J Comput Virol (2009) 5:321–334
DOI 10.1007/s11416-008-0097-x

EICAR 2008 EXTENDED VERSION

White-box attack context cryptovirology

Sébastien Josse

Received: 20 January 2008 / Revised: 16 June 2008 / Accepted: 26 June 2008 / Published online: 2 August 2008

Abstract This paper presents the use of cryptographic mechanisms that are suited to the white box attack context (the attacker is supposed to have full control of the target program's execution environment) and as we will demonstrate, to a viral context. Use of symmetric and asymmetric cryptography by viruses has been popularized by polymorphic viruses and cryptoviruses. The latter are specialized in extorsion. New cryptographic mechanisms, corresponding to a particular implementation of traditional (black box) cryptography have been recently designed to ensure the deep protection of legitimate applications. These mechanisms can be misappropriated and used for the purpose of doing extorsion. We evaluate these new cryptographic primitives and discuss their (mis)use in a viral context.

1 Introduction

We have observed for several years the use of specialized software protection applications by virus authors, with the aim of resisting reverse engineering more efficiently. The problem of content protection has triggered an important academic research in the field of software protection, under the hypothesis that an attacker and a legitimate end user of the protected software may be one and the same person. New software protection mechanisms and complete suites allow content providers to take drastic measures in order to secure their applications in depth.

In the same way virus designers have used specialized tools, such as packers, in order to reinforce virus protection, we believe that other more elaborated tools, such as specialized compilation chains, dedicated to in depth protection of DRM applications, can be misappropriated and used for the purpose of strengthen virus protection.

Among mechanisms brought into play by those software protection suites, several of them are of a cryptographic nature. In this paper, a novel use of one of these cryptographic mechanisms in a viral context is presented. The specific case of the design of a virus specialized in extorsion [20] is examined. The use of this kind of technology has to be taken into account by the anti-virus research community, in order to gain a broader vision of future viral threats.

Design and implementation of applications that are resilient against reverse engineering is both a crucial and difficult problem for many applications, especially when it is a matter of protecting proprietary algorithms and/or protecting the rights control function conditioning access to whole or part of its functionalities.

When the application to be protected cannot base its security on the use of an hardware component, or on a network server, we must make the hypothesis of an attacker able to execute the application in an environment that he perfectly controls. The attacker model matching this situation, called White-Box Attack Context (WBAC)[1] in this paper, imposes

This paper has been awarded with the EICAR 2008 Best Student Paper. Also with the Ecole Supérieure d'Informatique, d'Electronique, et d'Automatique, Laboratoire de virologie et de cryptologie opérationnelles, Laval, France.

S. Josse (✉)
Silicomp-AQL, 1 rue de la Châtaigneraie, CS 51766,
35517 Cesson-Sévigné Cedex, France
e-mail: Sebastien.Josse@aql.fr

[1] It should be noticed that WBAC context is the most restricting for software designers, insofar as a mechanism that is white box resiliant must also be resiliant against black box (BBAC—Black Box Attack Context) and gray box (GBAC—Gray Box Attack Context) attacks. The BBAC context is the most classical in cryptography: the attacker does not have access to information related to the implementation. The GBAC corresponds to logical attacks exploiting information that leaks

a particular software implementation of classical cryptographic primitives.

In this context, software protection lays on mechanisms covering several security objectives, among them the ability to control, in various execution points:

- Code, critical data and execution context integrity;
- Proprietary algorithms confidentiality;
- Diversification of software instances;
- Software anchorage to a personalized target execution platform, etc.

Viral context We can observe that malware not only have to be resilient against reverse engineering, they also have to evade detection. In the remainder of this paper, we will take an interest in the use of cryptographic mechanisms by a virus in WBAC context along with this additional constraint.

The remainder of this paper is organized as follows: Section 2 recalls fundamental theoretical results concerning obfuscation as a virtual black-box property. Section 3 presents the problem related to the use of cryptography by a virus for the purpose of doing extorsion and Sect. 4 gives examples of (mis)use of white box cryptography. In this section is also presented the need for a cryptographic mechanism adapted to the WBAC context. Section 5 presents principles of the white box implementation of two algorithms: DES and AES. Section 6 discusses the robustness of these algorithms against cryptanalysis. Section 7 concludes on the use that could be made of this type of technology by tomorrow's cryptoviruses and on the countermeasures and limitations of this technology.

2 Theoretical background

We will focus in this paper on cryptographic mechanisms tailor-made to ensure confidentiality of a secret key within an algorithm. Such a transformation (hiding a key in an encryption algorithm, with or without the help of environment interaction) can be formalized as an obfuscation transformation. We recall in this section some negative and positive results concerning code obfuscation, and their impact on this key management problem.

Footnote 1 continued
from the hardware (power consumption, instruction's execution time or certain CPU operations, such as cache setting, electromagnetic radiation, sound/noise spectrum, etc.). The attacker only has access to partial information about the implementation. This information is obtained by physical phenomenon's modelisation.

2.1 Ideal obfuscator

Let us denote Π a set of programs and PPT the set of polynomial time probabilistic Turing machines. An obfuscator can be formally defined as follows [7]:

Definition 2.1 A probabilistic algorithm $\mathcal{O}$ is an obfuscator if it satisfies the following properties:

1. $\forall P \in \Pi$, P and $\mathcal{O}(P)$ compute the same function;
2. $\forall P \in \Pi$, growing of execution time and space of $\mathcal{O}(P)$ is at most polynomial as regards to execution time and space of program P;
3. $\forall A \in PPT$, $\exists S \in PPT$ such as:

$$\forall P \in \Pi,\ p[A(\mathcal{O}(P) = 1] \backsimeq p[S^P(1^{|P|}) = 1].$$

Equality $p[A(\mathcal{O}(P) = 1] = p[S^P(1^{|P|}) = 1]$ is true up to a negligible[2] function μ of the program size $|P|$. The last property, called virtual black box property, can thus also be written: $\forall A \in PPT$, $\exists S \in PPT$ and $\exists$ a negligible function μ such as:

$$\forall P \in \Pi,\ |p[A(\mathcal{O}(P)) = 1] - p[S^P(1^{|P|}) = 1]| \leq \mu(|P|).$$

This property stipulate that the obfuscated version $\mathcal{O}(P)$ is perfectly unassailable, insofar as we cannot expect to learn more by reverse engineering $\mathcal{O}(P)$ than by the simple observation of its inputs/outputs.

It can be noticed first that the reverse engineering action is formalized as a predicate computation. We are thus taking into consideration the weakest requirement with regards to what can be calculated from $\mathcal{O}(P)$. The attacker is trying to decide a given property of program P.

The virtual black box property expresses the fact that the outputs distribution of any probabilistic analysis algorithm A applied to the obfuscated program $\mathcal{O}(P)$ is almost everywhere equal to the outputs distribution of a simulator S making oracle access to program P (program S does not have access to the description of program P, but for any entry x, it is given access to $P(x)$ in polynomial time as regard to the size of P. An oracle access to program P is equivalent to an access to sole inputs/outputs of the program P). Intuitively, the virtual black box property simply stipulates that everything that can be calculated from the obfuscated version $\mathcal{O}(P)$ can also be calculated via oracle access to P.

Such a generic compiler does not exist. The proof is based on the construction of a program that cannot be obfuscated.

