Journal of Applied Sciences 10 (18): 2161-2167, 2010
ISSN 1812-5654

An Overview: Theoretical and Mathematical Perspectives for Advance Encryption Standard/Rijndael

[1]A.A. Zaidan, [1]B.B. Zaidan, [2]Ali K. Al-Frajat and [2]Hamid A. Jalab
[1]Faculty of Engineering, Multimedia University, Jalan Multimedia, 63100, Cyberjaya, Malaysia
[2]Faculty of Computer Science and Information Technology,
University Malaya, 50603, Kuala Lumpur, Malaysia

Abstract: With the quick development of various multimedia technologies, more and more multimedia data are generated and transmitted in the medical, also the internet allows for wide distribution of digital media data. It becomes much easier to edit, modify and duplicate digital information. Besides that, digital documents are also easy to copy and distribute, therefore it will be faced by many threats. It is a big security and privacy issue, it become necessary to find appropriate protection because of the significance, accuracy and sensitivity of the information, which may include some sensitive information which should not be accessed by or can only be partially exposed to the general users. Therefore, security and privacy has become an important. Another problem with digital document and video is that undetectable modifications can be made with very simple and widely available equipment, which put the digital material for evidential purposes under question. Cryptography considers one of the techniques which used to protect the important information. In this study a theoretical and mathematical perspectives for Advance Encryption Standard (AES)/Rijndael of multimedia encryption schemes have been proposed in the literature and description. This overview for advance encryption standard achieving a flexibility and uncomplicated handle with this algorithm, which is a challenge of researchers when be used as implementation.

Key words: Cryptography, advance encrypting standard, mathematical preliminaries

ADVANCE ENCRYPTION STANDARD (AES)/ RIJNDAEL

AES is short for Advanced Encryption Standard and is a United States encryption standard defined in Federal Information Processing Standard (FIPS), published in November 2001. It was ratified as a federal standard in May 2002 (Xu and Dereje, 2004). One should not compare AES with RSA, another standard algorithm, as RSA is a different category of algorithm (Rabah, 2006). Bulk encryption of information itself is seldom performed with RSA. RSA is used to transfer other encryption keys for use by AES for example and for digital signatures. AES is a symmetric encryption algorithm processing data in block of 128 bits. Under the influence of a key, a 128-bit block is encrypted by transforming it in a unique way into a new block of the same size. AES is symmetric since the same key is used for encryption and the reverse transformation, decryption (Rabah, 2005a). The only secret necessary to keep for security is the key. AES may be configured to use different key-lengths, the standard defines 3 lengths and the resulting algorithms are named AES-128, AES-192 and AES-256, respectively to indicate the length in bits of the key. Each additional bit in the key effectively doubles the strength of the algorithm, when defined as the time necessary for an attacker to stage a brute force attack, i.e., an exhaustive search of all possible key combinations in order to find the right one (Rabah, 2005b). This is the second version of the Rijndael documentation. The main difference with the first version is the correction of a number of errors and inconsistencies, the addition of a motivation for the number of rounds, the addition of some figures in the section on differential and linear cryptanalysis, the inclusion of Brain Gladman's performance figures and the specification of Rijndael extensions supporting block and key lengths of 160 and 224 bits (Abomhara *et al*., 2010).

MATHEMATICAL PRELIMINARIES

Several operations in Rijndael are defined at byte level, with bytes representing elements in the finite field GF (2^8). Other operations are defined in terms of 4-byte words. In this section introduced the basic mathematical concepts (Alanazi *et al*., 2010).

Corresponding Author: A.A. Zaidan, Faculty of Engineering, Multimedia University, Jalan Multimedia, 63100, Cyberjaya, Malaysia

The Field GF (2^8): The elements of a finite field [LiNi86] can be represented in several different ways. For any prime power there is a single finite field, hence all representations of GF (2^8) are isomorphic. Despite this equivalence, the representation has an impact on the implementation complexity. For the classical polynomial had chosen representation. A byte b, consisting of bits b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0, is considered as a polynomial with coefficient in {0, 1}, $b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0x$ .Example; the byte with hexadecimal value '57' (binary 01010111) corresponds with polynomial[$x^6 + x^4 + x^2 + x + 1$] (Naji *et al*., 2009a).

