A Versatile Pipelined Hardware Implementation for Encryption and Decryption using Advanced Encryption Standard

Nadia Nedjah[1] and Luiza de Macedo Mourelle[2]

[1] Department of Electronics Engineering and Telecommunications,
Faculty of Engineering, State University of Rio de Janeiro, Brazil
nadia@eng.uerj.br
[2] Department of Systems Engineering and Computation,
Faculty of Engineering, State University of Rio de Janeiro, Brazil
ldmm@eng.uerj.br

Abstract. The Advanced Encryption System – AES is now used in almost all network-based applications to ensure security. In this paper, we propose a very efficient pipelined hardware implementation of AES-128. The design is versatile as it allows both encryption and decryption. The core computation of AES, which is performed on data blocks of 128 bits, is iterated for several rounds, depending on the key size. The security strength of AES has been proven proportional to the number of rounds applied. we show that if the required number of rounds must increase to defeat attackers, the proposed implementation stays efficient.

1 Introduction

Cryptographic algorithms used by nowadays cryptosystems fall into two main categories: symmetric key and asymmetric-key algorithms [8]. Symmetric-key ciphers use the same key for encryption and decryption, or to be more precise, the key used for decryption is computationally easy to compute given the key used for encryption. In turn, symmetric-key ciphers, fall into two categories: block ciphers and stream ciphers. Stream ciphers encrypt the plaintext one bit at a time, in contrast to block ciphers, which operate on a block of bits of a predefined length. Most popular block ciphers are DES, IDEA [7] and AES, and most popular stream cipher is RC6 [9].

The Advanced Encryption System – AES is a block cipher, adopted as the new encryption standard in substitution to its predecessor Data Encryption Standard – DES [2]. AES main scrambling computation is performed on a fixed block size of 128 bits with a key size of 128, 192 or 256 bits. This core computation is iterated for many rounds. The number of rounds depends on the key size. Currently, it is set to 10, 12 and 14 for the cited keys sizes respectively. The resistance of AES against breaking attacks depends entirely on the number of rounds used. So far, the best known attacks are on 7 rounds for 128-bit keys, 8 rounds for 192-bit keys, and 9 rounds for 256-bit keys [5]. The small

margin between these round numbers and the actual ones is very worrying for the cryptographer's community.

In this paper, we propose a novel hardware implementation of AES-128. The architecture allows one to perform the core computation of the algorithm is a pipelined manner. The throughput of the cryptographic hardware is 1Gbits per second. A unique hardware is used for encryption and decryption. The pipelined encryption and decryption allows an increase of the number of rounds without much loss of efficiency. Recall that increasing the number of rounds applied, increases the resistance of the AES algorithm.

This rest of this paper is organised in 4 subsequent sections. First, in Section 2, we give a brief description of the AES encryption and decryption algorithms as well as the modified version of these two algorithms, which are the basis of the proposed hardware architecture. Thereafter, in Section 3, we describe in a structured manner, the pipelined hardware architecture of AES-128 for encryption and decryption. Subsequently, in Section 4, we present some experimental result and compare our implementation to existing ones. Last but not least, in Section 5, we draw some conclusions and introduce some directions for future work.

2 Advanced Encryption Standard

AES is an elegant and a so-far-secure cipher. Encryption using AES proceeds as described in Algorithm 1, wherein functions *SubBytes*, *ShiftRows*, *MixColumns* and *AddroundKey* are defined as follows:

- Function *SubBytes* yields a new state simply by substituting each of the 16 bytes of *state* using a substitution box. The four most significant bits of the byte in question is used as the S-box row index while the remaining four bits are used as the S-box column index.
- Function *ShiftRows* obtains a new state by cyclically shifting the state rows. The bytes of row i are shifted i times, where $0 \leq i \leq 4$.
- Function *MixColumns* operates on the states columns. The bytes of a given column are used as coefficients of a polynomial over GF(2^8). The formed polynomial is multiplied by a fixed polynomial $P(x)$ *modulo* $x^4 + 1$, wherein $P(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$. The details of the multiplication operation can be found in [3], [1].
- Function *AddRoundKey* computes the new state using a XOR of the columns bytes and the key schedule of the current round.

