Understanding the key-independent cryptanalysis technique used to break the RC4 based Office Encryption stream cipher

Elias Tatros
Betreuer: Heiko Niedermayer
Seminar Future Internet WS10/11
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik, Technische Universität München
Email: elias.tatros@cs.tum.edu

ABSTRACT

Stream ciphers are still widely used to efficiently encrypt large amounts of data. Contrary to asymmetric encryption, stream ciphers are easy to implement in hardware and offer a high encryption speed. Using information theory, it is possible to proof that an arbitrary level of security can be achieved. However small flaws in the implementation of stream ciphers can lead to severe weaknesses. These can cause even unconditionally secure stream ciphers, like the one-time pad (Vernam system), to become totally insecure. This can be demonstrated in a cryptanalysis of the Microsoft Office encryption up to the 2003 version. Office employs a flawed implementation of the widely used RC4 algorithm, exposing a severe vulnerability, which can be exploited to recover the plaintext from the encrypted documents.

Keywords

stream ciphers, data encryption, code breaking, applied cryptography, RC4

1. INTRODUCTION

Many cryptographic solutions, that are considered secure in a theoretical sense, do not necessarily offer the same security when implemented in a real cryptosystem. It is fairly common that the key size alone serves as the most important security argument. A cryptosystem based on AES-256 might not be secure if its implementation is flawed. It is most probably not secure, if intended implementation backdoors were deliberately introduced into the system. Thus, in practice, an argument like "AES-256 inside" can only be considered a necessary, but not a sufficient condition for a secure solution.

A different problem, that can have a dramatic impact on the security, lies in the misuse of a cryptosystem. When users ignore basic rules for handling the system, then even information theoretically secure cryptosystems can become totally insecure. One prominent example for such a misuse occurred in the 1940s, when US intelligence under the VENONA project was able to break the one-time pad encryption on many high-level soviet diplomatic messages. The successful cryptanalysis of many of those encrypted messages was possible, because the people handling the encryption process for the soviets made mistakes using the cryptosystem. They reused some of the secret keys (called pads, because at that time they were printed on pads of paper) on different messages, thus ignoring a basic rule for handling one-time pad systems stating that a secret key must never be used more than once. When a cryptosystem is labeled (information theoretically) secure its users tend to have high confidence in the system and may therefore be less likely to thoroughly question their own actions when handling it.

From these issues two problems immediately arise. For users of a cryptosystem the problem is to detect implementation weaknesses or trapdoors within the system without reverse engineering. Reverse engineering is not a viable solution to the problem because in general it is too time consuming and quite often also illegal. For an attacker the problem is to identify, among a large number of ciphertexts, those that were generated by a weak or misused cryptosystem and then to recover the corresponding plaintexts in a reasonable amount of time. This needs to be done without knowledge of the cryptographic algorithm. For example, in satellite transmissions it can be assumed that the attacker is able to intercept the ciphertexts but doesn't know the algorithm used for encryption.

2. SYMMETRIC ENCRYPTION BASICS

Whenever the confidentiality of data is of concern encryption must be used. For large amounts of data it is better to use symmetric encryption. Advantages over asymmetric encryption include faster encryption speed and in most cases a high level of error resilience, while still offering an arbitrary level of security. Symmetric encryption solutions can be categorized into stream ciphers and block ciphers.

When using stream ciphers bits (or bytes) are enciphered and deciphered on-the-fly. This enables the cryptosystem to rapidly encrypt large quantities of data. They are also very error resilient since transmission errors do not propagate during the decipherment. For these reasons they are widely used for satellite communications protection, telephony encryption (e.g. A5/1) and Bluetooth encryption (e.g. E0) [1].

Contrary to stream ciphers, bits are not enciphered or deciphered on-the-fly when using block ciphers. Instead data is first split into blocks of a certain size (the standard is 128-bit blocks). Each of these blocks is then enciphered or deciphered using the same secret key. Block ciphers can operate in a number of modes which can provide authentication, enable different encryption properties and limit error propagation. The most common modes are ECB, CBC, CFB, PCBC, OFB [2]. It is worth noting that block ciphers, except in OFB mode, are not naturally transmission error resilient. Block ciphers in output feedback mode (OFB) emulate stream ciphers and thus are susceptible to stream cipher attacks and misuses (e.g. key reuse) [1].

2.1 Stream Ciphers in Detail

A stream cipher bitwise combines a truly random or pseudo-random sequence $(\sigma_t)_{t\geq 0}$ with the plaintext $(m_t)_{t\geq 0}$ using the xor operation which results in the ciphertext $(c_t)_{t\geq 0}$:

$$c_t = m_t \oplus \sigma_t$$

Since the xor operation is involutive, it is sufficient to apply the same random bit σ_t to ciphertext bit c_t for the time instant t and thus recover the deciphered plaintext bit m_t:

$$m_t = c_t \oplus \sigma_t$$

where c_t, m_t and σ_t denote a ciphertext, plaintext and random sequence bit at time instant t respectively. The sequence $(\sigma_t)_{t\geq 0}$ is called the running-key. In the case of Vernam ciphers (one-time pad) the running-key is truly random and independent from plaintext or ciphertext (i.e. produced by hardware methods). The one-time pad encryption has been proven to be information theoretically secure, meaning that the ciphertext C provides no information about the plaintext M to a cryptanalyst.

2.2 Perfect Secrecy

Perfect secrecy, as defined in [3], means that after the interception of a ciphertext the *a posteriori* probabilities of this ciphertext representing various plaintexts be identical to the *a priori* probabilities of the same plaintext before the interception, implying that intercepting ciphertexts doesn't provide the cryptanalyst with any information about the plaintexts. It is possible to achieve perfect secrecy, but it requires that there is at least one key k transforming any plaintext M into any ciphertext C. Obviously any key from a fixed M to a different C must be different (as shown in figure 1), therefore the number of unique keys must be at least as high as the number of plaintexts.