Addition: In the polynomial representation, the sum of two elements is the polynomial with coefficients that are given by the sum modulo 2 (i.e., 1+1 = 0) of the coefficients of the two terms. Example; '57' + '83' = 'D4', or with the polynomial notation; in binary notation had; 01010111 + 10000011 = 11010100. Clearly, the addition corresponds with the simple bitwise EXOR (denoted by⊕) at the byte level. All necessary conditions are fulfilled to have an Abelian group; internal, associative, neutral element (00), inverse element (every element is its own additive inverse) and commutative. As every element is its own additive inverse, subtraction and addition is the same (Naji *et al*., 2009b).

Multiplication: In the polynomial representation, multiplication in GF (2^8) corresponds with multiplication of polynomial modulo an irreducible binary polynomial of degree 8. A polynomial is irreducible if it has no divisors other than 1 and itself. For Rijndael, this polynomial is called m(x) and given by $m(x) = x^8 + x^4 + x^3 + x + 1$, or '11B' in hexadecimal representation. Clearly, the result will be a binary polynomial of degree below 8. Unlike for addition, there is no simple operation at byte level. The multiplication defined above is associative and there is a neutral element ('01'). For any binary polynomial b(x) of degree below 8, the extended algorithm of Euclid can be used to compute polynomials a (x), c(x) such that:

$$b(x)\,a(x) + m(x)\,c(x) = 1 \quad (1)$$

Hence, a(x). b(x) mod m(x) = 1 or

$$b^{-1}(x) = a(x) \bmod m(x) \quad (2)$$

Moreover, it holds that a(x). b(x) + c(x) = a(x). b(x) + a(x) + c(x). It follows that the set of 256 possible byte values, with the EXOR as addition and the multiplication defined as above has the structure of the finite field GF(2^8) (Alaa *et al*., 2009).

Multiplication by x: If there is multiply b(x) by the polynomial x, then; $b_7x^8 + b_6x^7 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x$, x. b(x) is obtained by reducing the above result modulo m(x). If b7 = 0, this reduction is the identity operation, If b7 = 1, m(x) must be subtracted (i.e., EXORed). It follows that multiplication by x (hexadecimal 02) can be implemented at byte level as a left shift and a subsequent conditional bitwise EXOR with '1 B'. This operation is denoted by b = x time (a). In dedicated hardware, xtime takes only 4 EXORs. Multiplication by higher powers of x can multiplication by repeated application of xtime. By adding intermediate results, multiplication by any constant can be implemented (Alaa *et al*., 2009).

Polynomials with coefficients in GF (2^8): Polynomials can be defined with coefficients in GF (2^8). In this way, a 4-byte vector corresponds with a polynomial of degree below 4. Polynomials can be added by simply adding the corresponding coefficients. As the addition in GF (2^8) is the bitwise EXOR, the addition of two vectors is a simple bitwise EXOR. Multiplication is more complicated. Assume that has two polynomials over GF (2^8) (Zaidan *et al*., 2010a, b):

$$a(x) = a_3x^3 + a_2x^2 + a_1x + a_0 \text{ and } b(x) = b_3x^3 + b_2x^2 + b_1x + b_0 \quad (3)$$

Multiplication by x: If multiply b(x) by the polynomial x_1 then:

$$b_3x^4 + b_2x^3 + b_1x^2 + b_0x \quad (4)$$

$x \otimes b(x)$ is obtained by reducing the above result modulo $1 + x^4$. This gives:

$$b_2x^3 + b_1x^2 + b_0x + b_3 \quad (5)$$

The multiplication by x is equivalent to multiplication by a matrix as above with all:

a_i = '00' except a_1 ='01'. Let $c(x) = x \otimes b(x)$. Then:

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 00 & 00 & 00 & 01 \\ 01 & 00 & 00 & 00 \\ 00 & 01 & 00 & 00 \\ 00 & 00 & 01 & 00 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Hence, multiplication by x, or powers of x, corresponds to a cyclic shift of the bytes inside the vector (Zaidan *et al*., 2009a, b).