This impossibility result demonstrates that a virtual black box generator—which could be able to protect the code of

[2] A function μ: $\mathbb{N} \rightarrow \mathbb{N}$ is said negligible if for all polynomial $\pi \geq 0$, $\exists N \in \mathbb{N}$ such that $\forall n \geq N$, $\mu(n) \leq 1/\pi(n)$. A negligible function is thus a function that grows much slower than the inverse of any polynomial.

any program by preventing it to reveal more information than it is revealed by its inputs/outputs—does not exist. This impossibility result naturally leads to important outcomes for designers of obfuscation mechanisms adapted to WBAC context. Let us consider a practical application of obfuscation that consists in transforming a symmetric encryption into an asymmetric encryption, by obfuscating the private key encryption scheme.

A private key encryption scheme (G, E, D) (where G is the key generation algorithm, E the encryption algorithm and D the decryption algorithm) is said unobfuscatable if there exist $A \in PPT$ and a negligible function μ such as:

$$p_{K \overset{R}{\leftarrow} \mathbb{F}_2^k}[A(\widetilde{E}_K) = K] \geq 1 - \mu(k)$$

where $\widetilde{E}_K$ is any circuit computing the encryption function with the key K (and $K \overset{R}{\leftarrow} \mathbb{F}_2^k$ refers to a random variable uniformly distributed over $\mathbb{F}_2^k$). An attacker is thus able, given any circuit calculating the encryption function, to recover key K. Unobfuscatable private key encryption scheme does exist if private key encryption scheme does. This result clearly states that all private key encryption scheme are not well suited for obfuscation. However, it should be noticed that this result does not prove that there does not exist some private key encryption scheme such that we can give to the attacker a circuit calculating the encryption algorithm without security loss. It proves however that there is not a general method enabling to transform any private key encryption scheme into a public key encryption system by obfuscating the encryption algorithm.

The problem of the construction of a private key encryption scheme verifying the virtual black box property (thus resilient in the WBAC context) is so an open problem, even if the impossibility result concerning a generic way to manage it may seem discouraging for the security designer. As we will see in Sect. 5, obfuscation by using a network of encoded lookup tables makes it possible to obtain from DES and AES algorithms versions that are more resilient in white box. However, effective cryptanalysis of DES and AES white box implementations establish that the problem of the construction of a private key encryption scheme verifying the virtual black box property remains complete.

The ideal model of an obfuscator enabling to transform any program into a virtual black box cannot be implemented. In particular, there is not any general transformation that enables, starting from an encryption algorithm and a key, to obtain an obfuscated version of this algorithm that could be published without leaking information about the key it contains. However, this formalism does not establish that it is impossible to drown a key in an algorithm in order to transform a private key algorithm into public key encryption. We set out to apprehend this problem in practice, by evaluating the most relevant practical propositions in this research field.

2.2 Notes on less restrictive obfuscator models

Several attempts have been made to relax the ideal model of obfuscator, in order to obtain positive results for obfuscation. It is possible to modify the virtual black box property in order to make it less unattainable. We can notably quote the τ-obfuscation [8], where the idea is not to search for perfect obfuscation, but rather for an efficient resilience at least for a certain time to deobfuscation transformations. More precisely, a τ-obfuscator satisfies the modified virtual black box property: $\forall A \in PPT,\ \exists S \in PPT$ such as:

$$\forall P \in \Pi,\ p[A(\mathcal{O}(P), 1^{\tau \times t(\mathcal{O}(P))}) = 1] \backsimeq p[S^P(1^{|P|}) = 1].$$

This property states that any result that can be computed in less than $\tau \times t(\mathcal{O}(P))$—where $t(\mathcal{O}(P))$ is the time needed to obfuscate P—is actually computable from an oracle program of P. Even if it seems that it is technically possible to implement the τ-obfuscation concept, the existence of τ-obfuscator remains an open problem.

It is also possible to express the characteristic properties of the ideal obfuscator in a less restrictive model. The random oracle model has been used to redefine the obfuscator notion and to obtain positive results of obfuscation. It is indeed possible to build a class of functions that are obfuscatable in this model: the point functions, namely the boolean functions $1_\alpha\colon \mathbb{F}_2^k \to \mathbb{F}$ defined as follows: $1_\alpha(x) = 1$ if $x = \alpha$, 0 otherwise. For random oracles $\mathcal{R} : \mathbb{F}_2^* \to \mathbb{F}_2^{2k}$, the obfuscator $\mathcal{O}^{\mathcal{R}}$ transforms the program 1_α into the program $\mathcal{O}^{\mathcal{R}}(1_\alpha)$ defined as follows: $\forall x \in \mathbb{F}_2^k,\ \mathcal{O}^{\mathcal{R}}(1_\alpha)(x) = 1$ if $\mathcal{R} = \mathcal{R}(\alpha)$, 0 otherwise. In other terms, in this model, the most classic method to conceal a password (storage of hash value $r = \mathcal{R}(\alpha)$) can be seen as obfuscation of a point function.

In the same way, the environmental key generation mechanism (see Sect. 3.2) leads to a true obfuscation of the key in the random oracle model.

Problem of implementation of the random oracle model we expect that any protocol designed in this ideal model remains secure when implemented by using a function easy to evaluate, such as a fixed hash function $f(k, .)\colon \mathbb{F}_2^* \to \mathbb{F}_2^{l(k)}$ in the place of the random oracle. It has been demonstrated in [14] that a system whose security lays on the correlation intractability of its random oracle can be secure in the random oracle model but does not remain secure anymore when implemented using a function or a functions set.

Theorem 2.1 (*Non-secure implementation of the random oracle* [14]). *There exist encryption* (*and signature*) *schemes that are secure in the random oracle model, but do not have any secure implementation by functions sets. Moreover, each of these schemes possesses a generic attacker that, knowing the description of an implementation, is able to break the scheme that uses this implementation.*

It should be noticed that this theorem confirms the result of Barak et al. When we try to modelize the reverse engineering action, we cannot assume that the only thing an attacker can do with the description of the oracle implementation is to invoke it on the entries of his choice: we shall not ignore that as it is usual in complexity theory, whole or part of the program code can be given as an entry to the program itself, and thus that disposing of the description of a function is far more powerful than having a black box access to this function.

The result of Barak et al. is furthermore complementary, insofar as it proves that a natural method[3] making it possible to obtain appropriate functions sets does not permit to obtain a secure implementation whatever the secure protocole in the random oracle model that is considered.

3 Use of cryptography by a virus for the purpose of doing extorsion

The study of viral mechanisms for the purpose of doing extorsion has been called cryptovirology [30,31]. After a virus has triggered its final charge, the effects on the target system can be irreversible for the victim but not for the virus author. The latter can therefore extort money from the victim in exchange for a way to restore its data. A first virus of this type was observed in 1989 (trojan horse AIDS). It used a simple substitution cipher.

3.1 Use of symmetric/asymmetric cryptography

Use of asymmetric cryptography makes it possible for a virus to avoid carrying a deciphering key that can be captured. The victim's data are encrypted using the public component K_{pub} of an asymmetric couple of keys (K_{pub}, K_{priv}). The virus author gives the private component K_{priv} in exchange for money.

The drawback of asymmetric cryptography is its slowness.

A first solution to this problem consists in ciphering only certain files (trojan horse PGPCoder). A second solution consists in randomly generating a key K and then in using a symmetric encryption algorithm E_K more efficient in order to cipher the victim's data. The virus next ciphers the key K with the public component K_{pub} of an asymmetric couple of keys (K_{pub}, K_{priv}). The victim must transmit $K_{pub}[K]$ and the extorted amount of money to the virus author. The latter can then send the key K to the victim, enabling him to restore its data without revealing the private key.