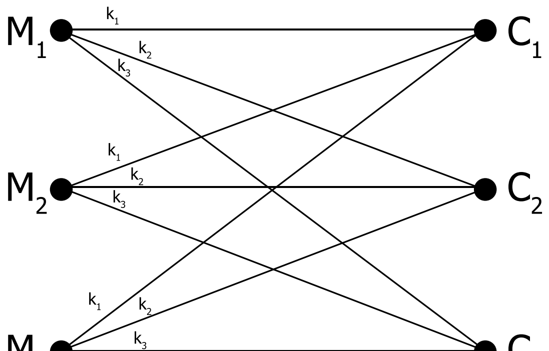


Figure 1: a perfect system as described in [3]

More precisely, if a finite key was used for encryption and a ciphertext consisting of N letters is intercepted, there will be a set of possible plaintexts with certain probabilities that this ciphertext could represent. C. E. Shannon has shown in [3] that as N increases the probabilities for all possible plaintexts except for one will approach zero. It is now possible to define a quantity $H(N)$ that measures by statistical means how near the average ciphertext of N letters is to the unique solution (i.e. how uncertain a cryptanalyst is of the plaintext after intercepting N letters of ciphertext). As shown in [4], this quantity can be measured by means of entropy. For a given set of possibilities with probabilities $p_1, p_2, ..., p_n$ the entropy H is given by:

$$H = -\sum p_i \log p_i$$

The perfect secrecy property can then be expressed as:

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

Intuitively this means that the uncertainty about the plaintext remains constant with each interception of a new ciphertext. The same holds true when applying the entropy function to the key:

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

As previously mentioned, it is necessary to have

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

in order to achieve perfect secrecy. If the key bits are truly randomly produced (i.e. by a true random number generator) then $H(K) = |K|$, meaning that the running-key $(\sigma_t)_{t\geq 0}$ must be at least as long as the message to encipher. From a practical point of view, this property becomes very difficult to handle for large-scale operations (e.g. soviets reused keys for one-time pad encryption and thus enabled cryptanalysis in VENONA project) because the hardware based generation of truly random sequences is somewhat complex and doesn't scale very well. This is the reason why one-time pads are generally used for short messages or strategic use only. Usually in stream ciphers the running-key is produced by a PRNG (pseudorandom number generator) as an expansion of a secret key that is reduced in size (up to 256-bits). The secret key only serves for the initialization of the algorithm at time $t = 0$. But now $H(K) \geq H(M)$ is no longer true. Yet, as stated in [1], in practice the number of messages enciphered by a cryptosystem using |K|-bit keys is always far lower than $2^{|K|}$ (the number of secret keys), thus it can be assumed that during the lifetime of the cryptosystem

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

holds true. However, like in one time pad systems, it is very important not to reuse secret keys. The security of stream ciphers becomes void whenever a secret key is reused to initialize the system.

2.3 Describing the Attack

The preceding chapters served as a basic introduction to symmetric encryption. In the following chapters the attack itself will be explained. At first in a general theoretical way applicable to any stream cipher and then specifically applied to the Microsoft Office 2003 RC4 encryption. Chapters 3 and 4 explain how to detect ciphertexts in which the key has been reused. These chapters correspond to sections 6.2 and 6.4 which highlight the vulnerability in the MS Word case. Here the IV is reused in revised versions of Word documents which leads to parallel ciphertexts that can be obtained from temporary files in Windows. Chapter 5 describes a general algorithm to recover the plaintexts from the ciphertexts by statistical means. The cryptanalysis consists of a language specific frequency analysis of character sequences. This chapter corresponds to section 6.5, which shows and evaluates the results obtained in tests performed on french texts in MS Word documents.

3. PARALLEL CIPHERTEXTS

As shown by C.E. Shannon in [4] the security of stream ciphers and block ciphers in OFB mode is nullified if the secret key is reused. In order to clarify why key reuse compromises the security of any such cipher, it is necessary to take a look at what exactly happens when the same secret key is used to encipher multiple plaintexts. Let m_1 and m_2 be plaintexts and let σ be the single secret key used to encipher both of them into the resulting ciphertexts c_1 and c_2:

$$c_1 = m_1 \oplus \sigma$$
$$c_2 = m_2 \oplus \sigma$$

Then the ciphertexts c_1 and c_2 are said to be *parallel*.

More generally, as defined in [1], two (or more) ciphertexts are said to be parallel if they are produced either by a stream cipher (Vernam cipher or finite state machine) or by a block cipher in OFB mode using the same running-key. Furthermore the parallelism depth of k parallel ciphertexts $c_1, c_2, ..., c_k$ is k. Whenever a key is reused and parallel ciphertexts occur, the perfect secrecy as defined by Shannon in [3] has been violated and the parallel ciphertexts should be susceptible to cryptanalysis.

The first step necessary in order to perform a successful cryptanalysis is to detect groups of these parallel ciphertexts among a large number of ciphertexts. Detection needs to work without knowledge of the underlying cryptosystem (other than that it is indeed a stream cipher or block cipher in OFB mode), meaning the algorithm used for encryption can remain unknown. This is important because, as covered in the introduction, when ciphertexts have been intercepted the crypotsystem is usually still unknown and most of the time it is not feasible to uncover the algorithm using reverse engineering. It is also very interesting to note that whenever parallel ciphertexts will be detected, one can assume that either a serious misuse has occurred or that an implementation flaw, or worse, trapdoor exists within the program.

Once parallel ciphertexts have been detected, the logical follow up step is to perform a cryptanalysis to break the encryption and recover the plaintexts. This needs to be done using the ciphertexts alone, meaning the underlying cryptosystem and the key used for encryption may remain unknown. Since there is no preliminary key recovery, this technique is described in [1] as a *key-independent cryptanalysis*.

