Achieving perfect secrecy in the modern days

Ang Ming Yi
School of Computing
National University of Singapore
13 Computing Dr, 177417
+65 65162727
ming_yi@u.nus.edu

Kaung Htet Aung
School of Computing
National University of Singapore
13 Computing Dr, 177417
+65 65162727
kaung_htet@u.nus.edu

Koh Jun Xiang
School of Computing
National University of Singapore
13 Computing Dr, 177417
+65 65162727
kohjx@u.nus.edu

Pang Yong He
School of Computing
National University of Singapore
13 Computing Dr, 177417
+65 65162727
yonghe@u.nus.edu

Yeo Quan Yang
School of Computing
National University of Singapore
13 Computing Dr, 177417
+65 65162727
quanyang@u.nus.edu

ABSTRACT

In this paper, we describe a proof-of-concept encryption scheme in which secret keys are generated from publicly available data.

Categories and Subject Descriptors

D.4.6 [**Security and Protection**] (K.6.5): Authentication, Cryptographic controls

E.4 [**Coding and Information Theory**]: Nonsecret encoding schemes

General Terms

Algorithms, Performance, Design, Experimentation, Security

Keywords

Perfect Secrecy, Entropy, Digital Media, Encryption scheme

1. INTRODUCTION

Data is mainly being encrypted using symmetric encryption schemes using much less bits than the asymmetric counter part. Despite the difference in bits, the security is considered equivalent. The caveat is that symmetric keys must be agreed upon before hand. In this paper, we aim to design a scheme that only involves an initial key agreement and will allow two parties to communicate as securely as existing schemes without the need to always renew the agreed key.

2. Idea

2.1 Components

The key components in our project are the two parties that require secure communication, the encryption scheme, and the proof-of-concept program that implements our encryption scheme.

2.2 Simplification

Instead of providing fully featured GUI software, we will be providing a simplified proof-of-concept version of the software where the users are required to know the constraints manually when starting an encryption process. Also, although our current encryption scheme generates and uses a 512-bit key, it is scalable to a 4096-bit key version in the future.

2.3 Key concept

The concept of our encryption scheme is to turn publicly available data into a suitable shared session key between two parties through a series of functions. By using publicly available data, such as videos and compressed files, we would be able to use them as a form of one-time pad (OTP) during encryption, and subsequently, decryption. The key idea is to mimic the one-time pad (OTP) encryption technique such that it is not possible to crack the encryption without an exhaustive brute force on every encrypted data transfer session.

2.4 Assumptions

We assume that the following are true:

- The two parties are able to agree on an initial shared secret key before using our encryption scheme.
- The two parties are able to both access and download the same publicly available data on the Internet.

3. Encryption Scheme

3.1 Introduction

In our encryption scheme, we hope to achieve the "necessary and sufficient condition for perfect secrecy" [1]:

- From the 512 bits key, we are able to generate a unique randomized key per session with the key length being as long as the plaintext.
- Nothing can be learn about the plaintext from the ciphertext

In the previous section, we have highlighted the aim of eliminating the need of initial pre-shared secret key renewal. In our encryption scheme, with the use of Exclusive or (XOR) operations and hashing algorithm, the shared secret key is manipulated into a unique session key for each session, eliminating the need to re-establish a new pre-shared secret key for each session.

The underlying design of our encryption scheme is based on the concept of a block cipher in which the encryption would operate on a fixed-length of bits. In our proof-of-concept version, the block size is fixed at 512-bits.

In each block, the generation of the session block key is done with an algorithm performing a series of block chaining operations from the session secret key through blocks of Internet files to compute the final session block key.

Last but not least, the final session block key would be XOR'ed with a block of the plaintext to output a block of ciphertext.

The mode of operation, Electronic Codebook (ECB) has been chosen and implemented to encrypt data larger than the block size.

3.2 Design

In this section, we will explain our encryption scheme in details through design diagrams.

