A Novel Method to Implement Book Cipher

Changda Wang
School of Computer Science and Telecommunication Engineering, Jiangsu University, Zhenjiang, China
Email: changda@ujs.edu.cn

Shiguang Ju
School of Computer Science and Telecommunication Engineering, Jiangsu University, Zhenjiang, China
Email: jushig@ujs.edu.cn

Abstract—**Armed with super computational ability, the most efficient attack on symmetric-key systems is an exhaustive key search. A novel encryption method with infinite key space is presented by modifying traditional book cipher. Infinite key space means the unbounded entropy of the key space, which can frustrate any exhaustive key search. Moreover, this book cipher is immune from frequency analysis. Experimental results show that both encryption and decryption have very high rates of data throughput while compared with DES. High efficiency makes it suitable for some computing power limited environments.**

Index Terms—**symmetric-key encryption, infinite key space, book cipher**

I. Introduction

Cryptographic techniques are typically divided into two generic types: symmetric-key and public-key. They have a number of complementary advantages. Current cryptographic systems exploit the strengths of each. Public-key encryption techniques maybe used to establish a key for a symmetric-key system being used by communicating entities *A* and *B*. In this scenario *A* and *B* can take advantage of the long term nature of the public/private keys of the public-key scheme and the performance efficiencies of the symmetric-key scheme [1].

For secure algorithms, armed with super computational ability, the most efficient attack on symmetric-key systems is an exhaustive key search [1]. The key space must then be large enough to make an exhaustive search completely infeasible. If an adversary is assumed to have unlimited computational resources, most of cryptographic primitives are not secure at all. Actually, by exhaustive key search, the American programmer Rocke Verser cooperated with thousands volunteers from Internet used 96 days to have DES broken in 1997. Two years later, the process was shortened dramatically to less than 24 hours [2].

Shannon proved that a necessary condition for a symmetric-key encryption scheme to be unconditionally secure is the uncertainty of the secret key must be at least as great as the uncertainty of the plaintext [3]. If the key string is randomly chosen and never used again, it’s called a one-time pad. One-time pad gives the unbounded entropy of the key space, i.e. infinite key space. The one-time pad is unconditionally secure regardless of the statistical distribution of the plaintext [1]. An obvious drawback of the one-time pad is that the key should be as long as the plaintext, which increases the difficulty of key distribution and key management. This motivates the design of stream ciphers where the key stream is pseudo randomly generated from a smaller secret key, with the intent that the key stream appears random to a computationally bounded adversary. No doubt, they do not offer unconditional security.

We present a novel method to implement book cipher in this paper. Although the key space is infinite, the length of the key is finite and short. That is to say, this encryption method can provide one-time pad without increases the difficulty of key distribution and key management. The remainder of this paper is organized as follows. Section 2 briefly introduces the foundation of our symmetric-key encryption method. Section 3 presents encryption and decryption algorithms. Section 4 describes a case study and its experimental results. Section 5 discusses the advantages of this encryption method. Section 6 we draw some conclusions and points to future works.

II. Foundation

Book cipher is a cipher in which the key is the identity of a book or other piece of text. Traditionally book ciphers work by replacing words in the plaintext of a message with the location of words from a book. It is generally essential that both correspondents not only have the same book, but also have the same edition. If a word appears in the plaintext but not in the book, it cannot be encoded. An alternative approach which gets around this problem is to replace individual letters rather than words. This approach has a long history. It was used by George Scovell for the Duke of Wellington's army in some campaigns of the Peninsular War (A.D. 1807-1814). Even in Cold War, we can saw it served as a secure communication method. For instance, in 1970s, Taiwan broadcast book cipher codes via radio to instruct some of their espionages in mainland China.

If used carefully, book cipher is very strong because it acts as a homophonic cipher with an extremely large number of equivalents[4]. However, this is at the cost of large cipher text expansion. In the early electronic era, book cipher faded out due to three main reasons: (1) both encryption and decryption parts need to digitalize the codebooks and organize them in a form of database, which were extremely time consuming work three decades ago; (2) not as today broad bandwidth networks available almost everywhere, cipher text expansion made it relatively difficult to transmit at that time; (3) if adversary knew the language of the plaintext, e.g. English or Chinese, which will be dramatically shrink the codebooks choice.

The contents and types of the computer files are various. If we observe them from storage layer as we do in operating system, they are all identical, i.e. a sets of some ordered bytes, fig.1. That is to say, computer uses 256 different symbols, i.e. bytes, from '00000000' to '11111111', to compose files, just like English use 26 letters and punctuation to write articles. From this point of view, any computer file can be used as a codebook to carry out book cipher for other computer files.

