Design and implementation of proposed 320 bit RC6-cascaded encryption/decryption cores on altera FPGA

Ashwaq T. Hashim, Ahmed M. Hasan, Haider M. Abbas
Department of Control and Systems Engineering, University of Technology, Iraq

Article Info

Article history:

Received Jan 4, 2020
Revised May 19, 2020
Accepted May 27, 2020

Keywords:

Block cipher
Cascaded design
Cryptography
FPGA design
RC6

ABSTRACT

This paper attempts to build up a simple, strong and secure cryptographic algorithm. The result of such an attempt is "RC6-Cascade" which is 320-bits RC6 like block cipher. The key can be any length up to 256 bytes. It is a secret-key block cipher with precise characteristics of RC6 algorithm using another overall structure design. In RC6-Cascade, cascading of F-functions will be used instead of rounds. Moreover, the paper investigates a hardware design to efficiently implement the proposed RC6-Cascade block cipher core on field programmable gate array (FPGA). An efficient compact iterative architecture will be designed for the F-function of the above algorithm. The goal is to design a more secure algorithm and present a very fast encryption core for low cost and small size applications.

Corresponding Author:

Ahmed Mudheher Hasan,
Department of Control and Systems Engineering,
University of Technology,
Sinaa Street, Baghdad-Iraq.
Email: 60163@uotechnology.edu.iq

1. INTRODUCTION

One of the most cryptographic elements is the symmetric-key block cipher applied widely in information security. In addition to its data confidentially purpose, it has been used as a main element to construct various cryptographic systems like pseudo random number generators, protocols, stream ciphers, message authentication and hash functions. It is one among various symmetric-key block ciphers used to provide security efficiently and with flexibility. There are some types such as Blowfish, Twofish, RC6, Rijndael, Mars, Serpent and DES, which have been received with the greatest practical interest [1-4].

Implying 256 distinct message blocks through choosing a small value of *m* (i.e. *m*=8) results in a low security system, since attackers can try to extract the plaintext from the ciphertext through frequency analysis if they have a basic knowledge of the statistical patterns in the plaintext. Moreover, attackers can build a codebook for the plaintext-ciphertext pairs that corresponds to a particular key if they have guessed the plaintext that relates to their ciphertext [5, 6].

Fortunately, the above problems have been overcome through increasing the block length that leads to increasing the number of possible messages. As a result, the attacker is unable to avail knowledge plaintext patterns that are shorter than the block length. Therefore, it is not useful for the attacker to fabricate a codebook.

The message block length (*m*) is normally chosen to be at least 64 bits. It is very important for it to be impractical for the attacker with a known plaintext (*P*, *C*), to be able to recover the key through exhaustive key-search (i.e. brute-force). Therefore, it is vital to make the determination of the key difficult when trying values of k ⊂ K until the key is found so that C = E (P; k). Basically, in order to make the brute-force attack ineffective and the key more secure, the key should be longer [7]. Nowadays, designers are encouraged to design with high-speed and high level of security required to implement the hardware cryptographic algorithms. Modern applied cryptographic algorithms exhaust high processing power, and thus, this issue is considered as a bottleneck in ultra-high-speed networks. Continued growth in the size and functionality of field programmable gate array (FPGA) over the last decade has resulted in an increasing trend in their use for private-key cryptographic algorithms. Private-key cryptographic algorithms are difficult to be achieved on a serial processor. This is due to the large data to be used when increasing the message block length (*m*).

FPGA structure is capable of exploiting spatial and temporal parallelism. Thus, it matches the essential operation for private-key cryptographic algorithms (i.e. Boolean functions, look-up table, and bit-substitutions). On one hand, more than one operation can be executed concurrently at cryptographic-round level. On the other hand, synchronous processing of multiple blocks of data performed in block-cipher level as well [8-10].

This paper is arranged as follows: Section 2 presents some of the previous works. Section 3 gives a concise introduction of RC6 algorithm. Section 4 briefly describes the proposed RC6-Cascaded system. Section 5 describes the architecture of the proposed Altera FPGA design. Section 6 shows the experimental and implementation results and finally, section 7 draws the conclusion observed from the results.

2. RELATED WORKS

Various symmetric-key block ciphers have been available offering various levels of security, flexibility, and effectiveness. In 2008, Krishnamurthy et al [11] proposed a secret-key block cipher that utilizes precious features of CAST-128 and Blowfish algorithms. Ashwaq [12] in 2009 introduced a block cipher which makes use of four 32-bit plaintext data as input and generates four 32-bit ciphertext data words. The cipher is word-oriented in that all the interior operations are carried out on 32-bit words. This algorithm is a type-3 Feistel network repeating the simple function 16 times. In 2010, Ashwaq et al [13] proposed a symmetric key block cipher known as '512 bits RC6' which is an evolutionary enhancement of the RC6 block cipher designed to get performance enhancement and increase security. The inner loop design is adapted from 128-bit RC6 round. In addition, Anand et al [14] in 2012 proposed a cipher of 512 bit for secure communication. It is improved based on a design principle known as substitution permutation network. While in Krishnamurthy [11], the VHDL implementation is utilized and carefully tested to manifest the percentage improvement in the performance of the modified Blow-CAST-Fish block cipher. Faez et al [15] implemented a traditional RC6 block cipher that includes the encryption and decryption on FPGA Vertix II device with highly solid architecture through reutilizing the same units for the comparable operation in both algorithms.