4. DETECTING PARALLEL CIPHERTEXTS

As explained in the previous chapter, parallel ciphertexts come into existence when multiple plaintexts are encrypted with the same secret key. They can then be exploited to break the encryption and recover the plaintexts without knowledge of the key or algorithm used. This chapter explains how to detect groups of parallel ciphertexts (if any) among a large number of ciphertexts.

4.1 Statistical Features in Languages

In order to understand the following steps it is important to know that plaintexts exhibit strong statistical features which depend on the language and encoding of the text. As explained by C.E. Shannon in [3] a language can be described as a stochastic process that produces a certain sequence of symbols according to a system of probabilities. He defines the parameter D as the redundancy of the language. Intuitively speaking, D measures how much text in a language can be reduced without losing any information. For example, in the English language the letter u may be omitted without loss of information when occuring after the letter q. This is possible because u always follows the letter q, thus it is sufficient to keep the q and discard u. Due to the statistical structure of the English language (i.e. high frequency of certain letters or words) many such reductions are possible. In fact, given a certain language with parameter D it is possible to calculate the number of intercepted ciphertext letters required to obtain a statistical solution:

$$\frac{H(K)}{D}$$

where H(K) is the size of the key space, which for n possible messages that are all a priori equally likely cannot exceed $\log_2 n$ bits. For example, in a very simple monoalphabetic substitution of letters $H(K) = \log_2(26!) \approx 88.3$ and $D \approx 3.2$ for the English language, hence about 28 letters are sufficient to break the encryption.

Furthermore it is interesting to note that encodings can hide or amplify certain statistical features and thus the choice of encoding is of importance when considering a trapdoor design. The next step is to build a statistical hypothesis test that serves as our general detection method and determines whether two ciphertexts are parallel or not.

4.2 Forming the Statistical Hypotheses

Consider the stream cipher xor encryption and let all operations be bitwise or bytewise as usual. Let

$$C_1 = M_1 \oplus \sigma_1$$
$$C_2 = M_2 \oplus \sigma_2$$

where M_1, M_2 are plaintexts, σ_1, σ_2 are the keys and C_1, C_2 are the resulting ciphertexts. From the previous chapter it is known that plaintexts M_1 and M_2 exhibit very strong statistical features depending on language and encoding (i.e. each character - letter, number, punctuation - has a different frequence of occurrence). Considering the ciphertexts C_1 and C_2 this frequence of occurrence is different. When using

ASCII encoding the probability for each character is $\frac{1}{256}$. Therefore the quantity

$$M_1 \oplus M_2$$

exhibits a very special statistical profile that can be detected and identified, whereas the quantity

$$\sigma_1 \oplus \sigma_2$$

exhibits a totally random statistical profile. The xor of the two ciphertexts C_1 and C_2 gives:

$$C_1 \oplus C_2 = M_1 \oplus \sigma_1 \oplus M_2 \oplus \sigma_2$$

If $\sigma_1 \neq \sigma_2$ the quantity $C_1 \oplus C_2$ will exhibit a totally random profile. But if the secret key has been reused (i.e. C_1 and C_2 are parallel and $\sigma_1 = \sigma_2$) then:

$$\begin{aligned} C_1 \oplus C_2 &= M_1 \oplus \sigma_1 \oplus M_2 \oplus \sigma_2 \\ &= M_1 \oplus M_2 \end{aligned}$$

Therefore the quantity $C_1 \oplus C_2$ exhibits a very strong statistical profile whenever $\sigma_1 = \sigma_2$ and a totally random statistical profile otherwise. Since this quantity behaves differently whenever key reuse occurs, it can be used to form statistical hypotheses in the statistical test for detection of parallel ciphertexts. The two hypotheses can now be defined as:

- Null hypothesis (H_0): $\sigma_1 \neq \sigma_2$. Ciphertexts C_1 and C_2 are not parallel, since the key has not been reused. Therefore the quantity $C_1 \oplus C_2$ exhibits a random statistical profile.
- Alternative hypothesis (H_1): $\sigma_1 = \sigma_2$. Ciphertexts C_1 and C_2 are parallel, the key has been reused. Thus the quantity $C_1 \oplus C_2$ is exactly equal to $M_1 \oplus M_2$ and therefore exhibits the same special statistical profile.

4.3 Choosing a Suitable Estimator

With the two hypotheses, established in the previous chapter, it is now possible to build an estimator that behaves differently in the cases (H_0) and (H_1) and therefore is suitable to detect whether two ciphertexts are parallel or not. Detecting parallel ciphertexts among a large number of ciphertexts is now as easy as detecting non-random files from random files. The chosen estimator in [1] performs a bitwise xor operation on each pair of ciphertexts and then counts the number of bits equal to null in the resulting sequence. Let n be the common part of the length of the two 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 at time instant i respectively, then the number of nullbits in the xor of C_1 and C_2 is denoted as Z:

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

Now let p be the probability that a bit in the sequence $C_1 \oplus C_2$ is equal to zero:

$$p = P[c_1^i \oplus c_2^i = 0]$$

Since every bit in the sequence $C_1 \oplus C_2$ is either null or one with a certain probability, p or $(1-p)$ respectively, and each xor result is independent from the previous one, Z has a binomial distribution with parameter n and p. For large n the binomial distribution can then, by application of the *de Moivre-Laplace theorem* [5], be approximated by a normal distribution with mean value np and standard deviation $\sqrt{np(1-p)}$. Therefore, assuming large enough n, Z has normal distribution:

$$Z \sim \mathcal{N}(np, \sqrt{np(1-p)})$$

Using this result it is possible to detect parallel ciphertexts from non-parallel ciphertexts, since the probability p is different with respect to hypotheses H_0 and H_1. For non-parallel ciphertexts, because of their random statistical profile, $p = \frac{1}{2}$. For parallel ciphertexts p depends on language and encoding. As stated by E. Filiol in [1], one can assume $p > 0.6$ for most languages. Therefore the setup for the statistical hypotheses test is now complete:

- If $Z \sim \mathcal{N}(\frac{n}{2}, \frac{\sqrt{n}}{2})$ then assume H_0: The key has not been reused ($\sigma_1 \neq \sigma_2$), consequently the ciphertexts C_1 and C_2 are not parallel.
- If $Z \sim \mathcal{N}(np, \sqrt{np(1-p)})$ with $p > \frac{1}{2}$ then assume H_1: The key has been reused ($\sigma_1 = \sigma_2$), consequently the ciphertexts C_1 and C_2 are parallel.

