Increasing the input data length of RSA cryptosystem by applying a hybrid lossless data compression algorithm

Worapat Siriboonpipattana¹, Chitsutha Soomlek², Pusadee Seresangtakul³
¹,²,³ Department of Computer Science, Faculty of Science, Khon Kaen University, Thailand
s_worpat@kkumail.com¹, chitsutha@kkumail.ac.th², pusadee@kkumail.ac.th³

Abstract. RSA is a well-known and widely used cryptosystem. One major limitation of RSA is its maximum input length depends on the chosen key size and the selected padding. This paper proposed a hybrid data compression algorithm to increase the input data length to be encrypted by RSA. Firstly, a plain text is compressed by the LZW algorithm. The compressed data is padded. After that, the size of the compressed data is further reduced by using continued fraction. Double-compressed data is, then, used as an input of RSA. When using a combination of continued fraction and LZW algorithm, the results showed that, on average, the input data length is increased by 18.91 per cent. In most cases, the processing time is much better than not doing any compression. In the case of decryption, a cipher text is decrypted by RSA first. The padding is removed. Then, the Euclidean algorithm and LZW are used respectively to restore the original data. The hybrid data compression is not only relieved the limitation of RSA but also enhances the security level.

1. Introduction

The Internet plays an important role in supporting long-distance communication and helps in reducing costs in many ways. However, the Internet is considered as an unsecured communication channel. There are many proposed cryptographic algorithms and cryptosystems to enhance security level when transmitting data over an insecure and untrusted network.

RSA is a high-security and well-known cryptosystem using asymmetric keys to protect the confidentiality of data [1], [2]. However, RSA has limitations [3]; for example, the selected prime number should be large enough to be secure and limited input size. This paper focuses on solving the limitation of input size of RSA. The maximum input size of RSA cannot exceed the chosen key size [3]. The input length can be calculated from the chosen key size and the selected padding. Currently, the recommended key size is at least 2048 bits long. As a result, RSA is not suitable for encrypting large-size data; plus, encryption and decryption speed is very slow [4]. Therefore, the researchers proposed to apply a hybrid lossless data compression algorithm to RSA as an alternative method to solve the limited input size of RSA. The plain text is double compressed by using the LZW algorithm [5] and continued fraction [6] before the compressed data is encrypted by using RSA. This approach not only relieves the limitation of RSA but also makes it more difficult for an intruder to get the original data.
This paper is organized as follows. The next section presents related works. Section III explains our research methodology. Section IV describes how experiments were set up and the evaluation results. Conclusions and future work are discussed in the last section.

2. Related works
Data compression can be applied to many research areas in order to achieve bit-rate and storage size reduction. Data compression can be either lossy or lossless [7]. In the case of lossless data compression, the original data can be entirely recovered from the compressed data. In this research, we focus on lossless data compression because we would like to preserve the integrity of data as well as solving the limitation of input data size when using RSA cryptosystem.

Arup Kumar Bhattacharjee et al. presented a comparison study of lossless data compression algorithms for text data [8] to make a comparison among Shannon-Fano coding [9], [10], Huffman coding [11], and LZW algorithm [5]. The experiments covered compression ratio, compression time, and decompression time by using 10 different text files. The results of the experiments showed that LZW algorithm produced the best compression ratio, but its compression process was slower than Huffman coding and Shannon-Fano coding because LZW algorithm needed to create a repetitive word dictionary before starting compression. In the decompression process, LZW was the fastest approach because of the dictionary. Note that the performance of the algorithms depends on the characteristic of the original data.

LZW compression algorithm, named after Abraham Lampel, Jacob Zev and Terry Welch, is a lossless dictionary coder or substitution coder, which means a dynamic dictionary is created regarding the presence of substring chosen from the original file. Then, the substring is matched to the dictionary [5]. If the string is found, then a reference of the dictionary is mentioned in the encoded file. If the string is not found, then a new dictionary entry is added with a new reference [5].

Even though the LZW algorithm seems to be promising to use with RSA, the major problem is the dictionary. For a receiver to obtain the original data back, the corresponding dictionary is required. If the dictionary is big, it will affect the size of the input to RSA and transmitting data.