Before the cipher operation takes place, a key schedule is generated. Four subkeys are required for each round of the cipher algorithm. The subkeys for the first round are the private cipher key. For a given round, the first subkey is obtained by first rotating once the last subkey form the previous round, then substituting each of byte using the S-box used by function *subBytes*, thereafter XORing the result with a given constant and finally XORing the result with first subkey of the previous round. The subsequent subkeys of the current round are

computed using a XOR of the previous key in the current round and the one inversely respective from the previous round.

Algorithm 1. AES-Cipher
input: Byte $T[4 \times nb]$, Word $K[nb \times (nr + 1)]$;
output: Byte $C[4 \times nb]$,
Byte $state[4, nb]$;
$state := T$;
AddRoundKey($state$, $K[0, nb - 1]$;
for $round := 1$ to $nr - 1$ do
SubBytes($state$);
ShiftRows($state$);
MixColumns($state$);
AddRoundKey($state$, $K[round \times nb, nb(round + 1) - 1]$);
SubBytes($state$);
ShiftRows($state$);
AddRoundKey($state$, $K[nr \times nb, nb(nr + 1) - 1]$);
$C := state$;
return C;
end

For hardware efficiency reasons, we modified the AES cipher algorithm as in Algorithm 2. Note that Algorithm 1 and Algorithm 2 are equivalent and yield the same output.

Algorithm 2. Modified-AES-Cipher
input: Byte $C[4 \times nb]$, Word $K[nb \times (nr + 1)]$;
output: Byte $T[4 \times nb]$,
Byte $state[4, nb]$;
$state := C$;
for $round := 0$ to $nr - 1$ do
AddRoundKey($state$, $K[round \times nb, nb(round + 1) - 1]$);
SubBytes($state$);
ShiftRows($state$);
if $round < nr - 1$ then MixColumns($state$);
AddRoundKey($state$, $K[nr \times nb, nb(nr + 1) - 1]$);
$T := state$;
return T;
end

The decryption of a text that was ciphered using AES can be performed by Algorithm 3. Comparing Algorithm 1 and Algorithm 3, one can note that each function was replaced by its inverse. However, the application sequence of these functions is slightly different. In order to have a unique versatile hardware for

encryption and decryption, this algorithm was modified as in Algorithm 4.

Algorithm 3. AES-Decipher
input: Byte $C[4 \times nb]$, Word $K[nb \times (nr + 1)]$;
output: Byte $T[4 \times nb]$,
Byte $state[4, nb]$;
$state := C$;
AddRoundKey($state$, $K[round \times nb, nb(nr + 1) - 1]$);
for $round := nr - 1$ downto 1 do
InvShiftRows($state$); InvSubBytes($state$);
AddRoundKey($state$, $K[nr \times nb, nb(nr + 1) - 1]$);
InvMixColumns($state$);
InvShiftRows($state$);
InvSubBytes($state$);
AddRoundKey($state$, $K[0, nb(nr + 1) - 1]$);
$T := state$;
return T;
end

Algorithm 3 and Algorithm 4 are equivalent as operations *InvSubBytes* and *InvShiftRows* commute. Moreover, function *InvMixColumns* is linear so we have expression *InvMixColumns*(x XOR y) is equivalent to *InvMixColumns*(x) XOR *InvMixColumns*(y). Recall that operation *AddRoundKey* is a XOR of its arguments. Using these two facts, we can swap operations *AddRoundKey* and *InvMixColumns*, provided that the columns of the decryption key schedule are modified using operation *InvMixColumns*. Note that functions *SubBytes* and *InvSusbytes* perm the same process but using distinct S-Boxes.

