INFORMATION TECHNOLOGY JOURNAL

Information Technology Journal 12 (7): 1434-1438, 2013
ISSN 1812-5638 / DOI: 10.3923/itj.2013.1434.1438

An Improved Rijndael Encryption Algorithm Based on NiosII

Tang Jun and Wang Liejun
School of Information Science and Engineering, Xinjiang University, P.R. China

Abstract: Through the data storage of Rijndael encryption algorithm and the analysis of the characteristics of column mixing and wheel transformation, the author put forward some suggestions for the optimization of Rijndael. Meanwhile, the author design and realize the optimized Rijndael encryption through the SOPC (System on Program Chip). Based on the SOPC of Altera FPGA (Field Programmable Gata Array), the algorithm design is put forward and the center is the embedded soft core NiosII. It is also realized the encryption and decryption verification by using the ModelSim software. Compared with the traditional design, Rijndael encryption algorithm undergoes a great breakthrough. The author also evaluated the Rijndael encryption algorithm performance and logic resources occupation.

Key words: Rijndael, encryption algorithm, FPGA, NiosII

INTRODUCTION

As the new generation AES (Advanced Encryption Standard) cipher algorithm, the Rijndael algorithm has the advantages of high efficiency in implementation, low demand for storage and high speed of encryption and decryption (Liu *et al*., 2011; Zhang, 2007).The algorithm can be implemented through software as well as hardware. The software implementation is too slow although it is flexible. It is even more so with the block encryption algorithms getting more and more complex (Liang *et al*., 2011; Spillman, 2005). Besides, the security system is difficult to avoid the appearance of the key plaintext in the computer during the implementation through software, which may result in the plaintext being stolen or modified. On the contrary, the hardware implementation has the advantages of high speed, high reliability and much higher security, so that an increasing number of applications require the implementation through hardware (Zhang, 2008; Babu *et al*., 2012). The key of hardware implementation operates in the interior of module and the algorithm solidifies in the hardware, which can guarantee that the key of plaintext doesn’t flow outside and can achieve the true sense of confidentiality. The design based on "FPGA+NiosII" project, on the one hand, can configure CPU flexibly according to the application, on the other hand, it can realize the configuration of hardware logic directly in the FPGA interior (Liao and Chao, 2008; Cao, 2006). The software realization can be used for the peripheral device that has not so high demand for data transmission speed. While for the configuration that requires high speed, it can use the hardware interface to realize directly and then use the software to control it. In this study, the Rijndael algorithm is optimized comprehensively using Look-up Table and then the algorithm is designed systematically using "FPGA+NiosII" project (Lv, 2005; Wang, 2008).

OPTIMIZED DESIGN OF RIJNDAEL ALGORITHM

The discussion of data storage condition: The statement of Rijndael algorithm can be done as 4 bytes matrix. It is stored with the order of the priority list. As shown in Table 1a, A0~A15 represents the bytes stored in the matrix.

And in the most advanced languages, usually in the order of priority do store a two-dimensional array, so if in Table 1a to the order of the data storage, the access state of a list of the corresponding four times to visit. But if the matrix transpose store data form, as shown in Table 1b, the access state line (original one column) so only visit only once, so efficiency is greatly improved (Shen, 2006).

Optimization to realize S-box: Instead of table (or S-box) is reversible and is made up of two reversible transformation compound.

First, the finite field GF(2^8) take inverse, zero 00 at inverse provisions for 00;

Second, the inverse again after the definition (at GF (2) affine transformation role:

Corresponding Author: W. Liejun, School of Information Science and Engineering, Xinjiang University, P.R. China

Table 1(a-b): Status data storage

(a)			
A0	A4	A8	A12
A1	A5	A9	A13
A2	A6	A10	A14
A3	A7	A11	A15
(b)			
A0	A1	A2	A3
A4	A5	A6	A7
A8	A9	A10	A11
A12	A13	A14	A15

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1&1&1&1&1&0&0&0 \\ 0&1&1&1&1&1&0&0 \\ 0&0&1&1&1&1&1&0 \\ 0&0&0&1&1&1&1&1 \\ 1&0&0&0&1&1&1&1 \\ 1&1&0&0&0&1&1&1 \\ 1&1&1&0&0&0&1&1 \\ 1&1&1&1&0&0&0&1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \tag{1}$$