Moreover, Mardiana et al [16] address efficiently the modification for implementing the RC6 block cipher algorithm for a digital image and describe the design and performance of the algorithm for color images and grayscale as well and it reduce the capacity (i.e. size) of variables to improve the algorithm in terms of memory consumed and time required.

Actually, advance encryption standard (AES) and Data Encryption Standard (DES) are widely used in block ciphers. AES is a modified version of Rijndael algorithm, which is fast in software and flexible to be implemented in hardware but suffers from brute force attack [17-19]. Meanwhile, DES is currently considered to be insecure for many applications due to its small key size. Moreover, according to today’s computing power, DES is not sufficient and may be exposed to huge force attacks [20]. Therefore, it is no longer suitable for security. For that reason, Chanchal et al, [21] present a blowfish cryptography algorithm to replace DES algorithm, which is a secure and fast block cipher algorithm, and offers a good performance. However, it suffers from weak keys problem. Nowadays, RC6 is a promising invented block cipher algorithm to replace the previous cipher algorithms due to the above-mentioned weakness. RC6 is presented as a powerful block cipher to be a new consideration of RC5.

There exists a considerable body of work done in the field of cryptography on embedded systems such as digital signal processors (DSP), microcontroller (µc), application specific integrated circuits (ASIC), and field programmable gate array (FPGA). From performance point of view, there is a trade-off between throughput and area utilization. Since the iterative architecture is a resource efficient approach consisting of simply one round. Therefore, the proposed hardware implementation using FPGA achieves iterative architecture.

3. RC6 ENCRYPTION ALGORITHM

RC6 is one of the finalists block cipher algorithm in the advanced encryption standard (AES) competition. The RC6 algorithm extends from its predecessor RC5. Since RC6 is the evolution of RC5, evolutionary differences will be observed accordingly. RC6 has a 128-bit block size and supports key sizes of 128, 192, and 256 bits up to 2040 bits.

RC6, like RC5, includes three components: the key expansion algorithm, the encryption and decryption algorithms. The parameter is viewed in the following specification: RC6-*w/r/b*, where *w* is the size of word, *r* is the non-negative number of rounds, and *b* is the length in byte of the encryption key. RC6 makes use of data-dependent rotations and it is based on seven primitive operations. Actually, there are only six primitive operations, while the parallel assignment is primitive and considered a basic operation to RC6. The subtraction, addition, as well as multiplication operations use two's complement representations for execution. While the integer multiplication is utilized to increase diffusion per round and boost the speed of the cipher. Figure 1 describes the overall structure of the RC6 encryption process [22, 23].

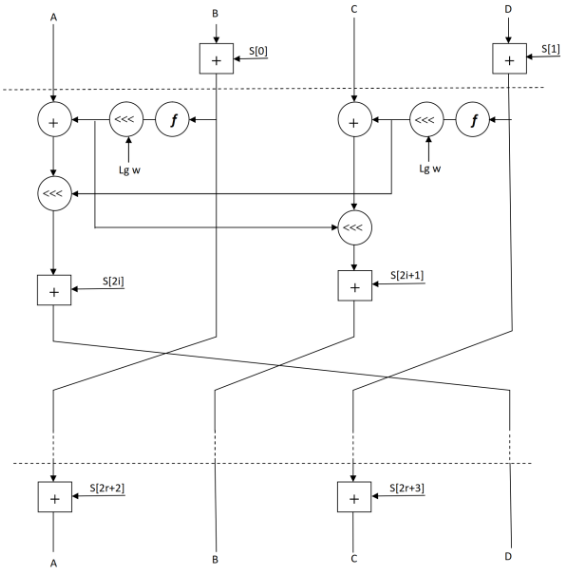

Figure 1. Overall structure of the RC6 algorithm [22]

4. THE PROPOSED SYSTEM

The proposed algorithm is 'RC6-Cascade' which is an RC6-like block cipher. The plaintext is 320 bit divided into five parts p_1, p_2, p_3, p_4 and p_5 each of which is 64 bits. The F-function will use cascaded design instead of round as it shown in Figure 2. The output is 320 bit c_1, c_2, c_3, c_4, and c_5 each of which is 64 bits.