[3] The method consists in applying a transformation (which modifies the code of a program without altering its functionalities) to a set of pseudo-random functions, namely a set of functions that cannot be distinguished from a random oracle when given only oracle access to these functions.

We can thus see that both use of symmetric and asymmetric cryptography make it possible to design cryptographic viruses specialized in extorsion.

3.2 Key management by environmental generation

Cryptography can be used to solve other problems that cryptographic viruses must face: key management and polymorphism.

If sole use of asymmetric cryptography solves the problem of key management, its main limitations are its slowness and its lack of discretion as regard to detection of its cipher function.

Both uses of asymmetric and symmetric cryptography beg the residual problem of key management: the key is first generated on the target platform, next written into a file. Traces may subsist in memory, enabling a specialized company to find back the key K. Moreover, laboratory study of virus allows to develop a virus detection procedure for random generation and ciphering functions.

Environmental key generation [17,18,27] specifically addresses the problem of key management and supplies a solution in the instance of directed viruses, namely viruses designed to execute only on a target platform possessing features already known from the virus author. Environmental key generation is a mechanism that avoid storing the key in the executable. The key is generated by application of a hash function to activation data existing in the software's execution environment. Let X be an integer corresponding to this environmental observation, Y the value needed for activation (and carried by the program), h a hash function and R_1, R_2 two nonces. Then possible constructions, among many others, are [27]:

Let key $K = X$ where the test is: does $Y = h(X)$?;
Let key $K = h(X)$ where the test is: does $Y = h^2(X) = h \circ h(X)$?;
Let key $K = h(X_1, \ldots, X_n)$ where the test is: does $h(X_n) = Y$?;
Let key $K = h(R_1, X) \oplus R_2$ where the test is: does $Y = h(X)$?

The most important feature of each construction is that knowledge of Y does not provide knowledge of K.

Drawn from this principle, environmental code generation [1] is a mechanism that enables to dynamically generate code, starting from activation data existing in the software's execution environment.

At the time of software protection, an instructions block I is deleted. Given a key K and a hash function h, a value S is brute force calculated such as the equation $h(K||S) = I$ is satisfied.

At the time of software execution, the key K is generated by application of a hash function to activation data existing in the software's execution environment. The instructions block I is then generated by $h(K||S) = I$.

At the time of static code analysis (or dynamic analysis in an environment that does not possess the same properties as the target environment), the analyst knows S and the K values domain. He does not know the generated code. In order to recover the code chunck, the attacker must cover the whole key space and for each value, test the generated code. According to the generated code (semantic) nature, this brute force attack can be very difficult to bring to fruition.

A critical analysis of these mechanisms has been developed in [18]. Observe also that at the time of dynamic analysis of the code in the target environment, the attacker can recover the key K and the related code.

3.3 Key management and diversification through white box cryptography

White box cryptography looks for a particular implementation of encryption algorithms in order to increase the security of key management. White box symmetric encryption algorithms aim at assuring keys confidentiality in the WBAC context. These implementations put forward an intrinsic mechanism for instances diversification, making it possible plentiful polymorphic versions of the encryption function. The implementation reduces the code portion to its simplest terms, banishing from assembler code any classical arithmetical operation. Such a code is far easier to diversify by using a polymorphism/mutator engine. A specific implementation of an iterated block cipher algorithm enables to obtain several crucial properties for a ciphering function used by a cryptographic virus: a key management mode adapted to the WBAC context, an asymmetrication of a symmetric algorithm, a diversification of the algorithm data (algorithm code must be diversified by using a mutator engine). We will see however that the code uses only a reduced portion of the CPU instructions set, and that it thus goes along easier with diversification by using a mutator engine.

Key concealing in the algorithm: It is difficult to recover the key, given encryption algorithm's code (or given decryption code);

Algorithm asymmetrication: It is difficult to forge the encryption algorithm starting from the decryption algorithm (and inversely);

Algorithm code and data diversification: It is difficult to forge a signature given the algorithm code, because it only uses non-arithmetical instructions and thus instances (code and data) can be very diversified;

Execution time/memory space trade-off: Execution time, even if more consequential than the execution time of black box algorithm (storage space of lookup tables is not negligible and imposes a memory load time before execution), remains far lower than the execution time of an asymmetric ciphering algorithm.

This kind of mechanism finds a place on the side of other key management cryptographic primitives (environmental key generation by using hash functions, symmetric/asymmetric cryptography) usable by a virus for the purpose of doing extorsion.

4 Examples of use in a viral context

4.1 White box integrity checking

Before presenting examples of viruses using white box cryptography for the purpose of doing extorsion, it is interesting to present the use that is done by the specialized compilation chain CSS (Cloakware Security Suite [15]). The white box integrity checking function comprises:

- A first part implementing the hash function H and the white box deciphering algorithm WBD_K;
- A hashes storage area $\mathrm{WBE}_K[H(\mathrm{BODY})]$, called Voucher.

White box integrity checking corresponds to the following test:

$H(\mathrm{BODY}) {==} \mathrm{WBD}_K[\mathrm{WBE}_K[H(\mathrm{BODY})]]$? OK : KO

Because it is difficult to recover the key K or to rebuild the encryption function WBE_K starting from the analysis (in WBAC context) of the decryption function WBD_K, the attacker is not in a position to substitute new hashes to the values stored in the Voucher, which would have allowed him to use the modified application without constraint.

Observe that the verification function must be protected against dynamic analysis.

4.2 Logic bomb

A first elementary example of malware using white box cryptography for the purpose of doing extorsion comprises (in addition to benign code portions and a possible trigger condition) a part WBE_K implementing the white box encryption algorithm and whose mission is to encrypt whole or part of the victim's data. The victim is not able to recover the key K by using a WBAC analysis of the function WBE_K. Conjugated use of a mutator engine MUT and a random bijection generator RNG make it possible to create a huge number of versions of this program, for a unique key setting K.

Table 1 Complexity of the detection problem of a Grammar G production $L(G)$, namely the problem: does x belong to $L(G)$?

Grammar type	Complexity
Type 0	Undecidable
Type 1	NP
Type 2	NP
Type 3	P

As stated in the introduction, in the viral context, a mutator, namely a polymorphic engine must reinforce both the diversity and the resilience against pattern recognition. Both polymorphism [26] and metamorphism [19] can be formalized as grammar productions. The difficulty to recognize a virus corresponds to the required expressiveness of the machine or automaton that is able to recognize the language $L(G)$ generated by the grammar G by applying its production rules (see Table 1).

When he first formalized generative grammars in 1956 [10], Chomsky gave the following classification:

- Type 0 grammars (unrestricted grammars), produce recursively enumerable languages, namely languages that can be recognized by Turing machines. Thus their productions simulate Turing machines. Consequently, deciding whether $x \in L(G)$ or not reduces the Halting problem;
- Type 1 grammars (context-sensitive grammars) or type 2 grammars (context-free grammars), produce languages that can be recognized by non deterministic finite automata;
- Type 3 grammars (regular grammars) produce languages that can be recognized by deterministic finite automata.

Thus the generative grammar type is crucial while designing a polymorphic engine [19]. This point will be discuss in more details in Sect. 6.4

4.3 Polymorphic virus

Another elementary example of a virus (polymorphic virus) using white box cryptography consists of

- A first part WBE_K implementing the white box encryption algorithm;
- A second part WBD_K(RNG||MUT) comprising a random bijections generator RNG and a mutator engine MUT, both encrypted by using decryption algorithm WBD_K.

