A Novel Approach of Speedy-Highly Secured Data Transmission using Cascading of PDLZW and Arithmetic Coding with Cryptography

Sankalp Prakash
Research Scholar
State Govt. Technical Education,
Jaipur (Rajasthan)

Mridula Purohit
Reader, Dept. of Mathematics
Vivekanand Institute of Tech. (East),
Jaipur (Rajasthan)

Abhishek Raizada
Research Scholar
Jaipur (Rajasthan)

ABSTRACT

The spread of computing has led to an outburst in the volume of data in the communication world. The paper proposes the cascading of two algorithms PDLZW and Arithmetic Coding with cryptography. With the hierarchical parallel dictionary set, the search time can be reduced significantly and all these dictionaries are operated independently. While the results generated by Arithmetic Coding are close to the optimal value. Since cascaded compression may achieve the higher compression ratio for the file but does not provide required security aspects. Therefore, the advantage of cryptography has been taken by XORing the compressed data with the one-time key, providing compression and security simultaneously. This paper proposes a new system JDCE, where compression is provided to the data twice before encrypted it for getting it ready for transmission.

General Terms

Arithmetic Coding, Cascaded Compression, Cryptography, Parallel Dictionary LZW (PDLZW), One-Time Pad

Keywords

Arithmetic Coding, Cascaded Compression, Cryptography, Parallel Dictionary LZW (PDLZW), One-Time Pad

1. INTRODUCTION

The field of Information Technology has grown up abruptly in last decade, which pivots around data/message transmission. There are two important factors to be considered firstly transmission speed i.e. time taken from source to destination and secondly data security or integrity of the data which means to ensure that the receiver is receiving the original message send by the sender. The length of the data/ message may vary from a few bytes to gigabytes. Therefore it becomes crucial to compress and cipher the data/message before transmission. Data compression is the process of encoding the data, so that fewer bits will be needed to represent the original data whereby the size of the data is reduced. Cryptography is an art of protecting information by transforming it (encrypting it) into an unreadable format, called cipher text. Only those who possess a secret *key* can decipher (or decrypt) the message into plain text. So, a new system JDCE (Joint Double Compression and Encryption) has been proposed in which compression is attained by cascading of PDLZW and Arithmetic Coding and then result of the compression is encrypted to provide rapid transmission and triple layer security.

Data compression has important applications in the area of data transmission as well as data storage despite of large capacity storage devices are available these days. Therefore, there is need for an efficient way to store and transmit different type of data such as text, image, audio and video to reduce execution time and memory size [11]. The general principle of data compression algorithm on text files is to conjures up an assortment of ad hoc techniques such as compression of spaces in text to tabs, creation of special codes for common words or run length coding of picture data to produce new text file which contains the same information but with new length as small as possible [7]. The effective data compression algorithm is chosen according to some scales like: Compression Size, Compression Ratio, Compression Time and Entropy [12]. Compression size means size of new compressed file. Compression ratio refers to the percentage that results from dividing the compression size by the original uncompressed file size and then multiply it by 100. Entropy is the number that results from dividing the compression size in bits by the number of symbols in the original file and scales as bits/symbol [11]. Shannon's fundamental theorem of coding states that, given messages randomly generated from a model, it is impossible to encode them into less bits (on average) than the entropy of that model [1].

LZW and the Arithmetic Coding (AC) are two of the most appropriate compression algorithms for communication because firstly, both are adaptive i.e. they do not require a prior knowledge on the text to be compressed and secondly, during the compression process they do not require transfer of extra information from the sender to the receiver in addition to the compressed text [5]. Many researchers published comparative studies on the data compression techniques such LZW, Huffman, FLC, AC, LZ-77, etc. and found LZW and AC are the best in their own territory. LZW is appropriate when aim is raw speed rather than compression performance. The conventional LZW is a dictionary based compression so it requires large amount of processing time for adjusting and searching through the dictionary [3]. The parallel dictionary LZW (PDLZW) has designed to overcome this problem. AC has its own limitations. Though PDLZW and AC work on different principles, they can be cascaded yielding higher compression ratio while being appropriate for communication purpose, but does not provide the full security of transmitting data. The proposed mechanism will overcome this issue by incorporating cryptography with the cascading of PDLZW and AC.

2. PARALLEL DICTIONARY LZW (PDLZW)

The basic idea of dictionary based compression technique given by Lempel and Ziv as LZ-77. The main disadvantage of LZ-77 is the size of buffers is very small so in 1984, Terry Welch suggested LZW algorithm. LZW is general compression algorithm capable of working on almost any type of data [11]. LZW compression creates a table of string commonly occurring in the data being compressed, searches the table to identify the longest possible input data string that exists in the table, and replaces the actual data with references into the table. The table is formed during the compression at the same time at which the data is encoded and during decompression at the same time as the data is being decoded [8]. It can typically compress large English text to about half of the original sizes. However, the limit is imposed in the conventional LZW by the fact that once the 4K dictionary is complete, no more strings can be added; and requires large amount of processing time for adjusting and searching through the dictionary [7].

To improve the limitations of conventional LZW, the dynamic LZW (DLZW) and word-based DLZW (WDLZW) algorithms were proposed. In DLZW, the dictionary has been initialized with different combinations of characters. It is organized in hierarchical string tables. The baseline idea is to store the most frequently used strings in the shorter table, which requires fewer bits to identify the corresponding string. The tables are updated using the move-to-front and weighting system with associated frequency counter. During the compression time, after the longest matching string is recognized in the table, it is moved to the first position of its block. The table updating process is based on the least recently used (LRU) policy to ensure that frequently used strings are kept in the smaller tables. This is to minimize the average number of bits required to code a string when compare with a single table implementation [2, 3, 7].

The WDLZW algorithm is a modified version of DLZW that focuses on text compression by identifying each word in the text and make it a basic unit (symbol). The algorithm encodes the input word into literal codes and copy codes. If the search for a word has failed, it is sent out as a literal code, which is its original ASCII code preceded by other codes for identification. The copy code is the address of the matching string in the string table. However, both algorithms are too complicated. To improve this, parallel dictionary LZW (PDLZW) was proposed. Since not all entries of the DLZW dictionary contains the same word size, this leads to the need of the entire dictionary search for every character. Consequently, the PDLZW has designed to overcome this problem by partitioning the dictionary into several dictionaries of different address spaces and sizes. With the hierarchical parallel dictionary set, the search time can be reduced significantly since these dictionaries can operate independently and thus can carry out their search operation in parallel [2].

2.1 Compression Algorithm for PDLZW

The PDLZW compression algorithm is based on a parallel dictionary set that consists of *m* small variable-word-width dictionaries, numbered from *0* to *m-1*, each of which increases its word width by 1 byte (1B). More precisely, dictionary0 has 1B word width; dictionary1 has 2B, and so on. The actual size of the dictionary set used in a given application can be determined by the information correlation property of the application.

In the algorithm, two variables and one constant are used. The constant *max_dict_no* denotes the maximum number of dictionaries, excluding the first single-character dictionary (i.e., dictionary0), in the dictionary set. The variable max_*matched_dict_no* is the largest dictionary number of all matched dictionaries and the variable *matched_addr* identifies the matched address within the *max_matched_dict_no* dictionary. Each compressed codeword is a concatenation of max_*matched_dict_no* and *matched_addr*.

Input: The string to be compressed.

Output: The compressed code words with each being a $\log_2 k$-bit codeword, which consists of *max_matched_dict_no* and *matched_ addr*, where *k* is the total number of entries of the dictionary set.

Begin:

1: Initialization.
- **1.1.** *string-1*← **null**.
- **1.2.** *max_matched_dict_no* ← *max_dict_no.*
- **1.3.** *update_dict_no* ← *max_matched_dict_no;. update_string* ← ∅{empty}.

2: **while** (the input buffer is **not empty**) **do**
- **2.1.** Prepare next *max_dict_no* + 1 character for searching.{*max_matched_dict_no* is reset to *max_dict_no* initially and the dictionary number of the dictionary set counts from 0 up to a constant *max_dict_no*}
  - **2.1.1.** *string-2* ← read next (*max_matched_dict_no*+1) characters from the input buffer.
  - **2.1.2.** *string* ← *string-1*‖*string-2*. (Where ‖ is the concatenation operator.)
- **2.2.** Search string in all dictionaries in parallel and set the *max_matched_dict_no* and *matched_addr*.
- **2.3.** Output the compressed codeword containing *max_matched_dict_no*‖*matched_addr*.
- **2.4.** **if** (*max_matched_dict_no* ≺ *max_dict_no* **and** *update_string* ≠ ∅) **then** add the *update_string* to the entry point by *UP[update_dict_no]* of *dictionary[update_dict_no]*.(*UP[update_dict_no]* is the update pointer associated with the dictionary).
- **2.5.** Update the update pointer of the *dictionary[max_matched_dict_no* + 1].
  - **2.5.1.** *UP[max_matched_dict_no* + 1] = *UP[max_matched_dict_no* + 1] + 1
  - **2.5.2.** **if** *UP[max_matched_dict_no* + 1] reaches its upper bound **then** reset it to 0. {FIFO update rule.}
- **2.6.** *update_string* ← extract out the first (*max_matched_dict_no* + 2) bytes from string; *update_dict_no* ← *max_matched_dict_no* + 1.
- **2.7.** *string-1* ← shift string out the first (*max_matched_dict_no* + 1) bytes.

End {End of PDLZW Compression Algorithm.}[2]

Here is an example to illustrate the operation of the PDLZW compression algorithm. It is assumed that the alphabet set ∑ is {a,b,c,d} and the input string is ababbcabbabbabc . The address space of the dictionary set is 16. The dictionary set initially contains only all single characters: a,b,c and d. The input string is grouped together by characters. These groups are denoted by a number with or without parenthesis. The number without parenthesis denotes the order to be searched

of the group in the dictionary set and the number with parenthesis denotes the order to be updated of the group in the dictionary set. After the algorithm exhausts the input string, the contents of the dictionary set and the compressed output code words will be {a,b,c,d,ab,ba,bc,ca,abb,,,,,abba,,,} and {0,1,4,1,2,8,8,4,2} respectively [2, 3].