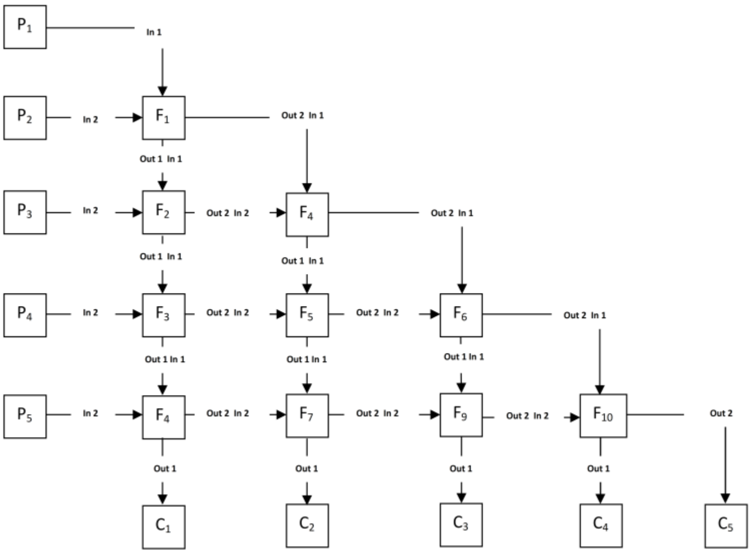

Figure 2. The proposed 320 bit RC6-cascade

4.1. The F-function of proposed algorithm

The F-function of the proposed algorithm uses two rounds of RC6-like algorithms in a Feistel network. The input of each function is two plaintexts of 64 bits instead of four subkeys S_i,S_{i+1},S_{i+2},S_{i+3} of 16 bits as shown in Figure 3. The Feistel network included splitting the input into two halves, and implementing a non-linear function only to the right half. Meanwhile, the result has been added to the left half, and subsequently, left and right half have been exchanged. Ciphers subsequenced in this approach are known as Feistel ciphers. In these ciphers, the result of one non-linear function is feed forward to the next one, where the propagation of local changes has been increased.

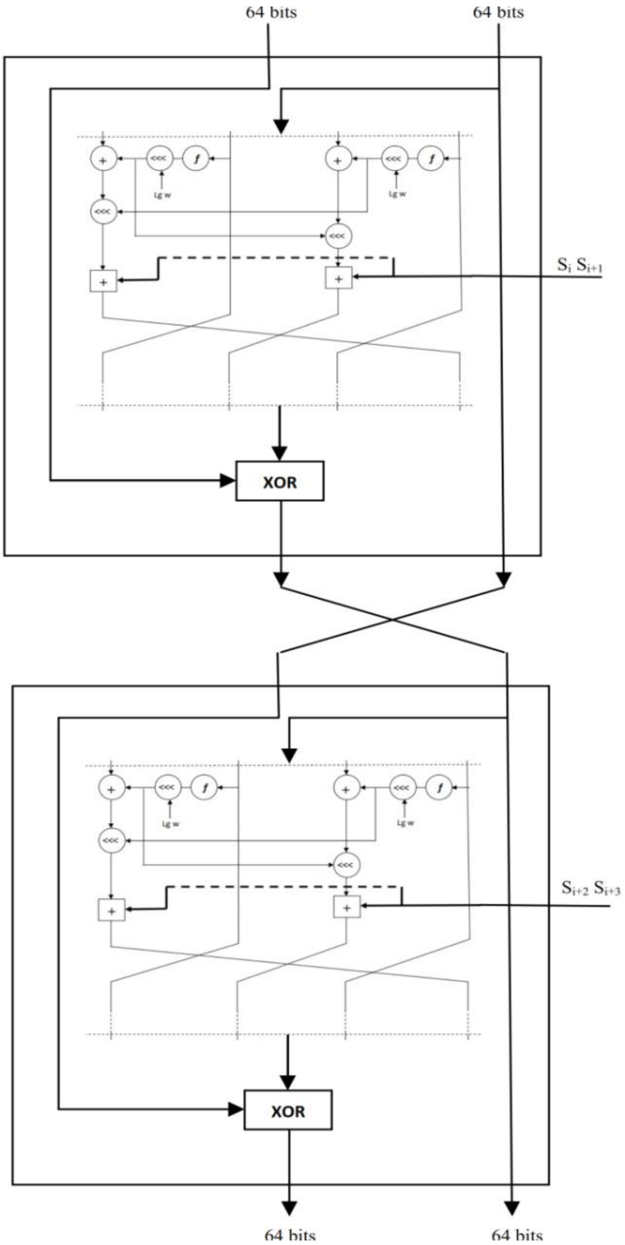

Figure 3. The F-function of proposed algorithm

4.2. The proposed algorithm

The key schedule of the proposed algorithm is basically identical to the key schedule of RC6 with some modifications. The user supplies a key of *b* bytes, where $0 \leq b \leq 255$. From this key, 4×10 subkeys (i.e, 4 subkeys for each F-function) are deduced and saved in the array S [0, 40]. The conserved array is utilized in both encryption and decryption.