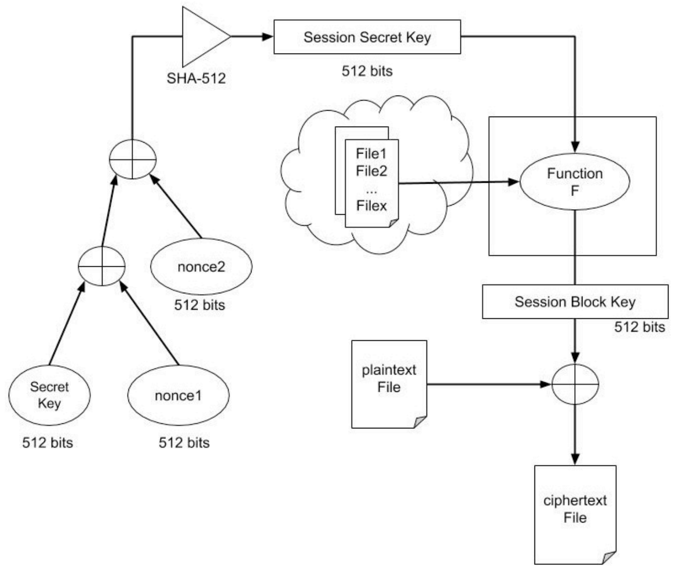


Figure 3-1 Encryption Scheme

The typical flow of our encryption scheme is depicted in the Figure 3-1.

Name	Description
Secret Key	The initial 512-bits pre-shared secret key
nonce1	A randomized 512-bits value for the session
nonce2	A randomized 512-bits value for the session
SHA-512	The hashing algorithm used to produce 512-bits Session Secret Key
Session Secret Key	A 512-bits protected secret key for the session
File1, File2... Filex	The files taken from the internet, e.g youtube video files
Function F	The function responsible in computing the final session block key
Session Block Key	The final session block key for the session
Plaintext File	The file to be encrypted.
Ciphertext File	The encrypted file.

The initial pre-process consists of XOR operations performed on the secret key, nonce1 and nonce2 values. SHA-512 hashing algorithm is an additional step used to ensure that the secret key goes through further transformation.

The purpose of the pre-process is to protect the secret key from any chosen-plaintext attack, known-plaintext attack and chosen-ciphertext attack. With a drastically change in the secret key for every session, it would be hard for the attackers to recover the secret key, thus, allowing the possibility of reusing the pre-shared secret key from the initial key agreement without the fear of exposing the secret key.

Now, with the session secret key ready, it is time to feed the key into the core function of our encryption scheme, function F.

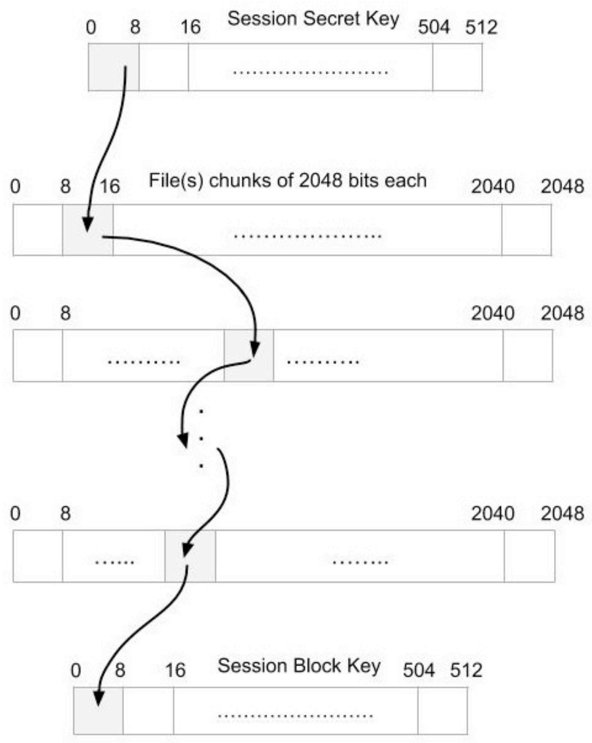


Figure 3-2 Function F

The block chaining operation of the function F is shown in Figure 3-2. The 512-bits session secret key and the 2048-bits file chunks are split into bytes block, with 128 and 256 blocks respectively.

The block chaining operation starting from a block of the session secret key, yielding a block index from the byte. This block index would be "chain" to the next block in the file chunk, which in turn yields another block index from the byte. This block chaining would repeatedly be done until the last file chunk, which would be used to form part of the session block key – the location of this byte would correspond to the location of the relevant block of session secret key for this chaining operation.

In total, the session secret key would go through 64 rounds of block chaining operation to complete a session block key, starting with the first byte of session secret key.

Each Internet file provided is only being used for one chain step, e.g. 1 file provides 1 chain step, 3 files provide 3 chain steps. The file would be split into 2048-bits chunk(s), the number of chunks is dependent on the number of blocks needed for the plaintext file-every 64 bytes of plaintext requires a 2048-bits chunk for encryption.

