International Journal of Computer Applications (0975 – 8887)
Volume 117 – No. 22, May 2015

A Novel Approach for Developing a Secure and Optimized Algorithm for Implementation of Digital Signature

Shivanshu Rastogi
Asst. Professor,
Department of CS&E
Moradabad Institute of Technology
Moradabad, India

Zubair Iqbal
Asst. Professor,
Department of CS&E
Moradabad Institute of Technology
Moradabad, India

Prabal Bhatnagar
Asst. Professor,
Department of CS&E
Moradabad Institute of Technology
Moradabad, India

Priyanka Saxena
Asst. Professor,
Department of ECE
Moradabad Institute of Technology
Moradabad, India

ABSTRACT

Digital signatures can be treated as a digital code that can be embedded with an electronically transmitted data and its property is that it uniquely identifies the sender and ensures that the data has not been modified after the process of digital signing of that data. There are various algorithms that have been proposed for the generation of digital Signature. In this paper we are proposing a novel approach for making secure and optimize algorithm than any other existing algorithm.

Keywords

Digital Signature, DES, Security.

1. INTRODUCTION

The algorithm we have designed uses the concept of private key cryptography for key generation because Public-key cryptography is relatively slow and is only suitable for encrypting small amounts of information while private key cryptography is much faster and is suitable for encrypting large amounts of information. Also we used hexadecimal S-box which implements a much faster processing while encryption and all the work are done in two-dimension rather than one-dimension.

The concept of digital signature starts by taking a mathematical sample from the data usually known as hash code of that message that is to be transmitted. It is a uniquely identifiable digital sample or fingerprint of the data having the property to change itself exponentially even on minute changes in the message. It means that there is a dramatic change in the hash code on change in even a single bit of data. Than we have to create digital signatures by signing the hash code with the help of a private key and embedding that to the original message to be transmitted.

After designing of the algorithm in the best possible manner we came out with many positive results. By our algorithm the time for digitally signing the document has been reduced to 60% than that of earlier used digital signature algorithms [1] which uses SHA for hash creation and DES [2] for encryption and decryption.

2. DIGITAL SIGNATURE

A digital signature or a digital signature scheme is used for authentication of a document or a digital message based on some mathematical approaches. When a digital signature is attached with a message the recent has reason to believe that the message has been created by a known sender and is not modified during the transmission. Digital signatures are generally used in financial transactions where it is necessary to rely on digital documents for commercial activities. The digital signatures are not much different from the paper signatures.

The basic motive behind using digital signatures is to provide authenticity of sender and receiver and security from modifications and non repudiation.

It confirms the origin of the document and also verifies that no modification has been done to the document after signing of the document.

PURPOSE OF DIGITAL SIGNATURE

1. **Signer Authentication:** A signer is verified by comparing the key of the sender associated with that of a intended receiver. If a match is found the message is attributed to the signer. Thus a signature cannot be duplicated unless the sender compromises his private key of loses it or divulging it.
2. **Message Authentication:** the digital signature also verify the signed message by comparing the hash values calculated during the signing and verification process. They are able to authenticate the messages with greater accuracy than the paper signature.
3. **Non-Repudiation:** it is a affirmative act on part of the sender. It also ensures the receiver that the document received by him is sent by an authenticated sender and the sender cannot legitimately deny on sending of the document. This also alerts the signer that he is making a transaction which may result in legal consequences.
4. **Integrity:** the creation and verification of digital signature is highly accurate and still provides a high level of assurance that the digital signature is genuinely the signer.