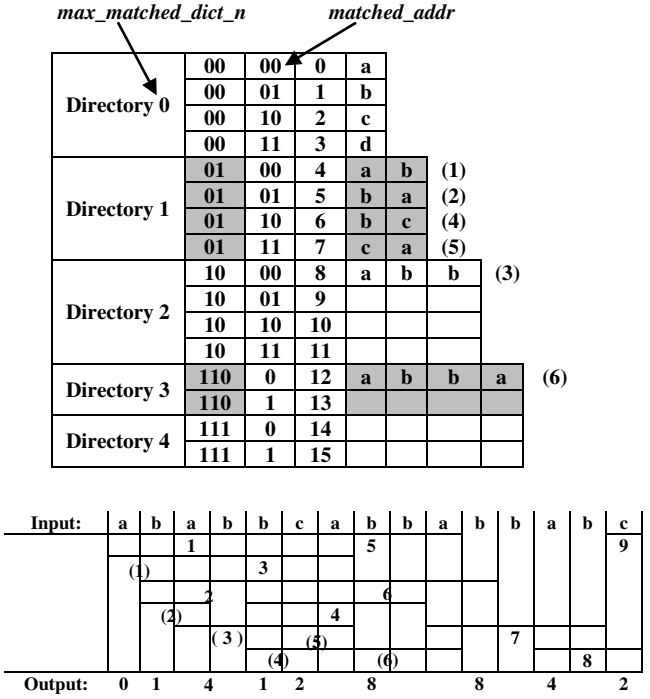


Figure 1: Example to illustrate the operation of PDLZW compression algorithm.

2.2 PDLZW Decompression Algorithm:

To recover the original string from the compressed one, reverse the operation of the PDLZW compression algorithm. This operation is called the PDLZW decompression algorithm. By decompressing the original substrings from the input compressed code words, each input compressed codeword is used to read out the original substring from the dictionary set. To do this without loss of any information, it is necessary to keep the dictionary sets used in both algorithms, the same contents. Hence, the substring concatenated of the last output substring with its first character is used as the current output substring and is the next entry to be inserted into the dictionary set. The PDLZW decompression algorithm has three variables and one constant. As in the PDLZW compression algorithm, the constant *max_dict_no* denotes the maximum number of dictionaries in the dictionary set. The variable *last_dict_no* memorizes the dictionary address part of the previous codeword. The variable *last_output* keeps the decompressed substring of the previous codeword, while the variable *current_output* records the current decompressed substring. The output substring always takes from the *last_output* that is updated by *current_output* in turn.

Input: The compressed codewords with each containing $\log_2$k-bits, where *k* is the total number of entries of the dictionary set.
Output: The original string.
Begin:
1: Initialization.

1.1. **if** (input buffer is not empty) **then** *current_output* ← **empty**; *last_output* ← **empty**; *addr* ← read next $\log_2$k-bit codeword from input buffer. {Where codeword = *dict_no* ‖ *dict_addr* and ‖ is the concatenation operator.}
1.2. **if** (*dictionary[addr]* is defined) **then** *current_output* ← *dictionary[addr]; last_output* ← *current_output; output* ← *last_output; update_dict_no* ← *dict_no[addr]* + 1.
2: **while** (the input buffer is not empty) **do**
2.1. *addr* ← read next log2k-bit codeword from input buffer.
2.2. {output decompressed string and update the associated dictionary.}
2.2.1. *current_output* ← *dictionary[addr].*
2.2.2. **if** (*max_dict_no* ≥ *update_dict_no*) **then** add (*last_output* ‖ the first character of *current_output*) to the entry pointed by *UP[update_dict_no]* of *dictionary[update_dict_no].*
2.2.3. *UP[update_dict_no]*←*UP[update_dict_no]*+ 1.
2.2.4. **if** *UP[update_dict_no]* reaches its upper bound **then** reset it to 0.
2.2.5. *last_output* ← *current_output; Output* ← *last_output; update_dict_no dict_no(addr)* + 1.
End {End of PDLZW Decompression Algorithm.}

The operation of the PDLZW decompression algorithm can be illustrated by the following example. Assume that the alphabet set ∑ is and input compressed codewords are {0, 1, 4, 1, 2, 8, 8, 4, 2}. Initially, the dictionaries numbered from 1 to 3 shown in Figure 1 are empty. By applying the entire input compressed codewords to the algorithm, it will generate the same content as is shown in Figure 1 and output the decompressed {a, b, ab, b, c, abb, abb, ab, c}substring [2, 3].

3. Arithmetic Coding Algorithm (AC)

In Arithmetic Coding, method for lossless data compression, a message is represented by an interval of real numbers between 0 and 1. As the message becomes longer, the interval needed to represent it becomes smaller, and the number of bits needed to specify that interval grows. Successive symbols of the message reduce the size of the interval in accordance with the symbol probabilities generated by the model. The more likely symbols reduce the range by less than the unlikely symbols and hence add fewer bits to the message [1]. This method is adaptive and does not need the probabilities of the symbols in the input in advance. These probabilities could be dynamically updated as input is read, and mapped into the interval [5].

The AC has advantages over Huffman Coding method (HC). HC indeed achieves "minimum redundancy." In other words, it performs optimally if all symbol probabilities are integral powers of ½. But this is not normally the case in practice; indeed, Huffman coding can take up to one extra bit per symbol. The worst case is realized by a source in which one symbol has probability approaching unity. Symbols emanating from such a source convey negligible information on average, but require at least one bit to transmit [1]. Arithmetic coding dispenses with the restriction that each symbol must translate into an integral number of bits, thereby coding more efficiently. It actually achieves the theoretical entropy bound to compression efficiency for any source

[13].In this article, the coding algorithm is adapted from an algorithm originally presented in C by Mark Nelson [16].

In order to construct the output number, the symbols being encoded have to have a set probabilities assigned to them. In general, using AC depends on creating a statistical model of the data. For example, to encode the random message "BILL GATES", would have a probability distribution. Once the character probabilities are known, the individual symbols need to be assigned a range along a *probability line*, which is nominally 0 to 1. It doesn't matter which characters are assigned which segment of the range, as long as it is done in the same manner by both the encoder and the decoder. The nine character symbol set use here would look like this:

Table 1. Initial Assignment

Character	Probability	Range
SPACE	1/10	0.00 – 0.10
A	1/10	0.10 – 0.20
B	1/10	0.20 – 0.30
E	1/10	0.30 – 0.40
G	1/10	0.40 – 0.50
I	1/10	0.50 – 0.60
L	2/10	0.60 – 0.80
S	1/10	0.80 – 0.90
T	1/10	0.90 – 1.00

Each character is assigned the portion of the 0-1 range that corresponds to its probability of appearance. Note also that the character "owns" everything up to, but not including the higher number. So the letter 'T' in fact has the range 0.90 – 0.9999….The most significant portion of an arithmetic coded message belongs to the first symbol to be encoded. When encoding the message "BILL GATES", the first symbol is "B". In order for the first character to be decoded properly, the final coded message has to be a number greater than or equal to 0.20 and less than 0.30. To encode this number is to keep track of the range that this number could fall in. So after the first character is encoded, the low end for this range is 0.20 and the high end of the range is 0.30.

After the first character is encoded, range for output number is now bounded by the low number and the high number. What happens during the rest of the encoding process is that each new symbol to be encoded will further restrict the possible range of the output number. The next character to be encoded, 'I', owns the range 0.50 through 0.60. If it was the first number in the message, set low and high range values directly to those values. But 'I' is the second character. So 'I' owns the range that corresponds to 0.50-0.60 in the new sub range of 0.2 – 0.3. This means that the new encoded number will have to fall somewhere in the 50^{th} to 60^{th} percentile of the currently established range. Applying this logic will further restrict number to the range 0.25 to 0.26.

The algorithm to accomplish this for a message of any length is shown below:

```
Set low to 0.0
Set high to 1.0
While there are still input symbols do
   get an input symbol
   code_range = high - low.
   high = low + range*high_range(symbol)
   low = low + range*low_range(symbol)
End of While
output low
```

So the final low value, 0.2572167752 will uniquely encode the message "BILL GATES" using present encoding scheme.

Given this encoding scheme, it is relatively easy to see how the decoding process will operate. Find the first symbol in the message by seeing which symbol owns the code space that the encoded message falls in. Since the number 0.2572167752 falls between 0.2 and 0.3, the first character must be "B". So remove the "B" from the encoded number. Since the low and high ranges of B, their effects can be removed by reversing the process that put them in. First, subtract the low value of B from the number, giving 0.0572167752. Then divide by the range of B, which is 0.1. This gives a value of 0.572167752. Now it can be calculated where that lands, which is in the range of the next letter, "I" and so on.

The algorithm for decoding the incoming number looks like this:

```
get encoded number
Do
   find symbol whose range straddles the encoded number
   output the symbol
   range = symbol low value - symbol high value
   subtract symbol low value from encoded number
   divide encoded number by range
until no more symbols
```

In summary, the encoding process is simply one of narrowing the range of possible numbers with every new symbol. The new range is proportional to the predefined probability attached to that symbol. Decoding is the inverse procedure, where the range is expanded in proportion to the probability of each symbol as it is extracted.

3.1 Practical Matters

The process of encoding and decoding a stream of symbols using AC is not too complicated. But at first glance, it seems completely impractical. Most computers support floating point numbers of up to 80 bits or so. As it turns out, AC is the best accomplished using standard 16-bit and 32-bit integer math. No floating point math is required, nor would it help to use it. What is used instead is an incremental transmission scheme, where fixed size integer state variables receive new bits in at the low end and shift them out the high end, forming a single number that can be as many bits long as are available on the computer's storage medium.

The previous section has shown how the algorithm works by keeping track of a high and low number that bracket the range of the possible output number. When the algorithm first starts up, the low number is set to 0.0, and the high number is set to 1.0. The first simplification made to work with integer math is to change the 1.0 to 0.999…., or .111… in binary.

In order to store these numbers in integer registers, first justify them so the implied decimal point is on the left hand side of the word. Then load as much of the initial high and low values as will fit into the word size. The implementation uses 16-bit unsigned math, so the initial value of high is 0xFFFF, and low is 0. The high value continues with FFs forever, and low continues with 0s forever, so those extra bits can be shifted in with impunity when they are needed. If imagine the "BILL

GATES" example in a 5 digit register, the decimal equivalent of setup would look like this:

HIGH: 99999
LOW: 00000

In order to find the new range numbers, it is needed to apply the encoding algorithm from the previous section. First calculate the range between the low value and the high value. The difference between the two registers will be 100000, not 99999. This is because it is assumed the high register has an infinite number of 9′s added on to it, so need to increment the calculated difference. Then compute the new high value using the formula from the previous section:

high = low + high_range(symbol)

In this case the high range was 0.30, which gives a new value for high of 30000. Before storing the new value of high, it is needed to decrement it, once again because of the implied digits appended to the integer value. So the new value of high is 29999. The calculation of low follows the same path, with a resulting new value of 20000. So now high and low look like this:

HIGH: 29999 (999…)
LOW: 20000 (000…)

At this point, the most significant digits of high and low match. Due to the nature of the algorithm high and low can continue to grow closer to one another without quite ever matching. This means that once they match in the most significant digit, that digit will never change. So the output can be obtained of that digit as the first digit of the encoded number. This is done by shifting both high and low left by one digit, and shifting in a 9 in the least significant digit of high.