The first part WBE_K of the viral program encrypts whole or part of the victim's data, next it decrypts the second part of the virus. The execution of the random bijections generator RNG and of the mutator engine MUT results in the generation of the function couple ($\mathrm{WBE}_{K'}$, $\mathrm{WBD}_{K'}$). The key K' is possibly transmitted to the virus author (with information about the target computer). The algorithm $\mathrm{WBD}_{K'}$ is used to encrypt RNG||MUT (or MUT(RNG||MUT)). The new virus instance is $\mathrm{WBE}_{K'}||\mathrm{WBD}_{K'}$(RNG||MUT) (or $\mathrm{WBE}_{K'}||\mathrm{WBD}_{K'}$ (MUT(RNG||MUT))).

4.4 Metamorphic virus

Another example, without self-modifying code (metamorphic virus), consists of:

- A first part WBE_K implementing the white box encryption algorithm;
- A second part RNG||MUT comprising the random bijections generator RNG and the mutator engine MUT.

The first part WBE_K of the viral program encrypts whole or part of the victim's data. The execution of the random bijections generator RNG and of the mutator engine MUT results in the generation of the function $\mathrm{WBE}_{K'}$. The key K' is possibly transmitted to the virus author (with information about the target computer). The new virus instance is $\mathrm{WBE}_{K'}$||MUT(RNG||MUT).

4.5 Comments

4.5.1 Diversification of the WBAC mechanism

The encryption or decryption primitive does not need to be protected by mechanisms bound to hamper dynamic analysis, insofar as a single step examination of the execution (context examination at each step of the execution) does not provide information about the key. However, code diversification is required in order to make the signature of the virus difficult. Performed operations are not arithmetical, instead they involve lookup tables runs. Lookup tables are diversified because of their design.

However, it should be noticed that we must face the problem of diversification of the random bijections generator RNG, the mutator engine MUT and the CPU instructions required to go through a lookup tables network. This problem is discussed in Sect. 6.4.

4.5.2 Comparison with the hybrid symmetric/asymmetric method

The strong points of white box cryptography with respect to joint use of symmetric and asymmetric cryptography as proposed in [30,31] are:

– We can avoid using asymmetric cryptography,
– The cipher diversification mechanism is intrinsic,
– The code is easier to obfuscate because it does not contain any arithmetical calculation.

4.5.3 Comparison with environmental key generation

The strong point of white box cryptography with respect to environmental key generation is that the virus preserves its freedom. It is not directed to a specific platform and does not closely depend on a specific environment.

As compared with the two mentioned mechanisms, the drawback of this mechanism is that it is not yet as robust against cryptanalysis, as we will see in Sect. 6.

5 DES and AES white box implementations

We present in this section the principles of the white box implementation of two well known algorithms: DES and AES.

5.1 WB-DES implementation

A method has been published in [12] to make the extraction of the key difficult in the white box context. The principle is to implement a specialized version of the DES algorithm that embed the key K, and which is able to do only one of the two operations encrypt or decrypt. This implementation is resilient in a white box context because it is difficult to extract the key K by observing the operations carried out by the program and because it is difficult to forge the decryption function starting from the implementation of the encryption function, and inversely.

The main idea is to express the algorithm as a sequence (or a network) of lookup tables, and to obfuscate these tables by encoding their input/output.

All the operations of the block cipher, such as the addition modulo 2 of the round key, are embedded in these lookup tables. These tables are randomized, in order to obfuscate their functioning. The representation of DES as a sequence of lookup tables requires to group together the transformations made along the 16 rounds in a different way. Figure 1 shows these boundary changes. Each round of the DES is cut in two layers. The first one is said to be non-linear and contains the S-Boxes, whereas the other one is said to be linear and gathers together the linear operations such as the expansion, the xor operation and the permutation. Inputs of this new representation are now 96-bits binary words. Three variables are introduced: X_{r-1}, $\overline{R_{r-1}}$ and Y_r.

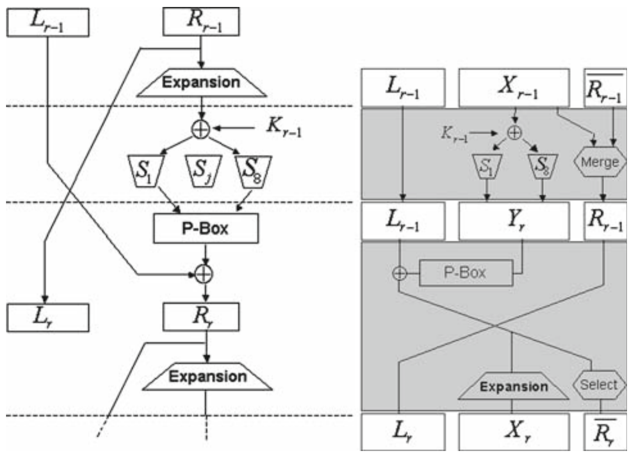

Fig. 1 One round of DES and its white box equivalent

– X_{r-1} represents the output of expansion, a 48 bits word;
– $\overline{R_{r-1}}$ represents the 16 bits of R_{r-1} that are not splitted by the expansion;
– Y_r represents the concatenation of the S-Boxes outputs, a 32 bits binary word.

It should be noticed that it is thus possible to untie the Feistel scheme of DES and to implement it as a substitution/permutation scheme,[4] as it is the case of AES.

The technique used to embed these keys is to represent DES as a network of lookup tables, and to apply input/output encodings in order to hide the keys. Using input/output encodings make each lookup table locally secure[5]: it is not possible to extract any information, in particular the embedded key. Thus the main idea of this obfuscation method is to be able to represent the whole DES as a unique lookup table that is locally secure, namely from which it is not possible to extract any information. Unfortunately it is not possible because the representation of a vector boolean function $\mathbb{F}_2^n \to \mathbb{F}_2^n$ requires an important memory space (exponential in the size of the input, namely the parameter n makes the memory space to rocket (see Table 2).

In order to take into consideration this constraint, smaller lookup tables are used. After having implemented the whole DES as lookup tables, the implementation is still not secure.

[4] More precisely, each DES round is splitted into a non-linear substitution step, bringing into play the S-Boxes, and a linear affine step.

[5] The lookup tables are randomized, in order to obfuscate their internal works: the input/output of these lookup tables are encoded by random bijections. The use of this encoding ensure a local security, namely the lookup table $g \circ T \circ f^{-1}$ encoded by bijections f and g does not provide any information about the original lookup table T. Given any lookup table T', there exists always two bijections f' and g' such that $g' \circ T' \circ f'^{-1} = g \circ T \circ f^{-1}$ (for example $f' = f \circ T^{-1}$ and $g' = g \circ T'^{-1}$). This local security is evaluated by an ambiguity measure, which expresses the difficulty that an attacker trying to suppress these parasite encodings must face (see Sect. 6.2 for a definition of the ambiguity measure).

