How to operationally detect misuse or flawed implementation of weak stream ciphers (and even block ciphers sometimes) and break them - Application to the Office Encryption Cryptanalysis

Eric Filiol

ESIEA Laval
Operational Cryptology and Virology Lab $(C+V)^O$
filiol@esiea.fr
http://www.esiea-recherche.eu/

Abstract. Despite the evergrowing use of block ciphers, stream ciphers are still widely used: satellite communications (military, diplomatic...), civilian telecommunications, software... If their intrinsic security can be considered as strong enough even for strategic use, the main drawback lies in the high risk of key misuse wich introduces severe weaknesses, even for unconditionnally secure ciphers like the Vernam system. Such misuses are still very frequent, more than we could expect.

In this paper, we explain how to detect such misuses, to identify ciphertexts that are relevant to this misuse (among a huge amount of ciphertexts) and finally how to recover the underlying plaintexts within minutes. This may also apply to (intendly or not) badly implemented block ciphers. In any cases, the cryptosystem can remain unknown to the cryptanalyst.

To illustrate the efficiency of our technique, we then apply it to the operational cryptanalysis of Office encryption up to the 2003 version (RC4-based). We focus on Word and Excel applications. The cryptanalysis was performed successfully and we managed to recover more than 90 % of the encrypted texts in a few minutes. The attack combined both a pure mathematical effort AND a few basic forensic approaches. In more general cases (e.g. satellite communications), we just need to intercept ciphertexts.

In the Office case, we explain in our sense that the attack does not rely on particular weakness but in a setting that can be seriously considered and described as a possible intended trapdoor. We develop this concept to explain how in a more general way such a trapdoor can be built.

1 Introduction

In any course of academic cryptography, the key size is often presented as a key condition for the security of a crypto system. For publishers of so-called secure solutions, the size of the key summarizes everything, hence becomes a sufficient condition and a marketing argument. How many times the single mention of "AES 256 inside" is considered as a key security argument! But if it works on paper, in practice it is quite different. Weak implementation – or worse intended implementation trapdoor – often reduces the security of an alleged application which at the end amounts very little. But the misuse of cryptographic system can also have a dramatic impact on the real security. If encryption means have been deregulated, the education of users has never been organized. They finally use systems that can turn against them due to their ignorance of basic rules. In the case of a typical user this situation does not matter so much than in the worrying case of professional use.

The problem then is twofold and exhibits dual aspects. On the user's side, the problem is to detect such an implementation weakness or, worse, the existence of a trapdoor. The problem lies in the fact that the user cannot perform reverse engineering – for legal reasons or since it is horribly time consuming especially for heavily obfuscated programs. On the attacker's side[1], most of the time the encryption algorithm is unknown (e.g. in satellite transmission) but the aim is first to identify every ciphertext produced by a misused/weak cryptosystem and second to recover the corresponding plaintext.

In this paper we show how to operationally solve all those issues in the context of any kind of stream cipher when misused, and of weakly implemented block ciphers. We present the technique we developped for that purpose. This technique was developped in 1994 by the author and is published in this white paper only now. Such a technique seems to have been developped by the NSA for the Venona project [11] to break the Soviet telegraph traffic. While this project was partially declassified in 1995, neither the plaintexts nor the decrypting algorithms have been ever made public.

[1] In the case of forensic analysis (e.g. of a hard disk) the forensic expert can be considered as an attacker with respect to the (prosecuted) user.

We apply this technique, as an illustrating example, to break 128-bit RC4 encryption of the Microsoft Office suite (up to version 2003). The Microsoft Office application (up to Office 2003) provides native encryption algorithms to protect documents ranging from simple, lame XOR encryption (the lowest ever) to the RC4 algorithm with a 128-bit key. The latter is considered by industry until recently as offering adequate security both in terms of key size and cryptographic design.

But Office 2003 still represents more than 75 % of domestic licences and yet nearly 80 % of professional licenses. This is explained by the fact that Microsft Office 2007 version failed to attract many people because of a disconcerting break of ergonomics and a lack of easy-to-use features.

In 2005, a Singaporian researcher [12] highlighted a weakness in Microsoft Office with respect to RC4 128: the keys remain the same for all modified versions (revisions) of a document thus violating a basic rule in cryptography. In the case of stream ciphers (and sometimes in block ciphers), reusing the key is dramatic. This error, as a consequence, precisely reduces the perfect security of a Vernam system [9] to a total insecurity. But this researcher never explained how to exploit in practice such a weakness. Considering existing works, only a very few publications addressed this issue. They were either too empirical [4] or addressed the issue in a very limited context (the plaintext type must be known) [8]. Those works do not address the critical issue of detecting the reuse of secret keys among a large number of ciphertexts at all.

But this error is even worse since it enables to break misused/weak cryptosystems without going through the academic step of preliminary key recovery or key search. In this paper, we detail the technique we developed in the context of the cryptanalysis of stream ciphers (or in some cases of block ciphers) when misused (or poorly implemented) thus enabling to recover the plaintext from an encrypted message in polynomial time. We then apply this technique to the operational cryptanalysis of Microsoft Word and Excel (version 11.0 or up to Office 2003).

This technical white-paper is organized as follows. Section 2 first recalls a few basics in cryptography to make this paper self-contained and thoroughly formalizes the problem to solve. In Section 3, then,

we address the issue of detecting the different groups of ciphertexts with respect to the cryptosystem misuse/vulnerability. Section 4 presents then how to recover the plaintext from the weak ciphertexts. Section 5 applies the cryptanalysis in the the case of Microsoft Word while Section 6 solves the issue in the case of Microsoft Excel. Finally Section 7 concludes while discussing how efficient trapdoors could be embedded to weaken cryptographic products.

2 A few Basics in Cryptology

In this section we are recalling a few concepts and definition which will help to understand the rest of the paper and made it self-contained. The reader is advised to refer to [9] for an in-depth coverage of cryptographic techniques.

As for data privacy is concerned, we must use encryption. As soon as data exceeds a few tens of bytes, we must consider symmetric encryption (asymmetric encryption generally considers very limited amounts of data like session key or text digest [hash value]). The reason is that contrary to asymmetric encryption (which relies on complexity theory):

1. it provides a fast encryption speed which is a critical features for large amounts of data;
2. it is possible to prove that an arbitrary level of security is achieved (symmetric cryptogtaphy is based on information theory and the core concept of entropy).

Two different families of symmetric cryptosystems are known:

- *Stream ciphers* where bits/bytes of data are enciphered/deciphered on the fly. These ciphers offer the best possible encryption speed. This is the reason why they are mainly used for satellite communication protection, telephony encryption (e.g. A5/1), Bluetooth encryption (e.g. E0 cryptosystem)... Moreover, they are very transmission error resilient (no propagation of errors during the decipherment).
- *Block ciphers* where data are first split into blocks (the standard is nowadays 128-bit blocks) and each of them is enciphered/deciphered by means of the same secret key. By construction block

ciphers are not naturally transmission error resilient and they are slower than pure stream ciphers. This is the reason why block ciphers have been designed in such a way that they can operate according to different modes: ECB, CBC, CFB, PCBC, OFB [9].

2.1 Stream ciphers

From a very general point of view, stream ciphers are inherently synchronous ciphers. In other words, every ciphertext bit, at time instant t is produced by xoring the corresponding plaintext bit at the same time instant with a random bit according to the formula:

$$c_t = m_t \oplus \sigma_t.$$

The bits σ_t are produced independently from the plaintext.

Conversely (hence the name "symmetric" for that class of cryptosystems), to decipher the ciphertext bit, we just apply the same random bit for the relevant time instant t since the xor operation is involutive:

$$m_t = c_t \oplus \sigma_t.$$

In this context, the security lies on the random sequence $(\sigma_t)_{t\geq 0}$ (we therefore denote the plaintext [respectively the ciphertext] by the sequence $(m_t)_{t\geq 0}$ [$(c_t)_{t\geq 0}$ respectively]). The sequence $(\sigma_t)_{t\geq 0}$ is referred as the running key.