In all experiments carried out by E. Filiol in [1], large peak values for Z (usually above 0.6) were observed whenever ciphertexts were parallel. Therefore this test has also been empirically verified.

4.4 Detection Algorithm and Error Reduction

To find and further reduce any errors during detection, it is possible to apply an equivalence relation over the set of parallel ciphertexts. Let C_i, C_j and C_k be any ciphertexts and $\mathcal{R}$ the relationship “be parallel to”. Then $\mathcal{R}$ is an equivalence relation over the set of parallel ciphertexts, i.e. it is:

- reflexive: $C_i \mathcal{R} C_i$, since obviously any ciphertext is parallel to itself.
- symmetric: $C_i \mathcal{R} C_j \rightarrow C_j \mathcal{R} C_i$, meaning the direction of the parallel relation is not relevant.
- transitive: $C_i \mathcal{R} C_j \wedge C_j \mathcal{R} C_k \rightarrow C_i \mathcal{R} C_k$, which can be used as a consistency check in the detection process.

The equivalence relation $\mathcal{R}$ partitions the set of parallel ciphertexts into equivalence classes. Any such class forms a group of parallel ciphertexts. This can be used for consistency checks, since any ciphertext in a certain equivalence class can never be in a different equivalence class at the same time. Furthermore, using the transitivity property of $\mathcal{R}$, if C_1 is detected to be parallel with C_2 and C_2 is parallel with C_3, then C_1 must also be parallel to C_3 and all three ciphertexts must be exclusively in the same equivalence class. Any violation of these rules would point to an error in the detection process and require further decision making by either the algorithm or the user.

The general detection algorithm only requires a set of ciphertexts as the input. It then simply compares all ciphertexts pairwise, using the statistical hypotheses test. Finally it builds groups of parallel ciphertexts using the $\mathcal{R}$ relation to check for consistency.

Letter	Frequency	Letter	Frequency
A	6.09	N	5.44
B	1.05	O	6.00
C	2.84	P	1.95
D	2.29	Q	0.24
E	11.36	R	4.95
F	1.79	S	5.68
G	1.38	T	8.03
H	3.41	U	2.43
I	5.44	V	0.97
J	0.24	W	1.38
K	0.41	X	0.24
L	2.92	Y	1.30
M	2.76	Z	0.03

Table 1: relative frequency of English letters

5. RECOVERING THE PLAINTEXTS

The question, how to detect parallel ciphertexts, may they be caused by key misuse, implementation flaws or intentional trapdoors, has been taken care of in the previous chapter. The next logical step is to proceed with a cryptanalysis in an effort to recover the plaintexts by exploitation of the detected flaws. As mentioned in chapter three and four, the cryptanalysis technique used by Filiol in [1] will recover the plaintexts without the key. In fact the algorithm and key used for encryption are irrelevant. It is however very important to generate a reliable and conclusive statistical model of the target language. The term language is considered in the general, formal sense and not limited to natural languages. Consequently this approach works for all languages generated by any of the four grammar types in the Chomsky hierarchy. Furthermore, when building the model, it is also important to consider the encoding used (ASCII, Unicode, ...).

5.1 Constructing the Corpus

The concept of a corpus is defined in [1] as the set of all possible n-grams with their respective frequency of occurrence in the target language. An n-gram is simply understood as a string of n characters of that language. The corpus will serve as a qualitative and quantitative model of the target language. Any language can be described by the frequency of occurrence of its characters. One can easily select a few relevant texts and build a table with single letters (or other characters) and their frequency of occurrence in those texts. As an example, table 1 shows the english letter frequency taken from a large text, as determined in [6].

The same can be done for all possible n-grams and their respective frequencies in order to build the final corpus. Filiol has shown in [1], that 4-grams are the best choice when considering memory, time and accuracy of the model.

The quantitative aspect obviously lies in its size N, which is given by the number of n-grams and in their assigned frequencies. But there is also an important qualitative aspect of the corpus, which must be considered carefully. The corpus must be representative of the target language. In order to build the corpus a set of text is searched and frequencies of the n-grams are extracted. The choice of those texts must be made wisely. There's a large number of texts from many different times using different levels of language (i.e. common, technical, political, diplomatic, military, ...) that must be considered. It is therefore a good idea to build a number of different corpora, which model these different levels of language very precisely and then choose whatever corpus is best suited for the cryptanalysis, depending on the operational context at hand. A further requirement is that the texts contain a statistically significant amount of characters. It is interesting to note, that the corpus of n-grams is generally compliant to Zipf's law when considering natural languages [7]. This means, that the frequency of any n-gram is inversely proportional to its rank in the corpus.

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	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	.	,	;	:	?	!	$\ll$	(
)	{	}	+	-	*	/	=	‘	à
â	é	ç	è	é	ê	î	ô	ù	

Table 2: French language character space

In order to limit the resources required for the corpus, it is important to limit the character space as much as possible without omitting critical characters needed to describe the language effectively. This step is especially crucial for languages that contain a large amount of characters (e.g. due to their accentuation). This is true for languages modeled in ASCII encoding, such as French, Turkish and several northern european languages. When considering Asian or Arabic languages the same approach can be used, but a different encoding must be considered. Table 2 shows the character space chosen by Filiol in [1] for the French language.

Another important criterion is the length of the n-grams, i.e. the choice of n. This choice directly influences the size of the corpus. Obviously for larger n more combinations of characters will be possible, thus significantly increasing size and memory requirements of the corpus while also increasing search times. As experimentally verified by Filiol in [1], the best choice for n is $n = 4$, since using tetragrams over trigrams ($n = 3$) greatly improved results but using pentagrams ($n = 5$) did not further improve results. Considering a character space with 95 characters (ASCII) will produce a corpus of 95^4 different tetragrams.