Therefore, the following conditions would applied to compute the session block key for each block of the plaintext file:

- The session secret key remained unchanged.
- The file(s) chunks used for chaining would be changed.
- The session secret key has to go through another 64 rounds of block chaining operation.

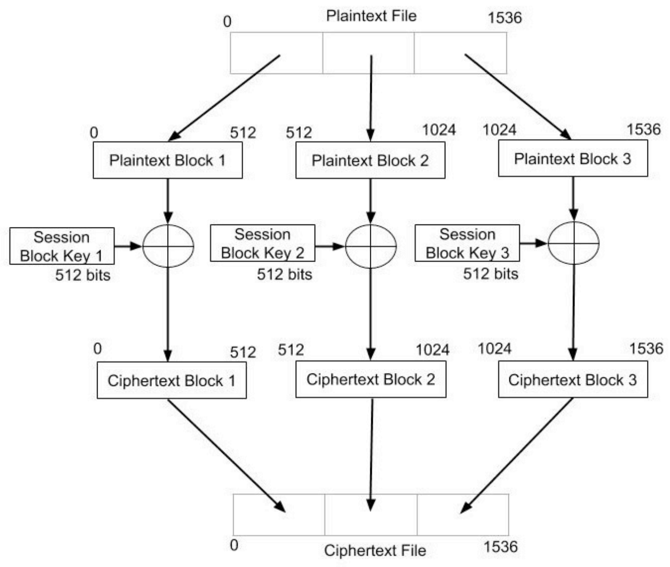


Figure 3-3 ECB

The mode of operation, ECB, for our encryption scheme is illustrated in Figure 3-3. The plaintext file is divided into blocks, and each block is XOR'ed with a session block key to output a cipbertext block. All the ciphertext blocks are then concatenated to get a ciphertext file.

In a normal ECB mode, the given key is used to encrypt all the blocks making the encryption deterministic since any identical plaintext block always get encrypted to the same ciphertext block. For that reason, it is always recommended to use other mode of operations, such as CBC, other than ECB mode [2].

However, in our encryption scheme, a new block key is generated for each block of plaintext file, giving our ECB mode the ability of making the encryption non-deterministic just like CBC mode. Any given plaintext block would not always get encrypted to the same ciphertext block, thus providing message confidentiality.

3.3 Implementation

The implementation of our proof-of-concept is written in C programming language. The actual source code could be found in the Appendix A.

Since we are providing a proof-of-concept on our encryption scheme, we are only focusing on the encryption and decryption process in the implementation.

To deal with the possibility that the multiple bytes of the session secret key could point to the same block index of the file chunk, we implemented collision resolution in our program. Linear probing is used to find the next available block index if block index collision occurred. This would ensure similar block index denoted by byte of the session secret key would not end up in the same location at the end of the block chaining operation.

To use our program, the following has to be prepared beforehand:

- Generate a 512-bits pre-shared key and saved as hexadecimal format in a file
- Generate a 512-bits nonce1 value and saved as hexadecimal format in a file
- Generate a 512- bits nonce2 value and saved as hexadecimal format in a file.
- Download videos or audio files from the Internet. The files must be large enough to split into the number of blocks required to encrypt the plaintext or decrypt the ciphertext.

The command-line program required 5 arguments to be passed to the program in the following orders:

1. A SecretKey File
2. A nonce1 file
3. A nonce2 file
4. A plaintext/ciphertext file
5. Any number of downloaded Internet files

Once executed, the program would run as follows:

1. Generate session secret key
2. Determine the number of blocks needed to encrypt the plaintext or decrypt the ciphertext.
3. Get the required number of file chunks from the files provided.
4. Generate the required amount of session block keys through Function F
5. Encrypt the plaintext or decrypt the ciphertext with the session block keys in ECB mode
6. Output the decrypted ciphertext or encrypted plaintext as file.