Clearly, this kind of encryption method will inherit most of advantages of book cipher. Moreover, it introduced some merits that traditional book cipher does not have.

III. ENCRYPTION AND DECRYPTION ALGORITHM

This is an improved book cipher method. The key is a set of public computer files, i.e. key : ={File_1, File_2 ,…, File_N}, where File_i denotes the ith file name or ID, the format of File_i can be arbitrary, $i \geq 1$, $i \in \mathbb{N}$, $\mathbb{N}$ denotes the set of nature numbers.

A. Encryption

(1) Choose a nature number $m \in N$, $m \geq 1$.

(2) Read the plaintext file by binary model and break it up into several pieces of bits with the length *m*. If the length of the last piece is less than *m*, then pad '0' at the tail to make it has the length of *m*. After this steps, the plaintext file is denoted by a set $\{P_0, P_1, \ldots, P_L\}$, where P_i represents the ith piece of bits within the plaintext file, $L \in \mathbb{N}$, $L \geq 1$.

(3) For each *file* belongs to the key, i.e. *file* $\in$ {File_1, File_2, …, File_N}, uses the binary model to read it and break it up into several pieces of bits with the same length *m*. If the length of the last piece is less than *m*, then pad '0' at the tail to make it has the length of *m*. To avoid confusion, a piece of bits is called a *block* here. Thereafter, record all of the *blocks*' addresses. Each *block*'s address is denoted as "*File_id + addr*", where *File_id* denotes the ID of *file* and *addr* denotes the piece's address in the *file.*

(4) For each P belongs to $\{P_0, P_1, \ldots, P_L\}$, substitutes P with the address of a randomly chosen *block* which has the same value as the P.

(5) After all the pieces in $\{P_0, P_1, \ldots, P_L\}$ were replaced, their addresses, i.e. $\{address_0, adress_1, \ldots, addres_L\}$ formed the ciphertext.

B. Decryption

(1) According to the key, i.e. {File_1, File_2 ,…, File_N}, collects these public available computer files.

(2) Read the addresses one by one in ciphertext, i.e. $\{address_0, adress_1, \ldots, addres_L\}$.

(3) For each $address_i \in \{address_0, adress_1, \ldots, addres_M\}$, $address_i$ is consisted of "*File_id + addr*". Locates the file among {File_1, File_2, …, File_N} by *File_id*, and then open it by binary model to read the *block*'s value according to *addr* indication.

(4) Replace each address in ciphertext with corresponding *block*'s value will obtain the set $\{P_0, P_1, \ldots, P_L\}$, reassemble these pieces can recover the plaintext.

IV. CASE STUDIES

For easier to implementation, here we choose *m*=8, i.e. each piece of bits is a byte. Since every computer file can break up into a set of bytes, the padding bits can be left out. Encryption and decryption algorithms are instantiations as follows.

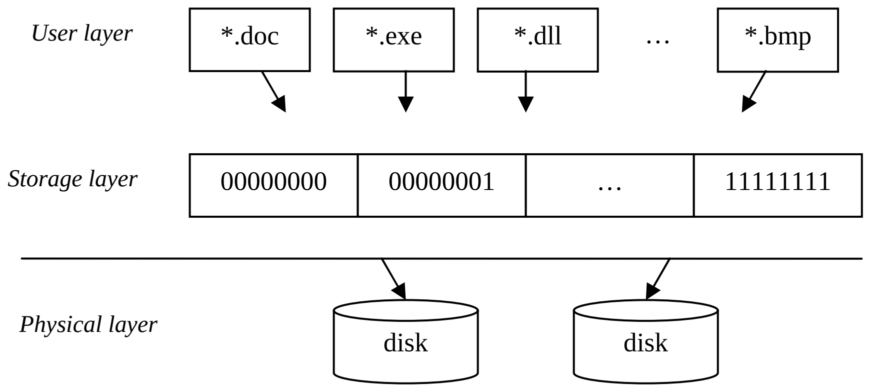

Fig.1. File structure

A. Encryption

(1) For each *file* belongs to the key, i.e. *file*∈{File_1, File_2, …, File_N}, uses the binary format to read it and record each byte's address in the *file*. Every byte's address is denoted as "*File_id* + *addr*", where *File_id* denotes the ID of *file* and *addr* denotes the byte's logical address offset in the *file*.

