Clarifying Obfuscation: Improving the Security of White-Box Encoding

Hamilton E. Link and William D. Neumann
Sandia National Laboratories[⋆]
P.O. Box 5800
Albuquerque, NM 87185-5800-0455
[helink,wneuman]@sandia.gov

Abstract

To ensure the security of software executing on malicious hosts, as in digital rights management (DRM) applications, it is desirable to encrypt or decrypt content using white-box-encoded cryptographic algorithms in the manner of Chow et al. Such encoded algorithms must run on an adversarys machine without revealing the private key information used, despite the adversarys ability to observe and manipulate the running algorithm. We have implemented obfuscated (white-box) DES and 3DES algorithms along the lines of Chow et al., with alterations that improve the security of the key, eliminating attacks that extract the key from Chow et al.s obfuscated DES. Our system is secure against two previously published attacks on Chow et al.s system, as well as a new adaptation of a statistical bucketing attack on their system. During implementation of white-box DES we found that a number of optimizations were needed for practical generation and execution. On a typical laptop we can generate obfuscated DES functions in a Lisp environment in under a minute allocating 11 MB, including the space required for the resulting function. The resulting function occupies 4.5 MB and encrypts or decrypts each block in approximately 30 ms on an 800 MHz G4 processor; slight run-time performance of the obfuscated DES could be traded to further reduce our algorithms representation to 2.3 MB. Although it is over an order of magnitude slower than typical DES systems, we believe it is fast enough for application to some DRM problems.

1 Introduction

The central threat to software in digital rights management (DRM) contexts is the malicious host. Software must run on an adversarys computer, putting them in complete control of execution. The provider must assume that the adversary is able to run, stop, and restart the software at any point, reverse engineer components of the system, and see and manipulate all data. For software that must run on traditional hardware, an adversary can reverse engineer the entire program. We are interested in a class of programs that implicitly compute functions and depend upon a data segment that is incomprehensible to an adversary.

A system that is intended to be run on a malicious host is, by definition, not a black box because the adversary is able to view the programs execution as well as any intermediate results that are generated during computation. The *white-box attack context* was introduced by Chow et al. as a setting where the adversary is allowed to not only make such observations about the software, but is also able to examine and alter the software at will [CEJv02a]. The system they implement in this context is a white-box program is a DES encryption algorithm. Ideally, a white-box encryption function would be so difficult to analyze that an adversary would be inclined to resort to a plaintext/ciphertext pairs attack, much as if the white-box implementation were a black box. Their implementation of DES does lend itself to some analysis, however, and a key can be extracted easily. We have built upon their ideas to create a similar white-box version of DES that is much less vulnerable to direct analysis.

One possible use for white-box cryptography would be the replacement of content encoding schemes such as the Content Scramble System (CSS) used to protect DVDs [css]. A chip could be made to perform decryption of DVD content using a white-box DES decryptor without revealing the encryption key, which an adversary would like to use to generate ad-

[⋆] Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

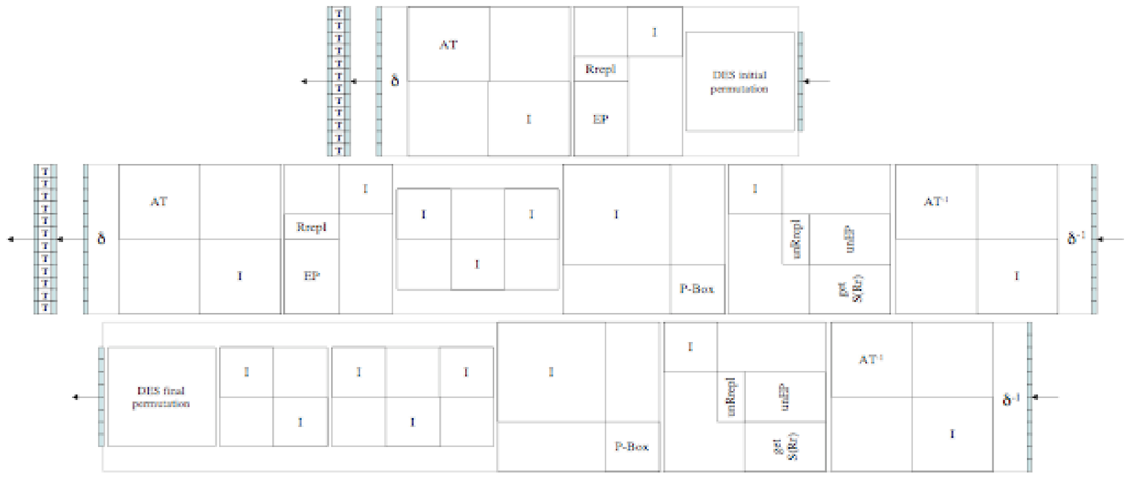

Figure 1 DES as a a sequence of table and matrix operations

ditional playable DVDs. As with CSS, such a system would still be vulnerable to duplication of raw data images, but it would be far more difficult for a criminal to generate properly encoded alterations or to extract the quantity of information needed to create a content player without the original content controls.