Algorithm 4. Modified-AES-Decipher
input: Byte $C[4 \times nb]$, Word $K[nb \times (nr + 1)]$;
output: Byte $T[4 \times nb]$,
Byte $state[4, nb]$;
$state := C$;
for $round := nr - 1$ to 0 do
AddRoundKey($state$, $K[round \times nb, nb(round + 1) - 1]$);
InvSubBytes($state$); InvShiftRows($state$);
if $round < nr - 1$ then InvMixColumns($state$);
AddRoundKey($state$, $K[nr \times nb, nb(nr + 1) - 1]$);
$T := state$;
return T;
end

3 Pipelined Hardware Implementation of AES

The overall architecture of the AES hardware mirrors the structure of Algorithm 2 and Algorithm 4. It is a synchronous implementation of both the processes of cipher and decipher. It uses four 128-registers. Every clock transition, these registers are loaded, except $Register_3$, which is loaded when an input state is completely ciphered. In the encryption/decryption process, $Register_0$ is loaded with the input data or the partially encrypted/decrypted plaintext/ciphertext; $Register_1$ with the result of the *AddRoundKey* component; $Register_2$ with the state after applying functions *SubBytes* (using the appropriate S-Box) and subsequently *ShiftRows*/*InvShiftRows*. The block architecture of the AES cipher and decipher hardware is shown in Fig. 1.

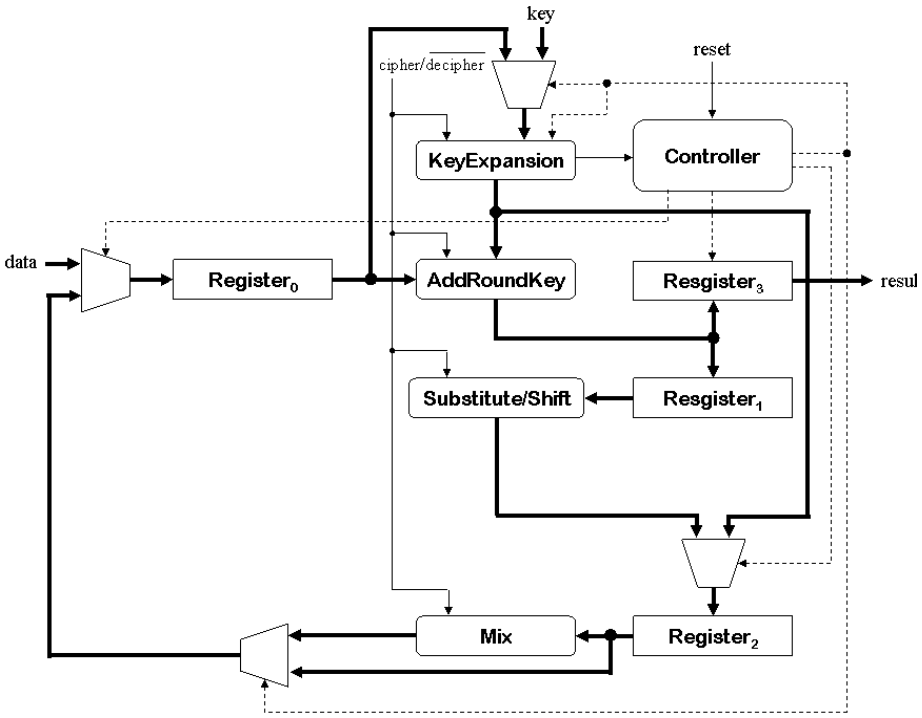

Fig. 1. Overall hardware architecture for the AES cipher/decipher

The component that implements function *AddRoundKey* is simply a net of XOR gates that adds in $GF(2^8)$ the key schedule to the current state. The component implementing function *SubBytes* uses 16 S-boxes (8 for ciphering and 8 for deciphering) stored in a Read-Only Memory (ROM). The obtained state is row-shifted before its storage in $Register_2$. The component architecture is given in Fig. 2.