Realize the affine transformation, to realize the require relatively large amount of computation, through the observation, finding that can carry on the transformation:

$$y = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_0 \end{bmatrix} + \begin{bmatrix} x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_0 \\ x_1 \end{bmatrix} + \begin{bmatrix} x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_0 \\ x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_7 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \tag{2}$$
$$= x + (x << 1) + (x << 2) + (x << 3) + (x << 4) + '63'$$

This can be done by cyclic shift to achieve.

Optimization to realize column hybrid: Column hybrid transformation is the state of the column as a finite field GF (2^8) four dimensional vector $\alpha(x) = A_{3,j}x^3+A_{2,j}x^2+A_{1,j}x+A_{0,j}$ and GF (2^8) on a fixed polynomial c(x) = '03'x^3+'01'x^2+01'x+'02' do multiplication again to take mold x^4+1, can use matrix representation is as follows:

$$\begin{bmatrix} b_{0,j} \\ b_{1,j} \\ b_{2,j} \\ b_{3,j} \end{bmatrix} = \begin{bmatrix} 02&03&01&01 \\ 01&02&03&01 \\ 01&01&02&03 \\ 03&01&01&02 \end{bmatrix} \begin{bmatrix} a_{0,j} \\ a_{1,j} \\ a_{2,j} \\ a_{3,j} \end{bmatrix} \tag{3}$$

Then:

$$\begin{bmatrix} b_{0,j} \\ b_{1,j} \\ b_{2,j} \\ b_{3,j} \end{bmatrix} = \begin{bmatrix} 2a_{0,j} \\ a_{0,j} \\ a_{0,j} \\ 3a_{0,j} \end{bmatrix} + \begin{bmatrix} 3a_{0,j} \\ 2a_{0,j} \\ a_{0,j} \\ a_{0,j} \end{bmatrix} + \begin{bmatrix} a_{0,j} \\ 3a_{0,j} \\ 2a_{0,j} \\ a_{0,j} \end{bmatrix} + \begin{bmatrix} a_{0,j} \\ a_{0,j} \\ 3a_{0,j} \\ 2a_{0,j} \end{bmatrix}$$
$$= 2\begin{bmatrix} a_{0,j} \\ a_{1,j} \\ a_{2,j} \\ a_{3,j} \end{bmatrix} + 3\begin{bmatrix} a_{1,j} \\ a_{2,j} \\ a_{3,j} \\ a_{0,j} \end{bmatrix} + \begin{bmatrix} a_{2,j} \\ a_{3,j} \\ a_{0,j} \\ a_{1,j} \end{bmatrix} + \begin{bmatrix} a_{3,j} \\ a_{0,j} \\ a_{1,j} \\ a_{2,j} \end{bmatrix} \tag{4}$$

Will be listed in the four bytes as 32 bytes α, the rest of the word on the level of byte cyclic shift and word multiplier operation (3α = 2α+α, namely 3α is multiplier and a plus combination) to achieve. Word of the multiplier is each byte elements by x (Muda *et al.*, 2010).

Optimization to realize column hybrid inverse operation: The polynomial which need to modular multiplication among the column transformation of the inverse operation is d(x) = '0b'x^3+'0d'x^2+'09x'+0e and d(x)c(x) = '01', the related matrix:

$$\begin{bmatrix} a_{0,j} \\ a_{1,j} \\ a_{2,j} \\ a_{3,j} \end{bmatrix} = \begin{bmatrix} 0e&0b&0d&09 \\ 09&0e&0b&0d \\ 0d&09&0e&0b \\ 0b&0d&09&0e \end{bmatrix} \begin{bmatrix} b_{0,j} \\ b_{1,j} \\ b_{2,j} \\ b_{3,j} \end{bmatrix} \tag{5}$$

The above carry on the corresponding deformation:

$$\begin{bmatrix} a_{0,j} \\ a_{1,j} \\ a_{2,j} \\ a_{3,j} \end{bmatrix} = \begin{bmatrix} 05&00&04&00 \\ 00&05&00&04 \\ 04&00&05&00 \\ 00&04&00&05 \end{bmatrix} \left(\begin{bmatrix} 02&03&01&01 \\ 01&02&03&01 \\ 01&01&02&03 \\ 03&01&01&02 \end{bmatrix} \begin{bmatrix} b_{0,j} \\ b_{1,j} \\ b_{2,j} \\ b_{3,j} \end{bmatrix} \right) \tag{6}$$

Visible in parentheses is the column transformation, so use the existing column transformation code.