Whenever the running key is produced randomly and independently from any other quantity (plaintext or ciphertext) we then refer to *One time pad* (Vernam ciphers). This kind of encryption method is the only one that can be proved as being unconditionally secure: it relates to the perfect secrecy defined by the third Shannon theorem. More precisely, if M, C and K describe the random variables denoting the plaintext, the ciphertext and the running key respectively (considered as sequences) then

$$H(M) = H(M|C)$$

where $H(.)$ is the entropy function[2]. In an equivalent way, it means that the transinformation (or mutual information) is null. In other

[2] For any discrete random variable (e.g. an information source) X taking values $X_1, X_2, \ldots, X_n$ with non null probabilities $p_1, p_2, \ldots, p_n$, then $H(X) = -\sum_{i=1}^{n} p_i \log_2 p_i$

words, the knowledge of the ciphertext (than can always be obtained by any attacker) does not provide any information on the plaintext. Similarly, we can define the same property with respect to the key:

$$H(K) = H(K|C).$$

C. E. Shannon [10] proved how perfect secrecy can be practically achived: we must have

$$H(K) \geq H(M)$$

It means that the uncertainty about the key (e.g. its entropy) must be at least as large as that with respect to the plaintext. If the key has length $|K|$ and if the key bits are randomly and independently produced, then we have $H(K) = |K|$. Hence the Shannon condition becomes $|K| \geq H(M)$. From a practical point of view, it implies that the running sequence $(\sigma_t)_{t\geq 0}$ must be at least as long as the message to encipher.

The main issue lies in the fact that this constraint is very difficult to manage. One time pad are produced by hardware methods (e.g. sampling a thermal resistance) and then must be duplicated. One time pad are then used for short message or for strategic use only.

This is the reason why most of the time, stream ciphers are only a good enough approximation of one time pads. In this case, the running sequence is produced as the expansion of a secret key whose size is reduced (up to 256 bits). The expansion is performed by a cryptosystem (an algorithm) which can be formalized by a finite state machine. The secret key is just the initialization of that machine at time instant $t = 0$.

But since in this case we have $H(K) << H(M)$, these systems no longer comply with the perfect secrecy condition. However, since the number of messages that are practically enciphered by a $|K|$-bit cryptosystem is always far lower than the number of possible secret keys ($2^{|K|}$), we can consider that **during the lifetime of the cryptosystem** we indeed have

$$H(K) \geq H(M).$$

But it is mandatory not to reuse secret keys otherwise the previous equation no longer holds. This also applies to one time pad cryptosystems.

Consequently, the security of stream ciphers is destroyed whenever key bits are reused to initialize the cryptosystem. We will formalize this later on in Section 2.3.

2.2 Block ciphers

Block ciphers follow a different design philosophy. The principle is to produce each ciphertext block directly from both the corresponding plaintext block and the secret key (which is reused from block to block). The reuse of the key is supposed to have no impact on the overall security[3].

However it is possible to emulate stream ciphers from block ciphers by using the latter in the so-called *output feedback mode* (OFB) which is described in Figure 1. In this mode, the secret key is the

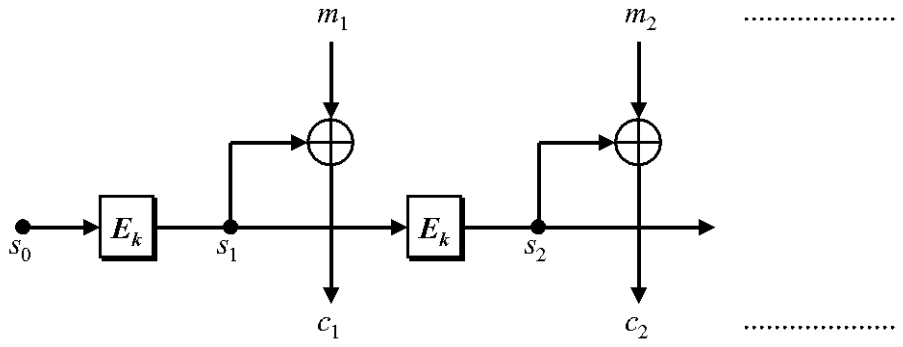


Fig. 1. Block cipher in OFB mode

block s_0 and the pseudo-running sequence is made of block $s_1, s_2, s_3 \ldots$ which are bitwise xored to the plaintext blocks. Consequently a block cipher in OFB mode is fully equivalent to a stream cipher.

[3] This assumption holds only for the open litterature in the field. However theoretical studies shows that the key reuse might be dangerous [7].

2.3 Definition and Formalization of the Problem

Now that basic definitions and concepts have been recalled, we can formally define the cryptanalytic and security problem we intend to solve.

When interpreting Shannon theory [10] in a suitable way, the reuse of secret key nullifies all the security of stream ciphers (or block ciphers in OFB mode). But no practical and operational exploitation of this interpretation has been ever published.

Let us suppose that we have two (or more) plaintext messages m_1 and m_2 (we drop off the time instantiation unless necessary). Since we reuse the secret key, we then consider a single (pseudo-random or not) running key σ. Then, we have for the corresponding ciphertexts:

$$c_1 = m_1 \oplus \sigma \text{ and } c_2 = m_2 \oplus \sigma.$$

We now have the following definition.

Definition 1 *Two (or more) ciphertexts are said* parallel *if they are produced from the same running key produced either by a stream cipher (Vernam cipher or finite state machine) or by a block cipher in OFB mode. If ciphertexts* $c_1, c_2 \dots c_k$ *are parallel, the* parallelism depth *is* k*.*

So the problem is twofold:

1. **Detection issue**.- Among a huge number of ciphertexts, how to detect the different groups of parallel messages?
2. **Cryptanalysis issue**.- Once parallel messages have been detected, how to break the encryption and recover the plaintexts?

The first problem is interesting in many ways. First we do not make any assumption about the underlying target cryptosystem. This means that it can remain totally unknown provided that it is indeed a stream cipher or a block cipher in OFB mode. In fact we will not care about the cryptosystem at all, under this simple assumption. It means that you can detect and break misused cryptosystem from this category without performing any reverse-engineering step (time consuming and illegal) or intelligence step to recover the encryption algorithm first.

More interestingly, suppose that we have a commercial encryption product based on a block cipher. We can either suspect an implementation flaw or, worse, a trapdoor to be present. In both case, the underlying block cipher is in OFB mode. Since we cannot reverse-engineer the product, we must use another approach to identify this critical issue. With our technique presented in this paper, we must have to produce a group of ciphertexts from the same secret key. It is important to recall that most users use the same secret key to encipher different documents. In this context, the concept of parallel messages is more than relevant.

From this, it becomes obvious that if we do not care about the cryptosystem, we must also work without the secret key itself. Consequently our technique must be efficient enough to recover the plaintexts without any preliminary key recovery step. This is the reason why we call our cryptanalysis technique *key-independent cryptanalysis*.

3 Detection of Weak Ciphertexts

3.1 General principle

Without loss of generality, let us consider two plaintexts M_1 and M_2 and their respective ciphertexts M_1 and M_2. Let be σ_1 and σ_2 two (pseudo random or not) running keys. Then we have (we do not consider time indexation; of course all operations are done bitwise or bytewise):

$$C_1 = M_1 \oplus \sigma_1 \quad \text{and} \quad C_2 = M_2 \oplus \sigma_2.$$

Now let us combine the two ciphertexts by a bitwise (or bytewise) xor. We have:

$$C_1 \oplus C_2 = (M_1 \oplus \sigma_1) \oplus (M_2 \oplus \sigma_2).$$

Plaintexts M_1 and M_2 exhibit very strong, biased statistical features corresponding to the language and the encoding[4]: each character (letter, number, punctuation...) has a different frequency of occurence.

[4] It is worth recalling and stressing on that during a cryptanalysis effort, considering the language only is a common but critical mistake. This mistake greatly reduces or

This frequency is itself different when considering ciphertext (uniform law with a probability $\frac{1}{256}$ for an ASCII encoding).

We then have two interesting features to consider:

- The quantity $M_1 \oplus M_2$ exhibits a very special statistical profile that can be easily detected and identified.
- The quantity $\sigma_1 \oplus \sigma_2$ exhibits a random statistical profile (each character appears with probability $\frac{1}{256}$ for an ASCII encoding).

These two features enable us, from the quantity $C_1 \oplus C_2$, to distinguish two cases, hence two statistical hypotheses. Our aim is to build a statistical testing to determine whether ciphertexts C_1 and C_2 are parallel or not (in other word, did the user or the application make the mistake to reuse the key or not?).

- **Null hypothesis (H_0).**- The two ciphertexts are not parallel (i.e.: σ_1 and σ_2 are different). The quantity $C_1 \oplus C_2$ exhibits a totally random statistical profile.
- **Alternative hypothesis (H_1).**- The ciphertexts are indeed parallel (σ_1 and σ_2 are the the same; the key has been reused). The quantity $C_1 \oplus C_2$ is then exactly equal to $M_1 \oplus M_2$ and hence exhibits a strongly biased statistical profile which depends on the underlying plaintext language.