Table 2 Memory space required for lookup tables storage

n (lookup table $\mathbb{F}_2^n \to \mathbb{F}_2^n$)	Memory space $n.2^n/8$ (bytes)
8	256 Bytes
16	128 KB
24	48 MB
32	16 GB

The next stage aims at encoding these lookup tables, in order to prevent any information leakage about the round keys. The technique involves composing the T-Box with non linear bijections in input and in output. Given two random bijections f and g (compatible with T), the T-Box is replaced with:

$$f \circ T \circ g.$$

Given three adjacent lookup tables L_1, L_2 and L_3 and f and g the input and output encodings applied to L_2, table L_2 is replaced with its encoded version $L_2' = f \circ L_2 \circ g$. It is thus required to encode the output of L_1 with g^{-1} and the input of L_3 with f^{-1} insofar as:

$$L_3 \circ f^{-1} \circ L_2' \circ g^{-1} \circ L_1 = L_3 \circ L_2 \circ L_1.$$

Because we need the local security property to be useful in our context, it is also required that the attacker should not be able to distinguish the non-linear T-Box (embedding a S-Box) from the bypass tables. A random permutation π_r must thus be applied on the order of the $T_i^{K_r}$, $i = 1, \ldots, 12$. From now on, the local analysis of the $T_{\pi_r(i)}^{K_r}$ requires $(12!)^{16}$ attempts.

At this stage of the obfuscation, inputs of first round's lookup tables are still exposed to a square-like attack [13]. Thus two external encodings F and G are integrated.

In summary, we were able to implement the whole DES algorithm as encoded lookup tables, in such a way that it seems difficult to extract any piece of information from any lookup table, by observing its input/output only.

5.2 WB-AES implementation

Obfuscation of AES [13] is done in a similar way as for DES. The goal is still to embed the round keys in algorithm code, in order to avoid storage of the key in static memory or its load in dynamic memory at the time of execution. The technique used to securely embed these keys is (as for DES) to represent AES as a network of lookup tables, and to apply input/output encodings in order to hide the keys.

Let us remember that AES starts with an initial AddRoundKey step and each further round of AES consists of four steps: SubBytes, ShiftRows, MixColumns and AddRoundKey for rounds $r = 1, \ldots, 9$ and three steps SubBytes, ShiftRows and AddRoundKey for round $r = 10$. In the white box implementation, this structure is reworked so that the initial AddRoundKey is part of a round. More precisely, if S is the S-Box that carries out the SubBytes operation, and $(k_{i,j}^r)_{(i,j)\in\{0,\ldots,3\}^2}$ the key of round r, then we first build the 10 T-Boxes:

$$T_{i,j}^r(x) = \begin{cases} S(x \oplus k_{i,j}^r), & \le r \le 9 \\ S(x \oplus k_{i,j}^{10}) \oplus k_{i,j-i}^{11}. \end{cases}$$

Observe that because of the linearity of the ShiftRows operation, it is possible to integrate the last AddRoundKey operation in a T-Box.

The next step for AES obfuscation is to represent Mixcolumns as a network of lookup tables. The T-Box is then replaced with the composition of the T-Box with the lookup table representing both the ShiftRows and MixColumns operations.

The technique used to encode the lookup tables is the same one as for DES obfuscation, namely to compose the T-Box with non linear bijections in input and in output. In order to thwart a square-like attack, it is required to reinforce the local security of the T-Box by using mixing bijections, namely linear bijections, in order to insert a diffusion step. Given two linear bijections m and M compatible with the preceding and succeeding operations, the T-Box is now replaced with:

$$f \circ M \circ MC_i \circ T \circ m \circ g.$$

Insertion of such mixing bijections m and M requires their cancelling by new lookup tables. Thus a new lookup table in inserted between rounds r and $r + 1$. The latter cancels both the mixing bijection M of round r and the mixing bijection m of round $r + 1$. We obtain the following additional composition: $g_{r+1}^{-1} \circ m^{-1} \circ M^{-1} \circ f_r^{-1}$.

At this stage of the obfuscation, inputs of first round's lookup tables are still exposed to a square-like attack. Thus two external encodings F and G are integrated.

In summary, we were able to implement the whole AES as encoded lookup tables, in such a way that it seems difficult to extract any piece of information from any lookup table by observing its input/output only. In order to fully implement the white box AES, we need four types of lookup tables:

- Type I External encoding, namely a function $\mathbb{F}_2^8 \to \mathbb{F}_2^{128}$ that composes two input decodings $\mathbb{F}_2^4 \to \mathbb{F}_2^4$, a linear bijection component $\mathbb{F}_2^8 \to \mathbb{F}_2^{128}$ and 32 output encodings $\mathbb{F}_2^4 \to \mathbb{F}_2^4$;
- Type II R-Box, namely a function $\mathbb{F}_2^8 \to \mathbb{F}_2^{32}$ that composes two input decodings $\mathbb{F}_2^4 \to \mathbb{F}_2^4$, a mixing bijection $m : \mathbb{F}_2^8 \to \mathbb{F}_2^8$, a T-Box, the mixcolums operation, a mixing bijection $M : \mathbb{F}_2^{32} \to \mathbb{F}_2^{32}$ and eight output encodings $\mathbb{F}_2^4 \to \mathbb{F}_2^4$;

Type III A function $\mathbb{F}_2^8 \rightarrow \mathbb{F}_2^{32}$ that composes a mixing bijection component $M_i^{-1} : \mathbb{F}_2^8 \rightarrow \mathbb{F}_2^{32}$, four times a mixing bijection $m^{-1} : \mathbb{F}_2^8 \rightarrow \mathbb{F}_2^8$ and eight output encodings $\mathbb{F}_2^4 \rightarrow \mathbb{F}_2^4$;

Type IV XOR-Box, namely function $\mathbb{F}_2^8 \rightarrow \mathbb{F}_2^4$ that composes two input decodings $\mathbb{F}_2^4 \rightarrow \mathbb{F}_2^4$, a XOR lookup table and an output encoding $\mathbb{F}_2^4 \rightarrow \mathbb{F}_2^4$.

6 Evaluation

Before presenting WB-DES and WB-AES cryptanalysis, criteria for security evaluation of a cryptographic primitive in black box context are presented. In the WBAC context, we can consider other criteria, such as diversity or ambiguity, which are able to account for cryptographic quality of a white box encryption algorithm component. Diversity and ambiguity are measures that are able to qualify supposedly the robustness of the white box implementation. We apply this criteria to WB-DES and WB-AES algorithms.

Finally, we present the main cryptanalysis of which these two algorithms were the subject.

6.1 Black box security criteria

Black box analysis of DES algorithm leaded to definition of general security criteria (such as strict avalanche or propagation criteria, xor value, output bit independence criterion [2,3,16,29]) on the confusion boxes of an iterated encryption system as regards to linear and differential cryptanalysis.

High nonlinearity provides resistance against linear attacks. Optimally nonlinear boolean functions are said to be bent. The nonlinearity of an s-box S is the minimal distance of all non trivial linear combinations of the columns of S to the set of affine functions. We expect the xor table to almost have a constant and uniformly small distribution. This second property will aid in protection against differential cryptanalysis. It should be noticed that the maximum order SAC criterion is guaranteed if bent functions are chosen for the columns of the s-box. In a nutshell,

- High nonlinearity provides resistance against linear attacks;
- An uniformly small distribution of the xor table provides resistance against differential attacks;
- The other properties (strict avalanche and output bit criteria) ensure that the S-box has good dynamic properties and are crucial in the design of the encyption algorithm, especially concerning its diffusion property.

Other criteria, such as balancedness, algebraic degree, and order of correlation immunity exist and are important for the black box security [9]. Unfortunately, not all of these properties can be achieved simultaneously.

These criteria are also fundamental for design and evaluation in the white box context for several reasons:

- In the first place, a white box implementation must also be resistant to black box cryptanalysis;
- Second, the white box cryptography uses black box methods, with finer granularity (here, each lookup table can be seen as a black box, of which we try to extract information or the key);
- Last, design of an encryption algorithm tailor-made to be white box resistant can lean upon these criteria to prove its security.