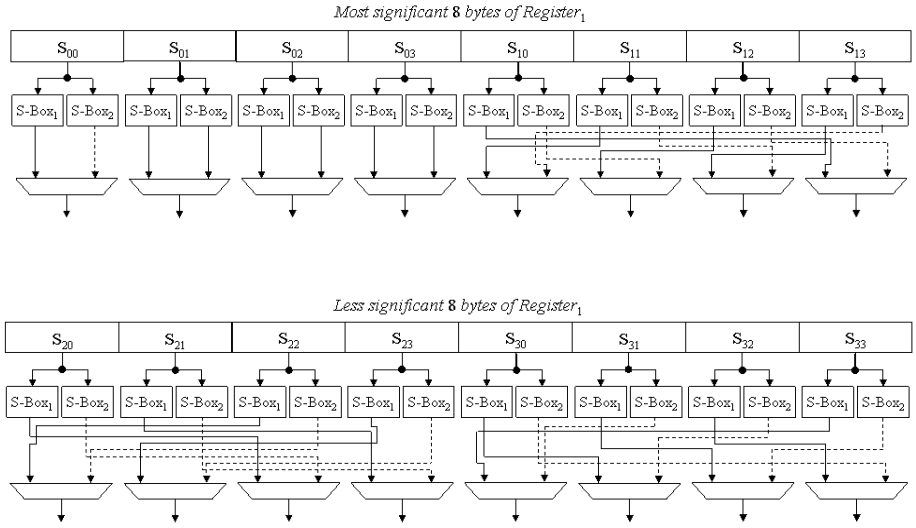

Fig. 2. The structure of *Substitute/Shif* component

Function *MixColumns* is implemented by a massively parallel component that computes all the bytes of the new state in a single clock. It uses four components of the same architecture. This basic component produces one column os the new state. Its architecture is described in Fig. 3, wherein component *mult* yields the a special product of a given byte from the state times {01}, {02}, {03}, {09}, {0B}, {0D} or {0E} (see [3], [1] for details on the operation). The architecture of component *mult* is presented in Fig. 4. Component *xtime* computes the *xtime* operation as defined in [3] and its architecture is given in Fig. 5.

For component synchronisation purposes, the architecture includes a controller. Among other actions, the controller determines when to reset the cipher hardware, accept input data, to register output results. As the excution of function *MixColumn*/*InvMixColumn* is conditional (see Algorithm 2), the controller decides when the result obtained by associated component can be used or must be ignored. Recall the hardware allows both encryption and decryption. When data is being deciphered, the key schedule generated by component *KeyExpansion* must be ordered differently [3]. The AES hardware of Fig. 1 takes advantage of component *MixColumn* to schedule the subkeys in the required order. The controller also controls this operation.

The controller is structured as in Fig. 6. The included combinational logic permits the conversion of the 5-bit count to a single bit that triggers state transition. The sate machine includes six states. As long as control signal *keyExpand* is set, the current state is kept unchanged in S_0. As soon as this signal is reset by the *keyExpansion* component, which means that the step of key schedule generation is complete, the machine transits to state S_1, wherein it stays for 3 clock cycles, which is the required time to complete the processing of one 128-bit