As this process continues, high and low are continually growing closer together, and then shifting digits out into the coded word.

This scheme works well for incrementally encoding a message. There is enough accuracy retained during the double precision integer calculations to ensure that the message is accurately encoded. However, there is potential for a loss of precision under certain circumstances.

The cumulative frequency table is stored in frequency orders to minimize the number of updates to it after every symbol is processed. Translation tables of character to index and index to character are used to simplify the process of sorting the cumulative frequency table. These translation tables are also adjusted whenever the cumulative frequency table updated. To overcome the overflow and underflow problems of the integer arithmetic, frequencies are scaled down by a normalization factor at regular interval [5].

4. Cascading of PDLZW and AC

Each compression algorithm has its own limitations and this is true with PDLZW and AC too. So solution to overcome the weakness of one compression technique has been found by combining it to another compression technique. This process is known as cascaded compression. In this process, the raw data is given to the PDLZW encoding algorithm. The output of the PDLZW is given to the AC for further compression. The decompression process is totally reversing [9]. Figure 2 shows the block diagram of cascading of PDLZW and AC.

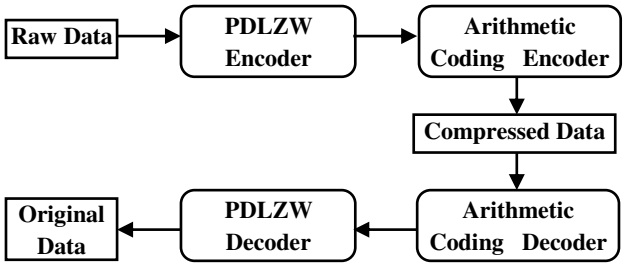


Figure 2: Block Diagram of Cascading of PDLZW and Arithmetic Coding.

Table 2 shows the results of implementation of Cascading of PDLZW and Arithmetic Coding on various text files.

Table 2. Implementation Results of Cascading of PDLZW and AC

File Name	Original Size (Bytes)	After Cascading Compression	
		File Size (Bytes)	Compression Ratio
new.txt	261	97	62.83
file1.txt	12288	948	92.88
file2.txt	68608	2560	96.26
file3.txt	88064	2867	96.74
main.txt	872448	4505	94.83

5. CRYPTOGRAPHY

In this fast-paced technological world the importance of information and communication systems is escalating with the increasing significance and quantity of data that is transmitted. Unfortunately systems and data are increasingly vulnerable to a variety of threats, such as unauthorized access and use, misappropriation, alteration, and destruction. Cryptography is the foundation of all data as well as information security aspects. Classical cryptosystems is very easy to understand, easily implemented and very easy to be broken. New forms of cryptography came after the widespread development of computer communications. In data and telecommunications, cryptography is necessary when communicating over any untrusted medium.

In the present scenario the cryptographic techniques have become the immediate solution to protect information against third parties. These techniques required that data and information should be encrypted with some sort of mathematical algorithm where only the party that shares the information could possible decrypt to use the information. Within the context of any communication, there are some specific security requirements includes (1) Authentication which means the process of providing one's identity; (2) Confidentiality that ensures no one can read the message except the intended receiver; (3) Integrity for assuring the receiver, the received message has not been altered in any way from the original and (4) Non-repudiation is a mechanism to prove that the sender really send this message [19].

Cryptography itself splits into here main branches:

(1) Symmetric (or Private-Key) Algorithm: two parties have an encryption and decryption method for which they share a secret key.

(2) Asymmetric (or Public-Key) Algorithm: a user possesses a secret key (private key) as in symmetric cryptography but also a public key.

(3) Hybrid Cryptography: symmetric and asymmetric algorithms (and often also hash functions) are all used together.

Further the symmetric cryptography can be divided in stream ciphers and block ciphers. Stream ciphers encrypt bits individually. This is achieved by adding a bit from a key stream to a plain bit. Block ciphers encrypt an entire block of plain text bits at a time with the same key. In stream ciphers each bit x_i is encrypted by adding a secret key stream bit s_i modulo 2. If arithmetic modulo 2 is done, the only possible values are 0 and 1 because if a number is divided by 2 the only possible remainders are 0 or 1. If the truth table of modulo2 addition is drawn then it is found that it is similar to the *exclusive-OR (XOR)* gate. So the XOR operation plays a major role in modern cryptography.

Though how good the compression techniques are, they do not provide security of data/message from intruders, hackers and code-breakers. So the compressed data must be encrypted to provide authenticity, confidentiality, integrity and non-repudiation.

6. NEW PROPOSED 3-TIER SYSTEM (JDCE)

This paper proposes a new 3-tier system that provides double compression with cryptography for speedy-highly secured data transmission. In this system, first compress the raw data using PDLZW encoder (Tier-1), and the output of this encoder is redirected to AC encoder (Tier-2). That is how the highly compressed data is achieved. Since the compression technique are not secured, so the cryptography has been integrated with the compression techniques. The output of Tier-2 is a number which is actually the compressed data. In this paper a truly unbreakable cipher: the One-Time Pad (OTP) is being used [21]. A stream cipher for which (1) the key stream s_0, s_1,s_2,… is generated by a true random number generator (2) the key stream is only known to the legitimate communicating parties, and (3) every key stream bit s_i is only used once, is called a One-Time Pad. The OTP is unconditionally secure [21]. Obtained number from Tier-2 is encrypted using private key encryption technique by XORing it with OTP which is a random key as long as message (Tier-3). This one time key is used to encrypt and decrypt a single message, and then discarded. Each new message requires a new key of same length as a new message. The output is highly compressed and secured. The Figure 2 has been modified here to achieve JDCE as Figure 3.

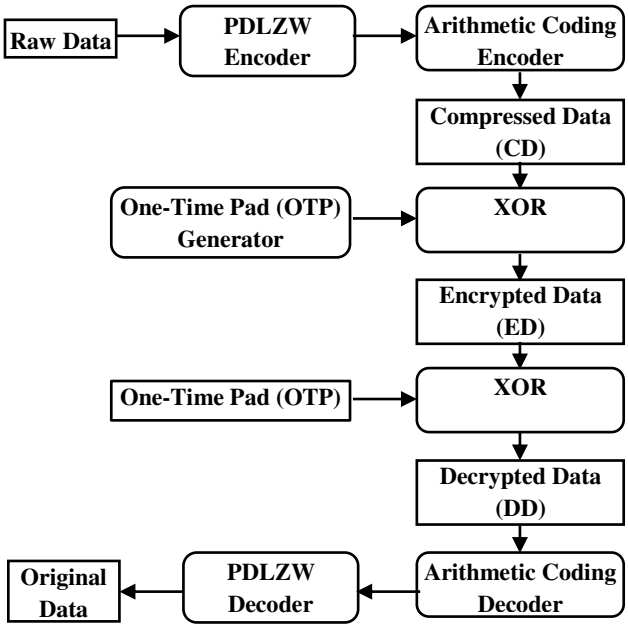


Figure 3: Block Diagram of Proposed 3-Tier System (JDCE).

7. CONCLUSION

The proposed technique 3-Tier System (JDCE) provides an excellent integration of data compression by cascading of PDLZW and Arithmetic Coding algorithms along with the cryptography to enhance the data security and transfer rate during data communication. In this technique the data size can be reduced by using cascaded compression technique and after that compressed data can be encrypted to provide the data security. The present network scenario demands exchange of information with reduction in both space requirement for data storage and time for data transmission along with security. The proposed technique fulfils all such requirements as this technique use the concept of data compression and encryption. This paper can be extended for the storage of files.

8. REFERENCES

[1] Ian H. Witten, Radford M. Neal, and John G. Cleary. Arithmetic coding for data compression. Commun. ACM, 30(6):520–540, June 1987.

[2] Ming-Bo Lin, Jang-Feng Lee, and Gene Eu Jan. A lossless data compression and decompression algorithm and its hardware architecture. IEEE Trans. Very Large Scale Integr. Syst., 14(9):925–936, September 2006.

[3] Ming-Bo Lin. A hardware architecture for the lzw compression and decompression algorithms based on parallel dictionaries. J. VLSI Signal Process. Syst., 26(3):369–381, November 2000.

[4] Mark Nelson and Jean-Loup Gailly. The Data Compression Book. M&T Books, 2nd edition, 1996.

[5] Yehoshua Perl, V. Maram, and N. Kadakuntla. The cascading of the LZW compression algorithm with arithmetic coding. In James A. Storer and John H. Reif, editors, Data Compression Conference, pages 277–286. IEEE Computer Society, 1991.

[6] Archana V. Nair. S, G. Kharmega Sundararaj and T. Sudarson Rama Perumal. Simultaneous compression and encryption using arithmetic coding with randomized bits. International Journal of Computer Technology and Electronics Engineering, 2:38–42, April 2012.

[7] P. Vichitkraivin and Orachat Chitsobhuk. An Improvement of PDLZW implementation with a Modified WSC Updating Technique on FPGA. World Academy of Science, Engineering and Technology, 2009.

[8] David Salomon. Data compression-The Complete Reference, 4th Edition. Springer, 2007.

[9] Nirali Thakkar and Malay Bhatt. Cascading of the PDLZW compression algorithm with arithmetic coding. International Journal of Computer Applications, 46(16):21–24, May 2012. Published by Foundation of Computer Science, New York, USA.

[10] Ajit Singh and Rimple Gilhotra. Data security using private key encryption system based on arithmetic coding, May 2011.

[11] Haroon Altarawneh and Mohammad Altarawneh. Data compression techniques on text files: A comparison study. International Journal of Computer Applications, 26(5):42–54, July 2011. Published by Foundation of Computer Science, New York, USA.

[12] Debra A. Lelewer and Daniel S. Hirschberg. Data compression. ACM Comput. Surv., 19(3):261–296, September 1987.