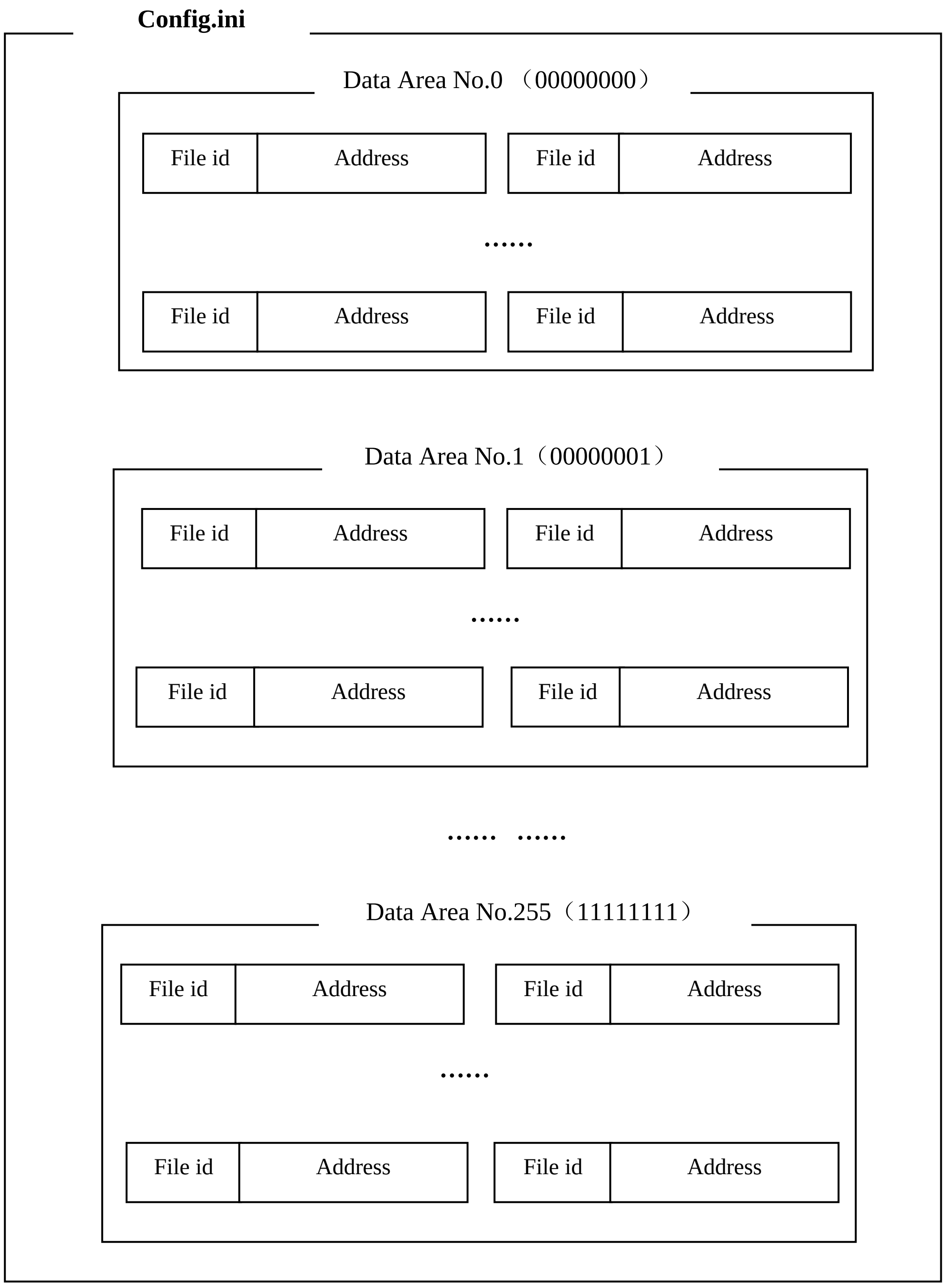

Fig.2 Structure of Config.ini

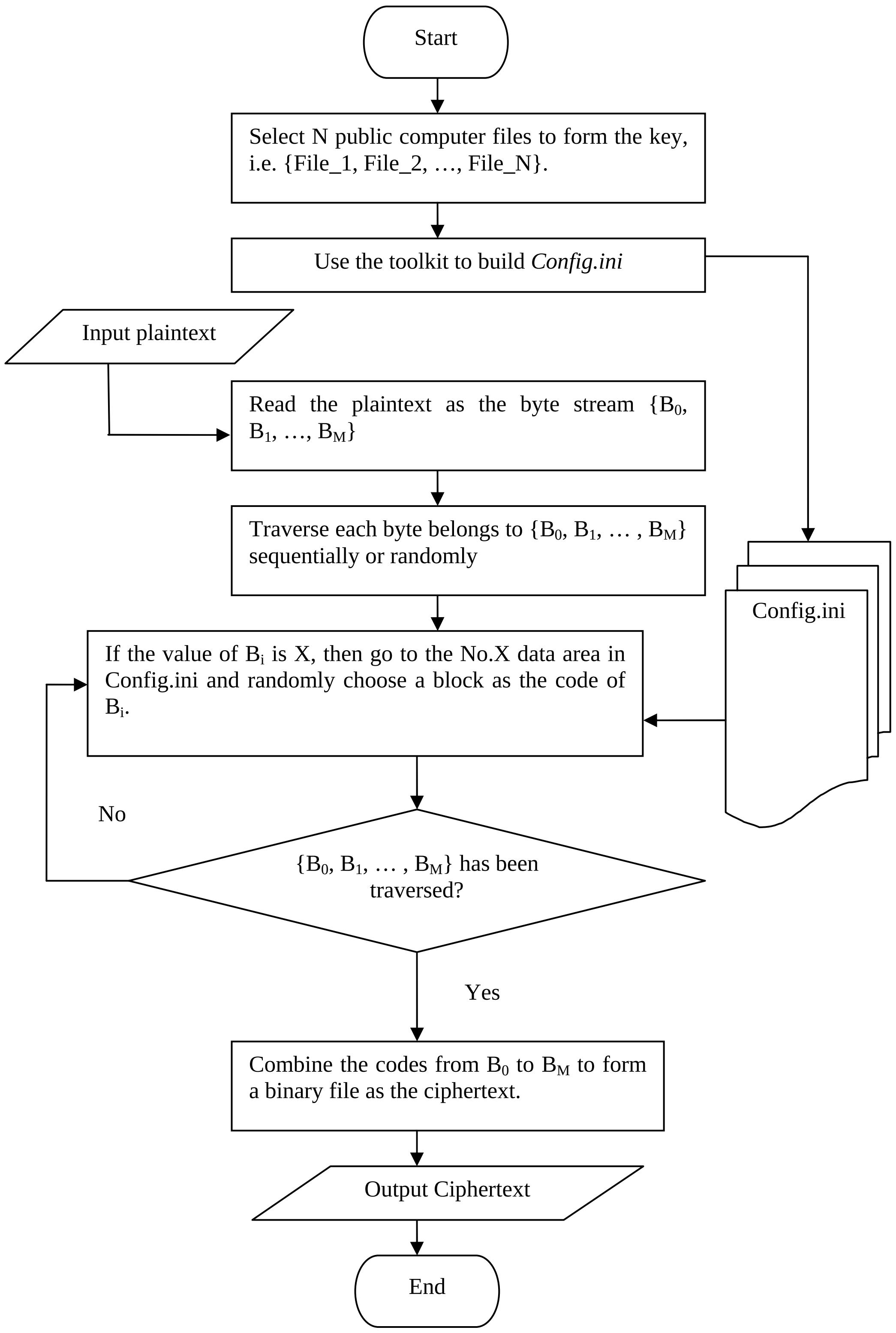
Start
Select N public computer files to form the key, i.e. {File_1, File_2, …, File_N}.
Use the toolkit to build *Config.ini*
Input plaintext
Read the plaintext as the byte stream {B_0, B_1, …, B_M}
Traverse each byte belongs to {B_0, B_1, … , B_M} sequentially or randomly
Config.ini
If the value of B_i is X, then go to the No.X data area in Config.ini and randomly choose a block as the code of B_i.
No
{B_0, B_1, … , B_M} has been traversed?
Yes
Combine the codes from B_0 to B_M to form a binary file as the ciphertext.
Output Ciphertext
End