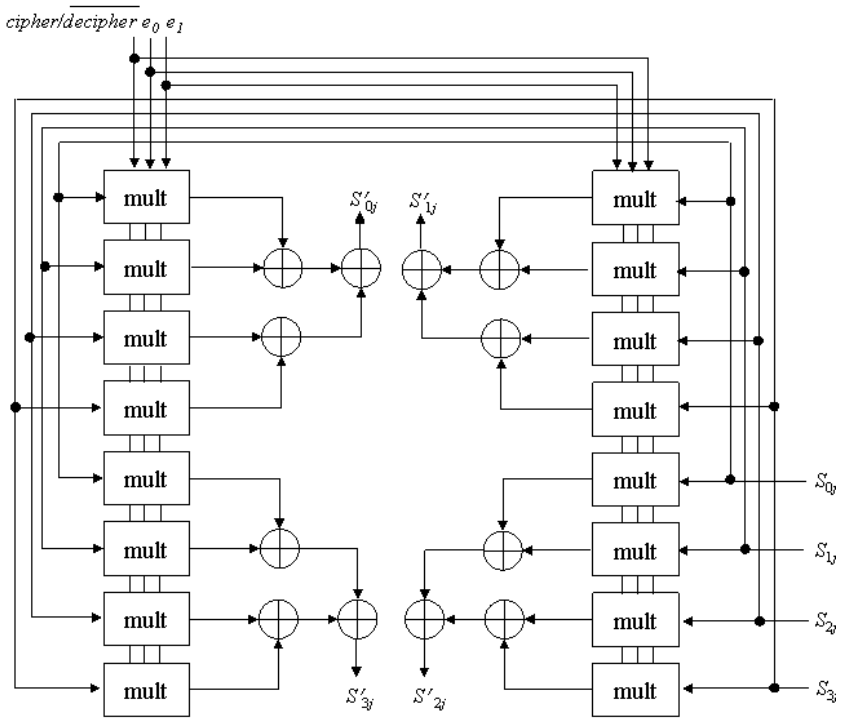

Fig. 3. Basic component in *Mix* component

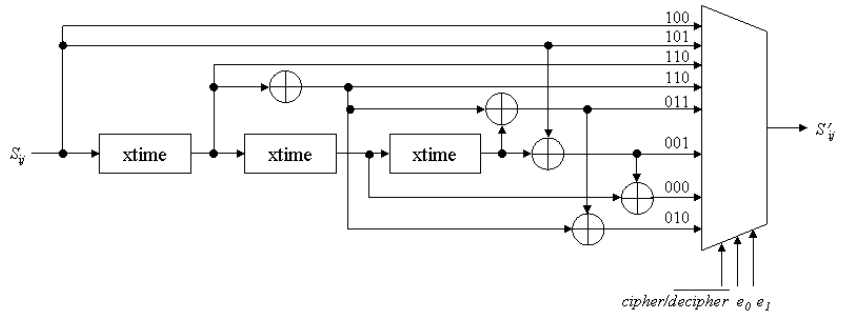

Fig. 4. Architecture of the *mult* component

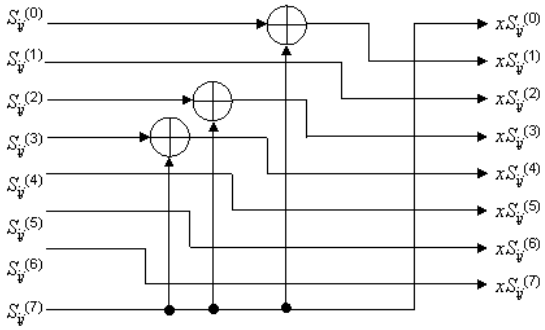

Fig. 5. Architecture of the *xtime* component

state. Also, during this period of time, the data input signal is active, which allows the hardware to accept the three states that will be ciphered/deciphered in pipelined manner. Synchronously with the fourth clock transition, the machine transits to state S_2 allowing to deactivate the data input signal and wait for the three accepted states are almost processed as only the last *AddRoundKey* is yet to be performed to complete the encryption/decryption process. At the 30th. clock transition, the machine state changes to S_3 to activate output result signal, which is maintained for the two subsequent clock periods. A the 33rd. clock transition, the encryption/decryption of the three accepted states is completed and therefore, the control is returned to state S_1, where in data input signal is reactivated to allow more date to be entered and processed. The state machine transition diagram is shown in Fig. 7.

