Project Ⅰ of Computer Organization and Design

C/C++ - based cracking for

Microsoft Office Word (2010 version) encrypted document

College of Information Science and Electronic Engineering, Zhejiang University

Han,Xiao 3160101136

hxiao@zju.edu.cn

11/11/2018

Abstract

As the innovative products of Office, Office2010 use Open XML language to describe and ZIP to pack, and choose the SHA1-AES encryption mechanism. The report started from the document structure, extracted the encryption information flow, and then cracked the password by password brute-force method, analyzed the result at last.

Keywords—MS Office Word, Agile encryption, brute force Crack, OpenXML

I . Introduction

Since the Microsoft Office series is widely used now, the security of Office document is more and more important. Office2003 and the version before are always criticized for the poor encryption schema, and we can find many crack tools that can crack the password of encryption document in a short time on the internet, such as Office recovery Toolbox, word key, etc. As the innovative products of Office, what Office2007 and Office2010 improved is more than the stronger humanized user interface, and the changed storage method of recommending which is "Open XML" that has been ECMA standardization[1], but the document protection is also increased through improving password inspection, rights restriction and digital signatures [2].

This report is aimed at the encryption mechanism research of Word in Office2010. Based on the document structure, I looked for encryption information flow, coded in C/C++ language on DevC++ 5.11, and then tested on single PC by brute force crack.

II . Office2010 Encryption Mechanism

1) Key generation

The key generation is as follows:

```
H0=H(salt+unicode(password))[5];
for(;i<100000;i++)
    Hn=H(i+Hn-1);
Hfinal=H(H99999+block_key);
```

block_key represents an array of bytes used to prevent different blocks from encrypting to the same cipher text. The algorithm can be SHA-1, SHA256, MD5, etc while the default is SHA-1. If size of the resulting Hfinal is shorter than the default key length, then the key must be padded by appending bytes with a value of 0x36. If the hash value is larger in size than key length, the key is obtained by truncating the hash value.[2]

2) Encryption

Office2010 adopted agile encryption, and the cipher algorithm can be AES, RC2, DES, etc while the default is AES128. The chaining mode used by the cipher algorithm is CBC. All the choice can be changed by register and Oct2010.[2]

The encryption as follows:

a) Generate a random array as verifier.

b) verifierHash = H(verfier)

c) Generate key1 by password and salt, and the block_key consist of the the following bytes 0xfe,0xa7,0xd2,0x76,0x3b,0x4b,0x9e,0x79.

d) The IV(initialization vector) CBC used is salt.[3]

e) EncryptedVerifierHashInput =Encrypt(verifier,key1,IV);
encode EncryptedVerifierHashInput by base64 and written to EncryptionInfo flow.

f) Generate key2 by password and salt, and the block_key consist of the the following bytes 0xd7,0xaa,0x0f,0x6d,0x30,0x61,0x34,0x4e.

g) The IV(initialization vector) CBC used is salt.

h) EncryptedVerifierHashValue=Encrypt(verifierHash,,key2,IV);
encode EncryptedVerifierHashValue by base64 and written to EncryptionInfo flow.

3) Decryption

The data should be decoded by base64 according to encryption as follows:

a) verifier1=Decrypt(base64decode(EncryptedVerifierHashInput),key1,IV) IV=salt.

b) verifierHash2=Decrypt(base64decode(EncryptedVerifierHashValue),key2,IV) IV=salt.

c) verifierHash1 = H(verifer1)

d) compare verifierHash1 and verifierHash2, if the two hash values do not match, the password is incorrect.

III. Process of recovering the password

The general idea is to determine whether the generated eigenvalues are consistent with those extracted from the original file by traversing all possible passwords and repeating the Word2010 encryption process. The algorithm flow has been mentioned above, and here are a few details that cannot be ignored. Preprocessing of pwd and salt refers to adding pwd after salt through a bitwise connection. It's worth mentioning that the password connected to the salt value is converted from ASCII to UNICODE and stored in Little-Endian. In the same way, the iterator should be stored in the form of Little-Endian while the SHA1 loop is running.

