discussion

In this study, we presented two new derivations of the AES cipher, called QuadriAES and SemiAES, which operate with a smaller block size of 32 bits and 64 bits, respectively. Subsequently, we attacked a 5-round version of them using impossible differential cryptanalysis.

Constraining our computational resources to only 6GB of RAM, we adopted a systematic approach wherein each phase of the cryptanalysis was executed independently, with the outcomes recorded to files using the concatenated pairs algorithm. This suggested algorithm effectively minimized the number of writing operations. Additionally, we examined the effect of utilizing multiple output differences within the impossible characteristic on the number of desired pairs. Similarly, by employing multiple first-round output differences, we increased the probability of discarding incorrect keys.

Furthermore, we optimized the resource utilization by applying multi-threading to each step of the attack.

The cryptanalysis of QuadriAES and SemiAES was conducted in a incremental manner, entailing the execution of the entire attack on multiple chosen plaintext pairs chunks, followed by the combination of their results. This iterative approach facilitated the assessment of the incremental impact of each additional chunk on the overall success rate.

Our work offers significant value to aspiring cryptanalysts, particularly those confronted with the formidable challenge of conducting cryptanalysis on encryption standards with substantial memory requirements. By developing methodologies that reduce the memory footprint while maintaining high success rates, we provide a pathway for cryptanalysts to navigate these challenges more effectively.

In future works, we plan to extend the methodologies and algorithms described in this study to attack QuadriAES-48, QuadriAES-64, SemiAES-96, and SemiAES-128. Moreover, we intend to explore the feasibility of employing limited computational resources for the cryptanalysis of the standard AES algorithm.

Refrences

[1] Pub, NIST FIPS. "197: Advanced encryption standard (AES)." *Federal information processing standards publication* 197.441 (2001): 0311.https://doi.org/10.6028/NIST.FIPS.197-upd1

[2] Biham, Eli, Alex Biryukov, and Adi Shamir. "Cryptanalysis of Skipjack reduced to 31 rounds using impossible differentials." *Advances in Cryptology—EUROCRYPT'99: International Conference on the Theory and Application of Cryptographic Techniques Prague, Czech Republic, May 2–6, 1999 Proceedings 18*. Springer Berlin Heidelberg, 1999.https://doi.org/10.1007/3-540-48910-X_2

[3] Biham, Eli, and Nathan Keller. "Cryptanalysis of reduced variants of Rijndael." *3rd AES Conference*. Vol. 230. 2000.https://csrc.nist.rip/encryption/aes/round2/conf3/papers/35-ebiham.pdf

[4] Cheon, Jung Hee, et al. "Improved impossible differential cryptanalysis of Rijndael and Crypton." *Information Security and Cryptology—ICISC 2001: 4th International Conference Seoul, Korea, December 6–7, 2001 Proceedings 4*. Springer Berlin Heidelberg, 2002.https://doi.org/10.1007/3-540-45861-1_4

[5] Phan, Raphael C-W. "Impossible differential cryptanalysis of 7-round Advanced Encryption Standard (AES)." *Information processing letters* 91.1 (2004): 33-38.https://doi.org/10.1016/j.ipl.2004.02.018

[6] Kakarla, Sourya, et al. "On the practical implementation of impossible differential cryptanalysis on reduced-round AES." *International Conference on Applications and Techniques in Information Security.* Singapore: SpringerSingapore, 2017.https://doi.org/10.1007/978-981-10-5421-1_6

[7] Pal, Debranjan, et al. "An Efficient Parallel Implementation of Impossible-Differential Cryptanalysis for Five-Round AES-128." *International Conference on Security, Privacy, and Applied Cryptography Engineering*. Cham:

Springer International Publishing, 2019.https://doi.org/10.1007/978-3-030-35869-3_9

[8] Pal, Debranjan, et al. "A cluster-based practical key recovery attack on reduced-round AES using impossible-differential cryptanalysis." *The Journal of Supercomputing* 79.6 (2023): 6252-6289.https://doi.org/10.1007/s11227-022-04872-y

[9] Phan, Raphael Chung-Wei. "Mini advanced encryption standard (mini-AES): a testbed for cryptanalysis students." *Cryptologia* 26.4 (2002): 283-306.https://doi.org/10.1080/0161-110291890948

[10] Phan, Raphael Chung-Wei. "Impossible differential cryptanalysis of Mini-AES." *Cryptologia* 27.4 (2003): 361-374.https://doi.org/10.1080/0161-110391891964