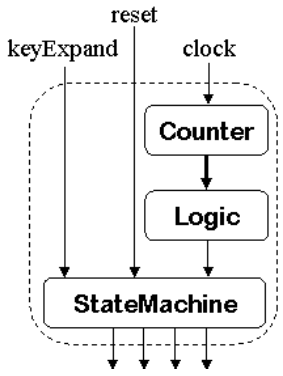

Fig. 6. Controller architecture

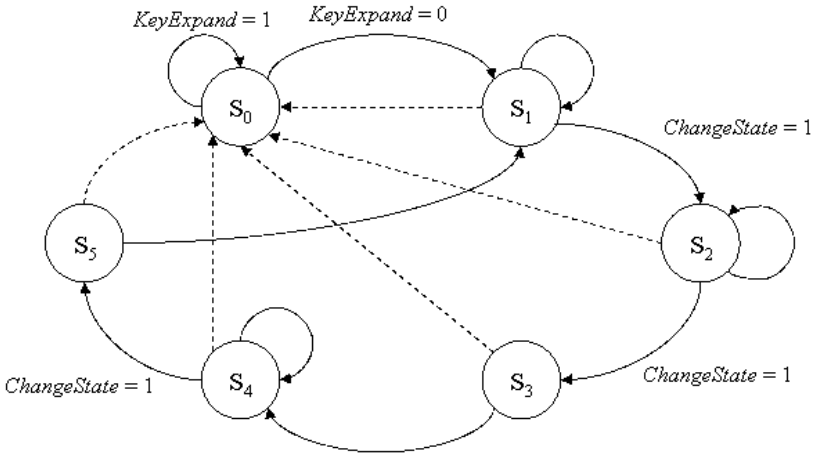

Fig. 7. State machine transition diagram

4 Experimental Results

The pipelined execution of the AES cipher using the architecture of Fig. 1 is illustrated in Fig. 8. We implemented the hardware described throughout this paper using reconfigurable hardware. The FPGA family used is VIRTEX-II. Component *KeyExpansion* introduces a delay of 78.3ns. The clock cycle is 10.44ns. Every 33 clock cycles, the hardware can yield an encrypted datastream of 3×128 bits. The throughput, say tp can then be calculated as in (1). The throughput is a little more than 1Gbps.

$$Tp = \frac{3 \times 128}{33 \times clockcycle} = \frac{128}{11 \times 10.44} = 1062.9Mbs \tag{1}$$

As far as the authors know, the versatile hardware implementation of AES algorithm that performs both encryption and decryption is novel. We compared our implementation to the ones from [6] and [10]. Note that these implementations are for the cipher algorithm only while our implementation ciphers and deciphers. One may think that the implementation proposed and those from [6] and [10] are incomparable. They are cited here for reference only. The throughput, expressed in Mbps, as well as the hardware area required, expressed in number of CLBs, are given in Table 1.

Recall that the resistance of AES-based encryption against cryptanalysis attacks depends entirely on the number of rounds used. The pipelined implementation we propose throughout this paper can be easily adapted to a higher round number. The chart of Fig. 9 shows that this can be done without much loss in efficiency and with much gain of security strength. To be able to increase the number of round, component *KeyExpansion* needs to generate more key schedules and therefore the delay introduced by it increases with the number of

Table 1. Performance comparison

Implementation	Throughput	Area	CLB/Mbs
Our's: cipher& decipher	1063	9937	9.35
[6]: cipher only	1911	8767	4.59
[10]: cipher only	1450	542	0.37

rounds. The throughput, say tp, can be expressed in terms of the round number, say rn, is as in (2). The security strength, say st is proportional to the number of rounds applied. So, considering the security strength provided by applying 10 rounds as a reference, st would be defined as in (3).