5.2 Algorithm for Plaintext Recovery

Assuming that a number of ciphertexts were intercepted of which p are detected to be parallel, let $C_1, C_2, ..., C_p$ be the p parallel ciphertexts and $M_1, M_2, ..., M_p$ be the corresponding plaintexts. Then using a corpus of N n-grams and the algorithm given by Filiol in [1], it is possible to recover the plaintexts $M_1, M_2, ..., M_p$, without knowledge of the encryption algorithm or key. Before applying the algorithm the p ciphertexts are split into a succession of x n-grams. This algorithm will result in xN p-tuples, meaning N p-tuples for each n-gram at position j in the ciphertexts. Each such p-tuple is of the form $(M_1^j, M_2^j, ..., M_p^j)$ and contains possible plaintext candidates for the n-grams at position j within the plaintext messages $M_1, M_2, ..., M_p$.

The first step is to make an assumption for the plaintext n-gram M_1^j which corresponds to the ciphertext n-gram C_1^j. This assumption is added to a p-tuple as the first element. Then the key n-gram at position j is given as

$$K_j = C_1^j \oplus M_1^j.$$

In the next step K_j is combined with every C_i^j in the $(p-1)$ remaining ciphertexts, using the xor operation thus generating the remaining $(p-1)$ M_i^j of the p-tuple.

$$M_i^j = C_i^j \oplus K_j, \text{ for } i \in [2, p]$$

These resulting M_i^j represent possible plaintext n-gram solutions at position j for the remaining (p-1) ciphertexts and are added to the p-tuple. These steps are done for every position j, meaning $j \in [1, x]$. Then the complete set of steps is exhaustively repeated N times, meaning each of the N n-grams in the corpus needs to serves as a guess for M_1^j.

After completion the algorithm will have generated xN p-tuples. N p-tuples for each n-gram at position j. Therefore the results are p-tuples of the form $(M_1^j, M_2^j, ..., M_p^j)$, $j \in [1, x]$ being the position of the n-grams within the plaintexts and $i \in [1, p]$ denoting the plaintext. N such p-tuples exist for every position j:

$$\begin{array}{ll} j = 1 & (M_1^1, M_2^1, ..., M_p^1),\ (M_1^{'1}, M_2^{'1}, ..., M_p^{'1}), \\ & (M_1^{''1}, M_2^{''1}, ..., M_p^{''1}),\ ... \\ j = 2 & (M_1^2, M_2^2, ..., M_p^2),\ (M_1^{'2}, M_2^{'2}, ..., M_p^{'2}), \\ & (M_1^{''2}, M_2^{''2}, ..., M_p^{''2}),\ ... \\ \vdots & \\ j = x & (M_1^x, M_2^x, ..., M_p^x),\ (M_1^{'x}, M_2^{'x}, ..., M_p^{'x}), \\ & (M_1^{''x}, M_2^{''x}, ..., M_p^{''x}),\ ... \end{array}$$

The next step is to select the most probable of the N p-tuples for each position. In order to do that a p-tuple of probabilities $(P[M_1^j], P[M_2^j], ..., P[M_p^j])$ is associated with each of the corresponding xN p-tuples generated by the algorithm. In order to find the most probable plaintext n-grams p-tuple one has to determine the p-tuple that maximizes the p-tuples of probabilities. For that purpose a suitable function must be chosen that can compute those probabilities in the most significant way:

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

As explained by Filiol in [1], this step is where the ability and experience of the cryptanalyst becomes important, since the choice of this function strongly depends on the nature and contents of the texts. This function is named the *frequency cumulative function* [1] and must always be strictly increasing and positive. The probability of success depends strongly on the frequency function and a few other parameters (n-gram processing mode and decryption mode) that will be explored later on.

Let f_i be the frequency of occurrence of n-gram i in the corpus, then the most efficient choices for the *frequency cumulative function* are the *additive function* given by

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

and the *multiplicative function* given by

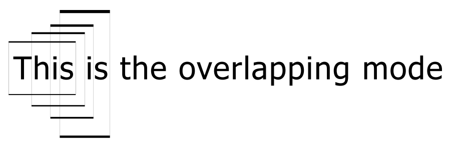
$$\prod_{i=1}^{p} (f_i^a + 1).$$

For texts containing many low frequency words, meaning low values of f_i (e.g. proper names, technical terms), the multiplicative function should be preferred, since it is much more efficient in this case. The optimal value for the parameter a is given by Filiol [1] as $a = 0.3$.

Another important factor for the successful recovery of the plaintexts is the *n-gram processing mode*. This refers to how the ciphertexts are split into n-grams. The first mode and most obvious solution is to simply split the ciphertext into a number of non-overlapping n-grams. This means that two consecutive n-grams have a void intersection. For example, using $n = 4$ this mode would produce the following framed n-grams:

$$\boxed{\text{This}}\boxed{\text{is}}\boxed{\text{the}}\boxed{\text{non-}}\boxed{\text{over}}\boxed{\text{lapp}}\boxed{\text{ing}}\boxed{\text{mode}}$$

In this mode n-grams do not share any characters which is not optimal if one wants to check for consistency. An advantage is that such a mode is very easy to implement. The disadvantage however is that whenever a wrong plaintext candidate is chosen, this error cannot be detected, since the n-grams are all independent. Therefore a different mode is introduced that splits the texts into n-grams by shifting one character position at a time: The overlapping mode allows



for consistency checks, since two consecutive n-grams always share $(n-1)$ characters. Using this property, one can verify that every n-gram candidate at position $j+1$ has $(n-1)$ common characters with the n-gram candidate at position j. This offers a huge help in selecting the correct plaintext n-gram candidates. However, this mode is a bit more complex to implement than the non-overlapping mode.