Design rationale: The three criteria taken into account in the design of Rijndael are the following:

- Resistance against all known attacks
- Speed and code compactness on a wide range of platforms
- Design simplicity

In most ciphers, the round transformation has the Feistel structure. In this structure typically part of the bits of the intermediates state are simply transposed unchanged to another position. The round transformation of Rijndael does not have the Feistel structure. Instead, the round transformation is composed of three distinct invertible uniform transformations, called layers. By uniform, means that every bit of the state is treated in a similar way. The specific choices for the different layers are for a large part based on the application of the Wide Trail Strategy, a design method to provide resistance against linear and differential cryptanalysis. In the Wide Trial Strategy, every layer has its own function:

- **The linear mixing layer:** Guarantees high diffusion over multiple rounds
- **The non-linear layer:** Parallel application of S-boxes that have optimum worst-case nonlinearity properties
- **The key addition layer:** A simple EXOR of the Round Key to the intermediate state

Before the first round, a key addition layer is applied. The motivation for this initial key addition is the following. Any layer after the last key addition in the cipher (or before the first in the context of known-plaintext attacks) can be simply peeled off without knowledge of the key and therefore does not contribute to the security of the cipher. (e.g., the initial and final permutation in the DES). Initial or terminal key addition is applied in several designs, e.g., IDEA, SAFER and Blowfish. In order to make the cipher and its inverse more similar in structure, the linear mixing layer of the last round is different from the mixing layer in the other rounds. It can be shown that this does not improve or reduce the security of the cipher in any way. This is similar to the absence of the swap operation in the last round of the DES (Zaidan *et al*., 2010a).

Specification: Rijndael is an iterated block cipher with a variable block length and a variable key length. The block length and the key length can be independently specific to 128, 192 or 256 bits (Zaidan *et al*., 2009a).

THE STATE, THE CIPHER KEY AND THE NUMBER OF ROUNDS

The different transformations operate on the intermediate result, called the state. The state can be pictured as a rectangular array of bytes. This array has

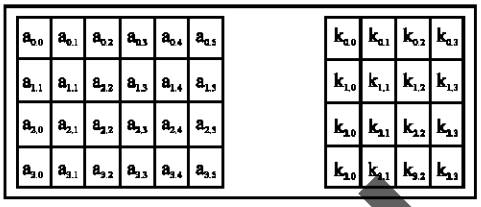

Fig. 1: Example of state (with Nb = 6) and cipher key (with Nk = 4) layout

four rows, the number of columns is denoted by Nb and is equal to the block length divided by 32. The Cipher Key is similarly pictured as a rectangular array with four rows. The number of columns of the Cipher Key is denoted by Nk and is equal to the key length divided by 32.These representations are illustrated in Fig. 1. In some instances, these blocks are also considered as one-dimensional array of 4-bytes vectors, where each vector consists of the corresponding column in the rectangular array representation. These arrays hence have lengths of 4, 6 or 8 respectively and indices in the ranges 0..3, 0..5, 0..7. 4-bytes vectors will sometimes be referred to as words. Where it is necessary to specify the four individual bytes within a 4-bytes vector or word the notation (a, b, c, d) will used where a, b, c and d are the bytes at positions 0, 1, 2 and 3, respectively within the column, vector or word being considered (Zaidan *et al*., 2010a).