$$Tp(rn) = \frac{128}{(rn+1) \times clockcycle} \tag{2}$$

$$St(rn) = \frac{rn}{10} \tag{3}$$

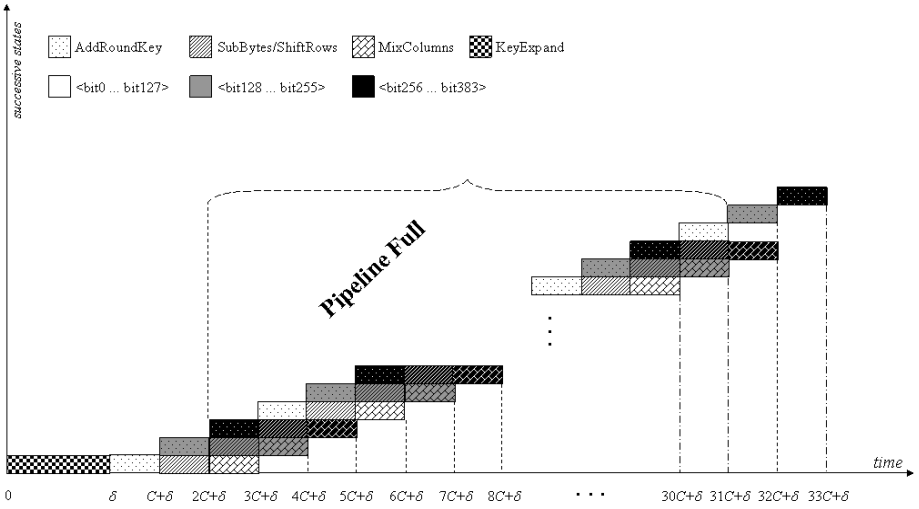

Fig. 8. Pipelined execution of the AES algorithm using the hardware of Fig. 1

5 Conclusion

In this paper, we propose a novel pipelined hardware implementation of AES-128 that can be used for both encryption and decryption. Besides, we show that if the required number of rounds must increase to defeat attackers, the

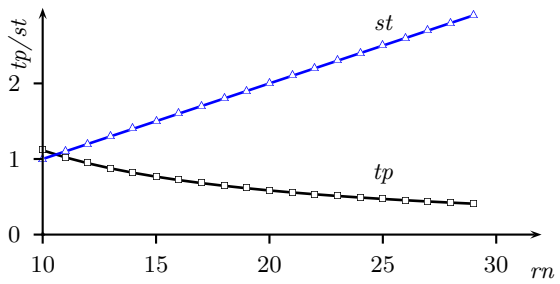

Fig. 9. The impact of increase in the round number

proposed implementation stays efficient. The hardware proposed is massively parallel and executes the four main steps of the algorithm in a pipelined manner, which allows a reasonable throughput fo a little more of 1Gbs. Compared to existing implementations of the cipher algorithm, this kind of throughput may be considered somehow low. However, considering the 2-in-1 aspect of the hardware as it allows encryption and decryption, it comes handy for devices with restricted hardware area with a not too bad throughput of 1Gbs.

In future research work, we intend to investigate further the proposed implementation, with the hope to improve the throughput without much increase in required hardware area.

References

1. J. Daemen and V. Rijmen, *The Design of Rijndael: AES – The Advanced Encryption Standard*, Springer-Verlag, 2002.
2. National Institute of Standard and Technology, *Data Encryption Standard*, Federal Information Processing Standards 46, November 1977.
3. National Institute of Standard and Technology, *Advanced Encryption Standard*, Federal Information Processing Standards 197, November 2001.
4. Nicolas Courtois, Josef Pieprzyk, *Cryptanalysis of Block Ciphers with Overdefined Systems of Equations*, Proceedings of ASIACRYPT 2002, pp 267-287, 2002.
5. N. Ferguson, J. Kelsey, S. Lucks, B. Schneier, M. Stay, D. Wagner and D. Whiting, *Improved Cryptanalysis of Rijndael*, Proceedings of FSE 2000, pp. 213-230, 2000.
6. A. Labbe, A. Perez, *AES Implementation on FPGA: Time and Flexibility Tradeoff*, in Proceedings of FPL, pp. 836-844, 2002.
7. X. Lai, J. L. Massey, *A Proposal for a New Block Encryption Standard*, EUROCRYPT'90, pp. 389–404, 1990.
8. A.J. Menezes, S.A. Vanstone and P.J. Van Oorschot, *Handbook of Applied Cryprography*, CRC Press, USA, 1997.
9. R. Rivest, M. Robshaw, R. Sidney, and Y.L. Yin. *The RC6 block cipher*, First AES Candidate Conference, 1998.
10. F. Standaert, G. Rouvroy, J. Quisquater, J. Legat, *A Methodology to Implement Block Ciphers in Reconfigurable Hardware and its Application to Fast and Compact AES RIJNDAEL*, in Proceedings of FPGA, 2003.