About the above remark, it should be noticed that the random generation of bent functions is not an easy task. S-boxes can be created either randomly or deterministically. The advantage of the former is that there is no simple mathematical structure that can be potentially used for cryptanalysis. Both methods can be used simultaneously for the construction of bent functions, but deterministic known algorithms are costly and reduce the diversity level obtained through randomization.

6.2 White box diversity and ambiguity criteria

The diversity measure consists in counting the number of different implementations that it is possible to generate (including the variation of embedded keys). This measure is important because it characterises the ability of the obfuscator to stave off large scale attacks (a priori). Attacks specific to an instance then only have a limited range.

Definition 6.1 (White box diversity [13]). The white box diversity metric counts the number of distinct constructions or decompositions, namely the number of possible encoded steps.

As an example, Table 3 gives the diversity measures of the four types of lookup tables that are used in the AES white box implementation.[6]

However, this measure does not account for an implementation robustness against an attack which aims to extract the embedded key. In order to better qualify the robustness of an implementation, it is more interesting to count the number of constructions, namely the number of keys and random bijections reaching the same lookup table. The bigger this

[6] For the reader who wants to check the calculus, let us remember that there are $2^n!$ bijections $\mathbb{F}_2^n \rightarrow \mathbb{F}_2^n$, and among them $2^n \prod_{i=0}^{n-1}(2^n - 2^i)$ affine bijections (and $\prod_{i=0}^{n-1}(2^n - 2^i)$ linear bijections). Moreover, the Moivre-Stirling formula gives the approximation: $n! \simeq \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$ (for n big enough).

Table 3 Diversity of WB-AES lookup tables

Type	Diversity
Type I	$(16!)^2 \times 20160^{64} \times (16!)^{32}$
Type II	$(16!)^2 \times 256 \times 2^{62.2} \times 2^{256} \times (16!)^8$
Type III	$(16!)^2 \times 2^{256} \times (16!)^8$
Type IV	$(16!)^2 \times 16!$

Table 4 Ambiguity of WB-AES lookup tables

Type	Ambiguity
Type I	$(16!)^2 \times 20160^{32}$
Type II	
Type III	$(16!)^2 \times 15!$ if the two blocks of the matrix are of null rank
	$(16!)^2 \times 20160^2$ if the two blocks of the matrix are of full rank
Type IV	$16! \times 16$

number is, the more ambiguity is introduced by the obfuscator. The ambiguity metric enables to account for the space of possibility the attacker must face in order to find the exact combination key/bijection used at the time of the generation of the white box instance that he holds.

Definition 6.2 (White box ambiguity [13]). The white box ambiguity metric is an estimate of the number of constructions that produce exactly a certain table (of a given type). It is defined as the ratio of its white box diversity and the number of distinct tables (of this type).

As an example, Table 4 gives an approximation of the ambiguity measures of the four types of lookup tables that are used in the AES white box implementation.

6.3 Cryptanalysis of two white box implementations

6.3.1 WB-DES cryptanalysis

Let us remember that in the DES white box implementation, a 96 bits word goes through lookup tables. This word or internal state is represented in Fig. 1 by the concatenation of blocks $L_{r-1} \in \mathbb{F}_2^{32}$, $X_{r-1} \in \mathbb{F}_2^{48}$ and $\overline{R_{r-1}} \in \mathbb{F}_2^{16}$: $L_{r-1}||X_{r-1}||\overline{R_{r-1}}$.

Let us subdivise the internal state $L_{r-1}||X_{r-1}||\overline{R_{r-1}}$ of round r into 8-bits words. It is thus represented as the concatenation of 12 binary words: $v_1^r||v_2^r||\ldots||v_{12}^r$. Each word v_i^r represents the encoded input of a T-Box T_i^r, which can be of two sorts: either a non-linear T-Box (embedding a S-Box) or a bypass T-Box. The attack [24] works directly on the vectors v_i^r, by the addition of differences denoted Δv (or equivalently by the substitution of v_i to a value v_i'. Indeed, $v_i \oplus v_i' = \Delta v \Leftrightarrow v_i \oplus \Delta v = v_i \oplus v_i \oplus v_i' = v_i'$). Because on the one hand the vectors v_i^r are encoded versions of true inputs $f_i^r(v_i^r)$ of the T-Box and on the other hand the encodings f_i^r are not linear, it is not possible to deduce from Δv the difference that is really applied to the input $f_i^r(v_i^r)$. Therefore, the attack observes the propagation of these differences on the T-Boxes of the next rounds ($r+2$, $r+3$ and $r+4$).

Thus the attack exploits the noteworthy properties of the DES round function: input bits do not affect all output bits of the round function. By analyzing the propagation of a difference $\Delta v = v \oplus v'$ on the input of an encoded T-Box (namely $g \circ T \circ f$) through several rounds, it is possible to obtain information about the internal behavior of this difference. When identified a set of differences: $\{\Delta v \mid f(\Delta v)$ corresponds to one or two bits flips on input to $T\}$, it is possible to recover the key embedded in the white box implementation.

The differential cryptanalysis described in [24] works as follows:

1. In the first place, distinguish the non-linear T-Boxes among the 12 T-Boxes;
2. Second, partially discover the random permutation π_r that is applied to the non-linear T-Boxes, by using a standard (black box) implementation of the DES algorithm;
3. Last, extract the key K_r.

In conclusion, this attack exploits the weakness of the DES round function. In order to thwart such an attack, the last resort to improve this design seems to be the randomization of the S-Boxes. If such a modification of the DES is clearly unacceptable (mainly because the S-Boxes have strong security properties, as stated in Sect. 6.1), it could be an interesting trail in the viral context, even if random generation of such vectorial boolean functions is not a trivial task. Indeed, an attacker would have to reconstruct the S-Boxes for each new version of the algorithm.

Let us see now the white box resilience of a much stronger encryption algorithm, namely AES, which round function seems to be immune to such a differential attack.

6.3.2 WB-AES cryptanalysis

As shown in Sect. 5.2, a round of the obfuscated AES is made of two lookup tables. The first one achieves the operations AddRoundKey, ShiftRows and MixColumns (the last round is slightly different), whereas the second, inserted between rounds r and $r+1$, reverse the linear bijections that are inserted both:

– Before the output encoding of the lookup table implementing round r, as well as
– After the input encoding of the lookup table implementing round $r+1$.

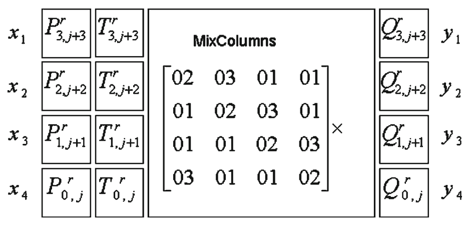

Fig. 2 R^r_j, $j = 0, \ldots, 3, r = 1, \ldots, 9$

A lookup table being a particular representation of a vectorial boolean function, it is very possible to compose the lookup tables between them when they match. This fact is exploited by the cryptanalysis [5]. Let us consider the four bijections that map 4 bytes of the current state to 4 bytes of the following state. Each of these bijections is noted R^r_j, $j = 0, \ldots, 3$, $r = 1, \ldots, 9$ (c.f. Fig. 2). The heart of R^r_j is the concatenation of 4 T-boxes, followed by the multiplication by MC. This core is protected in input and output by encodings (8 in total: 4 in input, 4 in output). Let us recall that the latter are made of:

– In input, the concatenation of two 4-bits encodings, followed by a 8-bits linear bijection;
– In output, a 8-bits linear bijection, followed by the concatenation of two 4-bits encodings.