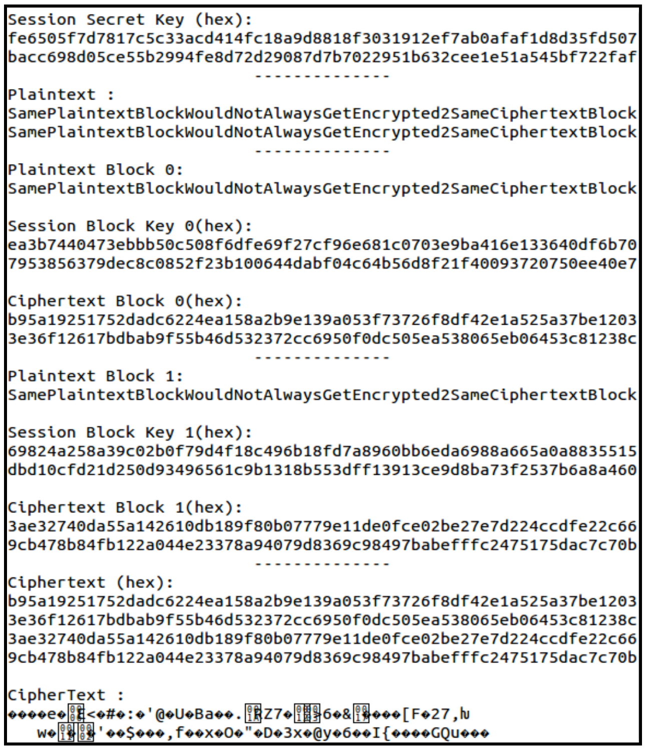
Our program is scalable and can be easily modified to generate 1024-bits, 2048-bits or even 4096-bits key and use them for encryption/decryption. However, the hashing algorithm currently limits it. A hashing algorithm that could hash to a 1024-bits, 2048-bits or 4096-bits session secret key has to be provided in order for our encryption scheme to work.

3.4 Results

As discussed earlier, the encryption is non-deterministic for the ECB mode in our encryption scheme. We would now use our implementation to demonstrate this characteristic with a few examples.

The steps to compute the Session Secret Key in Figure 3-1 and the steps to compute Session Block Key in Figure 3-2 would be omitted to give a clearer illustration of the non-deterministic characteristic.

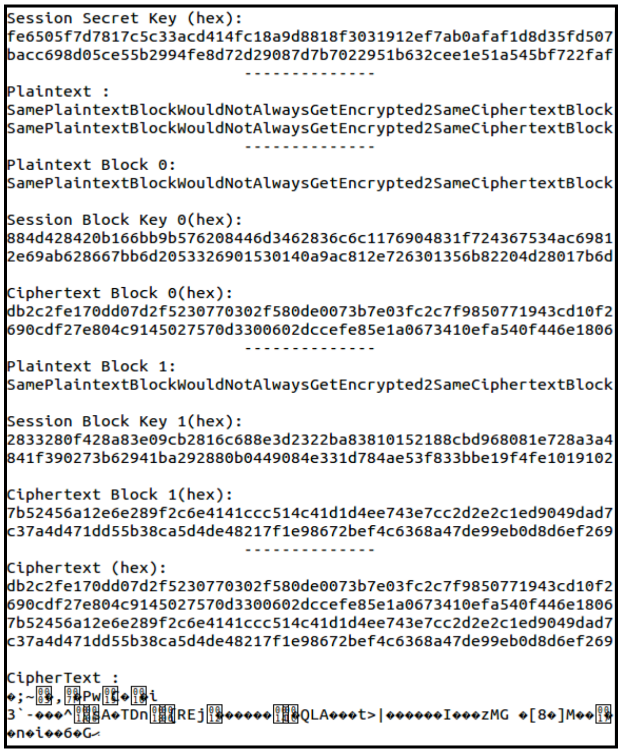
For easier comparison of the encrypted blocks, we would assume the same Session Secret Key in all examples. With reference to the diagram in Figure 3-3, we would show a few scenarios with the combination of Internet files being changed to see the effects of Function F and ECB mode on the encrypted blocks.



```
Session Secret Key (hex):
fe6505f7d7817c5c33acd414fc18a9d8818f3031912ef7ab0afaf1d8d35fd507
bacc698d05ce55b2994fe8d72d29087d7b7022951b632cee1e51a545bf722faf
                         --------------
Plaintext :
SamePlaintextBlockWouldNotAlwaysGetEncrypted2SameCiphertextBlock
SamePlaintextBlockWouldNotAlwaysGetEncrypted2SameCiphertextBlock
                         --------------
Plaintext Block 0:
SamePlaintextBlockWouldNotAlwaysGetEncrypted2SameCiphertextBlock

Session Block Key 0(hex):
ea3b7440473ebbb50c508f6dfe69f27cf96e681c0703e9ba416e133640df6b70
7953856379dec8c0852f23b100644dabf04c64b56d8f21f40093720750ee40e7

Ciphertext Block 0(hex):
b95a19251752dadc6224ea158a2b9e139a053f73726f8df42e1a525a37be1203
3e36f12617bdbab9f55b46d532372cc6950f0dc505ea538065eb06453c81238c
                         --------------
Plaintext Block 1:
SamePlaintextBlockWouldNotAlwaysGetEncrypted2SameCiphertextBlock

Session Block Key 1(hex):
69824a258a39c02b0f79d4f18c496b18fd7a8960bb6eda6988a665a0a8835515
dbd10cfd21d250d93496561c9b1318b553dff13913ce9d8ba73f2537b6a8a460

Ciphertext Block 1(hex):
3ae32740da55a142610db189f80b07779e11de0fce02be27e7d224ccdfe22c66
9cb478b84fb122a044e23378a94079d8369c98497babefffc2475175dac7c70b
                         --------------
Ciphertext (hex):
b95a19251752dadc6224ea158a2b9e139a053f73726f8df42e1a525a37be1203
3e36f12617bdbab9f55b46d532372cc6950f0dc505ea538065eb06453c81238c
3ae32740da55a142610db189f80b07779e11de0fce02be27e7d224ccdfe22c66
9cb478b84fb122a044e23378a94079d8369c98497babefffc2475175dac7c70b

CipherText :
�����e�□E<�#�:�'@�U�Ba��.□RZ7�□□6�&□����[F�27,ḫ
   w�□□'��$���,f��x�O�"�D�3x�@y�6��I{����GQu���
```