[13] Jiantao Zhou, Oscar C. Au, Xiaopeng Fan, and Peter H. W. Wong. Joint security and performance enhancement for secure arithmetic coding. In ICIP, pages 3120–3123, 2008.

[14] Raj S. Katti, Sudarshan K. Srinivasan, and Aida Vosoughi. On the security of randomized arithmetic codes against ciphertext-only attacks. IEEE Transactions on Information Forensics and Security, 6(1):19–27, 2011.

[15] Helen A. Bergen and James M. Hogan. A chosen plaintext attack on an adaptive arithmetic coding compression algorithm. Computers & Security, pages 157–167, 1993.

[16] Mark R. Nelson. Arithmetic coding and statistical modeling: achieving higher compression ratios. Dr. Dobb's J., 16(2):16–ff., December 1990.

[17] Whitfield Diffie & Martin E. Hellman. Privacy and authentication: An introduction to cryptography, 1979.

[18] Tarek M Mahmoud, Bahgat A. Abdel-latef, Awny A. Ahmed, Ahmed M Mahfouz, Tarek M. Mahmoud, Bahgat A. Abdel-latef, Awny A. Ahmed, and Ahmed M. Mahfouz. Hybrid compression encryption technique for securing sms. International Journal of Computer Science and Security, 2009.

[19] Gary C. Kessle. An overview of cryptography. [Online]. http://www.garykessler.net/library/crypto.html.

[20] William Stallings. Cryptography and Network Security: Principles and Practice. Prentice Hall Press, Upper Saddle River, NJ, USA, 5th edition, 2010.

[21] Christof Paar and Jan Pelzl. Understanding Cryptography - A Textbook for Students and Practitioners. Springer, 2010.

International Journal of Information, Communication and Computing Technology
Jagan Institute of Management Studies, New Delhi

Applied Hybrid Cryptography in Key-pair Generation of RSA implementation

[1]Sankalp Prakash,[2] Mridula Purohit

ABSTRACT

A key is a crucial factor in any cryptographic system. Devoid of a key, the algorithm would not produce any fruitful result. Key is being used in encryption, decryption and other cryptographic algorithms such as digital signature schemes and MAC (message authentication codes). The only thing that should be kept secret in a sound cryptosystem is the key so the key generation is important in implementation of cryptographic algorithms. In this paper the key generation module of the developed application package has been discussed which generates private and public keys in pairs using RSA and the corresponding private keys generated are encrypted using DES algorithm to apply hybrid cryptography.

KEYWORDS

Hybrid cryptography, Key-pair Generation, Public Key, Private Key, RSA, DES

INTRODUCTION

In this fast-paced technological world the importance of information and communication systems is escalating with the increasing significance and quantity of data that is transmitted. Unfortunately the vulnerability of systems and data are highly rising due to variety of threats, such as unauthorized access and use, destruction, alteration and misappropriation. Cryptography is the foundation of all data as well as information security aspects. Classical cryptosystems are quite simple to understand, implement and to be broken. Many new forms of cryptographic algorithms came after the extensive expansion of computer communications. Cryptography is being used as a tool in an information technology security environment to protect sensitive and high value data that is vulnerable to unauthorized disclosure or undetected modification during transmission or in storage.

In the present scenario the cryptographic techniques have become the immediate solution to protect information against third parties. These techniques required that data and information should be encrypted with some sort of mathematical algorithm where only the party that shares the information could possible decrypt to use the information [13]. Within the context of any communication, there are some specific security requirements includes (1) Authentication which means the process of providing one's identity; (2) Confidentiality that ensures no one can read the message except the intended receiver; (3) Integrity for assuring the receiver, the received message has not been altered in any way from the original and (4) Non-repudiation is a mechanism to prove that the sender really send this message. [14]

Cryptography itself splits into here main branches:

(1) Symmetric (or Private-Key) Cryptography: two parties have an encryption and decryption method for which they share a secret key.
Data Encryption Standard (DES) is perhaps the most well-known and widely used symmetric key cryptosystem in the world. It is a symmetric block algorithm written by IBM that encodes 64-bit blocks of data using a 56-bit key.

(2) Asymmetric (or Public-Key) Cryptography: a user possesses a secret key (private key) as in symmetric cryptography but also a public key.
RSA, developed by Ronald L. Rivest, Adi Shamir and Leonard M. Adleman in 1977 at MIT [11], is still most widely used and is believed to be secure given sufficiently long keys and the use of up-to-date implementations. [17] It was a practical public-key cipher based on the difficulty of factoring very large numbers. The idea is that it is easy to find two large primes but it is difficult to factor the product of the two primes.

(3) Hybrid Cryptography: Symmetric and asymmetric ciphers each have their own advantages and disadvantages. Symmetric ciphers are significantly faster (Schneier states "at least 1000 times faster") than asymmetric ciphers, but require all parties to somehow share a secret (the key) [1]. The asymmetric algorithms allow public key infrastructures and key exchange systems, but at the cost of speed. So a hybrid cryptosystem is protocol to combination of the specific advantages of the two presently used encryption methods – speed (symmetrical encryption) and security (asymmetrical encryption). In other words symmetric and asymmetric algorithms (and often also hash functions) are all used together.

In designing security systems, it is wise to assume that the details of the cryptographic algorithm are already available to the attacker. This principle is known as Kerckhoffs' principle – "only secrecy of the key provides security", or, reformulated as Shannon's maxim, "the enemy knows the system". The history of cryptography provides evidence that it can be difficult to keep the details of a widely-used algorithm secret. [15]

In the real world, key management is the hardest part of cryptography. Keeping the keys secret is much harder. Cryptanalysts often attack both symmetric and public key cryptosystems through their key management. So the security of a cryptographic algorithm rests in the key. If someone using a cryptographically weak process to generate keys then the whole system is weak [1].

[1] *Research Scholar, Department of Computer Science & Engineering, Jagannath University, Jaipur (Rajasthan)*
[2] *Professor, Department of Mathematics, Vivekanand Institute of Technology (East), Jaipur (Rajasthan)*
Email: [1] sankalp1973@gmail.com, [2] mridula_purohit@yahoo.co.in

KEY GENERATION OF RSA IMPLEMENTATION

Key generation consists of the following two tasks:

1. Determine two prime numbers, *p* and *q*,
2. Selecting e and calculating the *d*.

First consider the selection of *p* and *q* because the value $n = p \times q$ will be known to any potential opponent, to prevent discovery of *p* and *q* by exhaustive methods, these primes must be chosen from a sufficiently large set (i.e. p and q must be large numbers ~ 100 digits). But the method used for finding large primes must be reasonably efficient.

The following procedure is usually adopted for picking prime numbers:

1. Pick an odd integer at random using a pseudorandom number generator.
2. Pick an integer $a \prec n$ at random.
3. Perform the probabilistic primality test, such as Miller-Rabin. If *n* fails the test, reject *n* and go to step 1.
4. If *n* has passed a sufficient number of tests, accept *n;* otherwise go to step 2.

This is tedious procedure but it is performed only when a new key pair {KU, KR} in needed.

The traditional method of generating random numbers is to pass the results of a pseudorandom number generator through a hash function like MD5 or SHA and use the hash result. This is however much slower and instead faster approach was adopted in the development of this application. The developed application uses the shuffling method as outlined in Computer Generated Random Numbers [5]. Additionally it uses around 16 random entropy sources including different pseudorandom number generators. The basic method is outlines below:

Random Number Generation

- Start with an array of dimension around 200.
- Seed all the pseudorandom number generators by cascading one random number to seed to the next one. The first generator is seeded with system time.
- Fill the first 100 elements with the results of the first pseudorandom number generator.
- Fill the next 100 elements with the results of the second pseudorandom number generator.
- When the program wants a random number, randomly choose one from the array and send it to the program.
- Replace the number chosen in the array with a new random number from the main random number generator which uses all the 16 randomness entropy sources.

Primality Test

The random number values generated are made odd and then checked for primality using the probabilistic Miller-Rabin algorithm.

MILLER-RABIN (n, t)
INPUT: an odd number $n \geq 3$ and security parameter $t \geq 1$.
OUTPUT: an answer "PRIME" or "COMPOSITE" to the question: "Is n Prime?"

1. Write $n - 1 = 2^s r$ such that r is odd.
2. For I from 1 to t do the following
   2.1. Choose a random number integer a, $2 \leq a \leq n - 2$
   2.2. Compute $y = a^r \bmod n$
   2.3. If $y \neq 1$ and $y \neq n - 1$ then do the following:
   $j \leftarrow 1$.
   While $j \leq s - 1$ and $y \neq n - 1$ do the following:
   Compute $y \leftarrow y^2 \bmod n$
   If $y = 1$ then return ("COMPOSITE")
   $j \leftarrow j + 1$
   if $y \neq n - 1$ then return ("COMPOSITE")
3. Return ("PRIME").

Calculation of Private Key

Having determined prime numbers p and q, we select e and calculated using the Extended Eulid's algorithm:

EXTENDED_EULID(U,V)
INPUT: two nonnegative integers u and v.
OUTPUT: a vector (u1, u2, u3) such that uu1 + vv1 = u3 = gcd(u, v)

REMARK: Temporary vectors (v1, v2, v3) and (t1, t2, t3) are used in such a way that the following hold throughout the calculation:
ut1 + vt2 = t3 uu1 + vu2 = u3 uv1 + uv2 = v3

1. Initialize: set(u1, u2, u3) ← (1, 0, u) and (v1, v2, v3) ← (0, 1, v)
2. If v3 = 0 stop.
3. Set $q \leftarrow \lfloor u3 / v3 \rfloor$ and then set:
   (t1, t2, t3) ← (u1, u2, u3) – (v1, v2, v3)q
   (u1, u2, u3) ← (v1, v2, v3)
   (v1, v2, v3) ← (t1, t2, t3)
   Return to step2

THE HYBRID ENCRYPTION ALGORITHM

A hybrid encryption algorithm has the advantages of both the symmetric and asymmetric algorithms. This process involves the following steps:

(1) Generate key pair i.e. Public key KU= {e, n} value and Private Key KR= {d, n} using RSA key generation
(2) Save the Public Key {e, n}.
(3) Encrypt Private Key {d, n} by using DES algorithm of symmetric cryptography and then Save it in a file for better security.
(4) Now encrypt the message by using the Public Key KU.
(5) At the time of decryption, first the private key is decrypted and then the encrypted message can be decrypted using private key KR.

The complete process can be viewed in the figure 1.