RSA was developed by Rivest, Shamir, and Adleman in 1978 [2]. RSA uses a key pair: <public key, private key>; where the private key must be a secret and known only to the sender. In contrast, the public key can be disseminated over the network. The public key and private work as a pair. When one key is used for encryption, and another key is used for decryption. The security level of RSA depends on how the prime number used in the algorithm is selected and the key size.

RSA is the most widely used asymmetric key encryption algorithm in the world, but RSA execution time is slower than other asymmetric key encryption algorithms. Therefore, RSA is not usually used to encrypt large size data [4]. As a result, many researchers search for a solution to improve RSA such as maximizing input size [12] and reducing the calculate time to find a prime number [13].

Researchers suggest employing data compression algorithms to overcome the limitation of input data size and protect the confidentiality of data by compressing input data before encrypting with RSA [12], [14]. Data compression reduces the number of bits used for representing data and creates a dictionary for decompression. Therefore, it is more difficult to crack the captured data when a data compression algorithm is applied to a cryptographic algorithm.

Chang Ee Hung and Arif Mandangan [14] presented Compression-RSA, which can accept many sets of integer at a time. Then, all accepted integers are reduced to two integers by using the continued fraction to compress the data and using the Euclidean algorithm to restore the original data. However, this approach can only be used with an integer. If the input data is a string, it must be converted to integer first, for example, converting to an ASCII code. As a result, the input data size becomes larger than the
original because one character of input will be increased to two or more characters. Moreover, Compression-RSA cannot improve the security level of RSA because there is a possibility to find a plain text from a cipher text.

In 2016, Abdallah Karakra and Ahmad Alsadeh [12] presented Augmented RSA (A-RSA). A-RSA employed Huffman Code and Rabin cryptosystem to improved encryption and decryption execution time, space, and security. In their work, plain text is compressed with Huffman code before encryption which produces two output, i.e., a header and a binary file. Then, the header is encrypted with RSA, and a random number is used to blind the cipher text and the binary file, after that, the binary file was encrypted with Rabin cryptosystem. Thus, only the header is encrypted by RSA. The results indicated that the encryption time was reduced by 45%, and decryption time was reduced by 99%. The size of the plain text was reduced by 54% from the original size. However, the results of file size reduction depend on the number of symbols in a plain text.

3. Research methodology

3.1. The process

Figure 1 presents the overview of the process on both the sender side and receiver side. The details of each side are presented as follows.

![Figure 1. Overview of the algorithm](image)

3.1.1. Sender part. The hybrid lossless data compression is started on the sender side before the compressed data is used as an input of RSA. There are four major activities: LZW compression, padding, continued fraction, and data concatenation.

**LZW compression** - in this step, the size of the plain text is reduced by using LZW data compression. Code words and dictionary are the results produced by LZW. The dictionary is created from the input
plain text. It contains only characters presented in the plain text. For example, given 128byte.txt as an input plain text and it is a text file of 128 bytes. The file contains “475317695330560812726820864603 03358856487614440366719455394066808812156604739919498706094428441080040887917240 614057282090647” which is a long integer. The corresponding dictionary will use a set of numbers to represent the characters in 128byte.txt, i.e., 1-4, 2-7, 3-5, 4-3, 5-1, 6-6, 7-9, 8-0, 9-8, 10-2. Note that the index is not included in the dictionary.