Fig.3 The flow chart of encryption program

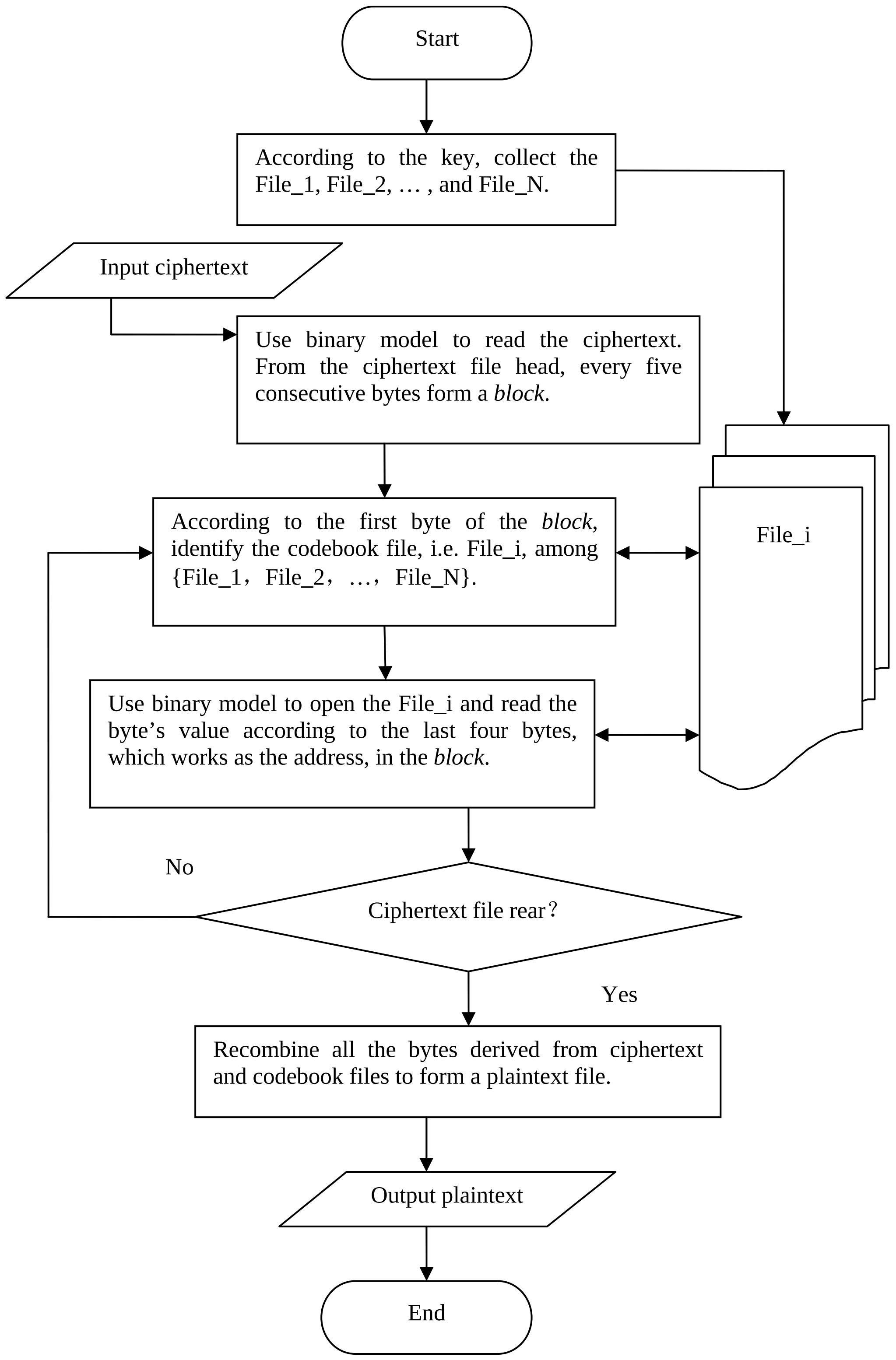
Start
According to the key, collect the File_1, File_2, … , and File_N.
Input ciphertext
Use binary model to read the ciphertext. From the ciphertext file head, every five consecutive bytes form a *block*.
According to the first byte of the *block*, identify the codebook file, i.e. File_i, among {File_1，File_2，…，File_N}.
File_i
Use binary model to open the File_i and read the byte's value according to the last four bytes, which works as the address, in the *block*.
No
Ciphertext file rear？
Yes
Recombine all the bytes derived from ciphertext and codebook files to form a plaintext file.
Output plaintext
End

Fig.4 The flow chart of decryption program

(2) Create a new file named *config.ini* which includes 256 different data areas. These data areas are marked with serial numbers from 0 to 255. They are used to store the bytes' addresses derived from the key. According to different byte values, the addresses stored in different data areas, e.g. the address of '00000000' is stored in No.0 data area and the address of '11111111' is stored in No.255 data area, fig.2, viz. the byte's value equal to its date area serial number.

(3) Open the plaintext file by binary model, and then treat it as a set of ordered bytes, i.e. {B_0, B_1 ,…, B_M}.

(4) For each byte B_i belongs to {B_0, B_1 ,…, B_M}, locates the data area in config.ini by the B_i's value. Thereafter, randomly chooses an address within such data area to replace the B_i's position.

(5) After all the bytes in {B_0, B_1 ,…, B_M} were replaced, their addresses, i.e. {$address_0$, $adress_1$, …, $addres_M$} formed the ciphertext.

The usage of the *config.ini* is that provides an efficient way to substitute each byte in {B_0, B_1 ,…, B_M} with many different addresses choice. Even though the total size of the {File_1, File_2 ,…, File_N} is not large, e.g. 2 MB, each data area has more than eight thousands addresses averagely. Our experimental results, which will be discussed later, corroborates that directly look up every byte address of plaintext file among {File_1, File_2 ,…, File_N} is inefficient and almost intolerable for time issues.

B. Decryption

(1) According to the key, i.e. {File_1, File_2 ,…, File_N}, collects these public computer files in a directory of local computer.