Figure 3-4 Scenario 1 with Internet file 1

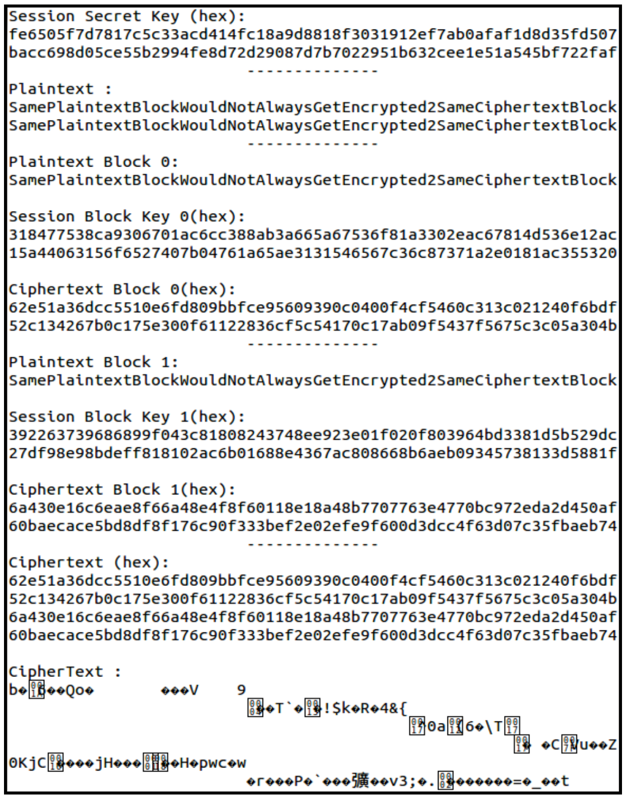
We can see from scenario 1 shown in Figure 3-4 that our ECB mode is able to encrypt the two identical plaintext blocks into two different ciphertext blocks



```
Session Secret Key (hex):
fe6505f7d7817c5c33acd414fc18a9d8818f3031912ef7ab0afaf1d8d35fd507
bacc698d05ce55b2994fe8d72d29087d7b7022951b632cee1e51a545bf722faf
                         --------------
Plaintext :
SamePlaintextBlockWouldNotAlwaysGetEncrypted2SameCiphertextBlock
SamePlaintextBlockWouldNotAlwaysGetEncrypted2SameCiphertextBlock
                         --------------
Plaintext Block 0:
SamePlaintextBlockWouldNotAlwaysGetEncrypted2SameCiphertextBlock

Session Block Key 0(hex):
884d428420b166bb9b576208446d3462836c6c1176904831f724367534ac6981
2e69ab628667bb6d2053326901530140a9ac812e726301356b82204d28017b6d

Ciphertext Block 0(hex):
db2c2fe170dd07d2f5230770302f580de0073b7e03fc2c7f9850771943cd10f2
690cdf27e804c9145027570d3300602dccefe85e1a0673410efa540f446e1806
                         --------------
Plaintext Block 1:
SamePlaintextBlockWouldNotAlwaysGetEncrypted2SameCiphertextBlock

Session Block Key 1(hex):
2833280f428a83e09cb2816c688e3d2322ba83810152188cbd968081e728a3a4
841f390273b62941ba292880b0449084e331d784ae53f833bbe19f4fe1019102

Ciphertext Block 1(hex):
7b52456a12e6e289f2c6e4141ccc514c41d1d4ee743e7cc2d2e2c1ed9049dad7
c37a4d471dd55b38ca5d4de48217f1e98672bef4c6368a47de99eb0d8d6ef269
                         --------------
Ciphertext (hex):
db2c2fe170dd07d2f5230770302f580de0073b7e03fc2c7f9850771943cd10f2
690cdf27e804c9145027570d3300602dccefe85e1a0673410efa540f446e1806
7b52456a12e6e289f2c6e4141ccc514c41d1d4ee743e7cc2d2e2c1ed9049dad7
c37a4d471dd55b38ca5d4de48217f1e98672bef4c6368a47de99eb0d8d6ef269

CipherText :
�;~□,□Pw□C�□i
3`-���^□□A�TDn□□REj□������□□QLA���t>|������I���zMG �[8�]M��□
�n�i��6�G⸗
```