The cryptanalysis aims at recovering the parasits safeguarding R-Boxes. To do so, the attackers proceeds in several steps:

1. In the first place, linearize the P-Boxes and Q-Boxes, by recovering their non-linear part.
2. After removing the non-linear parts of P-Boxes and Q-Boxes, we obtain unknown affine bijections. The second step of the attack consists in recovering the linear component along with the translation vector.
3. The recovery of the linear bijections makes it possible in the same time to recover the round keys that are integrated in the S-Boxes, but in an unknown order. The last step thus consists in exploiting the constraints that lean on these bytes in order to classify them in the right order. These constraints come from the cadencing algorithm.

In conclusion, it seems difficult to hide the algebrical structure of AES by only using encoded non-linear bijections. In order to thwart this attack, one could introduce a linear diffusion operation right after the first encoding (let us note this new 32×8 lookup table network D_1^{-1}), an operation D_2 (8×32 lookup table network) being inserted right after the substitution stage. We can expect that the noise introduced by these random permutations makes the second step of the cryptanalysis—i.e. after encodings linearization—more difficult.

6.4 Polymorphism

We give here a brief view of pros and cons of the use of lookup tables. Let us remember that polymorphism can be formalized as a generative grammar production. The more irregular the grammar is, the more difficult it is to derive an automaton from the grammar that is able to detect the cipher function.

From a theoretical point of view [33], the kernel of a polymorphic virus is made of an infection trigger condition $I(d, p)$, a payload function $D(d, p)$, the corresponding payload trigger condition $T(d, p)$ and a selection function $S(p)$ of target programs to infect. The latter function is in charge of the code mutation.

Metamorphic viruses differ from polymorphic viruses since while polymorphic forms of a virus share the same kernel, metamorphic forms of a virus do not. Using formal generative grammars, it is possible to give a more practical definition of metamorphism: Let $G = (N, T, S, R)$ be a formal grammar, where N is a set of non-terminal symbols, T is an alphabet of terminal symbols, $S \in N$ is the start symbol and R is a production (or rewriting or semi-Thue) system over $(N \cup T)^*$, namely a set of rule producing the langage $L(G)$. We define $G' = (N', T', G, R')$ where the alphabet of terminal symbols T' is a set of formal grammars, and R' is a set of production rules over $(N' \cup T')^*$.

Definition 6.3 (Metamorphic virus [19]). A metamorphic virus is a virus whose mutation engine is described by a grammar whose words are themselves a set of productions with respect to a grammar. A metamorphic virus is thus described by G' and every of its mutated form is a word in $L(L(G'))$.

Thus from one metamorphic form to another, the virus kernel is changing: the virus is mutating and changes the mutation rules at the same time.

With regards to the detection complexity of mutation techniques, several theoretical results have already been established:

– Detection of bounded-length polymorphic viruses is an NP-complete problem [28];
– The set of polymorphic viruses with an infinite number of forms is a Σ_3-complete set [32];
– Some code mutation technique embedding the word problem—which is known to be undecidable with respect to a semi-Thue system—leads to metamorphic viruses whose detection is undecidable [19]. The PBMOT engine's productions rules change from mutation to mutation and is specially designed to embed the word problem (with respect to a semi-Thue system).

The formal frame being presented, the main question that arises in our context is to prove that the white box implementation is suited to hinder sequence-based antiviral detection, by using a mutator engine. Because the implementation data is diversified by use of random bijections, only the code handling must cancel as much as possible any potential fixed element that would represent a potential detection pattern. Intuitively, because the instruction set required is very small and corresponds only to instructions needed to walk through a table, production rules can map such instructions to any chunk of code. Further investigations are required in order to check this assumption. Moreover, several behaviors may represent useful invariant that can be considered by antivirus, such as linear walk of lookup tables.

7 Conclusion

We presented in this paper a new use of white box cryptography in the viral context. WBAC cryptographic mechanisms enable an original way of key management, by embedding the keys in the implementation with a partial evaluation as regards to the key. This key management mode defines an original alternative to the environmental key generation (where the key is not embedded in the program body but dynamically generated starting from trigger information existing in the virus environment) or to the use of an asymmetric key infrastructure (where only the public key is stored in the virus body). This mechanism offers a trade-off between symmetric and asymmetric encryption, by asymmetrication of the implementation of an iterated block cipher.

WBAC cryptographic mechanisms enable a significant diversification of implementations, by integration of random bijections (used to encode the input/output of lookup tables or to insert an additional diffusion step by means of mixing bijections). Besides to ensure local security, this randomization of implementations enables to generate numerous partial evaluations of the encryption algorithm, for a single key setting. Furthermore, the algorithm implementation does not contain any arithmetical operation (it only contains operations enabling the dataflow to transit through a network of lookup tables). It is therefore easier to generate polymorphic instances of the algorithm by using a mutator engine (the CPU instruction subset used by such an implementation is reduced to basic memory handling instructions. These instructions are thus easy to diversify).

In the viral context, these properties are welcome. In particular, the corresponding obfuscation transformation could be applied to asymmetric cryptography and to hash functions, in order to increase the polymorphism level of encryption and environmental key generation functions.

WBAC cryptographic mechanisms enable a noteworthy strength against cryptanalysis in WBAC context: even if the proposed mechanisms are not as resilient in white box as in black box so far, the first attempts are encouraging, given that encryption algorithms are not so easy to break. If they do not yet offer a sufficient security level for digital right management or critical applications, their potential use in a viral context already poses a problem that must be taken into consideration by antivirus research. If compilation chains specialized in software protection are not widely available for end users, this fact could change in the next few years. In the same way as viral applications used specialized packers to increase their strength against reverse engineering, we can imagine that they use more complete solutions and integrating white box cryptographic mechanisms, enabling them to ensure an in depth protection.

7.1 Countermeasures and limitations

We made the conjecture that a virus whose code is only made of the CPU instructions that are required to walk through tables, the latter being wholly recomputed at each copy of the viral code, is a code that is easier to remodel than a code containing static immuable data and rich in arithmetical instructions. We observe that in order to be able to evade an antiviral detection program, a special attention must be paid to the mutator engine. It is also required to reinforce the robustness of the white box cryptographic primitive by other security mechanisms. In particular, other software protection mechanisms must thwart an attacker to extract the code implementing the cipher function, that he could use as a key. In the CSS software suite, these security hypotheses are covered by the use of a specialized compilation chain, making it possible to integrate additional protections at every step of the compilation, in order to ensure an in depth protection of the application.

7.2 Future works

Several additional tasks could be made in order to give a more complete overview of this technology:

– Investigate the robustness of white box implementations of AES algorithm with key sizes 192 and 256, along with the additional linear bijections described in Sect. 6.3.2 and with a possible randomization of the substitution boxes;
– Investigate a grammar-based implementation of the mutator engine, tailor-made to exploit the structure of white box implementations.

Concerning the S-boxes randomization, i'm currently working on random algorithms starting from properties of the Algebraic Normal Form (ANF) representation of boolean functions (remember that a boolean function can be represented equivalently as a truth table, a fourier spectrum or as a multivariate polynom called ANF). The main idea of such a random generation algorithm is to put constraints on the multivariate monom algebraic properties of the ANF of boolean functions, in order to guarantee a high nonlinearity (comparable with results obtained using known deterministic methods).