3. WORKING OF DIGITAL SIGNATURE

3.1 Signing Process

Private Key cryptography gives a reliable method for digital signing and signature verification based on private key. A person can sign a given digital message (file, document, e-mail, and so forth) with his private key as shown in figure 1. From a technical point of view, the digital signing of a message is performed in two steps:

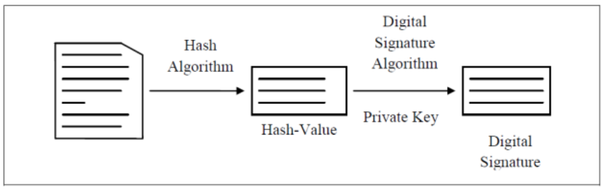

Figure 1 Digital Signing

STEP 1: CALCULATE THE MESSAGE DIGEST

In this step a hash-value of the message (often called the message digest) is calculated by applying some cryptographic hashing algorithm (for example, MD2, MD4, MD5, SHA1, or other). The calculated hash-value of a message is a sequence of bits, usually with a fixed length, extracted in some manner from the message. It is almost impossible, from a given hash-value of a given message, to find the message itself. This impossibility for retrieval of the input message is pretty logical if we take into account that a hash-value of a message could have a hundred times smaller size than the input message. Actually, the computing resources needed to find a message by its digest are so huge that, practically, it is unfeasible to do it.

It is also interesting to know that, theoretically, it is possible for two entirely different messages to have the same hash-value calculated by some hashing algorithm, but the probability for this to happen is so small that in practice it is ignored.

STEP 2: CALCULATE THE DIGITAL SIGNATURE

In the second step of digitally signing a message, the information obtained in the first step hash-value of the message (the message digest) is encrypted with the private key of the person who signs the message and thus an encrypted hash-value, also called digital signature, is obtained. For this purpose, some mathematical cryptographic encrypting algorithm for calculating digital signatures from given message digest is used .Often, obtained digital signature is attached to the message in a special format to be verified later if it is necessary.

3.2 Verification Process

Digital signature technology allows the recipient of given signed message to verify its real origin and its integrity. The digital signature verification cannot ascertain whether the given message has been signed by a given person. If we need to check whether some person has signed a given message, we need to obtain his real private key. This is possible by getting the private key in a secure way. From a technical point of view, the verification of a digital signature is performed in three steps:

STEP 1: CALCULATE THE CURRENT HASH-VALUE

In the first step, a hash-value of the signed message is calculated. For this calculation, the same hashing algorithm is used as was used during the signing process. The obtained hash-value is called the current hash-value because it is calculated from the current state of the message.

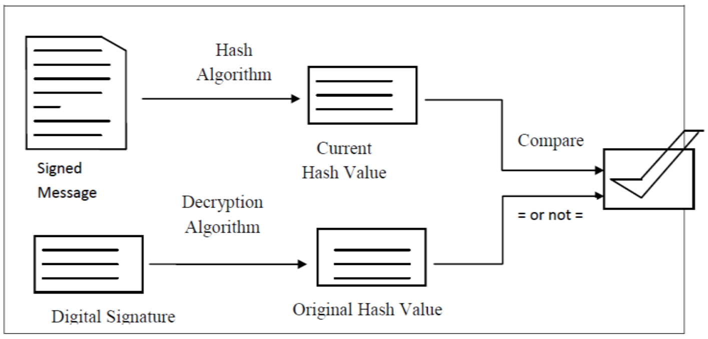

Figure 2 Verification of Digital Signing

STEP 2: CALCULATE THE ORIGINAL HASH-VALUE

In the second step of the digital signature verification process, the digital signature is decrypted with the same encryption algorithm that was used during the signing process. The decryption is done by the public key that corresponds to the private key used during the signing of the message. As a result, we obtain the original hash-value that was calculated from the original message during the first step of the signing process (the original message digests). This whole process is shown in figure 2.

STEP 3: COMPARE THE CURRENT AND THE ORIGINAL HASH-VALUES

In the third step, we compare the current hash-value obtained in the first step with the original hash-value obtained in the second step. If the two values are identical, the verification if successful and proves that the message has been signed with the private key that corresponds to the public key used in the verification process. If the two values differ from one another, this means that the digital signature is invalid and the verification is unsuccessful.