The input and output used by Rijndael at its external interface are considered to be one-dimensional arrays of 8-bit bytes numbered upwards from 0 to the 4*Nb-1. These blocks hence have lengths of 16, 24 or 3 bytes and array indices in the ranges 0..15, 0..23 or 0..31. The Cipher Key is considered to be a one- dimensional arrays of 8-bit bytes numbered upwards from 0 to the 4*Nk-1. These blocks hence have lengths of 16, 24 or 32 bytes and array indices in the ranges 0..15, 0..23 or 0..31.The cipher input bytes (the "plaintext" if the mode of use is ECB encryption) are mapped onto the state bytes. in the order $a_{0,0}$, $a_{1,0}$, $a_{2,0}$, $a_{3,0}$, $a_{0,1}$, $a_{1,1}$, $a_{2,1}$, $a_{3,1}$, $a_{4,1}$… and the bytes of the cipher key are mapped onto the array in the order $k_{0,0}$, $k_{1,0}$, $k_{2,0}$, $k_{3,0}$, $k_{0,1}$, $k_{1,1}$, $k_{2,1}$, $k_{3,1}$, $k_{4,1}$ … At the end of the cipher operation, the cipher output is extracted from the state by taking the state bytes in the same order. Hence if the one-dimensional index of a byte within a block is n and the two dimensional index is (i, j). Had: i = n mod 4; j = [n/4]; n = I + 4* j. Moreover, the index i is also the byte number within a 4-byte vector or word and j is index for the vector or word within the enclosing block. The number of rounds is denoted by Nr and depends on the value Nb and Nk. It is given in table one (Zaidan *et al*., 2009b).

Key schedule: The Round Keys are derived from the Cipher Key by means of the key schedule. This consists

Table 1: No. of rounds (Nr) as a function of the block and key length

Nr	Nb = 4	Nb = 6	Nb = 8
Nk = 4	10	12	14
Nk = 6	12	12	14
Nk = 8	14	14	14

```
Key expansion (byte key[4* Nk] word W[Nb*(nr+1)])
{

    for (i = 0; i<Nk; I++)
        W[i] = (Key[4*i], Key[4*i+1], Key[4*i+2, Key[4*i+3]]);

    for (i = Nk; i<Nb*(Nr+1); i++)
    {
        temp = w[i-1];
        if (i % Nk = = 0)
            Temp = SubByte(RotByte (temp) ^ Rcon[i/Nk];
        W[i] = W[i - Nk] ^ temp;
    }
}
```

Fig. 2: Function of key Expansion for Nk≤6

of two components: the key Expansion and the Round Key Selection. The basic principle is the following (Zaidan *et al*., 2010a).

The total number of round key bits is equal to the block length multiplied by the number of rounds plus 1 (e.g., for a block length of 128 bits and 10 rounds, 1408 Rounds Key bits are needed).

- The cipher key is expanded into an expanded key
- Round Keys are taken from this expanded key in the following way: the first Round Key consists of the first Nb words, the second one of the following Nb words and so on (Table 1)

Key expansion: The expanded key is a linear array of 4-byte words and is denoted by W [Nb* (nr+1)]. The first Nk words contain the cipher key. All other words are defined recursively in terms of words with smaller indices. The key expansion function depends on the value of Nk; there is a version for Nk equal to or below 6 and a version for Nk above 6. For Nk≤6, has (Zaidan *et al*., 2010b). Figure 2 shows the key expression function for Nk≤6.

In this description, SubByte (W) is a function that returns a 4-byte word in which each byte is the result of applying the Rijndael S-box to the byte at the corresponding position in the input word. The function Rot Byte (w) returns a word in which the bytes are a cyclic permutation of those in its input such that the input word (a, b, c, d) produces the output word (b, c, d, a). It can be seen that the first Nk words are filled with the cipher key. Every following word w[i] am equal to the EXOR of the previous word W [i-1] and the word Nk position earlier W [I-Nk]. For words in positions that are a multiple of Nk, a transformation consists of a cyclic shift of the bytes in

```
KeyExpansion (byte key[4* Nk] word W[Nb*(nr+1)])
{

    for (i = 0; i<Nk; I++)
        W[i] = (key[4*i], key[4*i+1], key[4*i+2, key[4*i+3]]);

    for (i = Nk; i<Nb*(Nr+1); i++)
    {
        temp = w[i-1];
        if (i % Nk = = 0)
            Temp = SubByte(RotByte (temp) ^ Rcon[i/Nk];
        else if (i % Nk = = 4)
            Temp = SubByte (tem);
        W[i] = W[i - Nk] ^ temp;
    }
}
```