Simplify the rest of the part:

$$\begin{bmatrix} 05&00&04&00 \\ 00&05&00&04 \\ 04&00&05&00 \\ 00&04&00&05 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} x+4(x+z) \\ y+4(y+w) \\ z+4(z+x) \\ w+4(w+y) \end{bmatrix} + \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} + \begin{bmatrix} 4(x+z) \\ 4(y+w) \\ 4(z+x) \\ 4(w+y) \end{bmatrix} \tag{7}$$

So this part can be through the use of xor and multiply (the column left 1 byte) to achieve the efficiency and to realize the relative gets greatly improved.

Optimization wheel transformation: According to the literature, the whole wheel transformation on four column operation, each column operation involves bytes instead, line shift, column transformation and wheel key added:

Table 2(a-b): Intermediate result state

(a)			
A0	A4	A8	A12
A5	A9	A13	A1
A10	A14	A2	A6
A15	A3	A7	A11

(b)			
A0	A4	A8	A12
A15	A3	A7	A11
A10	A14	A2	A6
A5	A9	A13	A1

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = S\left[a_{0,j}\right]\begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix} + S\left[a_{1,j-C1}\right]\begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} + S\left[a_{2,j-C2}\right]\begin{bmatrix} 01 \\ 03 \\ 02 \\ 01 \end{bmatrix} + S\left[a_{3,j-C3}\right]\begin{bmatrix} 01 \\ 01 \\ 03 \\ 02 \end{bmatrix} + \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix} \tag{8}$$

Through the production of look-up table in the S-box search $\alpha_{i,j}$ for the above four vector S [$\alpha_{i,j}$], defining table T_0 to T_3 as follows:

$$T_0[a] = \begin{bmatrix} S[a]\cdot 02 \\ S[a] \\ S[a] \\ S[a]\cdot 03 \end{bmatrix} \quad T_1[a] = \begin{bmatrix} S[a]\cdot 03 \\ S[a]\cdot 02 \\ S[a] \\ S[a] \end{bmatrix}$$

$$T_2[a] = \begin{bmatrix} S[a] \\ S[a]\cdot 03 \\ S[a]\cdot 02 \\ S[a] \end{bmatrix} \quad T_3[a] = \begin{bmatrix} S[a] \\ S[a] \\ S[a]\cdot 03 \\ S[a]\cdot 02 \end{bmatrix}$$

Then:

$$e_j = T_0\left[a_{0,j}\right] + T_1\left[a_{1,j-C1}\right] + T_2\left[a_{2,j-C2}\right] + T_3\left[a_{3,j-C3}\right] + k_j \tag{9}$$

For Eq. 8 application addition (xor) exchange rates (Wang, 2008):

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = \begin{bmatrix} S\left[a_{0,j}\right]02 + S\left[a_{3,j-C3}\right]01 \\ S\left[a_{0,j}\right]01 + S\left[a_{3,j-C3}\right]01 \\ S\left[a_{0,j}\right]01 + S\left[a_{3,j-C3}\right]03 \\ S\left[a_{0,j}\right]03 + S\left[a_{3,j-C3}\right]01 \end{bmatrix} + S\left[a_{2,j-C2}\right]\begin{bmatrix} 01 \\ 03 \\ 02 \\ 01 \end{bmatrix} + S\left[a_{1,j-C1}\right]\begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} + \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix} \tag{10}$$

Equation 8, 10 corresponding as shown in Table 2a and (b) line shift after the state, (b) is (a) second, four lines transposition results.