So far only the best plaintext n-gram candidate for every ciphertext n-gram at position j was kept (i.e. n-grams from the tuple that maximized the frequency cumulative function). This approach is called hard decoding. A further optimization can be made in keeping the b best candidates (soft decoding). This allows the use of backtracking, thus enabling the correction of wrong decisions.

The last effective optimization presented in [1] is to use the chosen character space as a limiting factor in the plaintext candidate selection. For example, if the plaintexts are known to contain only common language, one can allow only for p-tuples that contain printable characters exclusively to be accepted as candidates. This optimization is easy to implement, speeds up the plaintext recovery in general and also potentially prevents many wrong decisions. Since it is very likely that many p-tuples contain n-gram candidates with non-printable characters this increases the chances of a successful recovery greatly.

Finally, the best approach utilizes all the optimizations presented using parameters that were experimentally verified by Filiol in [1]. This means that the final cryptanalysis algorithm uses a multiplicative frequency function F with parameter $a = 0.3$:

$$F(f_1, f_2, ..., f_p) = \prod_{i=1}^{p}(f_i^a + 1),$$

the overlapping n-gram processing mode with consistency checks and a soft decoding, keeping the b best candidates with $b \in [5, 10]$.

6. BREAKING THE WORD ENCRYPTION

Now all the theoretical concepts needed for detecting a vulnerability (e.g. parallel ciphertexts caused by key misuse, implementation flaw or intentional trapdoor) and the cryptanalysis itself to recover the plaintexts from those parallel ciphertexts without a preliminary key recovery have been established. These concepts can be applied to the Microsoft Word encryption (up to Office version 2003). A number of different encryption methods are offered by Word, the very simple constant XOR encryption, the Office 97/2000 compatible encryption, which is a proprietary Office encryption, derived from the Microsoft Internet Explorer CryptoAPI method and several encryption services that are based on the RC4 stream cipher.

6.1 Word Encryption Methods

The default encryption method used by Office is the constant xor. In this encryption method the plaintext is combined with a constant 16-character string, derived from a user specified password, using the xor operation. The character string is simply repeated to cover all the plaintext. From a security point of view, this encryption method is very weak and in fact there already exist many tools that are dedicated to breaking this type of encryption in minutes (e.g. several products by the company Elcomsoft, which interestingly is also a Microsoft certified partner now). Although it might not be as efficient as dedicated software, it is also possible to utilize the previously discussed key-independent cryptanalysis, developed by Filiol, to break this type of encryption.

However this approach really shines when trying to break the RC4 based encryption methods offered by Word and other Office applications. These consist of several encryption standards (e.g. Diffie-Hellman with DSS and SHA-1), which theoretically offer adequate confidentiality, accountability and integrity. The strongest security is allegedly provided by the *RC4, Microsoft Enhanced Cryptographic Provider* service. *Microsoft Enhanced Cryptographic Prover* provides the same services as the *Microsoft Base Cryptographic Provider* services, but offers additional security through the use of longer keys and additional algorithms. In the RC4 case the key length of the enhanced provider is 128-bits, whereas the base provider only offers a 40-bits key. The *Microsoft Enhanced Cryptographic Prover* encryption (using RC4 with a 128-bit key and SHA-1) is the target of the attack described in [1].

RC4 is a stream cipher (symmetric key algorithm) that was developed by Ronald Rivest in 1987. It is still widely used in many applications that utilize stream ciphers. As usual for stream ciphers, RC4 uses a cryptographic bit stream that is combined with the plaintext using the xor function to produce the ciphertext. In the Office case the cryptographic bit stream is generated by a proprietary algorithm. A secret 128-bit key initializes a so called state table that is used to generate pseudo-random bytes, which are then utilized to generate a pseudo-random bit stream [8]. The 128-bit key is derived by a function F that takes the hash of a user specified password, concatenated with a “randomly” produced initialization vector (IV), to generate 128-bit values. Let F be the key generating function, H a cryptographic hash function (e.g. SHA-1) and IV the initialization vector, then the key K is given by:

$$K = F(H(IV||password))$$

This is a fairly standard approach to generate a key from a password (though usually an iterated hash is recommended) and a first step towards a strong encryption, since the key does not depend on the user's password alone. This is due to the randomly produced IV that is concatenated with the password, thus preventing the reuse of a key, even if the same password is used more than once. As stated by Filiol in [1] the IV plays the same role as a session key.

6.2 The Office Vulnerability

As discussed in chapter two, when using stream ciphers, it is imperative, not to reuse a secret initialization key, ever. One must assume, that many users will use the same password more than once, especially when considering the same document. Since Office generates the secret initialization key by the formula

$$K = F(H(IV||password)),$$

the security of the entire encryption becomes void whenever the initialization vector (IV) is reused. The vulnerability, first identified in [9], now lies in the fact, that Word 2003 (and Office 2003 in general) reuses the same IV for every revised version of a Word document. Since only multiple versions of a single document are considered, one can assume that the user will keep the same password for this document. Altogether this means, that the same secret key will be used for initialization of the RC4 stream cipher, whenever a revised (modified) version of a Word document is saved. As shown by Filiol in [1], this is the case even when a new file name is used to save the modified document. Thus all the revised versions of any single Word document form a set of parallel ciphertexts. Once detected by means of the detection method discussed in chapter four, these parallel ciphertexts can then be subjected to the cryptanalysis described in chapter five and finally the plaintexts can be recovered without any knowledge of the key. The vulnerability is caused by a flawed implementation of the RC4 algorithm in Microsoft Office, not by RC4 itself.