Fig. 3: Function of key expansion for Nk<6

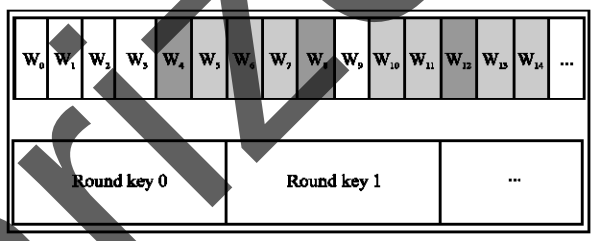

Fig. 4: Key Expansion and Round Key Selection for Nb = 6 and Nk = 4

a word (Rot Byte), followed by the application of a table lookup to all four bytes of the word (SubByte). For Nk<6, has (Zaidan *et al*., 2009a). Figure 3 shows the key expansion function for Nk<6.

The difference with the scheme for Nk > is that for i-4 a multiple of Nk, SubByte is applied to W[i-1] prior to the EXOR. The round constants are independent of Nk and defined by:

$$Rcon[i] = (RC[i], '00', '00', '00')$$

With RC [I] representing an element in GF (2^8) with a value of $x^{(i-1)}$ so that:

RC[1] = 1 (1.e., 01)
RC[i] = x (i.e., 02) •(RC|i-1) = $x^{(I-1)}$

Round key selection: Round key I is given by the round key buffer words W[Nb * i] to W[Nb * (i+1)]. This is illustrated in Figure 4. The Key schedule can be implemented without explicit use of the array W[Nb * (Nr + 1)]. For implementations where RAM is scare, the Round Keys can be computed on-the-fly using a buffer of Nk words with almost no computational overhead (Zaidan *et al*., 2010b). Figure 4 shows the key expansion and the round key selection as fellow.

THE CIPHER

The cipher Rijndael consists of:

- An initial round key addition
- Nr-1 rounds
- A final round

In Fig. 5, the pseudo C code for cipher Rijndael function has illustrated.

The key expansion can be done on beforehand and Rijndael can be specified in terms of the expanded key (Zaidan *et al*., 2010b). Figure 6 shows the code of expanded key function.

The Expanded Key shall always be derived from the Cipher Key and never be specified directly. There is however no restrictions on the selection of the Cipher Key itself.

Hardware suitability: The cipher is suited to be implemented in dedicated. There are several trade-offs between area and speed possible. Because the implementation in software on general-purpose processors is already very fast, the need for hardware implementations will very probably be limited to two specific cases (Zaidan *et al*., 2009a):

- Extremely high speed chip with no area restrictions, the T tables can be hardwired and the EXORs can be conducted in parallel
- Compact co-processor on a smart Card to speed up Rijndael execution, for this platform typically the s-box and the Xtime (or the complete MixColumn) operation can be hardwired

```
Rijndael (State, CipherKey)
{
KeyExpansion (CipherKey, Expandekey);
AddroundKey  (State, ExpandedKey);
For (l = 1; i<Nr; l++) Round (State, Expandedkey+Nb*l);
Final Round (State, ExpandedKey+Nb*Nr);
}
```

Fig. 5: Function of cipher rijndael

```
Rijndael (State, ExpandedKey)
{
AddroundKey  (State, ExpandedKey);
For (l = 1; l<Nr; i++) Round (State, Expandedkey+Nb*l);
Final Round (State, ExpandedKey+Nb*Nr);
}
```