5. FPGA DESIGN FOR THE PROPOSED SYSTEM

The hardware I/O specification for the proposed 320 bits Cascaded_RC6 encrypter and decrypter were shown in Figure 4 and Figure 5. It requires a PLAIN/CIPHER TEXT of 320 bits length. Also, a KEY of 128 bits length is required to be provided. The necessary control signals are START ENCRYPTION, START DECRYPTION, START KEY GENERATION, and ENCRYPTION/DECRYPTION. The mentioned signals are necessary to control the timely sequencing of the modules. This controlling is done via a controller that is interfaced with the Encrypter and Decrypter as shown in Figure 6 and Figure 7.

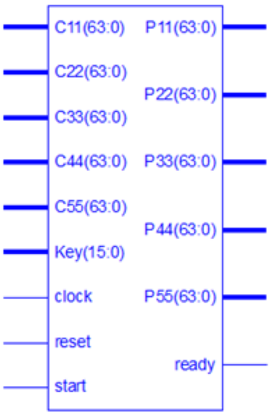

Figure 4. The RTL schematic of cascaded_RC6 encryptor

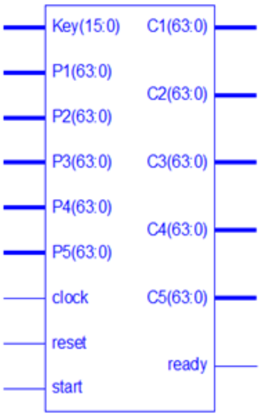

Figure 5. The RTL schematic of cascaded_RC6 decryptor

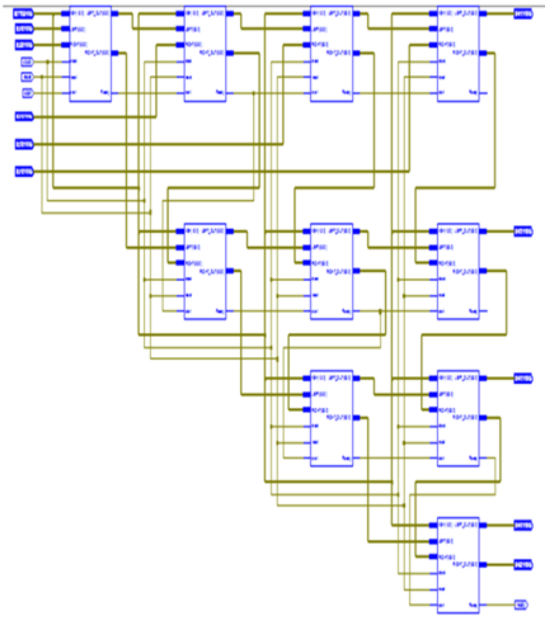

Figure 6. Encryptor diagram of the proposed system

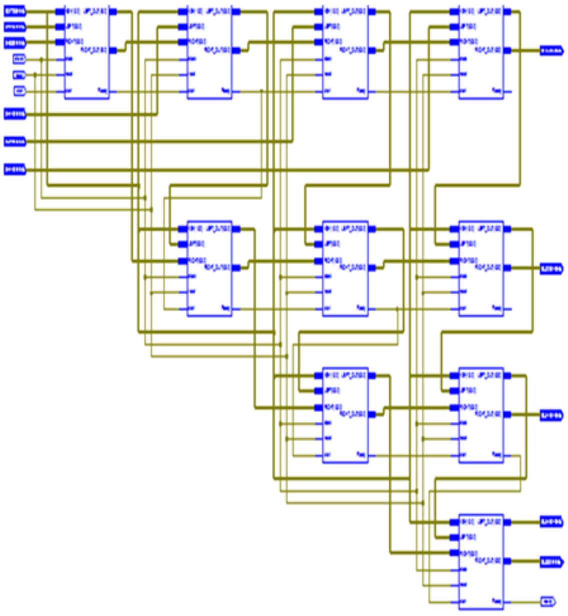

Figure 7. Decryptor diagram of the proposed system

6. EXPERIMENTAL RESULTS

6.1. Dictionary attacks

Here, the dictionary will require 2^{128} different texts since the block size is 128 bits. Therefore, a block size of 128 bits is required to allow the attacker to encrypt or decrypt arbitrary messages under an unknown key, while the proposed algorithm requires a 2^{320} plaintext. This attack applies to any arbitrary block encryption with 128-bit blocks in any design case.