(2) Read the addresses one by one in ciphertext, i.e. {$address_0$, $adress_1$, …, $addres_M$}.

(3) For each $address_i \in$ {$address_0$, $adress_1$,…, $addres_M$}, we known that $address_i$ is consisted of "*File_id + addr*". Locates the file among {File_1, File_2, …, File_N} by *File_id*, and then open it by binary model to read the byte's value according to *addr* indication.

(4) Replace each address in ciphertext with corresponding byte's value will obtain the set {B_0, B_1, …, B_M}, reassemble these bytes can recover the plaintext.

C. Experimental Results

We developed an independent toolkit by Visual Basic to build the file *config.ini*. For easier to implementation, every byte's address within different data areas of *config.ini* were denoted as five consecutive bytes, i.e. $B_0B_1B_2B_3B_4$, where B_0 denotes the *file* belonging to {File_1, File_2, …, File_N} and $B_1B_2B_3B_4$ denotes the byte's 32 bits logical address offset in the *file*. $B_0B_1B_2B_3B_4$ is called a *block* of data area in the *config.ini*. Since each byte has 8 bits, that is to say, we can use 256 different files at most as codebooks and each of them can be as large as 2^{32} bytes.

The flow charts of encryption and decryption programs are fig.3 and fig.4 respectively.

```
C:\WINDOWS\system32\user32.dll
C:\WINDOWS\system32\kernel32.dll
C:\WINDOWS\system32\mpr.dll
C:\WIDOWS\system32\gdi32.dll
C:\WINDOWS\system32\shell.dll
C:\WINDOWS\system32\winspool.drv
```

Fig.5. Key.txt

Both encryption and decryption programs were developed by Visual Basic. The key, i.e. {File_1, File_2, …, File_N}, is stored as a text file includes both file names and paths, e.g. fig.5.

Here, we selected six common files that almost available in every computer installed the Windows XP operating system. A 364KB pdf file downloaded from internet was chosen as the plaintext. On a Dell desktop computer in our lab, CPU 1.69GHZ dual core, RAM 512MB, we tested the encryption and decryption programs. Moreover, we tested DES and Twofish [5] algorithms with the same plaintext file on the same computer, table.1.

TABLE I.
BENCHMARK OF ENCRYPTION AND DECRYPTION

	Encryption (ms)	Decryption (ms)	CPU Occupation Rate (%)
Our algorithm	204	141	7
DES	7,282	7,266	51
Twofish	4,063	3,892	50

By our algorithm, encryption and decryption were cost 204 milliseconds and 141 milliseconds respectively. Use DES, 64 bits length of key, encryption and decryption were cost 7,282 milliseconds and 7,266 milliseconds respectively. Furthermore, use our algorithm the CPU occupation rate was 7% only, whereas use DES the CPU occupation rate was 51%.

Twofish was one of the five finalists of the AES contest. Since Twofish is unpatented, and the source code is uncopyrighted and license-free, we tested Twofish under the same environment. Its performance was better than DES.

The total size of six files in the key, fig.5, is 2.12MB. That is to say, each data area of the config.ini has 8,642 blocks averagely. For encryption, each byte from plaintext randomly chose a block from more than 8,000 blocks as its code. For decryption, just get the byte's value according to its code indication. Randomly chose a block from large candidates consumes times, this is the reason that decryption was faster than encryption by our algorithm.

V. ADVANTAGES

This scheme is said to be symmetric-key encryption method since both encryption and decryption primitives associate the same key, i.e. {File_1, File_2, …, File_N}. So it holds the common advantages of symmetric-key encryption as follows [1]: (1) It can be designed to have high rates of data throughput; (2) Keys are relatively short for the same key space; (3) It can be employed as primitives to construct various cryptographic mechanisms; (4) It can be composed to produce stronger ciphers; (5) It's perceived to have an extensive history.

Moreover, this method has further advantages.

A. Infinite key space

The size of the key space is the number of encryption/decryption key pairs that are available in the cipher system. As to the symmetric-key cipher system presented in this paper, the encryption key is the same as the decryption key. Shannon proved that a necessary, but usually not sufficient, condition for a symmetric-key encryption scheme to be secure is that the key space must be large enough to preclude exhaustive search. Large enough means the uncertainty of the secret key must be at least as great as the uncertainty of the plaintext. The one-time pad is an example of that by gives the unbounded entropy of the key space.

If the entropy of the key space is finite, theoretically an adversary could break it by exhaustive key search. High performance computing, such as grid computing and parallel computing have turned the theoretical threat into reality. DES was broken by strong computing power decade ago.