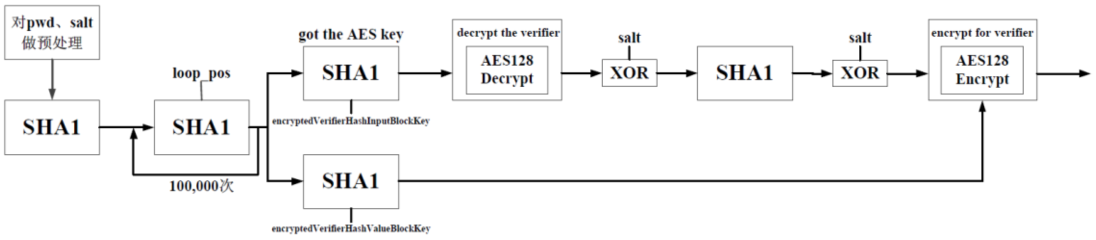
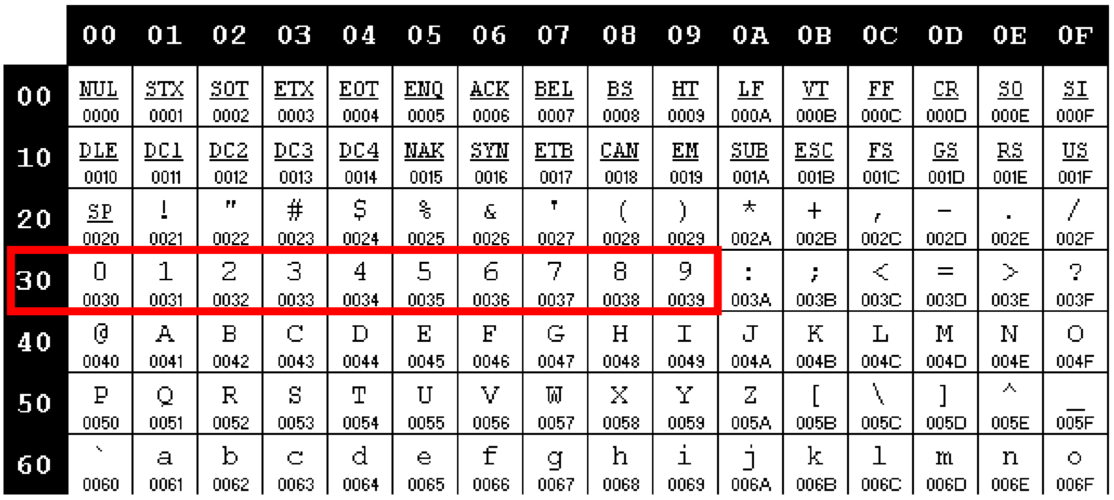


Figure1. Algorithm flow chart overview



	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	NUL 0000	STX 0001	SOT 0002	ETX 0003	EOT 0004	ENQ 0005	ACK 0006	BEL 0007	BS 0008	HT 0009	LF 000A	VT 000B	FF 000C	CR 000D	SO 000E	SI 000F
10	DLE 0010	DC1 0011	DC2 0012	DC3 0013	DC4 0014	NAK 0015	SYN 0016	ETB 0017	CAN 0018	EM 0019	SUB 001A	ESC 001B	FS 001C	GS 001D	RS 001E	US 001F
20	SP 0020	! 0021	" 0022	# 0023	$ 0024	% 0025	& 0026	' 0027	(0028	) 0029	* 002A	+ 002B	, 002C	- 002D	. 002E	/ 002F
30	0 0030	1 0031	2 0032	3 0033	4 0034	5 0035	6 0036	7 0037	8 0038	9 0039	: 003A	; 003B	< 003C	= 003D	> 003E	? 003F
40	@ 0040	A 0041	B 0042	C 0043	D 0044	E 0045	F 0046	G 0047	H 0048	I 0049	J 004A	K 004B	L 004C	M 004D	N 004E	O 004F
50	P 0050	Q 0051	R 0052	S 0053	T 0054	U 0055	V 0056	W 0057	X 0058	Y 0059	Z 005A	[005B	\ 005C	] 005D	^ 005E	_ 005F
60	` 0060	a 0061	b 0062	c 0063	d 0064	e 0065	f 0066	g 0067	h 0068	i 0069	j 006A	k 006B	l 006C	m 006D	n 006E	o 006F

Figure2. Unicode character table

Since the input here is over 128 bits, we need the piecewise operation for AES128 algorithm. Then remember to connect the results of the piecewise operation and compare them with the values extracted from the original file, namely comparing “Verifier” and “VerifierHashValue”.

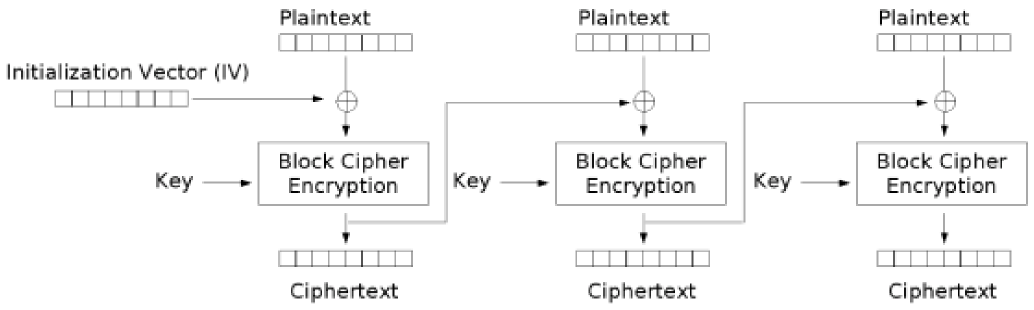


Figure3. Algorithm process of AES128 (CBC) mode encryption[6]

In the process of decryption, since the input is exactly 128bits, the second step of the algorithm process is not required, but considering the integrity of the report, the whole process is given as follows.