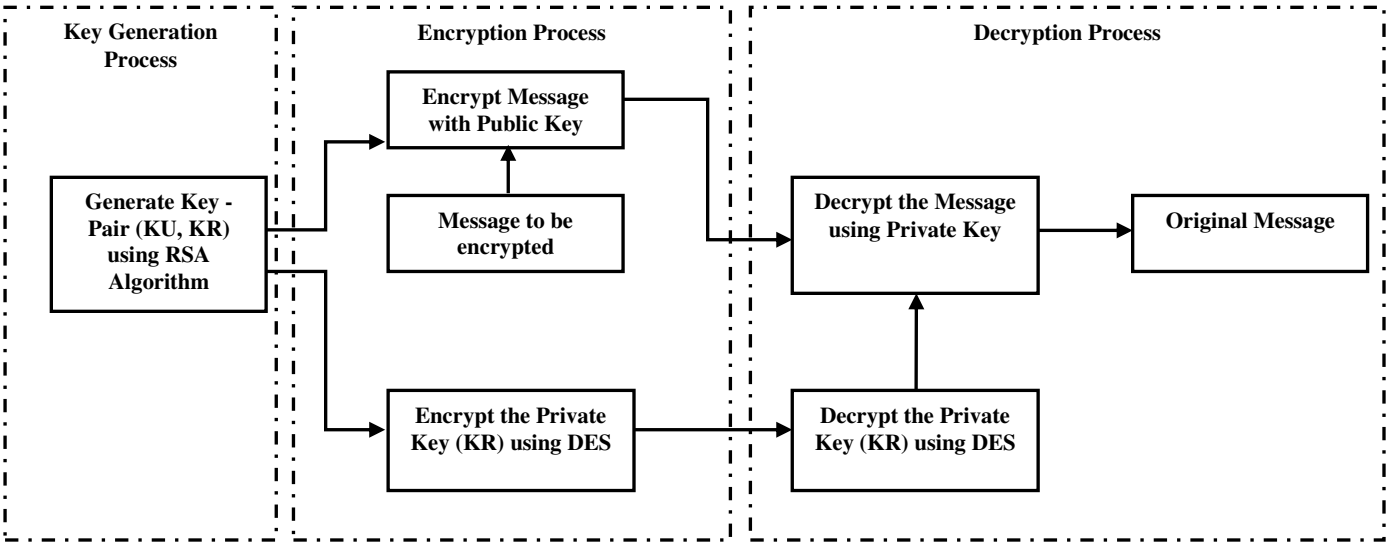


Figure 1: Hybrid Cryptography in RSAAPP

THE DEVELOPED APPLICATION (RSAAPP)

The key generation process generates the public and private keys in pairs. If required the keys can be viewed in hex format after generation. The corresponding private keys generated are encrypted using DES after taking 8-character password as user input. Both keys are made read only. The figures 2 to 5 below show the execution of the developed application i.e. RSAAPP-

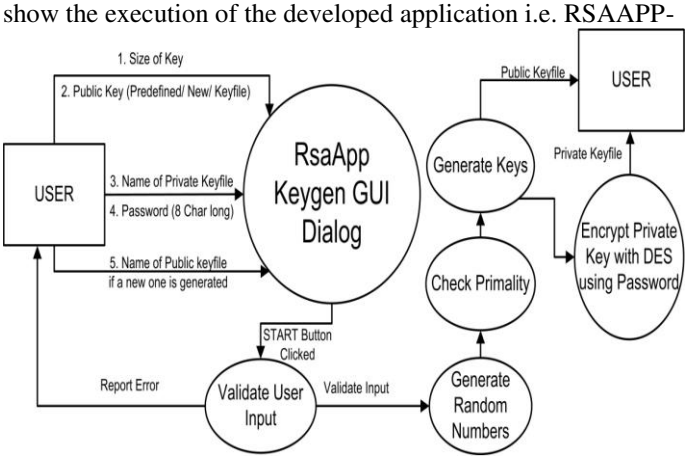


Figure 2: DFD for Key Generation Process in RSAAPP

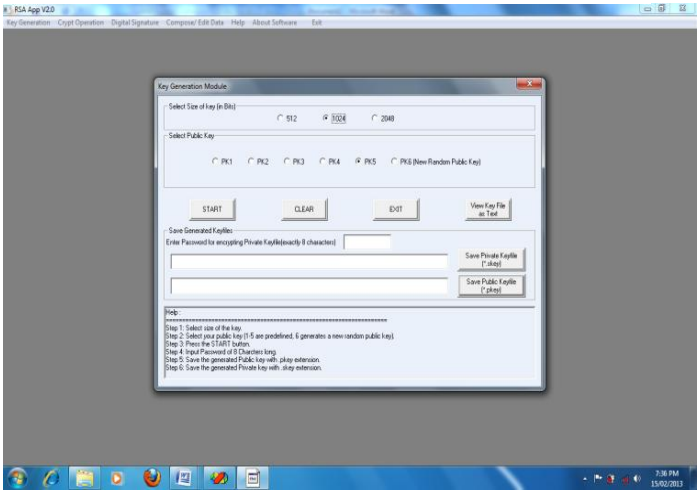


Figure 3: Key Generation User Interface

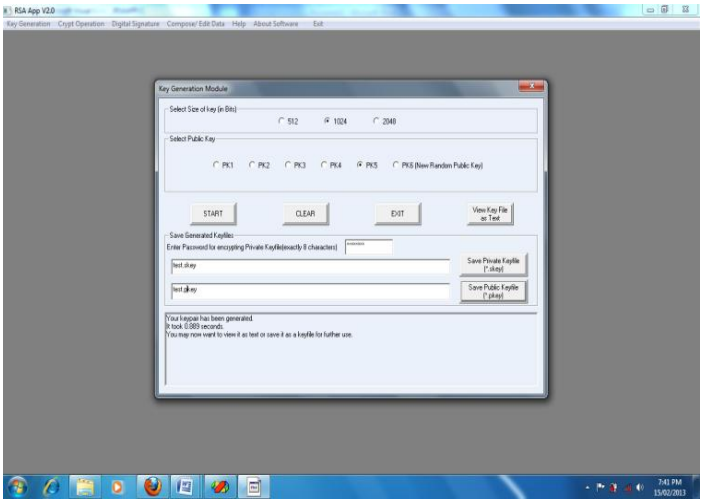


Figure 4 : After Completing Key Generation Process

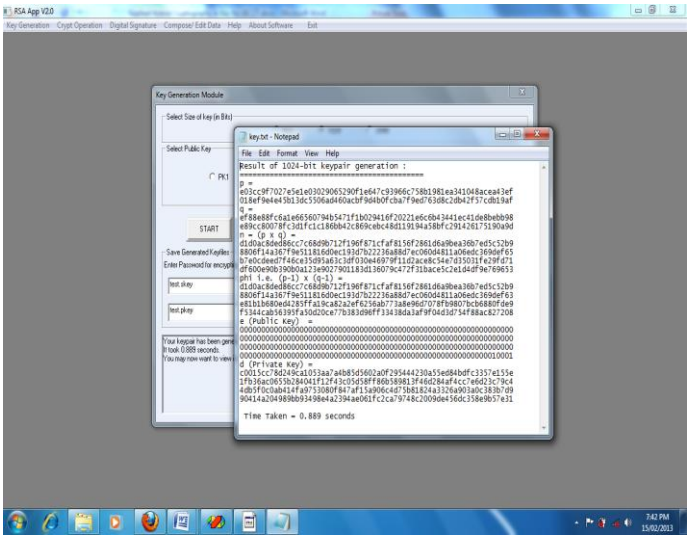


Figure 5: Result of the Key Generation

CONCLUSION

Soundness of Cryptosystem relies upon two basic factors: (1) algorithm and (2) key. The algorithm is a mathematical function, and the key is a parameter used by that function. A key is often easier to protect because it is a small piece of information, and easier to change if compromised. Thus, the security of an encryption system in most cases relies on some key being kept secret but practically it is very difficult. An attacker who obtains the key by theft, extortion, dumpster dives or social engineering can recover the original message from the encrypted data.

Cryptanalysts often attack both symmetric and public key cryptosystems through their key management. So the security of a cryptographic algorithm rests in the key. Therefore in the developed application the key generation process is made modular, efficient and fast enough so that it can generate the highly secured key-pairs in reasonable time and then the generated private key is being protected using DES which provides the advantages of hybrid cryptosystem in the RSAAPP.

REFERENCES

[1] Bruce Schneier, *Applied Cryptography: Protocols, Algorithms and Source code in C*, 2nd ed. John Wiley & Sons, New York, 1996.

[2] Alexander W. Dent, "Hybrid Cryptography," August 2004, information Security Group, Royal Holloway, University of London Egham Hill, Egham, Surrey, TW20 0EX,UK.

[3] Alfred Menezes, Paul C. van Oorschot and Scott A. Vanstone, *Handbook of Applied Cryptography*. CRC Press, Boca Raton, FL, 1996.

[4] ChristofPaar and Jan Pelzl, *Understanding Cryptography: A Textbook for Students and Practitioners*. London: Springer Monograph Series, 2009

[5] David W. Deley, "Computer Generated Random Numbers," 1991, at http://www.virtualschool.edu/mon/Crypto/RandomNumberMath.

[6] RSA Algorithm online at http://www.di-mgt.com.au/rsa_alg.html

[7] Dennis Hofheinz and EikeKiltz, "Secure Hybrid Encryption from Weakened Key Encapsulation," in *Advances in Cryptology - CRYPTO 2007*. Springer, 2007, pp. 553–571.

[8] L. Granboulan, "RSA hybrid encryption schemes," Cryptology ePrint Archive, Report 2001/110, 2001, at http://eprint.iacr.org/2001/110.ps

[9] K. Kurosawa and Y. Desmedt, "New Paradigm of Hybrid Encryption Scheme," in *Advances in Cryptology CRYPTO 2004*, ser. Lecture Notes in Computer Science, M. Franklin, Ed., vol. 4622. Springer-Verlag, 2004, pp. 426–442.

[10] MihirBellare and Phillip Rogaway, "Introduction to Modern Cryptography," in *UCSD CSE 207 Course Notes*, 2005, p. 207, http://www.cs.ucdavis.edu/rogaway/classes/227/spring05/book/main.pdf

[11] R. L. Rivest, A. Shamir, and L. M. Adleman, "A method for obtaining digital signatures and public key cryptosystems," *Comm. ACM*, vol. 21, issue 2, pp. 120–126, Feb. 1978.