Fig. 6: Function of expanded key

THE INVERSE CIPHER

In the table-lookup implementation it is essential that the only non-linear step (ByteSub) is the first transformation in a round and that the rows are shifted before MixColumn is applied. In the inverse of round, the order of the transformations in the round is reversed and consequently the non-linear step will end up being the last step of the inverse round and the rows are shifted after the application of (the inverse of) MixColumn. The inverse of a round can therefore not be implemented with the table lookups described above. This implementation aspect has been anticipated in the design. The structure of Rijndael is such that the sequence of transformations of its inverse is equal to that of the cipher itself, with the transformations replaced by their inverses and a change in the key schedule. This is shown in the following subsections (Alanazi *et al*., 2010). Figure 7 and 8 shows the code of the round A and the final round for inverse the Rijndael variant.

Inverse of a two-round rijndael variant

Algebraic properties: In deriving the equivalent structure of the inverse cipher, made using of two properties of the component transformations. First, the order of ShiftRow and ByteSub is indifferent. ShiftRow simply transposes the bytes and has no effect on the byte values. ByteSub works on individual bytes, independent of their position. Second, the sequence:

```
AddRoundKey (state, RoundKey);
InvMixColumn (State);
```

Can be replaced by:

```
InvMixColumn (State);
AddRoundKey (State, InvRoundKey);
```

```
InvRound (State, RoundKey)
{
AddroundKey (State, RoundKey);
InvMixColum (State);
InvShiftRow (State);
InvByteSub (State);
}
```

Fig. 7: The inverse of a round

```
InvFinalRound (State, RoundKey)
{
AddroundKey (State, RoundKey);
InvShiftRow (State);
InvByteSub (State);
}
```

Fig. 8: The inverse of the final round

With invRoundKey obtained by applying InvMixColumn to the corresponding RoundKey. This is based on the fact that for a linear transformation A, has A(x+k) = A(x) + A (k)

The equivalent inverse cipher structure: Using the properties described above, the inverse of the two-round Rijndael variant can be transformed into:

```
AddRoundKey (State, Expanded Key+2*Nb);

InvBytesSub (State);
InvShiftRow (State);
InvMixcolumn (State);
AddRoundKey (State, I-ExpandedKey+Nb);

InvBytesSub (State);
InvShiftRow (State);
InvMixcolumn (State);
AddRoundKey (State, I-ExpandedKey+Nb);
```

It can be seen that has again an initial Round Key addition, a round and a final round. The Round and the final round have the same structure as those of the cipher itself. This can be generalized to any number of rounds. Defined a round and the final round of the inverse cipher as follows:

```
I_Round (State, I_RoundKey)
{
InvByteSub (State);
InvShiftRow (State);
InvMixColumn (State);
AddroundKey (State, I_RoundKey);
}
I_FinalRound (State, I_RoundKey)
{
InvBytrSub (State);
InvShiftRow (State);
AddRoundKey (State, RoundKey0);
}
```

The Inverse of the Rijndael Cipher can now be expressed as follows:

```
I_Rijndael (state, CipherKey)
{
I_KeyExpansion
(CipherKey, I_ExpandedKey);
AddRoundKey
(State, I_ExpandedKey+ Nb*Nr);
For (i=Nr-1; i<0; i--)
Round (State, I_ExpandedKey+ Nb*i);
FinalRound (State, I_ExpandedKey);
}
```

The key expansion for the Inverse Cipher is defined as follows:

```
Apply the key Expansion.
Apply InvMixColumn to all Rounds Keys except the first
and the last one.
In pseudo C code, this gives:
I_KeyExpansion(CipherKey,I_ExpandedKey)
{
KeyExpansion(CipherKey,I_ExpandedKey);
For(i=1 ; i> Nr ; i++ )
InvMixColumn(I_ExpandedKey + Nb*i);
}
```

Implementation of the inverse cipher: The choice of the MixColumn polynomial and the key expansion was partly based on cipher performance arguments. Since the inverse cipher is similar in structure, but uses a MixColumn transformation with another polynomial and (in some cases) a modified key schedule, performance degradation is observed on 8-bit processors. This asymmetry is due to the fact that the performance of the inverse cipher is considered to be less important than that of the cipher. In many application of a block cipher, the inverse cipher operation is not used. This is the case for the calculation of MACs, but also when the cipher is used in CFB-mode.