These two hypotheses are sufficient to build our detection testing (simple hypotheses testing) [5]. For that purpose we have to choose a suitable estimator that behave differently under the two hypotheses H_0 and H_1.

3.2 Detection algorithm

Let us choose the following estimator:

$$Z = \sum_{i=0}^{n} (c_1^i \oplus c_2^i \oplus 1).$$

hinders the cryptanalysis result. The right approach is to consider both the language and its encoding at the same time. In particular, strongly biased statistical profiles appear or on the contrary remain hidden according to the encoding used. While an ASCII encoding generally erases those profiles, they appear when CCITTx encoding is used. So the choice of the encoding can also be part of a trapdoor design; refer to Section 7.

In this formula, n is the common size length of ciphertexts C_1 and C_2 (in bits), while c_1^i and c_2^i represent ciphertext bits of C_1 and C_2 respectively at time instant i. In fact, the estimator Z counts the number of bits equal to zero in the quantity $C_1 \oplus C_2$.

Then, using simple probability theory results [5], it is easy to prove that Z has a binomial distribution of parameter n and p where $p = P[c_1^i \oplus c_2^i = 0]$. This binomial distribution can be approximated (application of the central limit theorem) by a normal distribution of mean value np and of standard deviation $\sqrt{np(1-p)}$. Then asymptotically Z has normal distribution $\mathcal{N}(np, \sqrt{np(1-p)})$. With this result, we can build our detection test very easily. Indeed, the probability p is different with respect to hypotheses H_0 and H_1.

- Under H_0 (ciphertexts are not parallel), Z has normal distribution $\mathcal{N}(\frac{n}{2}, \frac{\sqrt{n}}{2})$.
- Under H_1 (ciphertexts are parallel) Z has normal distribution $\mathcal{N}(np, \sqrt{p(1-p)})$ with $p > \frac{1}{2}$ (from a practical point of view, for most languages, whatever may be their linguistic group, we even have $p > 0.6$).

Then the testing itself is very simple. We fix a decision threshold (which is theoretically defined by the error probabilities of types I and II) and then:

- whenever $Z < S$ then ciphertext are not parallel;
- otherwise ($Z > S$) ciphertexts are parallel.

From a practical point of view, in all our experiments we notice large peak values for Z whenever ciphertexts are parallel. Consequently, the use of a threshold is not really necessary. Everytime we have a peak value which is significantly higher than $\frac{n}{2}$, we can decide that ciphertexts are parallel indeed.

An additional mathematical property can be considered to get rid of any error. The parallelism binary relation is an equivalence relation. In other words, if we denote by $\mathcal{R}$ the relationship "*to be parallel to*". $\mathcal{R}$ is an equivalence relation since we have for any arbitrary ciphertexts C_i, C_j and C_k:

- $\mathcal{R}$ is reflexive: for all i, j, k we have $C_i \; \mathcal{R} \; C_i$.

– $\mathcal{R}$ is symmetric: for all pair of indices in $\{i, j, k\}$ we have $C_i \mathcal{R} C_j \Rightarrow C_j \mathcal{R} C_i$.
– $\mathcal{R}$ is transitive: if $C_i \mathcal{R} C_j$ and $C_j \mathcal{R} C_k$ then $C_i \mathcal{R} C_k$.

The transitivity property can be used to add consistency to the detection. If we detect C_1 being parallel with C_2 and C_2 being parallel with C_3 we must also have C_1 which is parallel to C_3. Any class of equivalence is in fact a group of parallel ciphertexts.

From an algorithmic point of view, the detection algorithm is very simple: we compare N ciphertexts pairwise and we extract the groups of parallel messages.

Once again it is worth mentionning that we do not care about the underlying cryptosystem which can remain unknown.

4 The Cryptanalysis: Recovering the Plaintexts without Key Recovery

The cryptanalysis technique aims at recovering the plaintexts without any preliminary step of key recovery. Since we are going to recover the plaintexts directly we need a very good model for the target language to validate the plaintext recovery. This model takes the form of a corpus[5]. Without loss of generality we consider the French language but our approach applies to any languages (natural or not[6]).

4.1 Modelling the language - Corpus contruction

We need to build a qualitative and quantitative model of the target language used for the plaintexts to be recovered. Any language can be described by the frequencies of occurence of characters: lower case,

[5] There exist a lot of theoretical approaches to model natural languages. The most relevant approach is based on the Zipf law which states that given a corpus of natural language words sorted according to their frequency, the frequency of any word is inversely proportional to its rank in the frequency table. We are not building our corpus directly according this law but our corpus of n-grams however complies to the Zipf law.

[6] We place ourself in the context of language in the Chomsky sense [3]. Natural languages are just a particular case in the Chomsky classification of languages (class 1). Other languages like those defined by processor opcodes (class 2) can be indifferently considered.

upper cases, punctuation signs, numbers.... To illustrate this, Table 2 gives the frequencies of the 26 letters (French language with no distinction between lower and upper cases). More generally, it is pos-

Lettre	Fréquence	Lettre	Fréquence
a	8,40	n	7,13
b	1,06	o	5,26
c	3,03	p	3,01
d	4,18	q	0,99
e	17,26	r	6,55
f	1,12	s	8,08
g	1,27	t	7,07
h	0,92	u	5,74
i	7,34	v	1,32
j	0,31	w	0,04
k	0,05	x	0,45
l	6,01	y	0,30
m	2,96	z	0,12

Fig. 2. Frequencies of occurence of French letters (from a 100,000-character XXth century novel)

sible to consider n-grams of characters (i.e. strings of n characters). n-grams provide more information than simple characters. Table 3 gives the frequencies of occurence for 3-grams in French texts. The

3-grammes	Fréquence	3-grammes	Fréquence
ENT	0,90	ELA	0,44
LES	0,80	RES	0,43
EDE	0,63	MEN	0,42
DES	0,61	ESE	0,42
QUE	0,60	DEL	0,40
AIT	0,54	ANT	0,40
LLE	0,51	TIO	0,38
SDE	0,51	PAR	0,36
ION	0,48	ESD	0,35
EME	0,47	TDE	0,35

Fig. 3. Frequencies of occurence of 3-grams in French language (from a 100,000-character XXth century novel)

set of all possible n-grams given with their respective frequency of

occurence is called a "*corpus*". We define then a discrete random variable X describing any n-gram value. We denote p_i the probability that X takes the value x_i: $P(X = x_i) = p_i$ with $i \in \{0, 1 \ldots, N\}$. N is the size of the corpus. The distribution law of X is entirely determined by probabilities p_i of events $\{\{X = x_i\}\}$ where the x_i are the different n-grams in the corpus.

To fit our cryptanalysis purposes, the corpus must meet critical constraints:

- it must be representative enough of the target language;
- it must have a suitable size to allow fast processing.

In order to avoid tedious statistical aspects, we will not expose the stastitical validation of the corpus as an efficient model. All details can be found in [2]. Let us however summarize the main constraints and features to manage:

- **Corpus representativity**.- To build a suitable corpus, we must gather a set of representative texts from which n-grams frequencies are extracted. Evaluating the quality of this corpus is a critical issue. The selected texts must first represent a statistically significant enough amount of characters. However we must also take into account the level of the target language. Our experiments and a large number of statistical analyses clearly show that to have a representative enough sample (i.e *the corpus*) of the target language (*the target population*), the most efficient approach consists in working with several corpus for the same target language to model all level of languages (common language, technical language, diplomatic or political language. . .) very precisely. Then the operational context generally dictates the choice of the corpus to use for the cryptanalysis.
- **Character space**.- Another very important criterion concerns the number of different characters to take into account. The aim is both to reduce it as much as possible (to limit the resources required) and to describe in the best way the target language, at the same time. The case of French language is a critical due to the accented characters (as for similar languages like Turkish or northern European languages). For Asian or Arabic languages, the approach is the same but with a different encoding. In the case

of the French language in ASCII encoding, we chose an alphabet of 96 characters (Table 4).

a	b	c	d	e	f	g	h	i	j	k	l	m
n	o	p	q	r	s	t	u	v	x	x	y	z
A	B	C	D	E	F	G	H	I	J	K	L	M
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	1	2	3	4	5	6	7	8	9	.	,	;
:	?	!	«	(	)	{	}	+	-	*	/	=
‘	à	â	ç	è	é	ê	î	ô	ù	espace		

Fig. 4. Working character space (French language)

– **Size of the corpus**.- It is obviously related to the corpus representativity. But here we also consider the computer resources (memory and time) to involve while keeping the model as accurate as possible. The best tradeoff is to consider $n = 4$ (tetragrams). To manage such a corpus, we use hash table (96 characters gives 96^4 different tetragrams thus requiring 1 Gb of memory). Considering tetragrams instead of trigrams greatly improves all our experimental results while it does not when considering pentagrams ($n = 5$) [2].