Figure 3-5 Scenario 2 with Internet file 2

Scenario 2 in Figure 3-5 show how another Internet file feed into our Function F would affect the resultant session block keys, which in turn, affect the encrypted blocks. A different session block key was generated for the block 1 and 2 compared to session block keys in scenario 1.



```
Session Secret Key (hex):
fe6505f7d7817c5c33acd414fc18a9d8818f3031912ef7ab0afaf1d8d35fd507
bacc698d05ce55b2994fe8d72d29087d7b7022951b632cee1e51a545bf722faf
                         --------------
Plaintext :
SamePlaintextBlockWouldNotAlwaysGetEncrypted2SameCiphertextBlock
SamePlaintextBlockWouldNotAlwaysGetEncrypted2SameCiphertextBlock
                         --------------
Plaintext Block 0:
SamePlaintextBlockWouldNotAlwaysGetEncrypted2SameCiphertextBlock

Session Block Key 0(hex):
318477538ca9306701ac6cc388ab3a665a67536f81a3302eac67814d536e12ac
15a44063156f6527407b04761a65ae3131546567c36c87371a2e0181ac355320

Ciphertext Block 0(hex):
62e51a36dcc5510e6fd809bbfce95609390c0400f4cf5460c313c021240f6bdf
52c134267b0c175e300f61122836cf5c54170c17ab09f5437f5675c3c05a304b
                         --------------
Plaintext Block 1:
SamePlaintextBlockWouldNotAlwaysGetEncrypted2SameCiphertextBlock

Session Block Key 1(hex):
392263739686899f043c81808243748ee923e01f020f803964bd3381d5b529dc
27df98e98bdeff818102ac6b01688e4367ac808668b6aeb09345738133d5881f

Ciphertext Block 1(hex):
6a430e16c6eae8f66a48e4f8f60118e18a48b7707763e4770bc972eda2d450af
60baecace5bd8df8f176c90f333bef2e02efe9f600d3dcc4f63d07c35fbaeb74
                         --------------
Ciphertext (hex):
62e51a36dcc5510e6fd809bbfce95609390c0400f4cf5460c313c021240f6bdf
52c134267b0c175e300f61122836cf5c54170c17ab09f5437f5675c3c05a304b
6a430e16c6eae8f66a48e4f8f60118e18a48b7707763e4770bc972eda2d450af
60baecace5bd8df8f176c90f333bef2e02efe9f600d3dcc4f63d07c35fbaeb74

CipherText :
b�□6��Qo�       ���V    9
                         □�T`�□!$k�R�4&{
                                        □0a□6�\T□
                                                 □ �C□u��Z
0KjC□����jH���□□�H�pwc�w
                         �r���P�`���彇��v3;�.□�������=�_��t
```

Figure 3-6 Scenario 3 with Internet file 1 and Internet file 2

The effect of 2 steps block chaining operation in Function F resulting in distinctive session block keys and ciphertext can be seen in Scenario 3 of Figure 3-6.

With these scenarios, we can see that any given plaintext block would not always get encrypted to the same ciphertext block. In addition, the nonce1 and nonce2 are randomized in each session. Given a scenario where the user decides to encrypt the same plaintext with same files, the end result would be a completely distinctive ciphertext.

More detailed analysis and evaluation on the strength of the individual encrypted block and our encryption scheme would be covered later in the section 5.