Padding - this step is mandatory before continued fraction because there is a possibility that the code words obtained from the previous step do not meet the requirements of continued fraction. If the last code word in the series is less than or equal to one, then it is impossible for the receiver to get the original data. Given $p$ as a padding, where $1 < p < 9$, the output of this step is a series of code words plus the padding. For example, by using 128byte.txt as the input plain text, the series of code words are 1, 2, 3, 4, 5, 2, 6, 7, 13, 4, 8, 3, 6, 8, 9, 5, 10, 2, 10, 6, 10, 25, 24, 6, 1, 23, 20, 4, 4, 3, 9, 9, 22, 1, 9, 16, 5, 1, 48, 8, 4, 6, 2, 5, 1, 3, 13, 56, 8, 52, 9, 24, 25, 10, 5, 22, 23, 11, 4, 7, 7, 55, 1, 7, 45, 61, 8, 56, 48, 31, 48, 5, 24, 8, 8, 49, 41, 2, 73, 28, 1, 61, 47, 21, 28, 9, 10, 79, 61, 11, 2. If $2$ is chosen to be the padding, then the output of this step is 1, 2, 3, 4, 5, 2, 6, 7, 13, 4, 8, 3, 6, 8, 9, 5, 10, 2, 10, 6, 10, 25, 24, 6, 1, 23, 20, 4, 4, 3, 9, 9, 22, 1, 9, 16, 5, 1, 48, 8, 4, 6, 2, 5, 1, 3, 13, 56, 8, 52, 9, 24, 25, 10, 5, 22, 23, 11, 4, 7, 7, 55, 1, 7, 45, 61, 8, 56, 48, 31, 48, 5, 24, 8, 8, 49, 41, 2, 73, 28, 1, 61, 47, 21, 28, 9, 10, 79, 61, 11, 2, 2.

Continued fraction [14] - the series of integers can be reduced to two integers, i.e., dividend and divisor. Given a series of codewords obtained from the previous step $q_1, q_2, \ldots, q_k$; where $q_1$ is the first codeword and $q_k$ is the last codeword in the series. If $q_1, \ldots, q_{k-1} > 1$ and $q_k > 1$, then:

$$x = \frac{M_1}{M_2} \text{ is the result of continued fraction; where}$$

$$\frac{M_1}{M_2} = q_1 + \frac{1}{q_2 + \frac{1}{q_3 + \frac{1}{\ddots + \frac{1}{q_k}}}}$$

Suppose “128byte.txt” is the input plain text, $M_1 = 26000374301743224639554 05418472572052961 7169475568986832770608011866102792985186456208399156$ and $M_2 = 181422652197379639385232949010919857423214070377819779342967683298710038272 41840115976780043$ are the results produced by this step.

Data concatenation - the three pieces of outputs obtained from the previous steps must be formed into a block of data before they are encrypted with RSA. First, the symbols in the dictionary are converted to corresponding ASCII codes. The block of data is $M_1 \oplus \cdots \oplus M_2 \oplus \cdots \oplus$ dictionary.

RSA-encryption - the concatenated data is encrypted by using the RSA encryption algorithm. In case that the concatenated data is larger than the chosen key size, it will be separated into blocks of data regarding the chosen key size. The last block could be smaller than the key size. Then, each block will be encrypted by using RSA. In practice, a few sets of same-length key pairs should be used to prevent certain attacks. In this research, we assume that the transmitting data is not lost during transmission, and all blocks of encrypted data arrive at the destination in the right order.

3.1.2. Receiver part. When all of the encrypted data arrive at the destination, there are five steps that the receiver needs to perform to get the original data: RSA-decryption, data integration and extraction, Euclidean algorithm, padding removal, and LZW decompression.

RSA-decryption - each block of cipher text is decrypted by using RSA.
Data integration and extraction – the deciphered data are placed in their order, after that, M1, M2, and the dictionary are extracted. Since the symbols in the dictionary were turned into ASCII codes, each ASCII code is changed back to the original symbol.

Euclidean algorithm [14] – at this step, the original code words and the padding are restored by using GCD of M1 and M2.

For example, we have GCD(26000374301743224639554054184725720529617, 16947556898632770608011860127929851864556208399156, 18142265219737963938523294901091985742321407037781977934296768329871003827241840115976780043), therefore, the code words and the padding are 1, 2, 3, 4, 5, 2, 6, 7, 13, 4, 8, 3, 6, 8, 9, 5, 10, 2, 10, 6, 10, 25, 24, 6, 1, 23, 20, 4, 4, 3, 9, 9, 22, 1, 9, 16, 5, 1, 48, 8, 4, 6, 6, 2, 5, 7, 1, 3, 13, 56, 8, 52, 9, 24, 25, 10, 5, 22, 23, 11, 4, 7, 7, 55, 1, 7, 45, 61, 8, 56, 48, 31, 48, 5, 24, 8, 8, 49, 41, 2, 73, 28, 1, 61, 47, 21, 28, 9, 10, 79, 61, 11, 2, 2

Padding removal – the padding is removed from the series of data obtained from the previous step.