4.2 The cryptanalysis: general algorithm

Let be $C_1, C_2 \ldots, C_i \ldots, C_p$ p parallel ciphertexts to decrypt. Let us consider a corpus of N n-grams (typically $n = 4$). Let us denote those n-grams by $x_0 \ldots x_N$. We split the ciphertexts into a succession of n-grams (according to the cryptanalysis mode we choose; refer to Section 4.3).

Then the main steps of our cryptanalysis algorithm are:

1. for each ciphertext n-gram C_1^j in the first ciphertext C_1, we make an assumption on the corresponding plaintext n-gram denoted M_1^j. This n-gram M_1^j is exhaustively searched through the set $\{x_0 \ldots, x_N\}$ of n-grams in the working corpus;
2. we compute the resulting key n-gram as follows: $K_j = C_1^j \oplus M_1^j$;
3. we apply K_j to each of the corresponding ciphertext n-grams in the $(p-1)$ remaining ciphertexts: $M_i^j = C_i^j \oplus K_j,\ i \in \{2, p\}$;

4. we exhaustively repeat the previous steps for every n-grams in the corpus.

This algorithm in fact computes N p-tuples $(M_1^j, M_2^j \ldots, M_p^j)$ for each ciphertext n-gram at index j. Each such p-tuple represents plaintext n-gram candidates for plaintext messages $(M_1, M_2 \ldots, M_p)$ at index j. We just have to determine which is the most probable one (the best candidate). For that purpose, we associate to each of the N p-tuples of n-grams, the corresponding p-tuple of probabilities $(P[M_1^j], P[M_2^j] \ldots, P[M_p^j])$.

The most probable plaintext n-grams p-tuple is the one which maximizes the p-tuples of probabilities. Then the issue is to choose a suitable function to compute those probabilities in the most significant way:

$$Z_j = f(P[M_1^j], P[M_2^j] \ldots, P[M_p^j])$$

The choice of this function strongly depends on the nature of the texts (presence of a many proper or geographical names, technical terms...). This is precisely where all the ability and the experience of the cryptanalyst express. The function must always be a strictly increasing positive function.

We can now give the cryptanalysis algorithm (pseudo-code) in Table 1.

From a complexity point of view, it is easy to verify that the cryptanalysis algorithm has complexity in $\mathcal{O}(pM)$ where M is the size (in bytes) of the ciphertexts. Recovering the plaintext has thus a polynomial complexity.

4.3 Critical parameters and optimizations

Algorithm 1 is a generic framework which can be tuned up with a lot of optimizations and parameters. It all depends on the target plaintexts and on the level of efficiency required. The two main aspects are the parallelism depth and the target language as well as its encoding (refer to the footnote in Section 3.1).

We are going to address the most significant parameters and optimization briefly. All details are available in [2, 6].

Algorithm 1 Parallel ciphertexts cryptanalysis

Require: p parallel ciphertexts $C_1, C_2 \dots C_p$
Require: A N n-grams corpus $\{x_0, x_1 \dots x_N\}$ of respective probabilities $\{P[x_0], P[x_1] \dots P[x_N]\}$.
Ensure: p plaintexts $M_1, M_2 \dots M_p$.

```
for all ciphertext n-gram C_1^j at index j in C_1 do
  Z_j = 0
  for all m_1^j ∈ {x_0, x_1 ... x_N} assume that M_1^j = m_1^j do
    Compute K_j = C_1^j ⊕ m_1^j
    for i ∈ {2, ..., p} do
      Compute m_i^j = C_i^j ⊕ K_j
      Store P[m_i^j]
    end for
    if f(P[m_1^j], P[m_2^j] ..., P[m_p^j]) > Z_j then
      Z_j = f(P[m_1^j], P[m_2^j] ..., P[m_p^j])
      for i ∈ {2, ..., p} do
        M_1^j = m_1^j
      end for
    end if
  end for
end for
```

– **Frequency cumulative function**.- We must consider a strictly increasing positive function. Among the different functions available [6], the most efficient ones are:
  - *the additive function* given by

$$\sum_{i=0}^{p} f_i^a.$$

  - *the multiplicative function* given by

$$\prod_{i=1}^{p} (f_i^a + 1).$$

  where f_i denotes the frequency of occurence of n-gram i in the corpus. We choose this function according to the level of language [6]. For example, for plaintexts containing a lot of proper names or technical terms that are not very frequent, the multiplicative function is far more efficient. The optimal value for a is $a = 0.3$.
– **n-gram processing mode**.- Two different modes can be used: overlapping or non overlapping mode. The first mode consists in decomposing ciphertexts in non overlapping consecutive n-grams.

In other words, two any consecutive n-grams have a void intersection. Consequently, each n-gram is processed independently from the others (Figure 5). From a programming point of view,

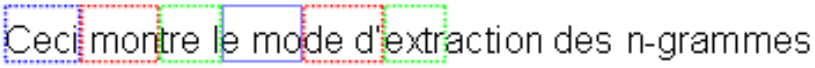


Fig. 5. Non overlapping processing mode for n-grams

this mode is very easy to implement. However it is not the most efficient one: if we keep a wrong plaintext n-gram candidate, no error correction is possible.
The second mode, on the contrary consists in processing n-grams by shifting them of one character position. In other words to consecutive n-grams share $n-1$ characters (Figure 6). The over-

Fig. 6. Overlapping processing mode for n-grams

lapping of n-grams enables to make a consistency check on plaintext n-grams candidates. The $n-1$ common characters must be present in the two best n-gram candidates (at index j and index $j+1$). While this technique is extremely efficient, it is also a little bit more complex at the implementation level. For this mode, a large number of optimizations are also available [2, 6].

– *Decoding techniques*.- These techniques come from the error-correcting theory. They are mainly based on the concept of maximum likelyhood measure. Instead of keeping only the suppos-

edly best plaintext n-gram candidate for every ciphertext n-gram (hard decoding), it is far more efficient yet complex to implement, to keep the b best candidates, for some arbitrary value of b (soft decoding). This enables to use backtraking method and hence to correct any previous wrong decision. From an algorithmic point of view, we use the data structure of decoding lattice. The plaintext recovery then is technically equivalent of searching for the optimal path in this lattice (it is directly inspired from the Viterbi decoding for convolutional codes).

– **Impossible character management**.- According to the working space character, it is also possible not only to speed up the plaintext recovery but also to prevent many wrong decisions and thus to increase the final success greatly. As an example, if the target plaintexts are written in common language, only printable characters are eligible. Most the plaintext n-grams candidate, however are likely to produce other plaintext n-grams containing non printable characters.

5 Application to the Microsoft Word Encryption

Now we have a cryptanalysis algorithm, let us prove its operational efficiency on a real example. We chose to consider the Microsoft Office Encryption.

5.1 The Microsoft Word Encryption

The Microsoft Word encryption requires a password from the user. To activate it, the user has to go through the menu `Tools` → `Options` → `Security tab` → `Advanced options`. Then we have to select the encryption mode (Figure 7). It is worth mentionning that if the user does not select the `Advanced tab` (what in fact most users do) then the lamest encryption method is by default selected (the constant XOR).

Three main categories of encryption are available.

5.1.1 Constant XOR

This data protection deals with encryption with respect to Office 4.x and is still present for backward compatibility (in both Word and

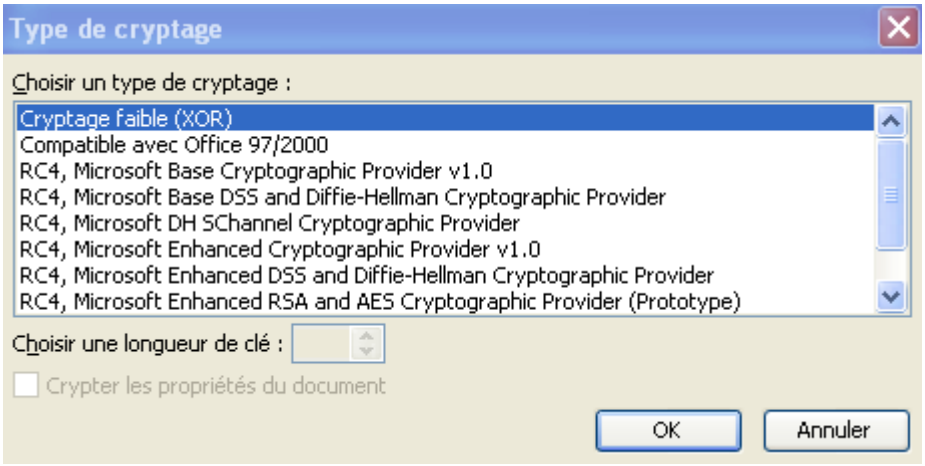


Fig. 7. Available Microsoft Word Encryption mode

Excel). This encryption method is simple and fast but very weak in terms of security. It consists in xoring the plaintext with a constant 16-character string, which has been derived from the password. This string is repeated many times to cover all the plaintext.