6.2. Avalanche effect

Avalanche effect is an important feature of the encryption algorithm. This property can be noticed when altering one bit in plaintext and then seeing the change in the outcome of at least half of the bits in the ciphertext. One of the purposes of the avalanche effect is that by altering a single bit there is massive change, so, it is difficult to fulfill an analysis of ciphertext, when seeking to come up with an attack. This section is dedicated to perform statistical tests on the ciphertext that result from encrypting 16 blocks of 320 bits as plaintexts as input to the proposed system and 16 blocks of 128 bits to the previous RC6 algorithm in hexadecimal, such as the following:

1. **0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000**

2. **1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111**

3. **2222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222
2222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222
2222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222**

4. **3333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333
3333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333
3333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333**

5. **4444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444
4444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444
4444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444**

6. **5555555555555555555555555555555555555555555555555555555555555555555555555555555555555555555555555555555555
5555555555555555555555555555555555555555555555555555555555555555555555555555555555555555555555555555555555
5555555555555555555555555555555555555555555555555555555555555555555555555555555555555555555555555555555555**

7. **6666666666666666666666666666666666666666666666666666666666666666666666666666666666666666666666666666666666
6666666666666666666666666666666666666666666666666666666666666666666666666666666666666666666666666666666666
6666666666666666666666666666666666666666666666666666666666666666666666666666666666666666666666666666666666**

8. **7777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777
7777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777
7777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777
77**

9. **8888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888
8888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888
8888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888**

10. **9999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999
9999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999
9999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999**

11. **aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa**

12. **bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbbbbbbnbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb**

13. **ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc**

14. **ddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddd
ddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddd
ddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddd
dddddddddddddddddddddddddddddddd**

15. **fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff**

The previous RC6 algorithm used a key length of 16 bytes (128 bits) and its input is 'ffffffffffffffff', while the key length of the proposed system is 32 bytes (256 bits) and its input is 'ffffffffffffffffffffffffffffffff'. Table 1 show the avalanche effect on plaintext when only one bit has been altered in the key after applying the previous 128-bit RC6 algorithm. At the same time, Table 1 lists the avalanche effect on the plaintext when only one bit of the key has been altered after applying the proposed 'Cascade-RC6' algorithm.

Table 1 show that the change was from 0.43 to 0.52 bits from 128 bits and 0.49 to 0.54 bits from 320 bits when the algorithm was implemented before and after the algorithm was improved. Table 1 show that the average of the avalanche effect of the previous RC6 algorithm is 0.48. On the other hand, it shows that the average of the avalanche effect of the proposed algorithm is 0.51. In addition, all the encrypted blocks except one have 50% avalanche effect after applying the proposed system.

Table 1. Avalanche effect of classical and proposed cascaded RC6 algorithm after changing one bit in key

Block no	Avalanche Effect	
	Classical RC6 algorithm	Proposed Cascaded RC6 algorithm
1	(0.44) 56	(0.51) 163
2	(0.50) 64	(0.53) 169
3	(0.51) 65	(0.50) 164
4	(0.47) 60	(0.49) 159
5	(0.46) 59	(0.52) 166
6	(0.48) 61	(0.50) 160
7	(0.47) 60	(0.49) 158
8	(0.48) 62	(0.51) 164
9	(0.52) 66	(0.51) 164
10	(0.45) 58	(0.50) 159
11	(0.48) 61	(0.51) 163
12	(0.50) 64	(0.52) 165
13	(0.43) 55	(0.49) 158
14	(0.52) 67	(0.54) 172
15	(0.48) 62	(0.50) 161
Average	(0.48) 61.34	(0.51) 163

6.3. Plaintext-ciphertext independence

The independence between the plaintext and ciphertext has been tested via the following steps:

- Test a large number of random plaintexts P_r where r = 1.....R and R is the number of plaintext.
- The selected plaintext should be encrypted using a key (K) in order to obtain the corresponding ciphertext (Cr).
- Use the plaintext (Pr) and the ciphertext (Cr) to generate the independent blocks (INr) through the equation:

$$I_{Nr}=P_r \oplus C_r \tag{1}$$

The algorithm to generate the independent blocks is as follows:

```
F1=plaintext.txt
F2=ciphertext.txt
For i= 1 to R do
    For j = 1 to n do
         Getbit(F1,b1)
         Getbit(F2,b2)
         If b1= b2
         Then
              IN-ARY[i,j]='0'
        Else
             IN-ARY[i,j]='1'
        Endif
   End for
End for
```

where *n* is the block length and *R* is the total number of blocks.

The distinct blocks must be random to assign that the ciphertext is independent to the plaintext [18, 24]. These tests focus on different non-randomness types that could exist in series. The most widely used of National Institute of Standards and Technology (NIST) tests are [25]:

- Test for the Longest-Run-of-Ones in a Block,
- Frequency Test within a Block,
- The Serial Test,
- The Frequency (Monobit) Test,
- The Cumulative Sums Test,
- The Runs Test,
- The Linear Complexity Test,
- The Approximate Entropy Test.

In addition, for each statistical test, a set of P values (i.e., the test value) is generated corresponding to the set of previous RC6 output sequences. The sequence is considered to be passed at the value of the statistical test result value P ≥ α, otherwise, it is considered a failure. The parameter α indicates the level of importance that locates the area of acceptance and rejection. In our work we set α = 0.01 as referenced by NIST. Table 2 lists the results of the randomness test of 20 blocks of a size of 128 bits, which result after being encrypted by the previous RC6 algorithm. Table 3 lists the values of encrypted blocks after applying the proposed algorithm. We notice that the P-values listed in Table 3 are greater than *α* value. Therefore, the proposed system output has passed NIST statistical test suite. While, some P-values listed in Table 2 failed to pass randomness tests.