4. PROPOSED ALGORITHM

The algorithm that we are proposing specifies a symmetric block cipher [3] that can process data blocks of 128 bits, using cipher key with length of 128 bits. The algorithm consists of ten rounds of encryption, as can be seen in Figure 3. First the 128-bit key is expanded into eleven so-called round keys, each of them 128 bits in size. Each round includes a transformation using the corresponding cipher key to ensure the security of the encryption.

After an initial round, during which the first round key is XORed to the plain text (Per Round key operation), ten equally structured rounds follow. Each round consists of the following operations:

- Replace bytes
- Reallocate rows
- Per round key

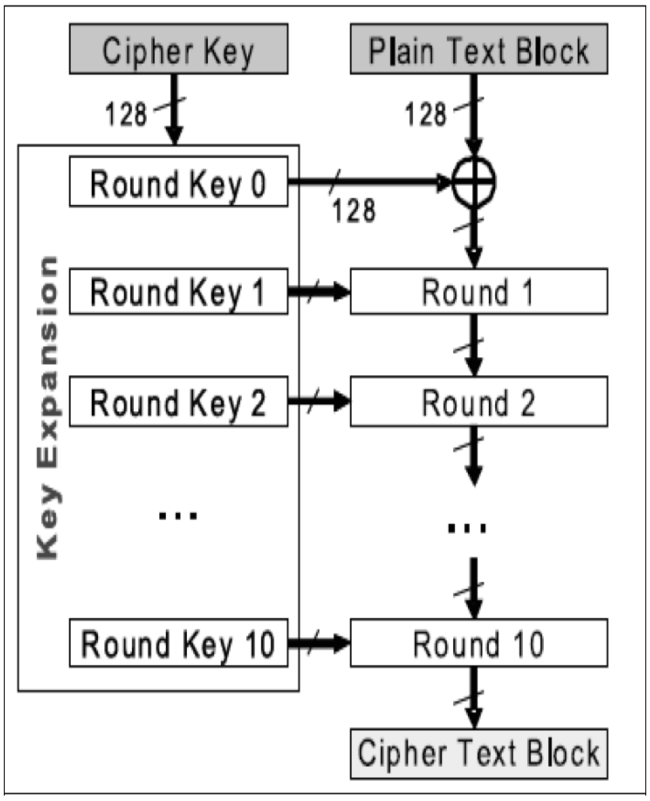

Figure 3 Algorithm Structure

Algorithm Specification

For this algorithm, the length of the input block, the output block and the State is 128 bits. And the number of 32-bit words (number of columns) in the State. For this algorithm, the length of the Cipher Key, K, is 128 bits. The key length reflects the number of 32-bit words (number of columns) in the Cipher Key.

Inputs And Outputs

The input and output for the algorithm each consist of sequences of 128 bits (digits with values of 0 or 1). These sequences will sometimes be referred to as blocks and the number of bits they contain will be referred to as their length. The Cipher Key for the algorithm is a sequence of 128 bits. Other input, output and Cipher Key lengths are not permitted by this standard. The bits within such sequences will be numbered starting at zero and ending at one less than the sequence length (block length or key length). Words(number of columns) in the State. For this algorithm, the length of the Cipher Key, K, is 128 bits. The key length reflects the number of 32-bit words (number of columns) in the Cipher Key.

The State

Internally, the algorithm's operations are performed on a two-dimensional array of bytes called the State. The State consists of four rows of bytes (block length divided by 32). In the State array denoted by the symbol s shown in figure 4, each individual byte has two indices, with its row number r in the range $0 \leq r \leq 4$ and its column number c in the range $0 \leq c \leq 4$. At the start of the Cipher and Inverse Cipher, the input the array of bytes in0, in1… in15 – is copied into the State array. The Cipher or inverse Cipher operations are then conducted on this State array, after which its final value is copied to the output.