Wrap in the state, there are three groups of continuous double byte (16 bits), i.e., A3 and A4, A7 and A8, A11 and A12, so each byte can all one-time access to, in the above Eq. (10) calculation, construct a table T_{01} [ab] (a and b are 8 bits of bytes), corresponding to the above Eq. (10) of the first paragraph may also structure the table below:

Table 3: The key of wheel change

i	1	2	3	4	5
W(i)	$w_4w_5w_6w_7$	$w_8w_9w_{10}w_{11}$	$w_{12}w_{13}w_{14}w_{15}$	$w_{16}w_{17}w_{18}w_{19}$	$w_{20}w_{21}w_{22}w_{23}$
i	6	7	8	9	10
W(i)	$w_{24}w_{25}w_{26}w_{27}$	$w_{28}w_{29}w_{30}w_{31}$	$w_{32}w_{33}w_{34}w_{35}$	$w_{36}w_{37}w_{38}w_{39}$	$w_{40}w_{41}w_{42}w_{43}$

$$T_{02}[ab] = \begin{bmatrix} S\left[a_{0,j}\right]02 + S\left[a_{3,j-C3}\right]01 \\ S\left[a_{0,j}\right]01 + S\left[a_{3,j-C3}\right]01 \\ S\left[a_{0,j}\right]01 + S\left[a_{3,j-C3}\right]03 \\ S\left[a_{0,j}\right]03 + S\left[a_{3,j-C3}\right]02 \end{bmatrix} \tag{11}$$

The key is divided into groups according to the column of matrix and 40 new columns are added to expand. If the former four columns (i.e., the initial key) is w (0), w (1), w (2) and w (3), then the new column will produce in a recursive way. The column 'i' is determined by Eq. 12 (Babu *et al.*,2012):

$$\begin{aligned} &w(i) = w(i-4)\ \text{XOR}\ w(i-1) \quad \text{i is the multiple 4} \\ &w(i) = w(i-4)\ \text{XORT}\ w(i-1) \quad \text{i isn't the multiple of 4} \end{aligned} \tag{12}$$

The round key 'i' can be expressed as Eq. 13:

$$W(i) = w_{4i+0}w_{4i+1}w_{4i+2}w_{4i+3} \tag{13}$$

With 4×4 matrix that is Eq. 14:

$$W = \begin{bmatrix} w_{0,4i+0} & w_{0,4i+1} & w_{0,4i+2} & w_{0,4i+3} \\ w_{1,4i+0} & w_{1,4i+1} & w_{1,4i+2} & w_{1,4i+3} \\ w_{2,4i+0} & w_{2,4i+1} & w_{2,4i+2} & w_{2,4i+3} \\ w_{3,4i+0} & w_{3,4i+1} & w_{3,4i+2} & w_{3,4i+3} \end{bmatrix} \tag{14}$$

If the initial key is w(0), w(1), w(2) and w(3), it can obtain w(4)~w(43) by Eq. 12. So the 10 round keys are composed of w(4)~w(43), as is shown in Table 3.

As can be seen from Table 3, if the initial key (w(0), w(1), w(2) and w(3)) is known, then the round key can be taken as a constant table and it can be implemented through Look-up Table circuit.

Although, need to use additional memory space to store structure form, but because the calculation of the process can be directly look-up table, the calculation efficiency has been greatly improved.

Rijndael algorithm based on niosII: The Rijndael algorithm based on the SOPC system is shown in Fig. 1. The standard Altera 32 bits NiosII embedded CPU provides guarantee for the large and systematic data processing. The system is composed of FPGA, memory