[12] S. Subasree and N. K. Sakthivel, "Design of a New Security Protocol Using Hybrid Cryptography Algorithms," *International Journal of Research and Reviews in Applied Sciences*, vol. 2, no. 2, February 2010.

[13] Onwutalobi Anthony-Claret Department of Computer Science University of Wollongong "Using Encryption Technique"

[14] Whitfield Diffie & Martin E. Hellman "Privacy and Authentication: An Introduction to Cryptography", *proceedings of the IEEE*, vol.67, no.3, 1979.

[15] Key (cryptography) - Wikipedia, the free encyclopedia at http://en.wikipedia.org/wiki/Key_(cryptography)

[16] RSA - Wikipedia, the free encyclopedia at http://en.wikipedia.org/wiki/RSA.

IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661, p- ISSN: 2278-8727Volume 15, Issue 6 (Nov. - Dec. 2013), PP 27-33
www.iosrjournals.org

An Efficient implementation of PKI architecture based Digital Signature using RSA and various hash functions (MD5 and SHA variants)

Sankalp Prakash[1], Mridula Purohit[2]
[1](Computer Science & Engineering, Jagannath University, Jaipur, Rajasthan, India)
[2](Mathematics Department, Vivekanand Institute of Technology (East), Jaipur, Rajasthan, India)

Abstract: *Digital Signature technique is widely being used to detect unauthorized modification to data and to authenticate the identity of the signatory. It is essential for secure transaction over unsecure/ open networks. Digital Signature schemes are mostly used in cryptographic protocols to provide services like entity authentication, authenticated key transport and key agreement. The PKI (Public Key Infrastructure) based digital signature architecture is related with RSA algorithm and secure Hash functions (MD5 &, SHA variants). RSA digital signature algorithm is an asymmetric cryptographic method whose security is associated with difficulty of factorization and hash function is applied to the message to yields a fixed-size message digest. This paper explores the PKI architecture based digital signature and presents an efficient way of its implementation and discusses various issues associated with signature schemes based upon RSA and hash functions. The results show that signing and verification are much faster in the developed application.*
Keywords: *Digital Signature, MD5, RSA, SHA1, SHA2*

I. Introduction

In this fast-paced technological world the importance of information and communication systems is escalating with the increasing significance and quantity of data that is transmitted to minimize operational cost and provide enhanced services. Unfortunately the vulnerability of systems and data are highly rising due to variety of threats, such as unauthorized access and use, destruction, alteration and misappropriation. Cryptography is the foundation of all data as well as information security aspects. A digital signature is an important type of authentication in the public key cryptographic system and it is widely used around the world. (Bruce Schneier, 1996; W.C.Cheng, C.F.Chou and L.Golubchik, 2002) .

By allowing the exchange of information more quickly, easily, and dependably than ever before, the Internet has forever changed the way of business and transactions. Electronic transactions are gaining in importance as nations around the globe because of significantly reducing the need for paper documentation while providing the opportunity for tremendous efficiency and productivity gains. As a result, digital signatures are poised to enter the mainstream as primary vehicle for establishing trust for a wide variety of electronic transactions. (Burton S. Kaliski, 2001; Rivest, Shamir, & Adleman, 1978) The information handled in electronic transactions is valuable and sensitive and must be protected against tampering by malicious third parties (who are neither the senders nor the recipients of the information). Sometimes, there is a need to prevent the information or items related to it (such as date/time it was created, sent and received) from being tampered with by the sender and/or the recipient. (S. R. Subramanya and Byung K. Yi, 2006)

A digital signature is a checksum which depends on the time period during which it was produced (Denning, 1984) . It is computed using a set of rules and a set of parameters such that the identity of the signatory and integrity of the data can be verified (Biometrics: the Future of Identification, 2000).

II. PKI Architecture Based Digital Signature

The notion of digital signatures goes back to the beginning of public-key cryptography. In their landmark paper Whitfield Diffie and Martin Hellman (W.Diffie & M.E.Hellman, 1976) introduced the idea that someone could form a digital signature using public-key cryptography that anyone else could verify but which no one else could generate. After it RSA (Rivest, Shamir, & Adleman, 1978) has become the most proven and most popular, and achieved the widest adoption by standards bodies and in practice. (Burton S. Kaliski, 2001)

PKI is mainly used for secure transactions between companies or governmental agencies. An ecommerce Web site that uses SSL for encryption is a portion of PKI system. Encrypted e-mail is also another transaction that may be a part of a PKI system. Some companies or agencies may want all staff to digitally sign any documents they have created. Because a digital signature is derived from a Digital Certificate and its key,

this is also part of a PKI system. There are so many possible scenarios and solutions it's almost impossible to list them all (Prakash Kuppuswamy, Peer Mohammad Appa and Saeed Q Y Al-Khalidi, 2012). PKI includes the mechanics described in this article as well as an ensemble of software, hardware and processes governed by rules and standards converging to the high level of Trust required and expected by the Industry. (CGI Group Inc., 2004) The RSA public-key cryptosystem and digital signature scheme are widely deployed today and have become essential building blocks for creating the emerging public-key infrastructure (PKI). (Burton S. Kaliski, 2001) In this paper the PKI (Public Key Infrastructure) based digital signature architecture has been discussed which is related with RSA algorithm and secure Hash functions (MD5 &, SHA variants).

Basically, the idea behind digital signatures is the same as handwritten signature which are traditionally used to validate and authenticate paper documents. A major difference between handwritten and digital signature is that a digital signature cannot be constant; it must be a function of the document that is sign. (Hemant Kumar, Ajit Singh, 2012) It is used to authenticate the fact that you promised something that you can't take back later. For electronic documents, a similar mechanism is necessary. Digital signatures, which are nothing but a string of ones and zeroes generated by using a digital signature algorithm, serve the purpose of validation and authentication of electronic documents. Validation refers to the process of certifying the contents of the document, while authentication refers to the process of certifying the sender of the document. (S. R. Subramanya and Byung K. Yi, 2006)

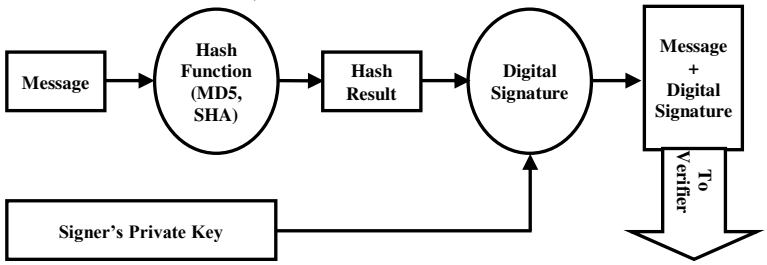
A digital signature is an electronic analogue of a written signature; the digital signature can be used to provide assurance that the claimed signatory signed the information. In addition, a digital signature may be used to detect whether or not the information was modified after it was signed i.e. to detect the integrity of the signed data. (Digital Signature Standard (DSS), June, 2009) In the situation where there is not complete trust between sender and receiver, something more than authentication is needed and the most attractive solution for this problem is the digital signature. Mainly digital signature is use in e-mail, electronic data interchange, software distribution, and other applications that require data integrity assurance and data origin authentication. The wireless protocols, like HiperLAN/2 (Martin Johnsson), and WAP (WAP Forum :), have specified security layers and the digital signature algorithm have been applied for the authentication purposes. (Hemant Kumar, Ajit Singh, 2012)

It must have some salient features such as verify the author and the date and time of signature; authenticate the contents at the time of signature; must be verifiable by third parties, to resolve disputes; it must be a bit pattern that depends on the message of being signed; must use some information unique to sender, to prevent both forgery and denial; must be relatively easy to produce to recognize and verify digital signature but computationally infeasible to forge it and must have legitimate concern.

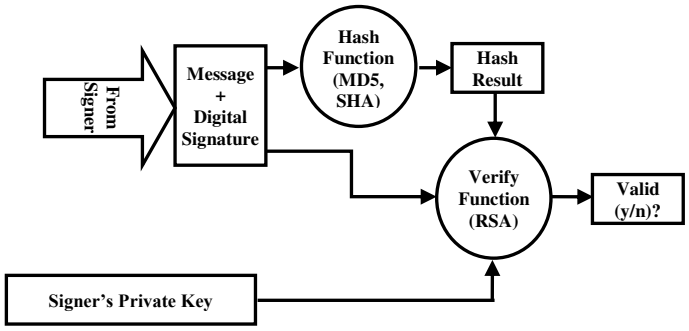
A digital signature can also be used to verify that information has not been altered after it was signed. A digital signature is an electronic signature to be used in all imaginable type of electronic transfer. Digital signature significantly differs from other electronic signatures in term of process and results. These differences make digital signature more serviceable for legal purposes.

Digital signatures are based on mathematical algorithms which includes a signature generation process and a signature verification process. A signatory uses the generation process to generate a digital signature on data; a verifier uses the verification process to verify the authenticity of the signature. These require the signature holder to have a key-pair (one private and one public key) for signing and verification. (Bruce Schneier, 1996; Rivest, Shamir, & Adleman, 1978; Digital Signature Standard (DSS), June, 2009)

Basic idea of digital signatures is each signer has a unique key called private key. There is also other part of key called public key. Whenever singer has to authenticate a document it creates a bit string called signature by applying his private key on the message or some hashed image of message as shown in Fig.1a. User who receives this message then applies his public key on the signature and checks the validity of the bit-string as shown in Fig.1b. If receiver is convinced that document is signed by legitimate signer, it accepts the document. Later if there is some dispute between sender and receiver regarding the validity of document, a third party inspects the signature and using the public key of signer verifies the signature. (Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone, 1997; William Stallings, Nov. 2005; Rania Salah El-Sayed, Moustafa Abd El-Aziem and Mohammad Ali Gomaa, 2008)



(a) Creating a Digital Signature



(b) Verifying a Digital Signature

Figure 1 : Generalized signature Generation and Verification

It has three phases namely (1) Key Generation (2) Signature Generation (3) Signature Verification. The Key Generation phase is the foundation phase for it.