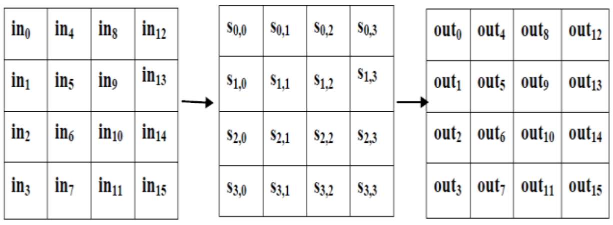
in_0 in_4 in_8 in_{12}
in_1 in_5 in_9 in_{13}
in_2 in_6 in_{10} in_{14}
in_3 in_7 in_{11} in_{15}
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
out_0 out_4 out_8 out_{12}
out_1 out_5 out_9 out_{13}
out_2 out_6 out_{10} out_{14}
out_3 out_7 out_{11} out_{15}

Figure 4 State Array Input And Output

Hence, at the beginning of the Cipher or Inverse Cipher, the input array, in, is copied to the State array according to the scheme and at the end of the Cipher and Inverse Cipher, the State is copied to the output array.

Cipher

At the start of the Cipher, the input is copied to the State array using the conventions described above. After an initial Round Key addition, the State array is transformed by implementing a round function 10 times. The final State is then copied to the output .The round function is parameterized using a key schedule that consists of a one-dimensional array of four-byte words derived using the Key Expansion routine described below. The individual transformations – Rep_Bytes(), Realloc_Rows(), PerRound_Key() process the State and are described in the following subsections.

REP_BYTES () TRANSFORMATION

The Rep_Bytes() transformation is a non-linear byte substitution that operates independently on each byte of the State using a substitution table (S-box). This S-box (Fig.5), which is invertible, is constructed by composing two transformations:

Take the multiplicative inverse; the element {00} is mapped to itself.

Apply the following affine transformation. In matrix form, the affine transformation element of the S-box can be expressed as:

$$\begin{bmatrix} b_0' \\ b_1' \\ b_2' \\ b_3' \\ b_4' \\ b_5' \\ b_6' \\ b_7' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0' \\ b_1' \\ b_2' \\ b_3' \\ b_4' \\ b_5' \\ b_6' \\ b_7' \end{bmatrix}$$

Figure 5 Affine Transformation Element of S-box

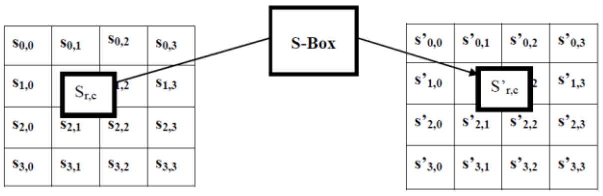
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $S_{r,c}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
S-Box
$s'_{0,0}$ $s'_{0,1}$ $s'_{0,2}$ $s'_{0,3}$
$s'_{1,0}$ $S'_{r,c}$ $s'_{1,3}$
$s'_{2,0}$ $s'_{2,1}$ $s'_{2,2}$ $s'_{2,3}$
$s'_{3,0}$ $s'_{3,1}$ $s'_{3,2}$ $s'_{3,3}$

Figure 6 Rep_Bytes() applies the S-box to each byte of the State.

The S-box used in the Rep_Bytes() transformation is presented in hexadecimal form in Fig. For example, if a1, 1 =

{53}, then the substitution value would be determined by the intersection of the row with index „5‟ and the column with index „3‟ in Figure 7. This would result in b1, 1 having a value of {ed}.

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
x	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Figure 7 S-Box: Substitution Values For The Byte (In Hexadecimal Format)

REALLOC_ROWS() TRANSFORMATION

As implied by its name, the Reallocate rows operation processes different rows. A simple rotate with a different rotate width is performed. The second row of the 4x4 byte input data (the state) is shifted one byte position to the left in the matrix, the third row is shifted two byte positions to the left, and the fourth row is shifted three byte positions to the left. The first row is not changed. This operation is shown in the figure 8.