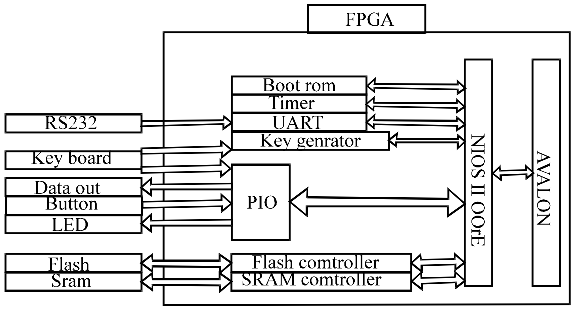

Fig. 1: Scheme of SOPC system based on FPGA

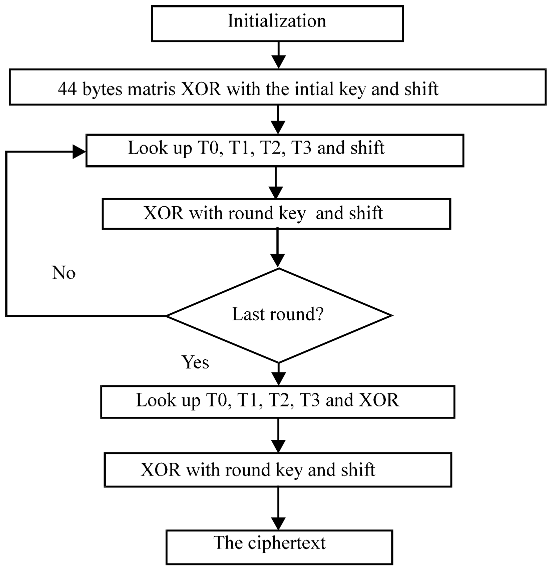
Initialization
44 bytes matris XOR with the intial key and shift
Look up T0, T1, T2, T3 and shift
XOR with round key and shift
No
Last round?
Yes
Look up T0, T1, T2, T3 and XOR
XOR with round key and shift
The ciphertext

Fig. 2: Design of optimized algorithm

and the external interface three parts. In the system, the peripheral circuit and NiosII CPU Soft-Core are integrated to realize the control functions. As the control core of the system, the NiosII CPU Soft-Core needs a balance between its resource occupation and function when it is created. The demand of system resources is greatly reduced by the SOPC Builder customization for NiosII CPU. The FPGA part is built in the FPGA chip, the core is NiosII processor core. A lot of data need to be processed in algorithm, so the algorithm round transformation is completed by using NiosII processor core and the key generation is conducted by the key generator in FPGA. The external interface of FPGA is a part including some interface devices and circuit modules, which is used for data input / output and display and so on.

The design of optimized algorithm is shown in Fig. 2.

RESULTS AND DISCUSSION

According to the initial key (w (0), w (1), w (2) and w (3)), the key generator generates w (4)~w (43)

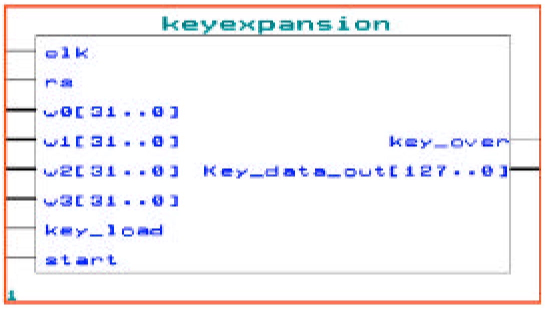

Fig. 3: Module of key expansion circuit

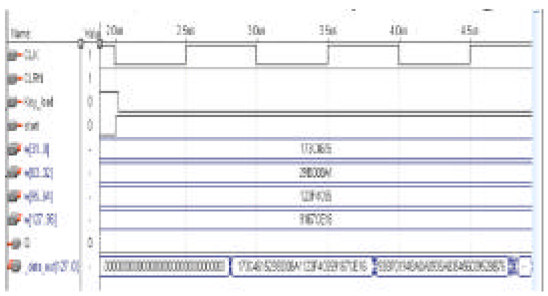

Fig 4: Simulation of the Encryption algorithm

automatically and stores them in the memory (complete the memory initialization). In the period of round transformation, the quadruple frequency of the round clock is conducted by the frequency multiplier and the counting value is taken as the Look-up Table circuit address. Four Look-up tables are implemented in a round clock period and w4i+0 ~ w4i+3 are sent out. At the same time, the 128 bits round key is exported through the serial-in parallel-out shift register. The function description for the key generation circuit is conducted by VHDL and the generated module symbols are shown in Fig. 3.