While the technique we have presented in this paper fully applies, it is even more efficient to use dedicated software (e.g. *Advanced Office Password Recovery* to break the XOR encryption. It is the default encryption!

5.1.2 Microsoft Office 97 / Office 2000 compatible encryption

Here the encryption is based on a proprietary Office encryption and is derived from the Microsoft Internet Explorer CryptoAPI method. It is mainly present for compatibility purposes.

5.1.3 RC4 stream cipher

Several encryption services based on RC4 are proposed. From an historical perspective, they correspond to the different export regulations to non US countries. The last service, which is denoted "*RC4 Microsoft Enhanced Cryptographic Provider*" is supposed to provide the strongest security by means of 128-bit keys. It is precisely the target of our attack.

RC4 is a finite state machine stream cipher: the plaintext is bitwise xored to a pseudo-random running key. The initial secret key is generated by the application from the user's password and from an initialization vector (IV) which is randomly produced by Word. The password and the IV are concatenated then hashed:

$$K = F(H(IV||\text{password}))$$

where F is a derivation function outputing 128-bit values. The hash function H is most of the time SHA-1. We can notice that the key does not depend on the user's password only, which is a necessary (but not sufficient) condition to have strong encryption. In this respect, even if the user always takes the same password, whatever may be the document to protect, the random IV is supposed to prevent the use of a key twice (or more). In this respect, the IV plays the same role as a session key or a message key.

5.2 Highlighting the Office Vulnerability

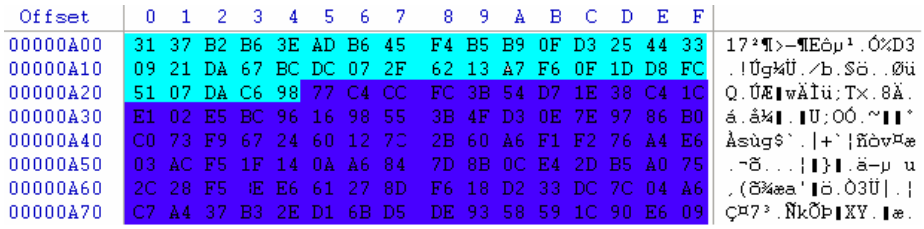
The flaw – first identified in [12] but only in a theoretical way – lies on the fact that Microsoft Office violates a fundamental rule in cryptogrphic implementation: the initialization vector remains the same for every revised version of a given Word document. Since most of the times, the user keeps using the same password then the secret key is the same as well. The set of revised versions of a document constitutes a parallel group of ciphertexts.

5.2.1 Technical analysis

Let us first create a Word document containing the following text: "*Ceci est un essai de construction de messages parallèles afin de montrer la vulnérabilité du chiffrement de Microsoft Word*". Then we operates as follows:

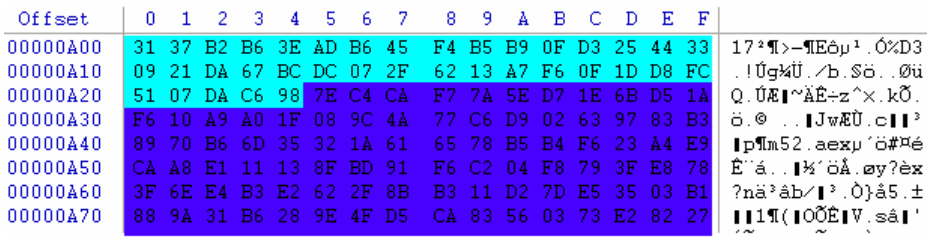
- We encrypt the document and save it (file "`message1`").
- We then modify the document as follows: "*Ceci est un essai de construction de deux messages parallèles afin de montrer la vulnérabilité du chiffrement de Microsoft Word.*". We also encrypt and save it with the same password (file "`message2`") as most users do.

Figures 8 and 9 show the hexadecimal view of encrypted files `message1` and `message2`.



Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00000A00	31	37	B2	B6	3E	AD	B6	45	F4	B5	B9	0F	D3	25	44	33
00000A10	09	21	DA	67	BC	DC	07	2F	62	13	A7	F6	0F	1D	D8	FC
00000A20	51	07	DA	C6	98	77	C4	CC	FC	3B	54	D7	1E	38	C4	1C
00000A30	E1	02	E5	BC	96	16	98	55	3B	4F	D3	0E	7E	97	86	B0
00000A40	C0	73	F9	67	24	60	12	7C	2B	60	A6	F1	F2	76	A4	E6
00000A50	03	AC	F5	1F	14	0A	A6	84	7D	8B	0C	E4	2D	B5	A0	75
00000A60	2C	28	F5	8E	E6	61	27	8D	F6	18	D2	33	DC	7C	04	A6
00000A70	C7	A4	37	B3	2E	D1	6B	D5	DE	93	58	59	1C	90	E6	09

Fig. 8. Hexadecimal view of `message1`



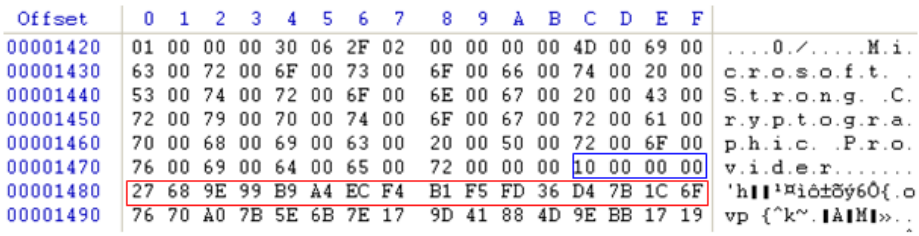
Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00000A00	31	37	B2	B6	3E	AD	B6	45	F4	B5	B9	0F	D3	25	44	33
00000A10	09	21	DA	67	BC	DC	07	2F	62	13	A7	F6	0F	1D	D8	FC
00000A20	51	07	DA	C6	98	7E	C4	CA	F7	7A	5E	D7	1E	6B	D5	1A
00000A30	F6	10	A9	A0	1F	08	9C	4A	77	C6	D9	02	63	97	83	B3
00000A40	89	70	B6	6D	35	32	1A	61	65	78	B5	B4	F6	23	A4	E9
00000A50	CA	A8	E1	11	13	8F	BD	91	F6	C2	04	F8	79	3F	E8	78
00000A60	3F	6E	E4	B3	E2	62	2F	8B	B3	11	D2	7D	E5	35	03	B1
00000A70	88	9A	31	B6	28	9E	4F	D5	CA	83	56	03	73	E2	82	27

Fig. 9. Hexadecimal view of `message2`

We easily notice that the 37 first bytes are the same. This is a clear proof of the fact that the same RC4 running sequence has been used and hence the IV remains unchanged. In an equivalent way we can compare the IVs which are located right after the `10 00 00 00` (Figures 10 and 11).

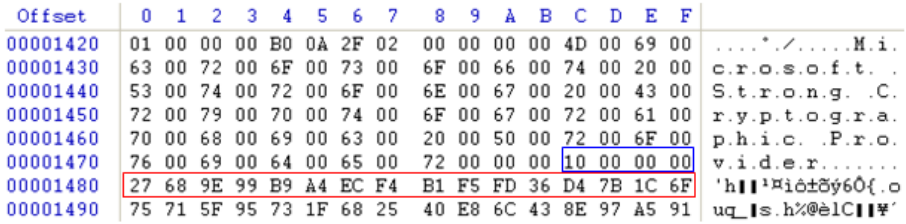
From a cryptanalytic point of view it is essential that the common part of the encrypted texts (before the first revision location; in light blue on Figures 8 and 9) is a short as possible. It is almost always the case: the first revision generally occurs with the date change which located at the beginning of the document.