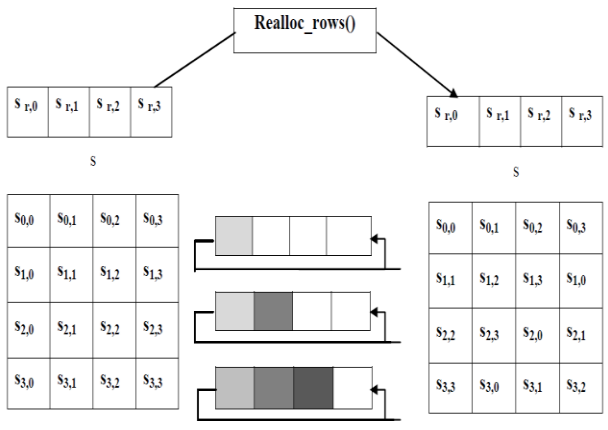
Realloc_rows()
S r,0
S r,1
S r,2
S r,3
S
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,1}$ $s_{1,2}$ $s_{1,3}$ $s_{1,0}$
$s_{2,2}$ $s_{2,3}$ $s_{2,0}$ $s_{2,1}$
$s_{3,3}$ $s_{3,0}$ $s_{3,1}$ $s_{3,2}$

Figure 8 Realloc_rows() Cyclically shifts The Last Three Rows In The State.

PERROUND_KEY() TRANSFORMATION

In the PerRound_Key() transformation, a Round Key is added to the State by a simple bitwise XOR operation. Each Round Key consists of 4 words from the key schedule. Those 4 words are each added into the columns of the State, the application of the PerRound_Key() transformation to the 10 rounds of the Cipher occurs. The action of this transformation is illustrated in Figure 9.

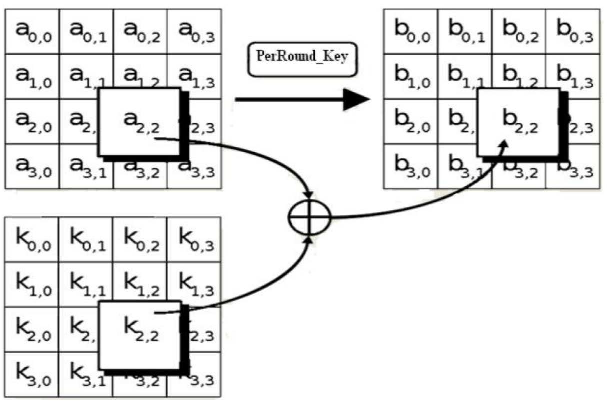
PerRound_Key
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$
$k_{0,0}$ $k_{0,1}$ $k_{0,2}$ $k_{0,3}$
$k_{1,0}$ $k_{1,1}$ $k_{1,2}$ $k_{1,3}$
$k_{2,0}$ $k_{2,2}$ $k_{2,3}$
$k_{3,0}$ $k_{3,1}$ $k_{3,2}$ $k_{3,3}$

Figure 9 PerRound_Key() XORs each column of the State with a word from key schedule

KEY EXPANSION

As previously mentioned, Key expansion refers to the process in which the 128 bits of the original key are expanded into eleven 128-bit round keys. To compute round key (n+1) from round key (n) these steps are performed:

A) Compute the new first column of the next round key as shown in Figure. First all the bytes of the old fourth column have to be substituted using the Repbytes operation. These four bytes are shifted vertically by one byte position and then XORed to the old first column. The result of these operations is the new first column. This is shown in figure 10.