Now, we have the original code words back, i.e., 1, 2, 3, 4, 5, 2, 6, 7, 13, 4, 8, 3, 6, 8, 9, 5, 10, 2, 10, 6, 10, 25, 24, 6, 1, 23, 20, 4, 4, 3, 9, 9, 22, 1, 9, 16, 5, 1, 48, 8, 4, 6, 6, 2, 5, 7, 1, 3, 13, 56, 8, 52, 9, 24, 25, 10, 5, 22, 23, 11, 4, 7, 7, 55, 1, 7, 45, 61, 8, 56, 48, 31, 48, 5, 24, 8, 8, 49, 41, 2, 73, 28, 1, 61, 47, 21, 28, 9, 10, 79, 61, 11, 2.

LZW Decompression – the original code words and the dictionary are used in LZW decompression algorithm to restore the plain text.

4. Experimental results
The experiments were divided into two parts. First, we compared the hybrid lossless data compression to LZW and continued fraction in term of reduced data size and processing time to confirm our idea. After that, the hybrid lossless data compression was combined with RSA and compared to other approaches in term of processing time. The experimental setup and results are as follows.

4.1. Experiment setup
Python 3.7.2 was employed in this research. All experiments were performed on an ASUS gl552jx laptop with an Intel Core i7-4750hq CPU 2.00 GHz, 16GB RAM, and 250 GB Transcend 750 SSD hard-disk. The operating system was Microsoft Windows 10 Pro 1803. We have prepared a dataset containing 15 files with different sizes, i.e., 128 bytes, 256 bytes, 512 bytes, 1024 bytes, and 2048 bytes, and in three different formats, i.e. .txt, .log, and .html files. The text files (.txt) contain a very-long random integer, and the integer’s length is similar to the file size. The log files (.log) contain strings in English alphabets, integers, and symbols. In case of the HTML files (.html), they contain HTML tags, strings, integers, and symbols.

4.2. A Comparison of data compression algorithms
Table 1 compares the number of output bits obtained from continued fraction, LZW algorithm, hybrid lossless data compression. Note that the outputs of the LZW algorithm already include the corresponding dictionaries. The results show that our hybrid approach outperforms both the continued fraction and LZW algorithm. The number of bits in the outputs, on average, can be reduced by 18.91 per cent. For continued fraction, the output sizes are increased, because each character in the text file must be changed to an ASCII code before compression. As mentioned above, this step is required since the continued fraction only supports integers. In the case of the LZW algorithm, the dictionary is the major reason why the outputs are significantly larger than the inputs. If a dictionary can be separated from their code words, the number of bits used for representation would be reduced.
Table 1. No. of output bits after compression

| Input Data  | Original size (bits) | Output size (Bits) | % Reduce |
|-------------|---------------------|--------------------|----------|
|             |                     | Continued Fraction | LZW      | Hybrid   |
| 128byte.txt | 1024                | 1457               | 1467     | 624      | -42.29   | -43.26   | 39.06    |
| 256byte.txt | 2048                | 2918               | 2744     | 1285     | -42.48   | -33.98   | 37.26    |
| 512byte.txt | 4096                | 5846               | 5036     | 2739     | -42.72   | -22.95   | 33.13    |
| 1024byte.txt| 8192                | 11696              | 9504     | 5784     | -42.77   | -16.02   | 29.39    |
| 2048byte.txt| 16384               | 23405              | 16902    | 11849    | -42.85   | -3.16    | 27.68    |
| 128byte.log | 1024                | 1669               | 1920     | 830      | -62.99   | -87.50   | 18.95    |
| 256byte.log | 2048                | 3211               | 3216     | 1748     | -56.79   | -57.03   | 14.65    |
| 512byte.log | 4096                | 6303               | 5944     | 3646     | -53.88   | -45.12   | 10.99    |
| 1024byte.log| 8192                | 12474              | 10940    | 7384     | -52.27   | -33.54   | 9.86     |
| 2048byte.log| 16384               | 24837              | 19693    | 14296    | -51.59   | -20.20   | 12.74    |
| 128byte.html| 1024                | 1504               | 2256     | 1000     | -46.88   | -120.31  | 2.34     |
| 256byte.html| 2048                | 3000               | 4026     | 1927     | -46.48   | -96.58   | 5.91     |
| 512byte.html| 4096                | 5904               | 6936     | 3854     | -44.14   | -69.34   | 5.91     |
| 1024byte.html|8192                | 11302              | 11270    | 6923     | -37.96   | -37.57   | 15.49    |
| 2048byte.html|16384               | 22762              | 19508    | 13049    | -38.93   | -19.07   | 20.36    |