Concerning the redesign of parts of AES and DES algorithms to obtain diversity, some works have already be made in this research area, mainly in the black box and gray box attack contexts:

– In case of AES, we can take an interest in the dual ciphers grey box secure design [4];
– In case of DES, we can cite the Fast Data Encryption (FDE) design proposal [22], which has a DES-like structure and comes with an algorithm to construct strong S-boxes.

A more ambitious investigation would be to design from scratch an encryption algorithm specially designed to be resilient in a white box context, namely not corresponding for example to the white box implementation of any iterated block cipher known for its black box resilience—the paradigm of an iterated round function is perhaps not a solution in the WBAC context, insofar as the cryptanalyst can work on an arbitrarily reduced number of rounds.

References

1. Aycock, J., deGraaf, R., Jacobson, M.: Anti-Disassembly using Cryptographic Hash Functions. University of Calgary, Canada. Available at http://pages.cpsc.ucalgary.ca/~aycock/ (2005)
2. Adams, C.M., Tavares, S.E.: Designing S-Boxes for ciphers resistant to differential cryptanaysis. In: Wolfowicz, W. (ed.) Proceedings of the 3rd Symposium on State and Progress of Research in Cryptography, pp. 181–190. Fondazione Ugo Bordoni (1993)
3. Adams, C.M., Mister, S.: Practical S-Box Design. Workshop on Selected Areas in Cryptography (SAC'96) Workshop Record, Queens University, pp. 61–76 (1996)
4. Barkan, E., Biham, E.: In how many ways can you write Rijndael? In: Proceedings of Asiacrypt'02, LNCS 2501, pp. 160–175 (2002)
5. Billet, O., Gilbert, H., Ech-Chatbi, C.: Cryptanalysis of a white box AES implementation. In: Helena, H., Anwar Hasan, M. (eds.) Selected areas in Cryptography. Lecture Notes in Computer Science, vol. 3357, pp. 227–240. Springer, Heidelberg (2004)
6. Boneh, D., Felten, E., Jacob, M.: Attacking an obfuscated cipher by injecting faults. In: Digital Rights Management Workshop, pp. 16–31. Available at http://www.cs.princeton.edu/~mjacob/papers/drm1.pdf (2002)
7. Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S., Yang, K.: On the (Im)possibility of Obfuscating Programs. Available at http://www.math.ias.edu/~boaz/Papers/obfuscate.html (2001)
8. Beaucamps, P., Filiol, E.: On the possibility of practically obfuscating programs. Towards a unified perspective of code protection. In: Bonfante, G., Marion, J.-Y. (eds.) Journal in Computer Virology, (2)–4, WTCV'06 Special Issue (2006)
9. Canteaut, A.: Cryptographic Functions and Design Criteria for Block Ciphers. In: Progress in Cryptology—Indocrypt 2001, no 2247 in LNCS, pp. 1–16. Springer, Heidelberg (2001)
10. Chomsky, N.: Three models for the description of languages. IRE Trans. Inform. Theory **2**(2), 113–123 (1956)
11. Chomsky, N.: On certain formal properties of grammars. Inf. Control **2**, 137–167 (1969)
12. Chow, S., Eisen, P.A., Johnson, H., van Oorschot, P.C.: A white-box DES implementation for drm applications. In: Security and Privacy in Digital Rights Management, ACM CCS-9 Workshop, DRM 2002, Washington, November 18, 2002, Revised Papers. Lecture Notes in Computer Science, vol. 2696, pp. 1–15. Springer, Heidelberg (2002)
13. Chow, S., Eisen, P.A., Johnson, H., van Oorschot, P.C.: White-box cryptography and an AES implementation. In: Nyberg, K., Heys, H.M. (eds.) Selected Areas in Cryptography. Lecture Notes in Computer Science, vol. 2595, pp. 250-270. Springer, Heidelberg (2002)
14. Canetti, R., Goldreich, O., Halevi, S.: The random oracle methodology, revisited. In: Proceedings of STOC 1998, pp. 209–218 (1998)
15. Cloakware Security Suite. Available at http://www.cloakware.com
16. Dawson, M.H., Tavares, S.E.: An expanded set of S-Box design criteria based on information theory and its relation to differential-like attacks. In: Advances in Cryptology, Eurocrypt '91, pp. 353–367 (1991)
17. Filiol, E.: (2004) Strong Cryptography Armoured Computer Viruses Forbidding Code Analysis : the BRADLEY virus. INRIA ISSN 0249-6399. In: Proceedings of EICAR 2005 Conference, StJuliens/Valletta, Malte. Available at http://papers.weburb.org/frame.php?loc=archive/00000136/
18. Filiol, E.: Techniques virales avancées. Springer, Collection IRIS, XXI, p. 283, ISBN 978-2-287-33887-8 (2006)
19. Filiol, E.: Metamorphism, formal grammars and undecidable code mutation. In: International Journal of Computer Science, vol. 2, Nb. 1, 2007 ISSN 1306–4428 (2007)
20. Gazet, A.: Comparative analysis of various ransomware virii. In: Proceedings of the 17th EICAR Conference, ESIEA, Laval (2008)
21. Goubin, L., Masereel, J.-M., Quisquater, M.: Cryptanalysis of white box DES implementations. Cryptology ePrint Archive, Report 2007/035, 2007. http://eprint.iacr.org/ (2007)
22. Hendessi, F., Gulliver, A., Shafieinejad, A.: A structure for fast data encryption. J. Contemp. Math. Sci. **2**(29–32), 1401–1424 (2007)
23. Link, H.E., Neumann, W.D.: Clarifying obfuscation: improving the security of white-box DES. In: ITCC (1), pp. 679–684 (2005)
24. Michiels, W., Gorissen, P., Preneel, B., Wyseur, B.: Cryptanalysis of white-box DES implementations with arbitrary external encodings (2007)
25. Preneel, B., Wyseur, B.: Condensed white-box implementations. In: Proceedings of the 26th Symposium on Information Theory in the Benelux, pp. 296–301. Brussels, Belgium (2005)
26. Qozah. Polymorphism and grammars, In: 29A E-zine, 4. Available at http://www.29a.net/ (1999)
27. Riordan, J., Schneier, B.: Environmental Key Generation towards Clueless Agents. School of Mathematics Counterpane Systems, University of Minnesota, Minneapolis. Available at http://www.schneier.com/paper-clueless-agents.pdf
28. Spinellis, D.: Reliable identification of bounded-length viruses is NP-complete. IEEE Trans. Inf. Theory **49**(1), 280–284 (2003)
29. Tavares, S.E., Webster, A.F.: On the design of S-Boxes. In: Crypto, Lecture Notes in Computer Science, vol. 218, pp. 523–534 (1985)
30. Young, A.L., Yung Cryptovirology: extortion based security threats and countermeasures. In: Proceedings of IEEE Symposium

on Security and Privacy, pp. 129–141. IEEE Computer Society Press, Oakland (1996)

31. Young, A.L., Yung, M.: Malicious cryptography, exposing cryptovirology. Wiley, New York (2004)
32. Zuo, Z., Zhou, M.: On the time complexity of computer viruses. IEEE Trans. Inf. Theo. **51**(8), 2962–2966 (2003)
33. Zuo, Z., Zhou, M.: Some further theoretical results about computer viruses. Comp. J. **47**(6), 627–633 (2004)