Table 2. NIST statistical tests after applying RC6 algorithm (the size of sequence is *n*=128 bits)

Frequency (Monobit)	Frequency Test within a Block M=16	Runs Test	Longest Run of Ones	Cumulative Sums	Serial M=3	Approximate Entropy M=2	Linear Complexity M=8
0.0037	0.3216	0.1537	0.0274	0.0027	0.0248	0.0288	0.0356
0.0265	0.0021	0.3577	0.0432	0.0027	0.0316	0.0459	0.0650
0.0182	0.0693	0.0550	0.0311	0.0543	0.0145	0.0397	0.0437
0.0041	0.0075	0.0804	0.0591	0.0137	0.0045	0.0095	0.0607
0.0107	0.0730	0.0036	0.0207	0.0069	0.0125	0.0218	0.0069
0.0186	0.0308	0.1142	0.0221	0.0162	0.0166	0.0106	0.0254
0.0339	0.0302	0.2008	0.0265	0.0081	0.0025	0.0046	0.0040
0.0129	0.0189	0.0242	0.0211	0.0107	0.0040	0.0060	0.0257
0.0412	0.4306	0.0110	0.0251	0.0182	0.0257	0.0002	0.0964
0.0124	0.0024	0.0756	0.0234	0.0762	0.0064	0.0227	0.0019
0.0059	0.0509	0.1149	0.0113	0.0082	0.0219	0.0309	0.0033
0.0897	0.0234	0.0794	0.0519	0.0014	0.0333	0.0303	0.0200
0.0485	0.0495	0.0070	0.0534	0.0097	0.0054	0.0264	0.0111
0.0038	0.0642	0.2168	0.0571	0.0239	0.0138	0.0032	0.0177
0.0857	0.0599	0.0036	0.0826	0.0499	0.0138	0.0190	0.0356
0.0072	0.0088	0.0955	0.0729	0.0314	0.0296	0.0189	0.0359
0.0867	0.0306	0.0887	0.0782	0.0927	0.0422	0.0100	0.0299
0.0288	0.0472	0.0926	0.0488	0.0459	0.0337	0.0119	0.0123
0.0028	0.1914	0.0025	0.0155	0.0097	0.0296	0.0094	0.0176
0.0159	0.0052	0.0010	0.0471	0.0159	0.0469	0.0037	0.0108

Table 3. NIST statistical tests after applying proposed algorithm (the size of sequence is *n*=320 bits)

Frequency (Monobit)	Frequency Test within a Block M=16	Runs Test	Longest Run of Ones	Cumulative Sums	Serial M=3	Approximate Entropy M=2	Linear Complexity M=8
0.3796	0.2657	0.4945	0.3079	0.3154	0.8826	0.8966	0.5438
0.5763	0.8333	0.3937	0.2457	0.5549	0.6764	0.7652	0.7613
0.4152	0.3219	0.8202	0.2873	0.2312	0.9243	0.7531	0.2886
0.6798	0.8093	0.5599	0.3558	0.1874	0.6068	0.8447	0.6392
0.8231	0.2318	0.9591	0.4922	0.7817	0.8593	0.9483	0.4131
1.0003	0.8094	0.3598	0.5517	0.7982	0.8596	0.9609	0.0942
0.9596	0.1877	0.9592	0.7514	0.9345	0.8592	0.9894	0.8315
1.0003	0.8091	0.8595	0.6513	0.2892	0.8598	0.9927	0.9857
0.6597	0.1872	0.2888	0.8632	0.4779	0.9242	0.8692	0.6881
0.7597	0.7833	0.4453	0.7957	0.5742	0.8689	0.8407	0.4135
0.7239	0.2018	0.7757	0.8287	0.7378	0.9392	0.8543	0.1367
0.9594	0.7032	0.2249	0.6523	0.8522	0.1966	0.3362	0.5912
0.7592	0.2839	0.0499	0.4476	0.3447	0.9842	0.2277	0.4981
0.8595	0.7033	0.0483	0.2338	0.2438	0.8827	0.1978	0.3045
0.5593	0.2838	0.0317	0.2439	0.4713	0.8163	0.1422	0.1543
0.2577	0.7038	0.0483	0.4531	0.2857	0.7785	0.1826	0.5676
0.2158	0.2835	0.2108	0.2637	0.4718	0.4362	0.3979	0.3741
0.2572	0.7039	0.1069	0.7879	0.2934	0.6066	0.2992	0.7617
0.7158	0.2834	0.2832	0.2471	0.1546	0.6768	0.4871	0.7853
0.5754	0.4839	0.5948	0.4594	0.1964	0.8822	0.9305	0.1364

6.4. Time requirement

In this section, the time requirements are computed for the proposed algorithm and the previous 128 bits RC6 algorithm. Table 4 shows this test.