A second application for white-box cryptography of this sort would be in authenticating communications in low power processors in wireless broadcast situations. In such a setup, it would be desirable if, for example, directives broadcast from some central authority could be authenticated prior to being carried out. The obvious approach would be to sign the directives with the central authority's private key, and require that the nodes verify the signature before processing the command. However, in many situations, the nodes may not be powerful enough to verify the signature efficiently. With such low-power nodes, we would prefer to be able to use a symmetric-key message authentication code to verify the authenticity of the sender. Unfortunately, if a MAC is used, an adversary need only compromise a single node and recover the shared symmetric key in order to pose as the central authority and issue bogus directives. Using white-box cryptography, we can hide the key from the adversary in a "verify-only" version of the code, which would prevent the attacker from issuing any bogus directives.

Chow et al. developed a white-box encoding for DES; their approach can be readily applied to other block ciphers such as Rijndael [CEJv02b]. The process of encoding reduces DES to small table lookups and then systematically reindexes and delinearizes those tables. In their publication of a white-box DES they admit vulnerability to a statistical bucketing attack on their encoding. They address this vulnerability by augmenting DES with a nonstandard input and output permutation, but this makes their implementation a non-DES cipher, which may be a reasonable approach for many DRM applications hat do not require the use of standard encryption schemes. Jacob et al. [JBF02] also performed some analysis of this DES encoding and described a fault-injection attack using differential cryptanalysis that reveals the DES key. This attack requires matching encodings of DES encryption and decryption, however, which would not be the case in many DRM applications.

We have improved upon the work of Chow et al. and have implemented a white-box DES that is resilient to both the statistical bucketing attack described to by Chow et al. and the differential cryptanalysis attack of Jacob et al. We have also implemented a modified statistical bucketing attack against Chow et al.s DES that requires fewer encryptions by the target function than either of these prior attacks and requires only access to a white-box DES encryption function. Our alterations protect against this new attack as well. Our white-box DES is comparable in time and space requirements to its predecessor. We have profiled our implementation and believe that while it is substantially slower than a typical DES, the cost is appropriate to the threat model and the performance is acceptable for some uses.

2 Related work

In recent years, research into obfuscation has produced a number of methods of protecting code from an adversary. A number of these attacks are described in [Hoh98]. Most of the obfuscation techniques attempt to prevent the adversary from performing static analysis on the code to learn about the underlying algorithm [WDHK01], [CTL97], [CTL98]. This approach differs from ours and that of Chow, van Oorschot, et al. in that, rather than hiding the workings of the obfuscated program, our goal is to obfuscate a portion of the data segment of the code. In addition, none of these techniques are believed to be secure in the white-box attack context, where dynamic analysis of the code is allowed. Barak, et al. have shown that it is not possible to obfuscate all programs [BGI+01]to create black-box equivalents for a computationally unbounded malicious host. Theoretical work remains to be done to identify useful subclasses of programs that can be obfuscated to black-box equivalents for a computationally bounded malicious host or to define other useful definitions of obfuscation.

The other major branch of research into obfuscation has been built up from Yao's encrypted circuits [Yao86]. This approach requires representing the code to be obfuscated as its equivalent circuit, and then using Yao's techniques to encrypt the circuit, its input, and all intermediate values [AF90]. While these techniques can be proven to be secure (modulo standard cryptographic assumptions), they are terribly inefficient to represent, with each gate in the circuit requiring 96 bytes of space in its encrypted form [ACCK00]. Another approach similar to this is that of Sander and Tschudin [ST98a], [ST98b], who encrypt an instance of a function along with its inputs using a homeomorphic encryption scheme. The encrypted inputs can then be processed on an untrusted host who can return the encrypted result to the original party for decryption. Unfortunately, this technique is restricted to a small class of problems – those that can be represented by a rational function.

Perhaps the most closely related approach to ours and Chow et al.'s is that of W.P.R. Mitchell [Mit99]. Mitchell describes a technique for performing encryptions with most round-based block ciphers while keeping the secret key hidden by a construct he calls "deceitful automata". While this technique is interesting from a theoretical standpoint, there are no known implementations of these deceitful automata, and it seems as if any such implementation would require an excessive increase in code size. Additionally, Mitchell describes this technique as being secure only against a "naive adversary" with access to a debugger and disassembler, and not against an attacker working within the white-box attack context.