4.3. Evaluation results

The results presented in Table 1 confirm that the hybrid lossless data compression has high potential in working with RSA to relieve the input limitation problem. The only problem left is the processing time. As mentioned earlier, that RSA’s processing time is slower than some algorithms in the same area. We need to ensure that the hybrid lossless data compression would not add too much processing time to RSA. In this experiment, we combined the hybrid lossless data compression with RSA, then, evaluate its performance in term of processing time. Note that the same pair of a public key and private key were used in all evaluations. The evaluation results are also compared to the original RSA on both sender and receiver sides.

Table 2 and 3 present the execution time on the sender and receiver sides, respectively. It turns out that our approach can reduce the processing time on both sender and receiver sides in most cases. On the sender side, the average of reduced execution time is 39.24 per cent. For the receiver side, the average of reduced execution time is 44.04 per cent. One of the reason is the input size to RSA is smaller than not doing any compression at all. Moreover, the characteristics of the input data affect the processing time. Since continued fraction is a part of the hybrid lossless data compression, the input data to RSA are in integer format; which works well with RSA. Additional steps are not required to pass the results from the hybrid lossless data compression to RSA encryption and from RSA decryption to the decompression process.
Table 2. Execution time on the sender side

| Input Data    | Original RSA (sec.) | hybrid data compression + RSA (sec.) | % Difference |
|---------------|---------------------|-------------------------------------|--------------|
| 128byte.txt   | 0.0019993782043457  | 0.001999616622925                   | -0.01        |
| 256byte.txt   | 0.0019855499267578  | 0.00099927520752                    | 49.64        |
| 512byte.txt   | 0.0019965171813965  | 0.003984212875366                   | -99.56       |
| 1024byte.txt  | 0.0039989948272705  | 0.009005308151245                   | -125.19      |
| 2048byte.txt  | 0.0079743862152100   | 0.011996030807495                   | -50.43       |
| 128byte.log   | 0.0020318031311035   | 0.002011537551880                   | 1.00         |
| 256byte.log   | 0.0090401172637939   | 0.000998258590696                   | 88.96        |
| 512byte.log   | 0.0386438369750976   | 0.004977941513062                   | 87.12        |
| 1024byte.log  | 0.0988972187042236   | 0.007004737854004                   | 92.92        |
| 2048byte.log  | 0.1958477497100830   | 0.019068002700806                   | 90.26        |
| 128byte.html  | 0.0110144615173339   | 0.00099450683594                    | 90.93        |
| 256byte.html  | 0.0159664154052734   | 0.001998901367188                   | 87.48        |
| 512byte.html  | 0.0410509109497070   | 0.002996683120728                   | 92.70        |
| 1024byte.html | 0.0941612720489502   | 0.008998632431030                   | 90.44        |
| 2048byte.html | 0.1838891506195060   | 0.013974666595459                   | 92.40        |