6.3 Word Document Specifics

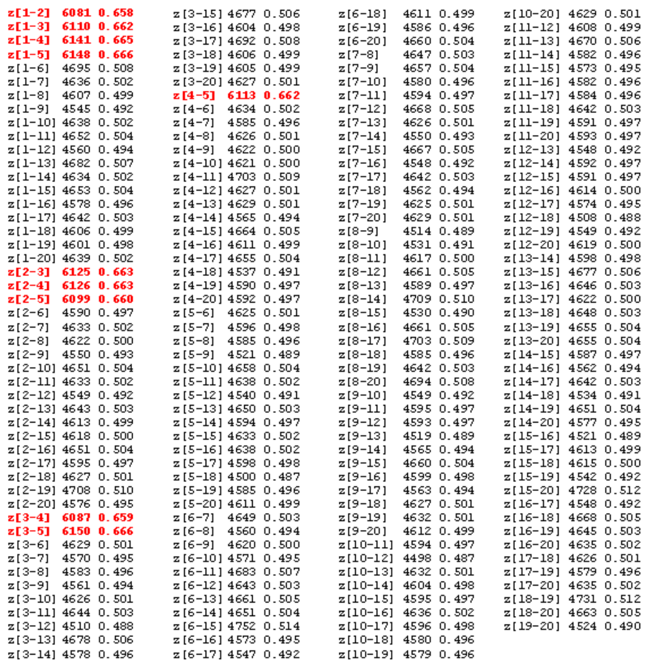
Before detection of parallel ciphertexts can occur, one must find the encrypted data within the Word documents. As determined by Filiol in [1], the beginning of the encrypted text in any Microsoft Word document is always located at offset $0xA00$. The size S of the encrypted text in bytes is calculated from two values x and y, located at offsets $0x21D$ and $0x21C$. Surprisingly those values are never encrypted, even if *document properties encryption* is enabled, thus the size in bytes can be calculated using the following formula:

$$S = 256x + y - 2048.$$

6.4 Detecting Parallel Word Documents

In order to apply the key-independent cryptanalysis described in chapter five, it is first necessary to locate several encrypted Word documents and determine whether their contents constitute a parallel group of ciphertexts. The flaw in the implementation of RC4 in Office is further amplified by the way the Windows operating system handles temporary files. Creating or modifying an Office document also creates temporary files, each containing a previous version of the document. The files will be deleted after closing Word. But as typical under Windows, this is done in an insecure way and the data still remains on disk and can be recovered (e.g. by using dedicated recovery software). Using these temporary files, by either intercepting them while the user is working on the document with Word opened or by recovering them from the disk after Word is closed, a parallel depth greater than one is achievable with relative ease. In fact, in the experiments conducted by Filiol [1], up to 4 temporary files were recovered. Any parallel depth above two is more than sufficient for a highly successful cryptanalysis. With an additional semantic validation step even two parallel ciphertexts are usually enough to recover the plaintexts.

Filiol has conducted many experiments for documents in almost all main languages. For example, a set of twenty encrypted 1500-character Word documents, of which the first five had the same password, were used in the detection test, resulting in the output shown in figure 2. The first column denotes which of the documents were compared for parallelism, for example z[1-2] means document number 1 and 2 were compared. The second column lists the number of compared bits and the third column shows the zero bits divided by the total number of bits, as used by the estimator developed in chapter four. The result clearly shows that peaks for files one to five, thus detection of parallel ciphertexts worked very well in this test.



```
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
```

Figure 2: Detection result of Filiol's experiment [1]

6.5 Testing the Cryptanalysis Algorithm

In further tests to verify the functionality of the cryptanalysis algorithm, multiple texts from different times and backgrounds were considered. These consisted of extracts from Jules Verne novels with a total length of 1200 bytes each, 1500 byte extracts from a speech of the Chief of Staff of the Army held in 2008, containing many technical terms, proper names and diplomatic language and finally extracts from a speech of the president of the French Republic, each 9700 bytes long. In this case the cryptanalysis was done using a multiplicative frequency cumulative function with parameter $a = 0.3$. For further optimization, as discussed in chapter five, overlapping mode, the printable characters only option and a hard decoding were used. The tests were conducted for different depths of parallelism. For a parallelism depth of

- two, about 40% of the plaintexts were recovered.
- three, above 80% of the plaintexts were recovered.
- four and five, above 90% of the plaintexts were recovered.

If soft decoding and semantic analysis were used Filiol [1] would expect a success rate of nearly 100% for the plaintext recovery, when dealing with more than two parallel ciphertexts. A linguist-driven analysis would be necessary to recover all the missing characters in the case of only two parallel ciphertexts.

6.6 The Excel Case

The application of the cryptanalysis technique to Microsoft Excel is more tricky than in the Word case. The detection part of the algorithm is harder because the structure of Excel files is somewhat more complex than the structure of Word documents. For example whenever an Excel document is modified the new content is located at the end of the data, not at the location where the modification occurred. Despite these complications it is, as shown in [1] and [9], still fairly easy to locate the data within the encrypted Excel document. The vulnerability is the same as for Word documents. When encrypting an Excel document the same IV will be used for all modified versions and thus, as long as the users password doesn't change, the key for the RC4 stream cipher will be reused, just like in the Word case. Parallel ciphertexts can then be detected from temporary files or separately saved versions of the same document.

The cryptanalysis part is also somewhat more difficult than in the Word case. Since usually Excel files deal with numerical data, one cannot expect sentences or semantically structured data. Therefore certain optimization features of the cryptanalysis algorithm, such as semantic analysis, might not be possible to the full extend in the Excel case.

This means a very particular corpus, which is specifically constructed to model the context of the Excel spreadsheet, must be used. Usually a higher parallelism depth than in the word case is required. However Excel spreadsheets also offer an advantage over Word documents. Data in Excel files are located between cell separators, which in the binary file are denoted as XX 00 00, where XX is the size of the data inside the next cell [1]. These separators constitute probable plaintext and therefore enable a more efficient recovery process, as described in [3] by the probable word method. According to Filiol, if all these Excel specifics are taken into account, the recovery of Excel plaintexts is as efficient as in the Word case.