In which, W0[31..0], W1[31..0] , W2[31..0] and W3[31..0]is the initial key, "Key_load" is loading the initial key, "start" is the beginning of the round key generation, "key_data_out[127..0]" is the output of Sub-key, "key_over" is the completion of round key generation. When the initial key is W_0 = 173c4615, W_1 = 29bd08a1, W_2 = 123f4c55 and W_3 = 9167ce16, the simulation result is presented in Fig. 4.

The design based on Altera EP2C35 chips is synthesized under the integrated environment of QUARTUSII9.0. The testing results are shown in Table 4. As can be seen, research the working frequency is 141 MHZ; the data flow is 1805 Mbps after improved. Compared with the results in literature (Table 4), the speed

Table 4: The results of experiment

Data flow (Mbps)	Frequency (MHZ)	LE	Device	Key length (bit)	Reference
254.0	37.9	2822(slice)	XC2V1000	128	(Zhang, 2007)
779.0	67.0	881(slice)	XC2S400E	128	(Shen, 2006)
376.5	29.4	3235	EP1S10	128	(Wang, 2008)
1805	141.0	33216	EP2C35	128	This paper

of data flow has been increased by 131% at least. The design has a clear superiority in speed and performance under the relatively low resource request. The study based on the on the optimization of the algorithm combined with the previous research about Amplify-and-Forward Cooperative Systems, can greatly improve the system performance (Tang and Wang, 2013).

CONCLUSION

According to the traditional Rijndael algorithm, this study proposes a realization of Rijndael algorithm more optimized solutions, this scheme will wheel transformation, key expansion algorithm with matrix said out, again to table query, using SOPC rich look-up table and storage resources to achieve the algorithm, the proposed algorithm reduced the complexity of the calculation. The testing results show that the design of the Rijndael algorithm based on "FPGA+NiosII" can achieve a higher data processing speed while it occupies relatively low resources. The design has a big breakthrough compared to the traditional realization based on FPGA.

ACKNOWLEDGMENT

The study is supported by the Natural Science Foundation of China (No. 61261036) and Natural Science Foundation of Xinjiang Province (No. 2013211A012).

REFERENCES

Babu, G.D., N.N. Anandakumar and D. Muralidharan, 2012. Countermeasures against DPA attacks on FPGA implementation of AES. J. Artif. Intell., 5: 186-192.

Cao, Z.G., 2006. The research and FPGA implementation of rijndael algorithm. Harbin, Harbin Eng. Univ., 37: 1461-1465.

Liang, W., X. Sun, Z. Ruan and J. Long, 2011. The design and FPGA implementation of FSM-based intellectual property watermark algorithm at behavioral level. Inform. Technol. J., 10: 870-876.

Liao, H.C. and Y.H. Chao, 2008. A new data encryption algorithm based on the location of mobile users. Inform. Technol. J., 7: 63-69.

Liu, C., Y. Zhou, Y. Xiao and G. Sun, 2011. Encryption algorithm of RSH (round sheep hash) chaoqun. Inform. Technol. J., 10: 686-690.

Lv, X., 2005. Design and implementation of AES crypto coprocessor based on FPGA. Microelectron. Comput., 22: 121-123.

Muda, Z., R. Mahmod and M.R. Sulong, 2010. Key transformation approach for rijndael security. Inform. Technol. J., 9: 290-297.

Shen, Q.F., 2006. FPGA optimization implementation for AES algorithm. J. Xian Univ. Technol., 22: 203-206.

Spillman, R.J., 2005. Classical and Contemporary Cryptology. Pearson Education, Beijing, ISBN-13: 9780131828315, Pages: 285.

Tang, J. and L. Wang, 2013. Non-combining incremental relaying protocol for amplify-and-forward cooperative systems. Inf. Technol. J., 12: 239-242.

Wang, J.Y., 2008. Reconfigurable design for encryption/decryption of AES based on FPGA. Comput. Eng., 34: 163-164, 167.

Zhang, D.X., 2007. Design and implementation of AES algorithm based on FPGA. J. Univ. Sci. Technol. China, 37: 1461-1465.

Zhang, J.Y., 2008. A very small FPGA implementation of application-specific instruction processor for AES. Microelectron. Comput., 25: 165-168.