3 Review of White-Box DES

Chow et al. and Jacob et al. described a white-box encoding of DES that is the basis for our implementation. In the interests of completeness, we will review the particulars of encoding DES here. Chow et al. also provided details on specific elements of the white-box encoding process that we do not need to reproduce.

DES is usually described as a sequence of permutations, s-boxes, and xors on bit vectors. The permutations and xors are linear and can be represented by affine transformations (ATs), but because the s-boxes in DES are nonlinear, we can not represent DES as a single AT. We can represent DES as alternating ATs and s-boxes. Chow outlines a process for re-representing an affine transformation with functionally equivalent code and data that cannot easily be used to recover the original AT. Once we represent an instance of DES (i.e., a key-specific DES) with ATs and s-boxes, we can prevent recovery of the ATs and simultaneously conceal the structure of the s-boxes to prevent recovery of the key that was used to generate them.

To represent DES permutations and xors for each round as a single AT, the s-box output must be accompanied by the original left and right input halves, L_r and R_r, for round r. This information is brought forward in parallel using 8×8 T-boxes that produce L_r, R_r, and the s-box results. R_r is included as the 16 duplicate bits passed into the s-boxes by the expansion permutation (EP), and 16 additional replicated bits of s-box input. The T-boxes allow the remaining DES components to be represented as matrix operations that are combined into a single AT in each round. In this manner, all of DES can be represented by alternating ATs and T-boxes (Figure 1).

Chow et al. used a technique of matrix decomposition (Figure 2) to implement io-block encoding of affine transformations. The technique divides each

$$y = Mx + v$$

$$\begin{bmatrix} y_o \\ \cdots \\ y_{k-1} \end{bmatrix} = \begin{bmatrix} M_{0,0} & M_{0,1} & \cdots & M_{0,j-1} \\ \cdots & \cdots & \cdots & \cdots \\ M_{k-1,0} & M_{k-1,1} & \cdots & M_{k-1,j-1} \end{bmatrix} * \begin{bmatrix} x_0 \\ x_1 \\ \cdots \\ x_{j-1} \end{bmatrix} + \begin{bmatrix} v_o \\ \cdots \\ v_{k-1} \end{bmatrix}$$

$$y_i = (\sum_{q=0}^{j-1} M_{iq} x_q) + v_i$$

$$m = ja, \quad n = kb$$

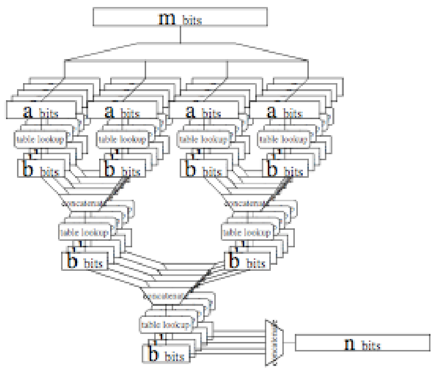

Figure 2 Decomposition of an affine transform

AT into independent equations that compute subvectors of the original result. Each subvector in the output is computed using a copy of the input vector, which is also divided into subvectors. Any subdivision of the input and output vectors will work. For clarity, Figure 2 depicts a uniform division of bit vector input into four a-bit vectors, and the creation of the output from five b-bit vectors. The first tables are 2^a-element multiplication tables with b-bit entries. The remaining tables are addition tables and have 2^{2b} b-bit entries, so tables that add two 4-bit values have 256 4-bit entries, while a table that adds two 8-bit values has 65,536 8-bit entries. We chose to divide input into 8-bit subvectors to match the 8-bit output of the T-boxes, and we divided the output into 4-bit subvectors to keep the addition tables from growing out of hand.

The delinearization step referred to by Chow et al. and Jacob et al. prevents an adversary from viewing the original contents of each table. Tables are delinearized by creating random permutations to rename their contents; for example, the elements of a table of 8-bit values would be renamed with a permutation of $[0 \ldots 2^8 - 1]$. The inverse of this permutation would be used to reindex the following table, which would subsequently have its contents renamed. For a system like DES, which can be represented entirely with ATs and table lookups, this process can be carried out on the entire implementation once the ATs have been tabularized.

Because the output of a table is delinearized, it is not possible to split the elements of the table into separate pieces without delinearizing the pieces separately. For this reason the block sizes available to us had to be divisors of eight. The smallest representation of the 96×96 matrices used to represent the rounds of DES would come from a 6×2 io-block encoding of the matrices, but such a division would have reduced the effectiveness of delinearization. For security, obfuscated blocks associated with the T-box input and output need to be as large as can be represented efficiently.