Generally, the key's length is the entropy of the key space. An obvious drawback of that is infinite key space means the key's length is infinite, which makes key distribution and key management almost impossible.

By our method, the entropy of the key space is not depending on the key's length. Although the key's length is short and finite, the entropy of the key space can be infinite. As mentioned above, the key was defined as {File_1, File_2, … , File_N}, where every File_i ∈ {File_1, File_2 ,…, File_N} is a file name or ID of any public computer file with arbitrary format. Not like traditional book cipher, if you know what kind of language is employed, e.g. English, the codebooks would be decreased dramatically since Chinese characters can't be used to substitute for English letters.

Mathematical infinite denotes an unbounded limit, which means something grows without bound [6]. Since we produce new computer files ceaselessly, the number of computer files is infinite according to this definition. Just think about how many digital pictures were taken everyday. So this encryption method can endure any kind of exhaustive key search.

B. Immune from frequency analysis

Shannon proved that a necessary condition for a symmetric-key encryption scheme to be unconditionally secure is the uncertainty of the secret key must be at least as great as the uncertainty of the plaintext [3]. That is to say, infinite key space only fulfills a necessary condition but sufficient condition to be unconditionally secure. Even with infinite entropy of key space, sometimes frequency analysis can be used to frustrate such encryption algorithms. For example, the one-time pad, which has infinite key space, was proofed unconditionally secure only regardless of the statistical distribution of the plaintext [1].

Frequency analysis is based on the fact that, in any given stretch of written language, certain letters and combinations of letters occur with varying frequencies. Moreover, there is a characteristic distribution of letters that is roughly the same for almost all samples of that language. The homophonic cipher was proposed to frustrate frequency analysis, which involves replacing each letter with a variety of substitutes, the number of potential substitutes being proportional to the frequency of the letter. In our cipher, any byte of plaintext file mapped to more than one ciphertext symbol, i.e. the byte's address among codebook files {File_1, File_2 ,…, File_N}. Since every data areas within config.ini could provide thousands or even more choices, so this symmetric-key encryption method can be treated as the type of homophonic cipher with pretty larger alphabets. Furthermore, in every data areas within the file config.ini, the substitute of each byte of the plaintext file is randomized. This non-determinate characteristic makes our encryption method can endure chosen-ciphertext attack also.

A chosen-ciphertext attack is an attack model for cryptanalysis in which the cryptanalyst chooses a ciphertext and causes it to be decrypted with an unknown key. Generally, non-randomized encryption schemes are vulnerable to chosen-ciphertext attack.

According to Shannon's theory, we knew that the entropy H of a discrete random variable X with possible values $\{x_1, ..., x_n\}$ is:

$$H(X) = -\sum_{i=1}^{n} p(x_i)\log_2 p(x_i)\,.$$

$H(X)$ is mathematical expectation about X, i.e. the average uncertainty about discrete random variable X. Here we have the following claims about $H(X)$[7],

Claim 1: $0 \leqslant H(X) \leqslant \log_2 n$. $H(X)=0$ iff exist an i, $p(x_i)=0$ and for the others $j \neq i$, $p(x_j)=0$; $H(X)=\log_2 n$ iff for all $1 \leqslant i \leqslant n$, $p(x_i)=1/n$.

Claim 2: $H(X,Y)=H(Y)+H(X|Y)=H(X)+H(Y|X)$, where $H(X,Y)$ is the *joint entropy* and $H(X|Y)$, $H(Y|X)$ are *conditional entropy*.

Claim 3: If (P,C,K,E,D) is a encryption system, then $H(K|C)=H(K)+H(P)-H(C)$, where P,C,K,E,D are plaint text space, cipher text space, key space, encryption method and decryption method respectively.

Since every computer files can be chosen as a member in the code book with the equal probability, under the assumption in the previous of this section, i.e. the numbers of computer files, we create everyday, can be infinite, then according to claim 1 we got,

$$H(K) = \log_2 n = +\infty \text{, when } n \rightarrow +\infty\,.$$

The entropy of chosen-ciphertext attack can be described as *H(K|(P,C))*, viz. if we have the correlation between some ciphertexts and plaintexts, how uncertainty the encryption key is.

By claim 2 and 3, we knew that

$$\begin{aligned} H(K|(P,C)) &= H(K)+H(P)-H(P,C) \\ &= H(K)+H(P)-[H(C)+H(P|C)] \\ &= H(K)+H(P)-H(C)-H(P|C) \end{aligned}$$

Since $H(K) \rightarrow +\infty$ ($n \rightarrow +\infty$), moreover, in our encryption method both *P* and *C* only have 256 different letters, so *H(P), H(C), H(P|C)* are limited numbers, that is to say, $H(K|(P,C)) \rightarrow +\infty$ ($n \rightarrow +\infty$), viz. if we select computer files in the code book randomly, even adversary knew the correlation between some ciphertexts and plaintexts, the encryption key cannot be revealed.