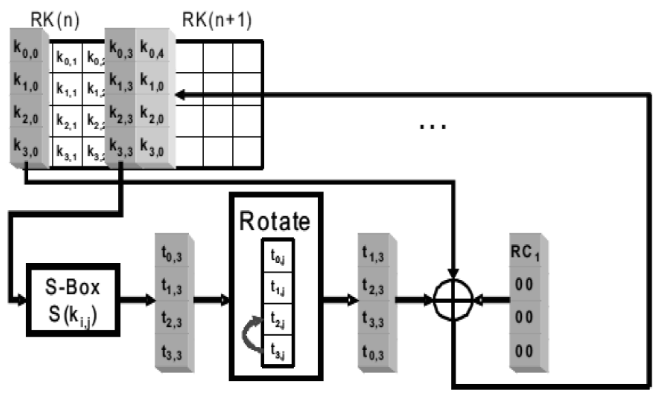
RK(n)
RK(n+1)
$k_{0,0}$ $k_{0,1}$ $k_{0,3}$ $k_{0,4}$
$k_{1,0}$ $k_{1,1}$ $k_{1,3}$ $k_{1,0}$
$k_{2,0}$ $k_{2,1}$ $k_{2,3}$ $k_{2,0}$
$k_{3,0}$ $k_{3,1}$ $k_{3,3}$ $k_{3,0}$
...
S-Box
$S(k_{i,j})$
$t_{0,3}$ $t_{1,3}$ $t_{2,3}$ $t_{3,3}$
Rotate
$t_{0,j}$ $t_{1,j}$ $t_{2,j}$ $t_{3,j}$
$t_{1,3}$ $t_{2,3}$ $t_{3,3}$ $t_{0,3}$
RC_1
00
00
00

Figure 10 Expanding First Column of Next Round Key

B) Columns 2 to 4 of the new round key are calculated as shown:

- [new second column] = [new first column] XOR [old second column]
- [new third column] = [new second column] XOR [old third column]
- [new fourth column] = [new third column] XOR [old fourth column]

Illustrates the calculation of columns 2-4 of the new round key.

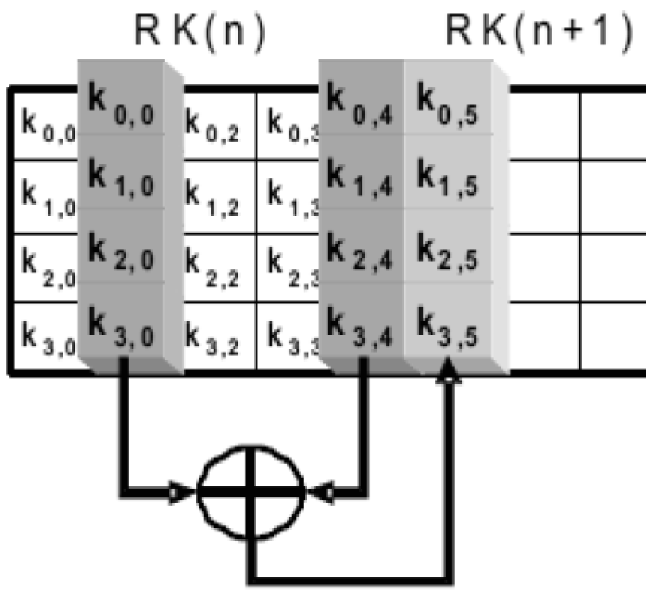
RK(n)
RK(n+1)
$k_{0,0}$ $k_{0,0}$ $k_{0,2}$ $k_{0,3}$ $k_{0,4}$ $k_{0,5}$
$k_{1,0}$ $k_{1,0}$ $k_{1,2}$ $k_{1,3}$ $k_{1,4}$ $k_{1,5}$
$k_{2,0}$ $k_{2,0}$ $k_{2,2}$ $k_{2,3}$ $k_{2,4}$ $k_{2,5}$
$k_{3,0}$ $k_{3,0}$ $k_{3,2}$ $k_{3,3}$ $k_{3,4}$ $k_{3,5}$

Figure 11 Expanding Other Columns of Next Round Key.

5. POSITIVE IMPACT OF THE ALGORITHM

- Multiple encryptions, such as triple-DES[4], will become unnecessary with this proposed algorithm. Since multiple encryption uses a plural number of keys, the avoidance of using multiple encryption will mean a reduction on the number of cryptographic keys that an application has to manage, and hence will simplify the design of security protocols and systems.
- Use of proposed algorithm will lead to the emergence of new hash functions of compatible security strengths. In several ways, block cipher encryption algorithms are closely related to hash functions. It can be a standard practice as block cipher encryption algorithms are often used to play the role of one-way hash functions. We have seen a typical "one-way transformation" usage of the DES function in the realization of the UNIX password scheme[5]. Another example is to use block cipher encryption algorithms to realize (keyed) one-way hash functions.
- As in the case that the DES's standard position had attracted much cryptanalysis attention trying to break the algorithm, and that these efforts have contributed to the advance of knowledge in block cipher cryptanalysis, the proposed algorithm as the new block cipher standard will also give rise to a new resurgence of high research interest in block cipher cryptanalysis which will certainly further advance the knowledge in the area.