From all this, we can conclude that **the Microsoft Office vulnerability is not related to the weakness of the underlying**



Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00001420	01	00	00	00	30	06	2F	02	00	00	00	00	4D	00	69	00	....0./.....M.i.
00001430	63	00	72	00	6F	00	73	00	6F	00	66	00	74	00	20	00	c.r.o.s.o.f.t. .
00001440	53	00	74	00	72	00	6F	00	6E	00	67	00	20	00	43	00	S.t.r.o.n.g. .C.
00001450	72	00	79	00	70	00	74	00	6F	00	67	00	72	00	61	00	r.y.p.t.o.g.r.a.
00001460	70	00	68	00	69	00	63	00	20	00	50	00	72	00	6F	00	p.h.i.c. .P.r.o.
00001470	76	00	69	00	64	00	65	00	72	00	00	00	10	00	00	00	v.i.d.e.r.....
00001480	27	68	9E	99	B9	A4	EC	F4	B1	F5	FD	36	D4	7B	1C	6F	'h▮▮¹¤ìô±õý6Ô{.o
00001490	76	70	A0	7B	5E	6B	7E	17	9D	41	88	4D	9E	BB	17	19	vp {^k~.▮A▮M▮»..▮

Fig. 10. Initialization vector for `message1`



Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00001420	01	00	00	00	B0	0A	2F	02	00	00	00	00	4D	00	69	00	....°./.....M.i.
00001430	63	00	72	00	6F	00	73	00	6F	00	66	00	74	00	20	00	c.r.o.s.o.f.t. .
00001440	53	00	74	00	72	00	6F	00	6E	00	67	00	20	00	43	00	S.t.r.o.n.g. .C.
00001450	72	00	79	00	70	00	74	00	6F	00	67	00	72	00	61	00	r.y.p.t.o.g.r.a.
00001460	70	00	68	00	69	00	63	00	20	00	50	00	72	00	6F	00	p.h.i.c. .P.r.o.
00001470	76	00	69	00	64	00	65	00	72	00	00	00	10	00	00	00	v.i.d.e.r.....
00001480	27	68	9E	99	B9	A4	EC	F4	B1	F5	FD	36	D4	7B	1C	6F	'h▮▮¹¤ìô±õý6Ô{.o
00001490	75	71	5F	95	73	1F	68	25	40	E8	6C	43	8E	97	A5	91	uq_▮s.h%@èlC▮▮¥'

Fig. 11. Initialization vector for `message2`

encryption algorithm (RC4) but to its implementation in Microsoft Office.

5.3 Microsoft Word cryptanalysis

5.3.1 Word document internals analysis

In order to perform the detection and the cryptanalysis we must precisely locate the encrypted data inside the metadata.

- In Microsoft Word, the beginning of the text is always located at offset `0xA00`.
- The size of the text (encrypted data bytes) can be computed from the two values x and y located at offsets `0x21D` and `0x21C` respectively. The size T (in bytes) is then given by the following formula:

$$T = (x - 8) \times 2^8 + y.$$

Those data are never encrypted (even if the user chooses the additional features "*document properties encryption*"!

5.3.2 Parallel encrypted documents detection

The critical aspect of our cryptanalysis technique lies on the fact that we must have a parallel depth strictly greater than one[7]. When dealing with Microsoft Word (or Excel), this apparent limitation is easy to bypass. We just have to use temporary files.

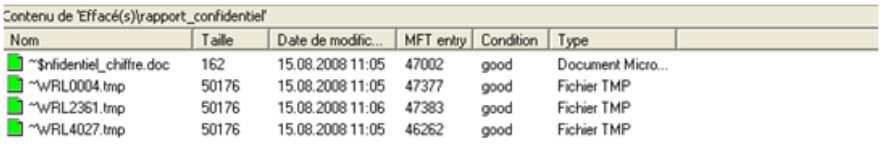
When we create or edit (for revision) a Microsoft Office document, temporaty files are created. Those files contains the previous versions of the documents (before saving the modifications). Those files are supposed to be erased whenever Word is closed... but in an insecure way as usual under Windows. Consequently, it is very easy to recover all those files by using a dedicated software (e.g. *Easy recovery Pro*, *Photorec* or *PC inspector file recovery*). It is worth mentioning that similarly this recovery step could be performed by a Trojan horse or a malicious USB key (*i.e* containing a spying malware), thus enabling to outsource the attack.

As an illustration, we have created a RC4-protected Word document denoted "`confidentiel_chiffré`". Then we have modified this document three times. We can see the corresponding temporary files (`WRL004, WRL2361` and `WRL4027`) on Figure 12. After a recovery

Fig. 12. Word temporary files

step we have an effective parallel depth of 4 (Figure 13).

[7] The case of a depth equal to 2 is not addressed here. While it relies on the same general technique, it is more tricky to manage due to some sort of plaintext rotation effect. To prevent this effect, we must perform an additional semantic validation step (especially by considering that any language, at a formal level, is a Markov process. More details can be found in [6].


Contenu de 'Effacé(s)\rapport_confidentiel'

Nom	Taille	Date de modific...	MFT entry	Condition	Type
~$nfidentiel_chiffre.doc	162	15.08.2008 11:05	47002	good	Document Micro...
~WRL0004.tmp	50176	15.08.2008 11:05	47377	good	Fichier TMP
~WRL2361.tmp	50176	15.08.2008 11:06	47383	good	Fichier TMP
~WRL4027.tmp	50176	15.08.2008 11:05	46262	good	Fichier TMP

Fig. 13. Word temporary files recovered

To validate the detection technique, we have peformed a large number of experiments for almost all main languages. Every time, the detection is successful.

As an example, we have built a set of twenty encrypted 1500-character Word documents. For five of them, we have used the same password (documents #1 to #5). The detection results are presented in Figure 14. We clearly observe statitical peaks for files #1 to #5 and the transitivity property is indeed verified.

5.3.3 Cryptanalysis experimental results

Let us now illustrate the cryptanalysis step. As an example, let us consider three test cases.

- **Test1**.- We have considered extracts from Jules Verne novels. Five parallel encrypted texts, each of them of 1,200 bytes.
- **Test2**.- We have considered the speech of the Chief of Staff of the Army before the Foreign Affairs Committee of the National Assembly in 2008 (high level technical, diplomatic language with a significant number of proper names). Five 1,500-byte parallel documents have been created from five extracts of this speech.
- **Test3**.- We have considered a speech of the President of French Republic. Five parallel texts have been created. Each text was 9700 bytes long.