4. Entropy of Media formats

The idea is to identify which media files would hold enough randomness quality to be used for generating the keys. Using entropy to measure the randomness is a highly accepted practice in information theory. [4] Different encoding formats convey various range of entropy. In this paper, we will focus on some common video and audio files. We tested the different existing sets collected from the Internet for randomness. We used a part of **ENT**[5] (A pseudorandom number sequence test program), and a python script to verify the output. We tested more files for individual format and selected results of five of each type.

4.1 Differences between formats

Different encoding formats convey various range of entropy. In this paper, we will focus on some common video and audio files. We tested the different existing sets collected from the internet for randomness. We tested more files for individual format and selected results of five of each type.

4.1.1 MKV

Test results for MKV file format are shown in **Table 1**. MKV files have very good results for our work. Entropy varies from 99.885% to 99.99%, which is nearly 100%.

Table 1

File Name	Size (byte)	Entropy (%)
Sample1.mkv	16658293	99.980522
Sample2.mkv	16094941	99.976234
Sample3.mkv	26246048	99.885612
Sample4.mkv	14117113	99.997154
Sample5.mkv	4124170	99.984505

4.1.1.1 AVI

Test results for AVI format is shown in **Table 2**.Another good results for our work. Entropy varies from 99.62% to 99.85%.

Table 2

File Name	Size (byte)	Entropy (%)
Sample1.avi	38709642	99.629494
Sample2.avi	17639654	99.850845
Sample3.avi	10713088	99.676903
Sample4.avi	19992576	99.806305
Sample5.avi	6410240	99.676407

4.1.1.2 MP4

Test results for MP4 format is given in **Table 3**. Not as good as MKV or AVI format as the entropy varies from 91.82% to 98.36%.

Table 3

File Name	Size (byte)	Entropy (%)
Sample1.mp4	87087	98.210075
Sample2.mp4	67644	98.365974
Sample3.mp4	42798	91.828133
Sample4.mp4	56345	96.916664
Sample5.mp4	49676	97.121902

4.1.1.3 MP3

Test results for MP3 format is shown in **Table 4**. According to the results, MP3 format is not very good for our work.

Table 4

File Name	Size (byte)	Entropy (%)
Sample1.mp3	204029	98.923233
Sample2.mp3	722304	98.165497
Sample3.mp3	9536	90.6446
Sample4.mp3	86058	99.093521
Sample5.mp3	5408	87.346275

4.1.1.4 WAV

WAV format does not seem to have enough randomness for our work.

Table 5

File Name	Size (byte)	Entropy (%)
Sample1.wav	491516	84.95134
Sample2.wav	331108	71.919235
Sample3.wav	355588	72.179588
Sample4.wav	409152	65.669975
Sample5.wav	728396	78.855354

4.2 Discussion

Our observations show that certain media formats are not suitable for our encryption scheme; therefore, we have to be very careful when we choose the file format that has better quality of randomness. Testing showed that MKV format brings the level of randomness that can be used for short-lived cryptographic keys.

5. Testing of Encryption Scheme

We will be testing on the randomness of our encryption method as well as to see if our encryption scheme can increase the entropy of the original media files.

We will be encrypting bmp images to illustrate how secure the encryption scheme is. Moreover we will be using tools available on NIST to illustrate how random our encryption scheme is as compared to other encryption schemes.

5.1 Testing Strategies

As mentioned above the testing component has 2 parts to it.

1. Using a tool available on **NIST** to test the randomness of our encrypted block.
2. Encrypt a bmp image to view the outcome of the image.

5.2 Tests conducted

The **Statistical Test Suite (v2.1.2)**[6] tool provided by **NIST** contains a lot of test to test for randomness. However we will just be testing our encryption scheme using their Frequency test and Approximate Entropy Test. We will also be conducting an image test to visually compare the result of each encryption technique. Lastly, we will experiment on how our encryption scheme would impact on files with 0% entropy. In all tests except the last, we will be using the same files, and order, to encrypt. They are: Sample1.mkv.

5.2.1 Frequency Test

The purpose of this test is to determine the number of 1s and 0s bits in a stream of bits is about the same as a stream of random bits. This is done by looking at the number of 1s and 0s in a stream of bit. If the number of 1s occupies about half of the bit stream it means that the bit stream is random.

5.2.1.1 Description of Test

1. The program will take in a bit stream and replace all 0 as -1.
2. Program will then add up the bit stream values.
   a. Eg, 1011010101, size of bit stream = 10. So sum of the bit stream = 1 + (-1) + 1 +1 +(-1) +1 +(-1)+1 +(-1) +1 = 2
3. Program then compute the statistic result of 0s and 1s and give a p value.
4. This p value will be determined whether a sequence of bits is random or not.
   a. If the P value < 0.01 then the input sequence is non – random otherwise is random.