1.1. Key-pair Generation

To generate key-pair for digital signature - RSA algorithm (Rivest, Shamir, & Adleman, 1978), most widely-used public key cryptography algorithm in the world, is used. The idea is that it is relatively easy to multiply prime numbers but much more difficult to factor. Multiplication can be computed in polynomial time where as factoring time can grow exponentially proportional to the size of the numbers. The algorithm is as follows:

a. Select p, q such that p and q both are primes and $p \neq q$.
b. Calculate $n = p \times q$.
c. Calculate $\Phi(n) = (p-1) \times (q-1)$.
d. Select integer e such that $\gcd(\Phi(n),e)=1$ and where $1 < e < \Phi(n)$.
e. Calculate $d = e^{-1} \bmod \Phi(n)$. i.e. $ed = 1 \bmod \Phi(n)$
f. Public key KU = {e, n}.
g. Private key KR = {d, n}.

The Fig.2 shows the data flow diagram of key-pair generation in proposed RSAAPP.

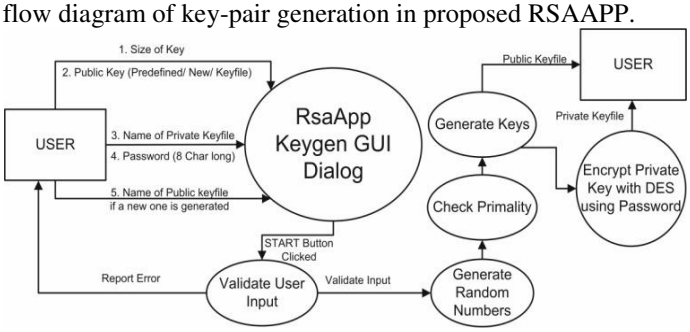


Figure 2: DFD for Key-Pair Generation in RSAAPP

1.2. Signature Generation

a. Given a message m, we apply a suitable hash function H (MD5, SHA1 or SHA2) to obtain the hash result M = H(m).
b. To sign a message *m*, we use M < n to compute Signature (S) = $M^d \pmod n$ where d is the private key of the signer.

The Fig.3 shows the data flow diagram of Signature Generation in proposed RSAAPP.

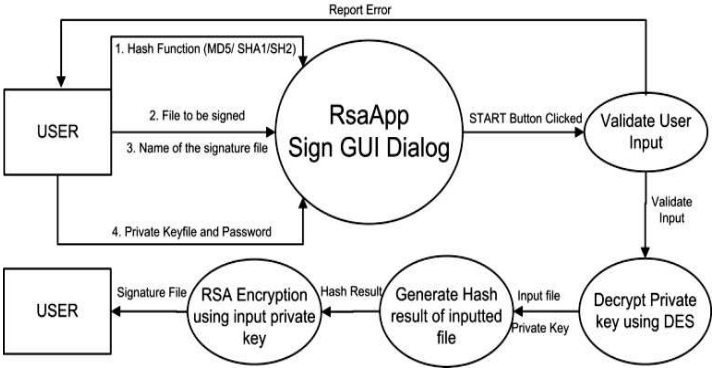


Figure 3: DFD for Signature Generation in RSAAPP

1.3. Signature Verification

a. To verify the message *m*, we use the digital signature S to compute $M = S^e \pmod n$ where *e* is the public key of the signer.
b. Then we obtain M' = H(M) and compare it with M.
c. If both are same then the message is authentic otherwise it is tempered.

The Fig.4 shows the data flow diagram of Signature Generation in proposed RSAAPP.

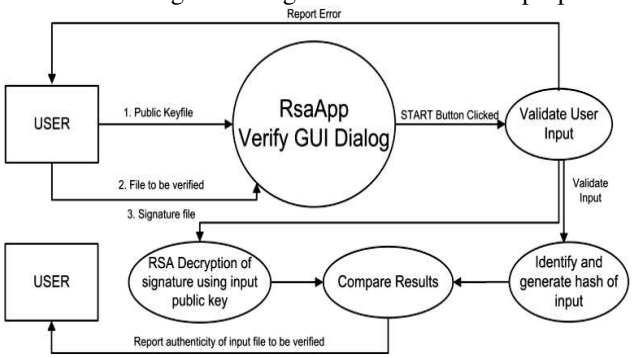


Figure 4: DFD for signature Verification in RSAAPP

III. HASH FUNCTIONS (MD5, SHA1 AND SHA2)

A typical hash function takes a variable length message and produces a fixed length hash. Given the hash, it is impossible to find a message with that hash; in fact one cannot determine any usable information about a message with that hash, not even a single bit. Hash function are used to digest or condense a message down to a fixed size, which then be signed, in a way that makes finding other messages with the same hash extremely difficult (so the signature would not apply easily to other messages). Any cryptographic hash function H has 3 important properties: (1) given message P, it is easy to compute H(P) (2) given H(P), it is effectively impossible to compute P and (3) no one can generate two messages that have the same message digest.

1.4. MD5 Hash Function

The algorithm takes as input a message of arbitrary length and produces as output a 128-bit "fingerprint" or "message digest" of the input. It is conjectured that it is computationally infeasible to produce two messages having the same message digest, or to produce any message having a given prespecified target message digest. The MD5 algorithm is intended for digital signature applications, where a large file must be "compressed" in a secure manner before being encrypted with a private (secret) key under a public-key cryptosystem such as RSA. (Ronald L.Rivest) The summery of the MD5 hash function is:

F(x, y, z) = (x AND y) OR ((NOT x) AND z)
G(x, y, z) = (x AND z) OR (y AND (NOT z))
H(x, y, z) = x XOR y XOR z
I(x, y, z) = y XOR (x OR (NOT z))

1.5. SHA1 Hash Function

NIST, along with NSA, designed the Secure Hash algorithm (SHA1) for use with the digital signature standard. The algorithm published in 1995 in FIPS PUB 180-1 is commonly referred to as *SHA-1*.

When a message of any length < 2^64 bits is input, the SHA-1 produces a 160-bit output called a message digest. The message digest can then, for example, be input to a signature algorithm which generates or verifies the signature for the message. Signing the message digest rather than the message often improves the efficiency of the process because the message digest is usually much smaller in size than the message. The same hash algorithm must be used by the verifier of a digital signature as was used by the creator of the digital signature. Any change to the message in transit will, with very high probability, result in a different message digest, and the signature will fail to verify. (Donald E.Eastlake and Paul E.Jones, Sept. 2001) SHA-1 uses a sequence of logical functions, f_0, f_1,…, f_{79}. Each function ft, where $0 \leq t < 79$, operates on three 32-bit words, x, y, and z, and produces a 32-bit word as output. The function $f_t(x, y, z)$ is defined as follows: (Secure Hash Standard, United States of America, National Institute of Science and Technology, Federal Information Processing Standard (FIPS) 180-1, April 1993)

$$f_t(x, y, z) = \begin{cases} Ch(x, y, z) = (x \text{ AND } y) \text{ OR } ((\text{NOT } x) \text{ AND } z) & (0 <= t <= 19) \\ Parity(x, y, z) = x \text{ XOR } y \text{ XOR } z & (20 <= t <= 39) \\ Maj(x, y, z) = (x \text{ AND } y) \text{ OR } (x \text{ AND } z) \text{ OR } (y \text{ AND } z) & (40 <= t <= 59) \\ Parity(x, y, z) = x \text{ XOR } y \text{ XOR } z & (60 <= t <= 79) \end{cases}$$

1.6. SHA2 Hash Function

In 2005, cryptanalysts found attacks on SHA-1 suggesting that the algorithm might not be secure enough for ongoing use. (Bruce Schneier, 2005) NIST required many applications in federal agencies to move to SHA-2 after 2010 because of the weakness.

SHA-2 is a set of cryptographic hash functions (SHA-224, SHA-256, SHA-384 and SHA-512) designed by the U.S. National Security Agency (NSA) and published in 2001 by the NIST as a U.S. Federal Information Processing Standard (FIPS) PUB 180-2. But in the proposed RSAAPP we use SHA-256.

SHA-256 uses six logical functions, where each function operates on 32-bit words, which are represented as x, y, and z. The result of each function is a new 32-bit word. (Secure Hash Standard, United States of America, National Institute of Science and Technology, Federal Information Processing Standard (FIPS) 180-2, 2002)

$$Ch(x,y,z) = (x \wedge y) \oplus (\neg x \wedge z)$$
$$Maj(x,y,z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$$

$$\sum_0^{\{256\}}(x) = ROTR^2(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x)$$
$$\sum_1^{\{256\}}(x) = ROTR^6(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x)$$
$$\sigma_0^{\{256\}}(x) = ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x)$$
$$\sigma_1^{\{256\}}(x) = ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x)$$

IV. THE DEVELOPED APPLICATION (RSAAPP)

To test and compare the performance characteristics of the RSA and discussed signature algorithms, we developed application RSAAPP using C language with windows API (Charles Petzold, Nov. 1998). The key generation process generates the public and private keys in pairs. If required the keys can be viewed in hex format after generation. The corresponding private keys generated are encrypted using DES after taking an 8-character password as user input. Both keys are made read only. The proposed application can be used encrypt any kind of data i.e. text or binary. It uses SHA2, SHA1 and MD5 hash algorithms for the digital signature. SHA2 is much more secure then SHA1 and MD5. After key generation the Signature Generation module is processed for the file which is to be signed by the signer and sent to the recipient. Further at the receiving end the recipient verifies the signature by execution of Signature Verification process to authenticate that the file has been send by the authentic sender and to validate that the file has been tampered or not. The figures below shows the execution of the developed application i.e. RSAAPP-