VI. Conclusions

The most interesting characteristic of this encryption method is the infinite key space doesn't mean the length of key is infinite. Actually, the length of key is finite and short, which makes key distribution easier than already known one-time pad encryption methods. Infinite key space depends on the number of computer files is infinite. So this encryption method can endure any kind of exhaustive key search. Moreover, any byte of plaintext file randomly mapped to more than one ciphertext symbol, i.e. the byte's address among codebook files. Theoretically, the number of ciphertext symbols can be as large as 2^{40} in our experiment since each block in config.ini has five bytes. This non-determinate characteristic makes our encryption method not only immune from frequency analysis, but also can endure chosen-ciphertext attack.

High efficiencies of encryption and decryption make this method suitable for some computing power limited environment. For example, in some ad hoc networks and sensor networks, processor is considered as more energy-consuming part than signal transmits and receives. According to table I, our method has the least CPU occupation time and rate while compared with some other symmetric encryption methods. Further discussion beyond the scope of this paper, readers are encouraged refer to [8, 9,10] for details.

As most of book ciphers, the safety of this encryption method is also at the cost of ciphertext expansion. For easier to implementation, in our experiment, we use a byte for codebooks and four bytes for addresses. So 256 different codebook files at most can be employed and each of these files can be as large as 2^{32} bytes. They are pretty large computer files. Actually, if we use less codebook files and require the size of these files is relatively smaller, e.g. no more than 2^{16} or 2^{24} bytes, the length of ciphertext will be shrunk a lot. Furthermore, choose different block size will affect the length of ciphertext also. In our experiment, we choose m=8. Bigger block size will decrease the length of ciphertext, but at the same time that will increase the number of data areas in config.ini. We suppose there is a tradeoff between the block size and the numbers of data area to form an optimization cipher system. Our future work is to mitigate the ciphertext expansion of this encryption method.

Acknowledgment

Project (No. 60773049) supported by the Nature Science Foundation of China, (No.BK2010192) supported by the Nature Science Foundation of Jiangsu province, (No. 07KJB520016) supported by the Education Department of Jiangsu province, (No. 07JDG053) supported by Jiangsu University.

References

[1] A. Menezes, P. van Oorschot, S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1996.
[2] D. Feng, Status quo and trend of cryptography, J*ournal of China Institute of Communications*, Vol. 23 No.5, May 2002, pages: 18-26
[3] C.E. Shannon, *Communication theory of secrecy systems*, Bell Syst Tech J, 1949, Vol.28, No.3, pages:656-715.
[4] Maurer Ueli, *Conditionally-perfect secrecy and a provably-secure randomized cipher*, Journal of Cryptology, vol.5, no.1, 1992, pages: 53-66
[5] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, Niels Ferguson, *The Twofish Encryption Algorithm: A 128-Bit Block Cipher*, Wiley, 1999.
[6] Eric Gossett, *Discrete mathematics with proof*, Pearson Education, 2003
[7] Rongxi Que, Yue Sun, *The Principle of Information Security and its Applications*, Tsinghua Press, 2006
[8] Lee Tsung-Han, Marshall Alan, Zhou Bosheng, *Modeling energy consumption in error-prone IEEE 802.11-based wireless ad-hoc networks*, vol. 4267 LNCS, pages:61-73, 2006
[9] Palit Rajesh,t; Naik Kshirasagar, *Modeling the energy cost of applications on portable wireless devices*, Proceedings of the 11th ACM International Conference on Modeling, Analysis, and Simulation of Wireless and Mobile Systemspages, 2008, pages:346-353
[10] Ioannis Caragiannis, Christos Kaklamanis, Panagiotis Kanellopoulos, *Power Consumption Problem in Ad-Hoc Wireless Networks*, WAOA 2003, LNCS 2909, 2004, pages: 252–255.

Changda Wang received the B.S. degree from mathematic department of Soochow University in 1994, and the M.S. degree and the Ph.d degree from school of computer science of Jiangsu University in 1999 and 2006, respectively.

He is an associate professor of computer science, Jiangsu University. From April 2004 to April 2005, he was supported by the scholarship of the Jiangsu province government to conduct his research in school of computer, Carleton University, Ottawa, Canada.

Shiguang Ju received his M.S. degree in 1988, from Nanjing University of Science and Technology (China) and the Ph.d degree in 1996 from National Polytechnic Institute (Mexico). From September 2003 to April 2004, he was a visiting professor at Texas Tech University (U.S.A).

He is now a professor of Jiangsu University. His current research interests include information security and spatial database.