Table 3. Execution time on the receiver side

| Input Data    | Original RSA (sec.) | hybrid data compression + RSA (sec.) | % Difference |
|---------------|---------------------|-------------------------------------|--------------|
| 128byte.txt   | 0.01399374008179    | 0.0139913558959960                   | 0.02         |
| 256byte.txt   | 0.02798271179199    | 0.0309667587280273                   | -10.66       |
| 512byte.txt   | 0.04097938537598    | 0.0559813976287841                   | -36.61       |
| 1024byte.txt  | 0.06995844841003    | 0.127915620838330                    | -82.85       |
| 2048byte.txt  | 0.14192438125610    | 0.2458415031433100                   | -73.22       |
| 128byte.log   | 0.12289595603943    | 0.0279726982116699                   | 77.24        |
| 256byte.log   | 0.24181556701660    | 0.0439610481262207                   | 81.82        |
| 512byte.log   | 0.65596199035645    | 0.0850193500518798                   | 87.04        |
| 1024byte.log  | 1.50427746772766    | 0.1574318408966060                   | 89.53        |
| 2048byte.log  | 3.03911161422729    | 0.3007440567016600                   | 90.10        |
| 128byte.html  | 0.20388460159302    | 0.0279746055603027                   | 86.28        |
| 256byte.html  | 0.3148562923584     | 0.0439746379852294                   | 86.03        |
| 512byte.html  | 0.61056756973267    | 0.0869536399841308                   | 85.76        |
| 1024byte.html | 1.43404412269592    | 0.1439135074615470                   | 89.96        |
| 2048byte.html | 2.77042746543884    | 0.2748720645904540                   | 90.08        |
5. Conclusions and future work
In this research, we proposed a hybrid lossless data compression which can be combined with RSA to relieve the limited input size of RSA. The input data is double compressed by using the LZW data compression algorithm together with a continued fraction. To restore the original data, the Euclidean algorithm and LZW data decompression algorithm is employed.

The experimental results show that the hybrid lossless data compression could reduce the input data size by 18.9 per cent. Its execution time when creating a cipher text is faster than the original RSA by 39.24 per cent. The execution time for restoring the plain text is faster than the original RSA by 44.04 per cent. However, we have not done an experiment on text files containing different types of Unicode. This is left for future work.

Moreover, if we could reduce the size of the dictionary enclosed in the cipher text, there is a high possibility that the input size to RSA will be further reduced. This could be an open research problem in the community.

Acknowledgment
This work was supported by Computer and Information Science Interdisciplinary Research Grant from Department of Computer Science, Faculty of Science, Khon Kaen University, Thailand.

References
[1] F Piper and S Murphy 2013 Understanding cryptography: p 173
[2] R. L. Rivest A. Shamir and L. Adleman 1978 A Method for Obtaining Digital Signatures and Public Key Cryptosystems (ACM vol 21) pp 120-126
[3] J. Jonsson and B. Kaliski (2003 February) Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1 Retrieved from https://tools.ietf.org/rfc/rfc3447.pdf
[4] NaQi Wei Wei Jing Zhang Wei Jinwei Li Peiyi Shen, Xiaoyan Yin, Xiangrong Xiao and Jie Hu, 2013 Analysis and Research of the RSA Algorithm (Information Technology Journal) pp 1818-1824.
[5] A Benjamin and J Quinn 2015 Continued Fractions Proofs that Really Count pp. 49-62
[6] H N Dheemanth 2014 Am J Eng Res. vol 3 no 2 LZW Data Compression pp 22-26
[7] Pu, I.M. 2006 Fundamental Data Compression (Elsevier)
[8] C E Hung and A Mandangan 2013 Proc. AIP Conf. vol 1522 Compression RSA: New approach of encryption and decryption method pp. 50-54
[9] Shannon C. E. (1948 July) Bell System Technical Journal 27 A Mathematical Theory of Communication pp 379-423.
[10] Fano and R M 1949 The transmission of information. Technical Report No 65 (Cambridge (Mass.) USA Research Laboratory of Electronics at MIT).
[11] D A Huffman associate IRE Minimum-Redundancy Codes vol. 27, 1951.
[12] A Karakra and A Alsadah 2016 Proc. 2016 SAI Comput Conf. SAI 2016 A RSA: Augmented RSA pp 1016-1023
[13] Y Wu and X. Wu 2018 Implementation of efficient method of RSA key-pair generation algorithm pp. 72-73
[14] S Apodaca Continued Fractions, Euclid's Algorithm, and Euclidean domains MATH 414
[15] Stefano Zacchiroli The Debsources Dataset Zenodo. https://zenodo.org/record/16106
[16] A Warntjen United National General Assembly Voting, 1946-2013 https://dataverse.harvard.edu/da tase t.html?persistentId doi:10.7910 DV N/7UBUUI