Table 4. Time comparisons of proposed algorithm and previous RC6 algorithm

Algorithm	Number of Bytes	Time in Second
RC6 algorithm	10000	0.087
	100000	0.76
	1000000	0.95
Proposed algorithm	10000	0.0017
	100000	0.0156
	1000000	0.153

6.5. Hardware testing

The design has been tested and verified using Xilinx ISE Foundation tool. All encryption and decryption designs are first coded in Verilog and then synthesized by using the Xilinx XST synthesis tool. At the end, the behavior of the designed systems is verified by ISE simulator through assigning the test vectors. Figure 8 and Figure 9 show the details of FPGA implementation results of the encryption and decryption respectively.

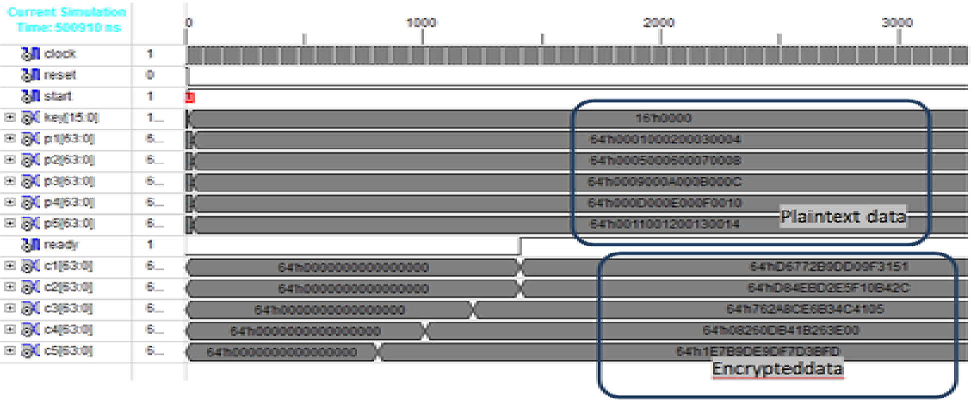

Figure 8. Encryption simulation waveforms

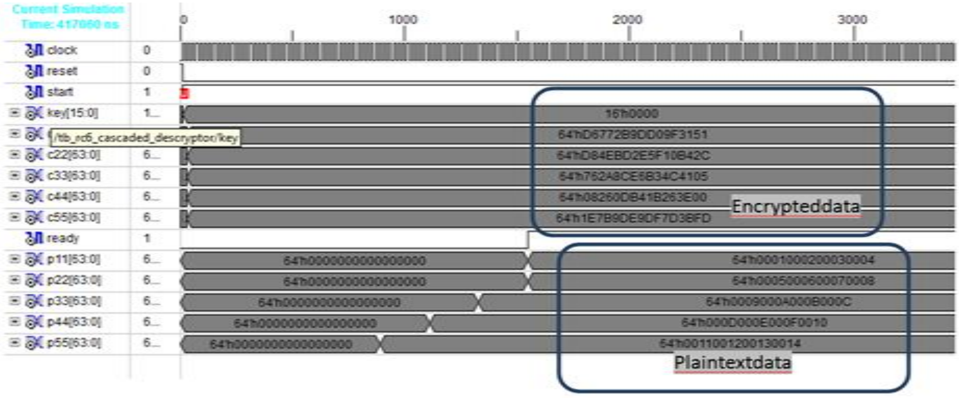

Figure 9. Decryption simulation waveforms

7. CONCLUSION

The proposed algorithm is a simple, compact and very secure block cipher. It comes out with good performance and a much improved security/performance over RC6 cipher by increasing the block size to 320 bits while retaining the simplicity in implementation and design. The proposed system increases the complexity for attackers. Computer network security ensures a secure information exchange and enables communication through the internet. Therefore, we proposed an FPGA based RC6-cascaded encryption/decryption scheme aiming at a high throughput and compact design for low cost applications.

The Proposed algorithm can be adjusted for processors of variable word-lengths. For example, as 64-bit processors become available, it should be possible for RC6-Cascade to exploit their longer word length. Consequently, the number w of bits in a word is a parameter of RC6-Cascade. Diverse choices of this parameter result in diverse RC6-Cascade algorithms.

ACKNOWLEDGEMENTS

The authors wish to thank Dr. Mohanad Dawood for his thoughtful comments and especially about the design of FPGA for the proposed system.

REFERENCES

[1] E. M. De L. Reyes, et al., “Modified AES Cipher Round and Key Schedule,” *Indonesian Journal of Electrical Engineering and Informatics (IJEEI)*, vol. 7, no. 1, pp. 28-35, 2019.

[2] H. V. Gamido, et al., “Modified AES for Text and Image Encryption,” *Indonesian Journal of Electrical Engineering and Computer Science (IJEECS)*, vol. 11, no. 3, pp. 942-948, 2018.