5.2.2 Image test

The purpose of the test is to identify if there are any visual patterns in the encrypted bitmap image, which would then be used to gauge the effectiveness of the encryption scheme.

5.2.2.1 Description of Test

1. We select a bmp image to be encrypted.
2. We then do the same with our encryption scheme.
3. We compare the result of the images in terms of visibility of original message, or any information leakage such as pattern formation on the result of encrypted bmp image file.

5.2.3 0% Entropy test

The purpose of the test is to see how well our encryption scheme can perform by encrypting a file with 0% entropy. The test would be conducted with the tool and method described in Section 3.

5.2.3.1 Description of Test

1. We will create a file filled with 10000 0x00.
2. We will encrypt the created file using our encryption scheme multiple times, using different files and different orders selected from our sample files.
3. We will conduct entropy test on the encrypted file.

5.3 Tests results

In this section we discuss our test results and compare and contrast with other encryption scheme.

5.3.1 Frequency Test

The results from the frequency test are as follows.

p-value from the Frequency Test: **0.739918**

Interpretation of p-value, using the **NIST STS** tool, if p-value is >= 0.01, we would accept the sequence as random. [6]

5.3.2 Image test

The results from the image test are as follows.

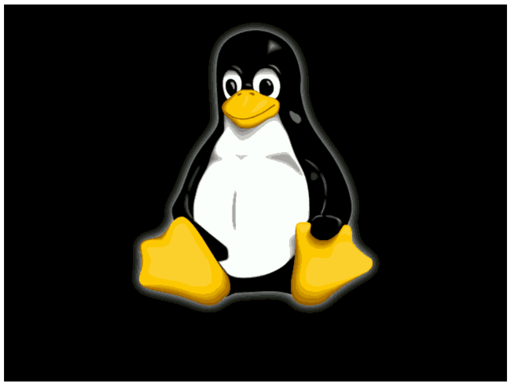


Figure 4-1Sample.BMP before encryption

Figure 4-2 Sample.BMP after encryption

In this test, the penguin is visually non-existent.

5.3.3 0% Entropy test

The results from the 0% entropy test are given in the following table. It shows the final amount of entropy after running our encryption scheme on a file with 0% entropy.

Files used	Before	After
Sample1.mkv Sample1.avi	0%	84.6324993421%
Sample1.avi Sample1.mkv	0%	92.2750846085%
Sample1.mkv	0%	93.4417328351%
Sample1.avi	0%	84.284006621%
Sample1.mkv Sample2.mkv Sample3.mkv	0%	93.3992040658%
Sample1.avi Sample2.avi Sample3.avi	0%	84.0028814941%

5.4 Discussion

Although the test results above are quite optimistic, there may be certain files, when used as key to encrypt, would leak certain information.

6. Conclusion

We have came up with an encryption scheme that would allow people to encrypt large files with an effective key size of 512-bits, but scalable to higher bit security in the future. From our tests, using certain media formats may have higher effectiveness than others. The resultant entropy of the encrypted file using our current encryption scheme may be affected by the choice of file used as keys.

7. ACKNOWLEDGMENTS

Our thanks to Prof. Hugh Anderson for allowing us to design and develop our own encryption scheme.

8. REFERENCES

[1] Shannon, C. E. (1949). A Mathematical Theory of Communication. *Bell System Technical Journal*, 623.

[2] Dworkin, M., & National Institute of Standards and Technology (U.S.). (2001). *Recommendation for block cipher modes of operation: Methods and techniques* (2001 ed.). Gaithersburg, MD: U.S. Dept. of Commerce, Technology Administration, National Institute of Standards and Technology.

[3] C. E. Shannon and W. Weaver*, A Mathematical Theory of Communication. Champaign*, IL, USA: University of Illinois Press, 1963.

[4] *Calculate File Entropy. (2013, May 18).* Retrieved November 5, 2015.

[5] J. Walker, "Ent - a pseudorandom number sequence test program," http://www.fourmilab.ch/random/, 2008.

[6] *Bassham, L. A Statistical Test Suite for Random and Pseudorandom Number Generators for Crytographic Applications [PDF Document]*. Retrieved from http://csrc.nist.gov/groups/ST/toolkit/rng/documents/SP800-22rev1a.pdf