6. EXPERIMENTAL RESULTS

Earlier algorithms takes huge amount of time for implementing digital signature, our algorithm implements digital signature in almost one-fourth time as taken by the DES and this is shown in following tables. We have shown the result considering 4 types of files for creation of digital signature and same amount of time is also taken at the receiver end for verification. In these tables we have shown time corresponding to various size files.

1. **PDF**

S.NO	**SIZE OF FILE (KB/MB)**	**TIME TAKEN (min: sec: hund)**
1.	**824KB**	**0:09:99**
2.	**6.87MB**	**1:31:01**
3.	**19.1MB**	**4:22:87**
4.	**54.3MB**	**9:07:44**

2. **Image Files**

S.NO	**SIZE OF FILE (KB/MB)**	**TIME TAKEN (min: sec: hund)**
1.	**566KB**	**0:05:11**
2.	**1.42MB**	**0:13:18**
3.	**4.56MB**	**0:42:30**
4.	**6.92MB**	**1:07:06**

3. **Audio Files**

S.NO	**SIZE OF FILE (KB/MB)**	**TIME TAKEN (min: sec: hund)**
1.	**1.38MB**	**0:13:68**
2.	**9.93MB**	**1:45:18**
3.	**22.1MB**	**4:11:50**
4.	**41.3MB**	**7:14:95**

4. **Video Files**

S.NO	**SIZE OF FILE (KB/MB)**	**TIME TAKEN (min: sec: hund)**
1.	**3.85MB**	**0:38:79**
2.	**21.7MB**	**3:39:82**
3.	**33.1MB**	**5:11:91**
4.	**77.3MB**	**16:37:45**

7. CONCLUSION

The widespread adoption of Internet as a secure medium for communication and e-commerce has made digital signature implementation to play a vital part of today's information systems. Now-a-days we need we have very large size documents that need to be transferred from one place to another with high security and with minimum time considerations for securing it by the use of digital signature. So, the demand of present scenario is that to develop a secure and efficient implementation of digital signature.

We have developed a digital signature implementation algorithm which exhibits processing power of high performance, efficiency and in minimum amount of time. In this we used private key cryptography for key generation because Public-key cryptography is relatively slow and is only suitable for encrypting small amounts of information while private key cryptography is much faster and is suitable for encrypting large amounts of information. Also we used hexadecimal S-box which implements a much faster processing while encryption and all the work are done in two-dimension rather than one-dimension.

We have applied the most efficient way for providing the highest security but we can further increase by taking larger key size and by using the concept of certificate authority for the distribution of keys. And for the concept of multiuser we can use the concept of different keys at sender’s or receiver’s end but we have to compromise with the confidentiality.

8. REFRENCES

[1] Chen Hai-peng, Shen Xuan-Jing, Wei Wei, "Digital Signature Algorithm Based on Hash Round Function and Self-Certified Public Key System", Education Technology and Computer Science, 2009.

[2] Sombir Singh, Sunil K. Maakar, Dr.Sudesh Kumar, "Enhancing the Security of DES Algorithm Using Transposition Cryptography Techniques", IJARCSSE Volume 3, Issue 6, June 2013.

[3] Cse.iitkgp.ac.in/~deepdeep/courses_iitkgp/crypto/slides/symmetricciphers.pdf.

[4] Mandeep Singh, Narula Simarpreet Singh, "Implementation of Triple Data Encryption Standard using Verilog" ", IJARCSSE Volume 4, Issue 1, January 2014.

[5] www.ee.usyd.edu.au/people/philip.leong/userfiles/files/papers/crypt_usenix91.pdf.