[3] N. A. N. Hashim, et al., “Memristor based ring oscillators true random number generator with different window functions for applications in cryptography,” *Indonesian Journal of Electrical Engineering and Computer Science (IJEECS)*, vol. 14, no. 1, pp. 201-209, 2019.

[4] M. Aljohani, et al., “Performance Analysis of Cryptographic Pseudorandom Number Generators,” *IEEE Access*, vol. 7, pp. 39794-39805, 2019.

[5] A. Abouchouar, et al., “New concept for cryptographic construction design based on noniterative behaviour,” *IAES International Journal of Artificial Intelligence (IJ-AI)*, vol. 9, no. 2, pp. 229-235, 2020.

[6] F. Noorbasha, et al., “FPGA Design and Implementation of Modified AES Based Encryption and Decryption Algorithm,” *International Journal of Innovative Technology and Exploring Engineering (IJITEE)*, vol. 8, no. 6S, pp. 132-136, 2019.

[7] R. Oppliger, “Contemporary Cryptography,” *Artech House*, INC, 2005.

[8] S. Abed, et al., “FPGA Modeling and Optimization of a SIMON Light Weight Block Cipher,” *Sensors*, vol. 19, no. 4, pp. 1-28, 2019.

[9] S. Dessai and Sandeep G., “Embedded Hardware Circuit and Software Development of USB based Hardware Accelerator,” *International Journal of Reconfigurable and Embedded Systems*, vol. 7, no. 1, pp. 21-33, 2018.

[10] Shailaja A. and Krishnamurthy G. N., “FPGA Implementation And Analysis Of RC7 Algorithm Using Reversible Logic Gates,” *International Journal of Engineering and Advanced Technology*, vol. 8, no. 6, pp. 769-776, 2019.

[11] Krishnamurthy G. N., et al., “Blow-CAST-Fish: A New 64-bit Clock Cipher,” *IJCSNS International Journal of Computer Science and Network Security*, vol. 8, no. 4, pp. 282-290, 2008.

[12] A. T. Hashim, “Type-3 Feistel Network of The 128-bits Block Size Improved Blowfish Cryptographic Encryption,” *Engineering and Technology Journal*, vol. 27, no. 2, pp. 235-246, 2009.

[13] A. T. Hashim, et al., “A Proposal 512-bits RC6 Encryption Algorithm,” *IJCCCE*, vol. 10, no. 1, pp. 11-25, 2010.

[14] M. A. Kumar and S. Karthikeyan, “Investigating the Efficiency of Blowfish and Rejindael (AES) Algorithms,” *International Journal of Computer Network and Information Security (IJCNIS)*, vol. 2, pp. 22-28, 2012.

[15] F. F. Shareef, et al., “Compact Hardware Implementation of FPGA Based RC6 Block Cipher,” *Journal of Engineering and Applied Sciences*, vol. 3, no. 7, pp. 589-601, 2008.

[16] Mardiana, et al., “Modification of RC6 Block Cipher Algorithm on Digital Image,” *Journal of Physics: Conference Series, International Conference on Information and Communication Technology*, vol. 930, pp. 1-6, 2017.

[17] Z. Hercigonja, et al., “Comparative Analysis of Cryptographic Algorithms,” *International Journal of Digital Technology and Economy*, vol. 1, no. 2, pp. 127-134, 2016.

[18] E. M. De L. Reyes, et al., “File encryption based on reduced-round AES with revised round keys and key schedule,” *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 16, no. 2, pp. 897-905, 2019.

[19] S. U. Jonwal and P. P. Shingare, “Advanced Encryption Standard (AES) implementation on FPGA with hardware in loop,” *International Conference on Trends in Electronics and Informatics (ICOEI 2017)*, 2017.

[20] K. Kamalam, “A Survey on Various Algorithmic Methods for Encryption and Decryption,” *International Advanced Research Journal in Science, Engineering and Technology*, vol. 4, no. 5, pp. 83-89, 2017.

[21] C. D. Pande and S. S. Mungona, “High Speed Data Cryptography Technique of Blowfish Algorithm using VHDL,” *International Journal on Recent and Innovation Trends in Computing and Communication*, vol. 4, no. 12, pp. 113-116, 2016.

[22] R. L. Rivest, et al., “The RC6TM Block Cipher,” *First Advanced Encryption Standard (AES) Conference*, Ventura, CA, 1998.

[23] R. Soni and S. Singh, “FPGA Implementation of Optimized the 64-BIT RC5 & RC6 Cryptography Encryption Algorithm,” *International Journal of Recent Technology and Engineering (IJRTE)*, vol. 4, no. 2, pp. 68-71, 2015.

[24] R. Anderson, et al., “Serpent: A Proposal for the Advanced Encryption Standard,” *First Advanced Encryption Standard (AES) Conference*, Ventura, CA, 1998.

[25] L. E. Bassham, et al., “A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications,” *NIST Special publication (NIST SP) - 800-22 Rev 1a*, 2010.