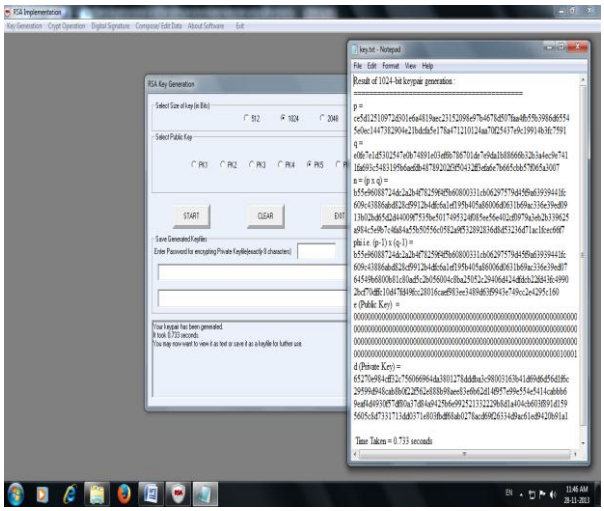
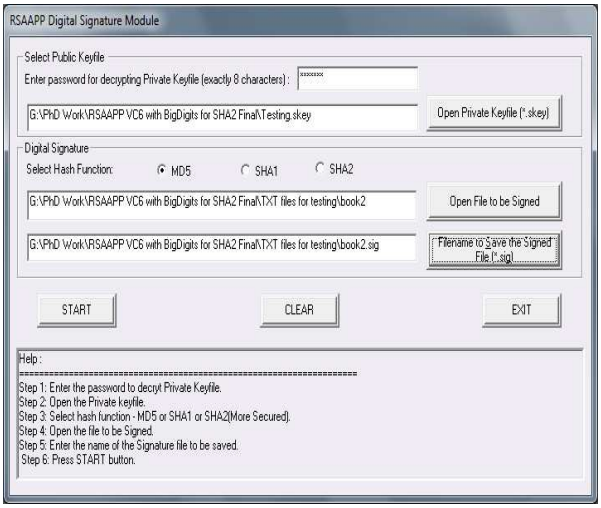
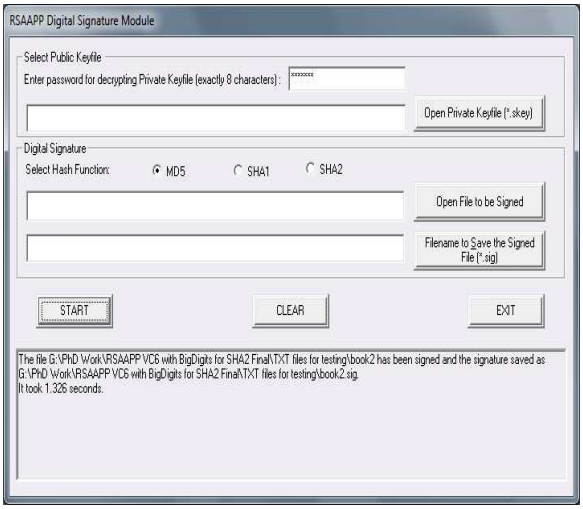


Figure 5: Result of the Key Generation



(a) Signature Generation module of RSAAPP



(b) Completion of Signature Generation of RSAAPP

Figure 6: Signature Generation module of RSAAPP

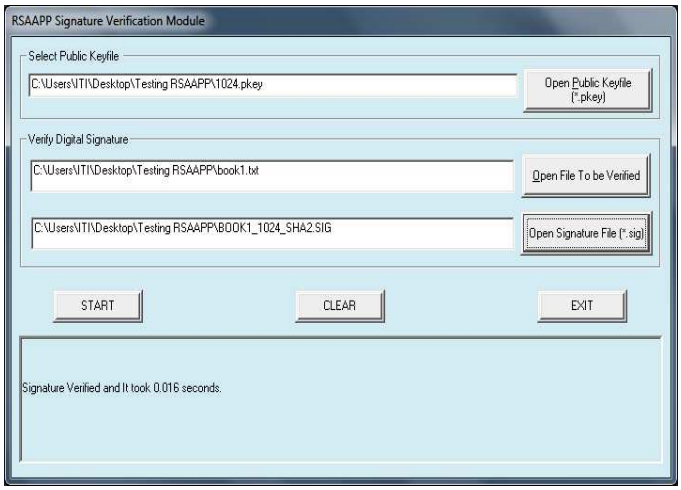


Figure 7: Signature Verification module of RSAAPP

V. EXPERIMENTAL RESULTS

The test results of the developed program RSAAPP for signature generation and signature verification using MD5, SHA1 and SHA2 are tabulated in seconds as shown in Table1. Three files of different size are chosen randomly from Calgary Corpus (Ian Witten, Timothy Bell and John Cleary, 2013) which is a collection of text and binary data files. Tests are preformed on an Intel P4 1.7GHz machine with 1GB of RAM with key size 1024 bits. Time taken in RSA 1024 bit key generation is 2.2 seconds. The experiment results with MD5, SHA1 and SHA2 hash functions are tabulated in seconds as shown in Table1.

Table 1: Experimental Results

Filename with size	MD5		SHA1		SHA2	
	Sign. Gen.	Sign. Verify	Sign. Gen.	Sign. Verify	Sign. Gen.	Sign. Verify
paper4 (12.9 KB)	0.381	0.02	0.381	0.03	0.481	0.02
news (368 KB)	0.430	0.04	0.511	0.05	0.411	0.03
book1 (596 KB)	0.461	0.06	0.571	0.06	0.421	0.06

VI. CONCLUSION

The experimental results shows that the RSA key (1024 bits), signature generation and signature verification with different hash functions – MD5, SHA1 and SHA2 are quite fast in developed RSAAPP. The cost of signature generation can be considered as a factor in the choice of signature system.

The developed RSAAPP system achieves high security for digital signature in addition to decrease processing time and computational overheads. And an intruder cannot pose the message sent since the sender's private key is unknown for him. Accordingly, the sender cannot be impersonated. On the receiver part, the message is verified by using sender's public key and his private key to decrypt the message successfully.

Acknowledgements

The authors are extremely express gratitude to all those people and everyone who support directly or indirectly during the research work that "May god gives them long and prosperous life to spread the light of their intellectuality" and we are forever indebted for their efforts on my behalf, we have learned a great deal from them.

REFERENCES

[1] Bruce Schneier, *Applied Cryptography Protocols, Algorithms, and Source Code in C*, 2nd ed. John Wiley & Sons, 1996.
[2] W.C.Cheng, C.F.Chou and L.Golubchik, "Performance of Batch-based Digital Signatures," in *10th IEEE International Symposium on Modeling* , 2002.
[3] Burton S. Kaliski, "RSA Digital Signatures," *Dr. Dobb's Journal*, May 2001, collabroation.cmc.ec.gc.ca/science/rpn/biblio/ddj/Website/articles/DDJ/2001/0105/0105c/0105c.htm.
[4] R. L. Rivest, A. Shamir, and L. M. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Comm. of the ACM*, vol. 21, no. 2, pp. 120-126, 1978.
[5] S. R. Subramanya and Byung K. Yi, "Digital Signatures," *IEEE Potentials*, vol. 06, pp. 5-8, Apr. 2006.
[6] D. E. Denning, "Digital signature with RSA and other Public-key cryptosystems," *Comm. of the ACM*, vol. 27, no. 4, pp. 388-392, Apr. 1984.
[7] "Biometrics: the Future of Identification," *IEEE Computer*, vol. 33, pp. 46-81, 2000.
[8] W.Diffie and M.E.Hellman, "New Directions in Cryptography," *IEEE Transactions Information Theory*, vol. 22, no. 6, pp. 644-654, Nov. 1976.
[9] Prakash Kuppuswamy, Peer Mohammad Appa and Saeed Q Y Al-Khalidi, "A New Efficient Digital Signature Scheme Algorithm based on Block cipher," *IOSR Journal of Computer Engineering*, vol. 7, no. 1, pp. 47-52, Nov. 2012.
[10] CGI Group Inc. (2004) Public Encryption and Digital Signature: How do they work?. [Online]. http://www.cgi.com/files/white-papers/cgi_whpr_35_pki_e.pdf
[11] Hemant Kumar, Ajit Singh, "An Efficient Implementation of Digital Signature Algorithm with SRNN Public Key Cryptography," *International Journal of Research Review in Engineering Science and Technology*, vol. 1, no. 1, pp. 54-57, Jun. 2012.
[12] "Digital Signature Standard (DSS)," National Institute of Standards and Technology FIBS PUB 186-3, June, 2009.
[13] Martin Johnsson. "HiperLAN/2 – The Broadband Radio Transmission Technology Operating in the 5 GHz Frequency Band", HiperLAN/2 Global Forum,Version 1.0 white-paper, 1999. [Online]. http://www.hiperlan2.com/technology.asp
[14] WAP Forum :. "Wireless Application Protocol Architecture Specification" and "WAP White Paper". www.wapforum.org.
[15] Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1997.
[16] William Stallings, *Cryptography and Network Security Principles and Practices*, 4th ed. Prentice Hall, Nov. 2005.
[17] Rania Salah El-Sayed, Moustafa Abd El-Aziem and Mohammad Ali Gomaa, "An Efficient Signature System using Optimized RSA Algorithm," *International Journal of Computer Science and Network Security*, vol. 8, no. 12, 2008.
[18] Ronald L.Rivest. The MD5 Message-Digest Algorithm, RFC 1321, April 1992,. [Online]. http://www.faqs.org/rfcs/rfc1321.html
[19] Donald E.Eastlake and Paul E.Jones, "US Secure Hash Algorithm 1 (SHA1)," Sept. 2001, RFC 3174.
[20] Secure Hash Standard, United States of America, National Institute of Science and Technology, Federal Information Processing Standard (FIPS) 180-1, April 1993. [Online]. http://www.itl.nist.gov/fipspubs/fip180-1.htm
[21] Bruce Schneier. (2005, Feb.) Schneier on Security: Cryptanalysis of SHA-1. [Online]. https://www.schneier.com/blog/archives/2005/02/cryptanalysis_o.html
[22] Secure Hash Standard, United States of America, National Institute of Science and Technology, Federal Information Processing Standard (FIPS) 180-2, 2002. [Online]. http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf
[23] Charles Petzold, *Programming Windows*, 5th ed. Microsoft Press, Nov. 1998.
[24] Ian Witten, Timothy Bell and John Cleary. (2013) The Data Compression Resource on the Internet. [Online]. http://www.data-compression.info/Corpora/CalgaryCorpus/

Sankalp Prakash is a Research Scholar. He has completed M. Tech. in Computer Science from Rajasthan Vidyapeeth, Udaipur (Rajasthan), India in the year 2006 and pursuing PhD in Computer Science and Engineering from Jagannath University, Jaipur (Rajasthan), India. His major field of study is Cryptography and Computer Networks. Mr. Prakash is member of Computer Society of India (CSI), International Association of Engineers (IAENG) and Institution of Engineers (INDIA).

Dr. Mridula Purohit got post graduation degree in 1996 and doctorate in mathematics in 2000 from university of Rajasthan, Jaipur (Rajasthan), India. Her major field of study is Discrete Mathematic, Differential Equations, Special Functions, Polynomials, Cryptography and Communications. Presently she is a professor in Department of Mathematics at Vivekanand Institute of Technology (East), Jaipur (Rajasthan), India. She got more than 15 years of teaching experience. She has published more than 12 papers in International and National Journals and 04 text books. Recently she is working on research project titled "Applications of wavelet transforms in various fields of Science & Technology" awarded by All India Council for Technical Education, Government of India, New Delhi (India).