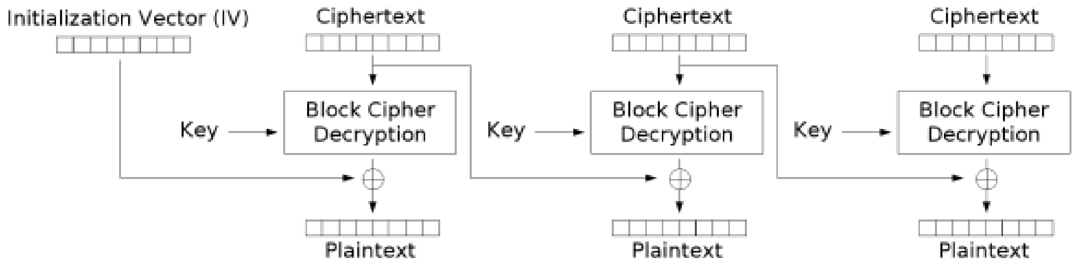


Figure4. Algorithm process of AES128 (CBC) mode decryption[6]

IV. Analysis of the results

1) Accelerating techniques

After writing the program, I cracked different encrypted documents, and the program was able to find the password correctly. The time statistics for each run result are as follows. In addition, the verification time for each password is approximately 0.120 seconds. Since the program is small in size, only 1,000 possible passwords have been traversed, so I did not generate trying passwords by random numbers, but by incrementing from 000. It is obvious that the running time of the program is linearly increasing with the size of the password, which is approximately equal to the password times the single running time.

Password	001	201	399	505	813	998
Time/s	0.254	26.853	58.821	69.770	108.595	132.282

In order to speed up the running of the program, I also provide some functional modules of OpenMP that support C/C++. The flow chart is as follows.

a) OpenMP implements parallelization with the fork-join model.

b) all OpenMP programs start with one main thread.The main thread is executed serially until the first parallel region is encountered.

c) fork: the main thread then creates a set of parallel threads.

d) the code in the parallel region is surrounded by braces, and then executed in parallel on multiple parallel threads.

e) merge: when the parallel threads execute the code in the parallel region, they synchronize and finish automatically, leaving only the main thread.

f) the number of parallel zones and the number of parallel threads can be arbitrary.

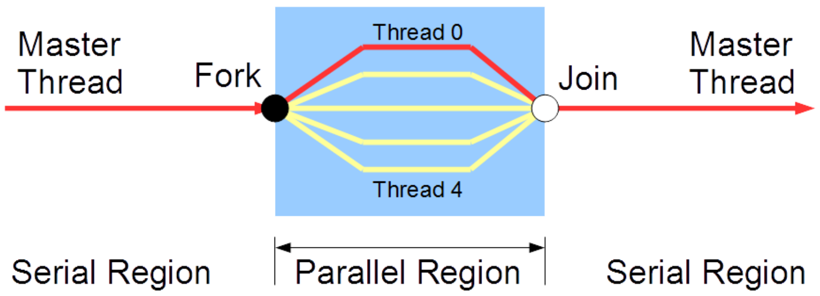


Figure5. Flow chart of the fork-join model of OpenMP

However, for this program, OpenMP can't play its advantages, it will only speed up the expected speed of the whole program, and not necessarily speed up for some practical problems.For example, when the password is 000, I believe the sequential traversal method will be faster than any other method. Of course, I also did a lot of basic algorithm optimization, do not add here, please read the source code.

2) What can be improved

Due to limited time and my ability, this procedure can be improved. For example, when extracting the Hash value, I used the brute-force method to complete the string matching. We can replace it with more efficient algorithms, such as Horspool algorithm, BM algorithm and so on. In addition, this program can only crack the 2010 version of MS Office Word file, and the password can only be 000 -999. We can enrich the possibility of traversing the password, and can add more algorithm modules, so that this program can crack other versions of more complex passwords Word documents. Since this aspect involves so much else, I won't expand it here, and I'll stop there.

V.Reference

[1] Microsoft Office File Format Documentation Introduction, Microsoft Corporation, 2011

[2] Office Document Cryptography Structure Specification, Microsoft Corporation, November 13, 2010

[3] X. Wu, J. Hong and Y. Zhang, "Analysis of OpenXML-based office encryption mechanism," in 2012, . DOI: 10.1109/ICCSE.2012.6295128.

[4] J. Hong, Z. Chen and J. Hu, "Analysis of encryption mechanism in office 2013," in 2016, . DOI: 10.1109/ICASID.2015.7405655.

[5] https://msdn.microsoft.com/en-us/library/dd950165(v=office.12).aspx

[6] https://blog.csdn.net/charleslei/article/details/48710293