6.7 Recap of the Office Attack

Clearly the vulnerability of the Office 2003 encryption lies in the reuse of the IV for revised documents. This leads to key-reuse in these revised documents which form a set of parallel ciphertext files. Through a weakness in the Windows operating system these files can be obtained even after their supposed deletion. This satisfies all the conditions for the application of the cryptanalysis algorithm described in chapter five. The test results show that by means of a language specific frequency analysis of character sequences it is generally possible to recover large parts of the plaintexts. It is possible to further enhance these results, for example by performing a linguist driven analysis subsequently.

The test results show that good results (over 80% recovery) can be expected when at least three parallel documents are obtained. It is also vital to know the language and general context of the texts in order to obtain satisfactory results. Furthermore the attack is only possible if the user does not change their password for every revised version of a document, since IV and password must be reused to produce the same key. It is however very unlikely that a user would change their password after making any modification to the document, meaning the attack would be applicable in most cases.

7. CONCLUSION

Filiol has designed a very interesting and operational technique to detect and break any type of misused or wrongly implemented stream cipher and block cipher in OFB mode. One can imagine many uses for such a technique, that go beyond the simple scenario of an attacker, who wants to detect and break weak ciphertexts. Advanced users might be interested in running such a detection method for parallel ciphertexts against their encrypted documents, in order to make sure that no key reuse has occurred. On the other hand, since this works with any stream cipher, the detection technique can be of interest to companies, who want to employ it as an additional experimental check for the correct implementation of their stream cipher encryption algorithm. Furthermore the key-independent cryptanalysis really shines when one needs to identify and break messages encrypted by an unknown proprietary cipher. Using an USB-key, malware programs or a trojan horse an attacker could possibly detect and gather parallel plaintexts without direct access to the system in question and employ this technique to break them without the need for a time consuming key recovery or any knowledge about the underlying cryptosystem.

Considering the gravity of such an attack, it is surprising that after so many versions of Microsoft Windows and Office, a serious flaw like this still exists in the most widely used office application. However it is worth noting, that Microsoft Office 2007 SP2 and 2010 apparently have experienced a large rework of their security features. These newer versions support any encryption algorithm offered by the Microsoft Cryptographic Application Programming Interface (CryptoAPI), such as AES, DES, ESX, 3DES, and RC2. They also offer a wide selection of cryptographic hashing functions and use much more secure defaults (usually AES with 128-bit key in CBC mode and SHA-1 hashing) than the constant XOR of Office 2003 and previous versions.

Interestingly Microsoft openly acknowledges many of the short comings in previous Office versions in the *Office Document Cryptography Structure Specification* [10]. For example the constant XOR encryption is more fittingly called “XOR Obfuscation”. This document also highlights the flaws of the Office RC4 implementation. In fact in this specification it is said that “The Office binary document RC4 CryptoAPI encryption method is not recommended, and ought to only be used when backward compatibility is required” [10]. Not only is the implementation of this encryption method susceptible to the key-independent cryptanalysis described by Filiol but also to several other attacks, as outlined by the specification itself. For example the password may be subject to rapid brute-force attacks, because of the weak key derivation algorithm, using a single hash function instead of an iterated hash, which is recommended by *RFC2898*. Furthermore the key is derived from an input only 40-bits in length, thus the encryption key may be subject to brute force attacks even on current hardware. It is also stated that “some streams might not be encrypted” and “document properties might not be encrypted”, which would explain several plain values that were found in [1] and [9] when analyzing the encrypted Office documents. Finally it is also said that “key stream reuse could occur”, which is the flaw that allowed for the key-independent cryptanalysis, after detection of the parallel ciphertexts.

Filiol also touches the subject of how such implementation flaws can be used in a trapdoor design. While one flaw alone might not be sufficient for the recovery of plaintexts, it can, when combined with another flaw become a huge security issue and in fact also act as an intended trapdoor. Although the Office case is probably not an intended trapdoor it demonstrates how a combination of flaws can lead to a security problem. 50% of the flaw is at the application level (i.e. incorrect RC4 implementation, reuse of key streams) and the other 50% at the operating system level (i.e. Windows temporary files are insecurely deleted). However it is unlikely that the Office RC4 flaw is an intended trapdoor, since several other serious flaws exist in the Office encryption, which are documented by Microsoft themselves. Still, Filiol’s hints on trapdoor design give very interesting insight on how such trapdoors might be build by distributing the security breach over several layers.

Conclusively, it can be said that any product labelled with “X-encryption secure”, where X is a well known and approved encryption algorithm or even said to be information theoretically secure (e.g. Vernam system), might in reality not be secure at all. Implementation flaws or even intended trapdoors, that may consist of flaws distributed across multiple levels and above all misuse of the system by its users may nullify any means of security.

8. REFERENCES

[1] E. Filiol: *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*, Black Hat Europe 2010, April 12-15th

[2] A.J. Menezes, P.C. van Oorschot and S.A. Vanstone: *Handbook of Applied Cryptography*, CRC Press, 1996

[3] C.E. Shannon: *Communication Theory of Secrecy Systems*, Bell System Technical Journal 28, 1949, p. 679-683

[4] C.E. Shannon: *A Mathematical Theory of Communication*, Bell System Technical Journal 27, 1948, p. 379-423 July, p. 623-656 October

[5] A. Papoulis, S.U. Pillai: *Probability, Random Variables, and Stochastic Processes*, 4th Edition, McGraw-Hill Europe, 2002, p. 72-123

[6] E.S. Lee: *Essays about Computer Security*, Centre for Communications Systems Research Cambridge, p. 187

[7] W. Li: *Random Texts Exhibit Zipf's-Law-Like Word Frequency Distribution*, IEEE Transactions on Information Theory, 38(6), 1842-1845, 1992

[8] *Changes in encryption file properties in Office 2003 and Office 2002*, `http://support.microsoft.com/kb/290112`

[9] H. Wu: *The misuse of RC4 in Microsoft Word and Excel*, Preprint IACR, 2005

[10] *Office Document Cryptography Structure Specification*, Microsoft Corporation, 2011