CONCLUSION

In this study an overview for theoretical and mathematical Perspectives for Advance Encryption Standard (AES)/Rijndael, which are content the several operations in rijndael are defined at byte level, with bytes representing elements in the finite field GF, Polynomials with Coefficients, Design Rationale, The State of the Cipher Key and The Number of Rounds which is include (key schedule, Key Expansion, Round Key Selection), cipher phase and The Inverse Cipher. This study has been provided flexibility and simplify handle with this algorithm.

ACKNOWLEDGMENT

Thanks in advance for the entire worker in this project and the people who support in any way, also I want to thank MMU for the unlimited support which came from them.

REFERENCES

Abomhara, M., O. Zakaria, O.O. Khalifa, A.A. Zaidan, B.B. Zaidan and O.A. Hamdan, 2010. Overview: Suitability of using symmetric key to secure multimedia data. J. Applied Sci., 10: 1656-1661.

Alaa, T., A.A. Zaidan and B.B. Zaidan, 2009. New framework for high secure data hidden in the MPEG using AES encryption algorithm. Int. J. Comput. Electr. Eng., 1: 566-571.

Alanazi, H.O., B.B Zaidan, A.A. Zaidan, A.H. Jalab, M. Shabbir and Y. Al-Nabhani, 2010. New comparative study between DES, 3DES and AES within nine factors. J. Comput., 2: 152-157.

Naji, A.W., A.A. Zaidan, B.B. Zaidan and I.A.S. Muhamadi, 2009a. Novel approach for cover file of hidden data in the unused area two within EXE file using distortion techniques and advance encryption standard. Proc. World Acad. Sci. Eng. Technol., 56: 498-502.

Naji, A.W., A.A. Zaidan and B.B. Zaidan, 2009b. Challenges of hidden data in the unused area two within executable files. J. Comput. Sci., 5: 890-897.

Rabah, K., 2005a. Theory and implementation of data encryption standard: A review. Inform. Technol. J., 4: 307-325.

Rabah, K., 2005b. Theory and implementation of elliptic curve cryptography. J. Applied Sci., 5: 604-633.

Rabah, K., 2006. Implementing secure RSA cryptosystems using your own cryptographic JCE provider. J. Applied Sci., 6: 482-510.

Xu, Q.Z. and Y. Dereje, 2004. Theoretical Analysis of linear cryptanalysis against DES (Data Encryption Standard). Inform. Technol. J., 3: 49-56.

Zaidan, A.A., B.B. Zaidan and M. Anas, 2009a. High securing cover-file of hidden data using statistical technique and AES encryption algorithm. World Acad. Sci. Eng. Technol., 54: 468-479.

Zaidan, B.B., A.A. Zaidan, F. Othman and A. Rahem, 2009b. Novel approach of hidden data in the unused area 1 within exe files using computation between cryptography and steganography. Proceedings of the International Conference on Cryptography Coding and Information Security, June 24-26, Academic and Scientific Research Organizations, Paris, France, pp: 1-22.

Zaidan, A.A., B.B. Zaidan and A.H. Jalab, 2010a. A new system for hiding data within (Unused Area Two + Image Page) of portable executable file using statistical technique and advance encryption standard. Int. J. Comput. Theory Eng., 2: 220-227.

Zaidan, B.B., A.A. Zaidan, A.K. Al-Frajat and H.A. Jalab, 2010b. On the differences between hiding information and cryptography techniques: An overview. J. Applied Sci., 10: 1650-1655.