Then we have applied a moderately optimized cryptanalysis algorithm (multiplicative function with $a = 0.3$, overlapping mode, printable character feature enabled, hard decoding without semantic step). Table 1 summarizes the results. If we consider soft decoding and semantic analysis, the rate of successful cryptanalysis is close to 100 % for a parallel depth stricly greater than 2 (and 75 % for two texts

```
Première colonne   : fichiers comparés
Deuxième colonne   : nombre de bits à « 1 »
Troisième colonne  : proportion de bits à « 0 »

z[1-2]  6081 0.658     z[3-15] 4677 0.506     z[6-18]  4611 0.499     z[10-20] 4629 0.501
z[1-3]  6110 0.662     z[3-16] 4604 0.498     z[6-19]  4586 0.496     z[11-12] 4608 0.499
z[1-4]  6141 0.665     z[3-17] 4692 0.508     z[6-20]  4660 0.504     z[11-13] 4670 0.506
z[1-5]  6148 0.666     z[3-18] 4606 0.499     z[7-8]   4647 0.503     z[11-14] 4582 0.496
z[1-6]  4695 0.508     z[3-19] 4605 0.499     z[7-9]   4657 0.504     z[11-15] 4573 0.495
z[1-7]  4636 0.502     z[3-20] 4627 0.501     z[7-10]  4580 0.496     z[11-16] 4582 0.496
z[1-8]  4607 0.499     z[4-5]  6113 0.662     z[7-11]  4594 0.497     z[11-17] 4584 0.496
z[1-9]  4545 0.492     z[4-6]  4634 0.502     z[7-12]  4668 0.505     z[11-18] 4642 0.503
z[1-10] 4638 0.502     z[4-7]  4585 0.496     z[7-13]  4626 0.501     z[11-19] 4591 0.497
z[1-11] 4652 0.504     z[4-8]  4626 0.501     z[7-14]  4550 0.493     z[11-20] 4593 0.497
z[1-12] 4560 0.494     z[4-9]  4622 0.500     z[7-15]  4667 0.505     z[12-13] 4548 0.492
z[1-13] 4682 0.507     z[4-10] 4621 0.500     z[7-16]  4548 0.492     z[12-14] 4592 0.497
z[1-14] 4634 0.502     z[4-11] 4703 0.509     z[7-17]  4642 0.503     z[12-15] 4591 0.497
z[1-15] 4653 0.504     z[4-12] 4627 0.501     z[7-18]  4562 0.494     z[12-16] 4614 0.500
z[1-16] 4578 0.496     z[4-13] 4629 0.501     z[7-19]  4625 0.501     z[12-17] 4574 0.495
z[1-17] 4642 0.503     z[4-14] 4565 0.494     z[7-20]  4629 0.501     z[12-18] 4508 0.488
z[1-18] 4606 0.499     z[4-15] 4664 0.505     z[8-9]   4514 0.489     z[12-19] 4549 0.492
z[1-19] 4601 0.498     z[4-16] 4611 0.499     z[8-10]  4531 0.491     z[12-20] 4619 0.500
z[1-20] 4639 0.502     z[4-17] 4655 0.504     z[8-11]  4617 0.500     z[13-14] 4598 0.498
z[2-3]  6125 0.663     z[4-18] 4537 0.491     z[8-12]  4661 0.505     z[13-15] 4677 0.506
z[2-4]  6126 0.663     z[4-19] 4590 0.497     z[8-13]  4589 0.497     z[13-16] 4646 0.503
z[2-5]  6099 0.660     z[4-20] 4592 0.497     z[8-14]  4709 0.510     z[13-17] 4622 0.500
z[2-6]  4590 0.497     z[5-6]  4625 0.501     z[8-15]  4530 0.490     z[13-18] 4648 0.503
z[2-7]  4633 0.502     z[5-7]  4596 0.498     z[8-16]  4661 0.505     z[13-19] 4655 0.504
z[2-8]  4622 0.500     z[5-8]  4585 0.496     z[8-17]  4703 0.509     z[13-20] 4655 0.504
z[2-9]  4550 0.493     z[5-9]  4521 0.489     z[8-18]  4585 0.496     z[14-15] 4587 0.497
z[2-10] 4651 0.504     z[5-10] 4658 0.504     z[8-19]  4642 0.503     z[14-16] 4562 0.494
z[2-11] 4633 0.502     z[5-11] 4638 0.502     z[8-20]  4694 0.508     z[14-17] 4642 0.503
z[2-12] 4549 0.492     z[5-12] 4540 0.491     z[9-10]  4549 0.492     z[14-18] 4534 0.491
z[2-13] 4643 0.503     z[5-13] 4650 0.503     z[9-11]  4595 0.497     z[14-19] 4651 0.504
z[2-14] 4613 0.499     z[5-14] 4594 0.497     z[9-12]  4593 0.497     z[14-20] 4577 0.495
z[2-15] 4618 0.500     z[5-15] 4633 0.502     z[9-13]  4519 0.489     z[15-16] 4521 0.489
z[2-16] 4651 0.504     z[5-16] 4638 0.502     z[9-14]  4565 0.494     z[15-17] 4613 0.499
z[2-17] 4595 0.497     z[5-17] 4598 0.498     z[9-15]  4660 0.504     z[15-18] 4615 0.500
z[2-18] 4627 0.501     z[5-18] 4500 0.487     z[9-16]  4599 0.498     z[15-19] 4542 0.492
z[2-19] 4708 0.510     z[5-19] 4585 0.496     z[9-17]  4563 0.494     z[15-20] 4728 0.512
z[2-20] 4576 0.495     z[5-20] 4611 0.499     z[9-18]  4627 0.501     z[16-17] 4548 0.492
z[3-4]  6087 0.659     z[6-7]  4649 0.503     z[9-19]  4632 0.501     z[16-18] 4668 0.505
z[3-5]  6150 0.666     z[6-8]  4560 0.494     z[9-20]  4612 0.499     z[16-19] 4645 0.503
z[3-6]  4629 0.501     z[6-9]  4620 0.500     z[10-11] 4594 0.497     z[16-20] 4635 0.502
z[3-7]  4570 0.495     z[6-10] 4571 0.495     z[10-12] 4498 0.487     z[17-18] 4626 0.501
z[3-8]  4583 0.496     z[6-11] 4683 0.507     z[10-13] 4632 0.501     z[17-19] 4579 0.496
z[3-9]  4561 0.494     z[6-12] 4643 0.503     z[10-14] 4604 0.498     z[17-20] 4635 0.502
z[3-10] 4626 0.501     z[6-13] 4661 0.505     z[10-15] 4595 0.497     z[18-19] 4731 0.512
z[3-11] 4644 0.503     z[6-14] 4651 0.504     z[10-16] 4636 0.502     z[18-20] 4663 0.505
z[3-12] 4510 0.488     z[6-15] 4752 0.514     z[10-17] 4596 0.498     z[19-20] 4524 0.490
z[3-13] 4678 0.506     z[6-16] 4573 0.495     z[10-18] 4580 0.496
z[3-14] 4578 0.496     z[6-17] 4547 0.492     z[10-19] 4579 0.496
```

Fig. 14. Detection results for parallel ciphertexts

Parallel depth	Decryption rate		
	Test1	Test2	Test3
2	40.07 %	40.66 %	39.80 %
3	88.29 %	82.40 %	89.78 %
4	92.71 %	90.81 %	91.87 %
5	93.76 %	91.71 %	93.12 %

Table 1. Cryptanalysis results on the three test sets

only; in this particular case, a final linguist-driven analysis will succeed in recovering the missing characters, most of the times).

6 Application to the Microsoft Excel Encryption

The Microsoft Excel case is more tricky to address than the Microsoft Word case, due to the nature and structure of Excel files. Nevertheless our technique applies in the same way and with an equal efficiency. We present in this section the main differences compared to the Microsoft Word case. Technical details are available in [2].

6.1 Excel internals and highlighting the flaw

Microsoft Excel does not locate spreadsheet data at a fixed location (offset). The reader can easily check this point by saving an arbitrary spreadsheet under two different names.

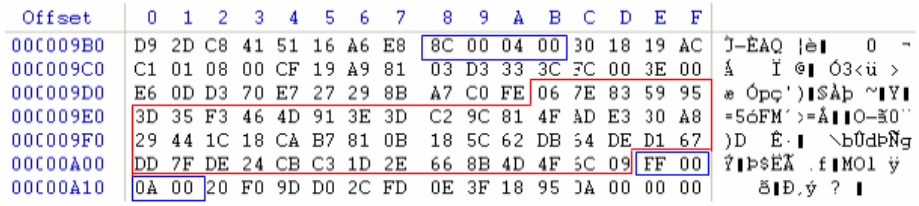
The second tricky features of Excel lies in the fact that data modifications are managed in a very different way. Indeed, whenever a user modifies a spreadsheet cell, then the new cell content is located at the end of the data, not at the cell location.

Those two special features have a significant impact on the detection and cryptanalysis of parallel Excel encrypted messages.

In [12], Hongjun Wu mentions the fact that Excel data are beginning 31 bytes right after the hexadecimal string `0x8c000400` and are ending right before the hexadecimal string `0xff001200`. In fact he was partially wrong: if indeed data offset is after the `0x8c000400` marker, the end marker depends on the number of spreadsheet cells

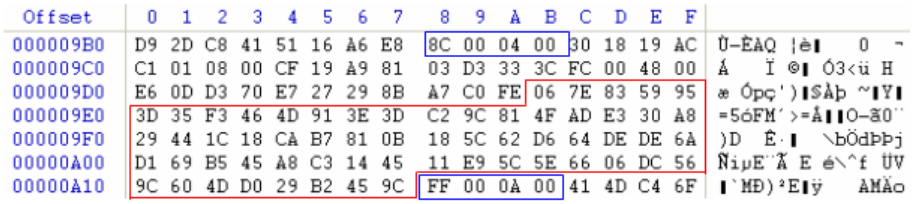
that have been used. We have successively `0xff000a00, 0xff001200, 0xff001a00, 0xff002200`.... In fact the third byte of the marker is incremented by 8 everytime: $0a + 08 = 12, 12 + 08 = 1a$.... All this enables us to locate precisely the user's data inside the spreadsheet.

Let us now highlight the Office vulnerability with respect to Excel. As for Microsoft Word, we have enciphered a spreadsheet (Figure 15), then modified it and enciphered it under a different name (Figure 16).



Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00C009B0	D9	2D	C8	41	51	16	A6	E8	8C	00	04	00	30	18	19	AC
00C009C0	C1	01	08	00	CF	19	A9	81	03	D3	33	3C	FC	00	3E	00
00C009D0	E6	0D	D3	70	E7	27	29	8B	A7	C0	FE	06	7E	83	59	95
00C009E0	3D	35	F3	46	4D	91	3E	3D	C2	9C	81	4F	AD	E3	30	A8
00C009F0	29	44	1C	18	CA	B7	81	0B	18	5C	62	DB	64	DE	D1	67
00C00A00	DD	7F	DE	24	CB	C3	1D	2E	66	8B	4D	4F	6C	09	FF	00
00C00A10	0A	00	20	F0	9D	D0	2C	FD	0E	3F	18	95	1A	00	00	00

Fig. 15. Hexadecimal version of the enciphered original spreadsheet (in red frame, the enciphered data; in blue frame, the beginning and end markers)



Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000009B0	D9	2D	C8	41	51	16	A6	E8	8C	00	04	00	30	18	19	AC
000009C0	C1	01	08	00	CF	19	A9	81	03	D3	33	3C	FC	00	48	00
000009D0	E6	0D	D3	70	E7	27	29	8B	A7	C0	FE	06	7E	83	59	95
000009E0	3D	35	F3	46	4D	91	3E	3D	C2	9C	81	4F	AD	E3	30	A8
000009F0	29	44	1C	18	CA	B7	81	0B	18	5C	62	D6	64	DE	DE	6A
00000A00	D1	69	B5	45	A8	C3	14	45	11	E9	5C	5E	66	06	DC	56
00000A10	9C	60	4D	D0	29	B2	45	9C	FF	00	0A	00	41	4D	C4	6F

Fig. 16. Hexadecimal version of the enciphered modified spreadsheet

We indeed notice in Figure 15 and 16 that the 32 first bytes of data are the same in both versions of the spreadsheet (common part of the unmodified data). Thus we have a clear proof of the vulnerability (reuse of the pseudo-running key).

6.2 Excel parallel messages detection

The detection is as easy as for the Microsoft Word case. The algorithm remains the same. Figure 17 exhibits the same statistical peaks that enables to identify clearly every parallel group. Let us

```
z[1-2] 1728 0.651584
z[1-3] 1372 0.500730
z[1-4] 1347 0.511002
z[1-5] 1091 0.507914
z[1-6] 952 0.501053
z[2-3] 1358 0.512066
z[2-4] 1332 0.505311
z[2-5] 1028 0.478585
z[2-6] 974 0.512632
z[3-4] 1322 0.501517
z[3-5] 1083 0.504190
z[3-6] 947 0.498421
z[4-5] 1048 0.487896
z[4-6] 927 0.487895
z[5-6] 929 0.488947
```

Fig. 17. Detection results for Excel

now address the issue of Excel parallel documents cryptanalysis. Basically the technique is globally the same. However, there are also fundamental differences and constraints to take into account for a successful cryptanalysis.

– Data are located between cell separators. These separators are made of groups of three bytes: `XX 00 00` where `XX` is the size of the data inside the next cell (see Figure 18). From a cryptanalysis

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
000008F0	FC	00	6A	00	08	00	00	00	08	00	00	00	0A	00	00	63	ü j c
00000900	6F	6C	6F	6E	6E	65	20	32	20	0A	00	00	63	6F	6C	6F	olonne 2 colo
00000910	6E	6E	65	20	31	20	07	00	00	6C	69	67	6E	65	20	31	nne 1 ligne 1
00000920	07	00	00	6C	69	67	6E	65	20	32	0A	00	00	64	6F	6E	ligne 2 don
00000930	6E	E9	65	73	20	31	31	0A	00	00	64	6F	6E	6E	E9	65	nées 11 donnée
00000940	73	20	31	32	0A	00	00	64	6F	6E	6E	E9	65	73	20	32	s 12 données 2
00000950	32	0A	00	00	64	6F	6E	6E	E9	65	73	20	32	31	FF	00	2 données 21ÿ

Fig. 18. Excel cell separator (in hexadecimal)

point of view, cell separators are representing probable/known plaintext and thus they enable an efficient plaintext recovery.
– Use of a specific corpus. We have to model very precisely the particular nature of Excel data spreadsheets. These data have the main general features:
  • Almost neither sentences nor semantically structured data (in particular, there are only a very few verbs).
  • Very limited punctuation.
  • Data are essentially numbers and figures.

Whenever those particular features and constraints are taken into account, recovering the Excel plaintext is as efficient as in the Microsoft Word case.

7 Conclusion: What About Trapdoors?

The issue that arises immediately relates to the nature of the Microsoft Office flaw and above all on the technical conditions that make possible its operational exploitation. It is more than surprising that year after year, version after version (both for the Windows operating system and Office), such a weakness still exists.

The Office flaw is in this respect very interesting because it sheds a new light on the way to embed trapdoors very efficiently. As we already noticed it for PDF security [1], part of the security for aplications nowadays lies at the operating system level.

In the context of embedding invisible trapdoors inside a cryptosystem, the issue has many aspects. One very efficient way to do it, is first to introduce a weakness that will be considered most of the time as an unintentional implementation flaw. But this flaw must be designed in such a way that its exploitation depends on users' misuses that generally occur with a very high probability. In the context of cryptography, the very likely misuse is the reuse of the same secret password or secret key to encrypt several different documents. But worse, the implementation weakness can be split in two parts: one part at the application level the other one at the operating system level.

The choice of the encoding is also part of the game. Former satellite encodings concentrate information too much contrary to ASCII

encoding, thus making strong biases appear. On the contrary, while ASCII encoding is relatively well suited for accented languages (like French), it can also exhibit strong biases for non accented languages. So the choice of the encoding should be carefully made relatively to the character space. The future generalization of unicode may have dramatic impact on the security of encryption systems.

In the case of commercial encryption software embedding a block cipher (e.g. AES-256), it would consist in considering OFB mode and flawing the generation of the quantity s_0 (refer to Figure 1).

The technique of cryptanalysis presented in this paper clearly shows once more that when cryptography works on paper, the real security can be very still very far. The implementation can be (intentionally or not) flawed and we need methods to detect it without performing a time-consuming, complex reverse-engineering step.

Another interesting point lies in the fact that sometimes recovering the key is not always an essentiel part of the security, as well as the intrinsic mathematical security of the cryptosystem. And the knowledge about the latter is not always a necessary cryptanalysis condition. We have implemented this attack against a misused Vernam cryptosystem – which is theoritically supposed to offer perfect security – very efficiently. Beside the implementation flaw, we must remind that users' wrong use of cryptography sometimes can nullify the overall security.

References

1. Alexandre Blonce, Eric Filiol and Laurent Frayssignes (2008). *PDF Security Analysis and Malware Threats*. Black Hat Europe 2008. `http://www.blackhat.com/presentations/bh-europe-08/Filiol/Presentation/bh-eu-08-filiol.pdf`
2. Franck Bonnard et Eric Filiol (2008). *Cryptanalyse du chiffrement de la suite bureautique Microsoft Office*. ESIEA Technical Report (in French). Available on `http://www.esiea-recherche.eu/`
3. Chomsky N. (1956), Three models for the description of languages, *IRE Transactions on Information Theory*, 2, pp. 113–124.
4. Ed Dawson and Lauren Nielsen (1996). Automated cryptanalysis of xor plaintext strings, *Cryptologia*, vol 20, issue 2, pp. 165–181.
5. Yadolah Dodge (1999). *Premiers pas en statistiques*. Springer France Publishing.
6. Eric Filiol. *Cryptanalysis course.* Graduate courses.
7. Eric Filiol. Repetition Codes Cryptanalysis of Block Ciphers. *Journal of the Indian Statistical Association*, 42 (9), 2004.
8. Joshua Mason, Kathryn Watkins, Jason Eisner and Adam Stubblefield (2006). A natural language approach to automated cryptanalysis of two-time pads, *CCS*

'06: Proceedings of the 13th ACM conference on Computer and communications security.

9. Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone (1996). *Handbook of Applied Cryptography*. CRC Press, ISBN 0-8493-8523-7.
10. Claude E. Shannon (1948). A Mathematical Theory of Communication. *Bell System Journal*, Vol. 27 pp. 379-423 (Part I) & pp. 623-656 (Part II).
11. Peter Wright (1987). *Spy Catcher - The Candid Autobiography of a Senior Intelligence Officer*. Dell Publishing.
12. Hongjun Wu (2005). *The misuse of RC4 in Microsoft Word and Excel*. Preprint IACR. `http://eprint.iacr.org/2005/007`