4 Attack on Split T-Box Output

Chow et al. described a statistical bucketing attack on their white-box DES that exploits the nonlinearity of the s-boxes to expose the key in under ten seconds. Their attack tracks individual bit changes in the input to the second round of s-boxes, generating and comparing preimage sets of input to expose the key. For their T-box implementation, which divides the 8-bit T-box output into two 4-bit chunks, one of which is the obscured 4-bit s-box output, it is actually possible to generate more detailed partitions in the input and expose the key more efficiently. We implemented this new attack and found it to be several times faster than that of Chow et al.

Our implementation of the statistical bucketing attack identifies the T-box corresponding to each s-box, exposes the first round key 6 bits at a time, and then uses brute force search to reveal the full DES key. Our software precomputes the 4-element sets of 6-bit preimages for each s-box using an all zero key. The 6-bit elements of each preimage are run backwards through the expansion permutation (EP) and the DES initial permutation (P1) to produce the corresponding zero-key preimages.

To identify the T-box corresponding to each s-box given a white-box DES, we encrypt 0 through the first T-boxes. We then turn on individual input bits and observe the changes in the T-box output to identify the 4-bit T-box output corresponding to each

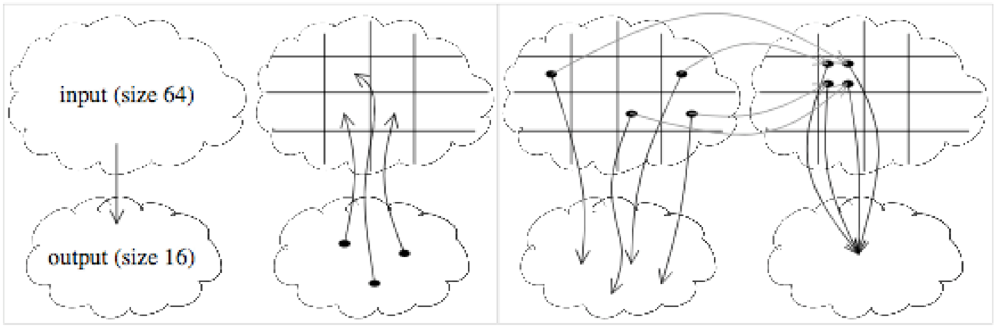

Figure 3 The structure of the domain (left) causes a correct round subkey to produce recognizable preimage sets.

s-box. Next we begin testing the 6-bit hypotheses for each portion of the first round key. Hypotheses are reversed through EP and P1, and this value is xored with all elements of each zero-key preimage before being passed to the encryptor. If the processed preimages each map to a single 4-bit T-box output (i.e., the set of preimages remain the same), the hypothesis is correct and 6 bits of the 48-bit first round key have been identified (Figure 3). When the first round key is known, we record the full white-box DES encoding of an arbitrary input. This is used to perform brute force search of the remaining bits, comparing the recorded result with a conventional DES encoding for each potential key until a match is found.

As part of the attack implementation, a reference DES is created that produces first-round s-box results. All 64 6-bit s-box inputs (corresponding to 6 bits of message encrypted with a zero key) are passed through all eight s-boxes and mapped backwards through EP and P1. The DES inputs corresponding to each of the 16 4-bit results for each s-box are recorded as preimage sets. Each s-box has 16 zero-key preimage sets with 4 elements.

The first step of the attack is to identify which of the twelve T-boxes correspond to the eight s-boxes. This is a matter of encrypting a zero block through the first round of T-boxes, and then encrypting individual bits and observing which T-box outputs change. For the bits $b_0b_1b_2b_3b_4b_5$ passed into an s-box, bits b_0 and b_5 are passed as bypass bits in the T-box to create a bijection, b_1 and b_4 are produced as bypass bits for other T-boxes, and b_2 and b_3 are replicated in an arbitrary T-box as part of the replicated bits (those bits not duplicated by EP) for R_r. The T-box that is changed by both b_1 and b_4 for an s-box is the matching T-box, and in particular the 4-bit T-box output block that changes corresponds to the 4-bit s-box output. Our attack uses this process to build s-box output bitmasks.

Once the s-box outputs have been identified, the program runs through each s-box and each of 64 corresponding candidate 6-bit portions of the first round key. For each candidate key, the 6 bits are reversed through EP and the DES initial permutation to get a plaintext that effectively cancels those key bits for that s-box. This key preimage is xored with the elements of a zero-key preimage set, and these values are passed into the white-box DES. The s-box output bitmask is used to isolate the results, and the four results are compared. If they are not all equal, then the candidate key bits are incorrect, and the algorithm moves on. If all four results are equal for all preimage sets, the candidate key bits are correct[1].

At this point we have the first round key and one alternative hypothesis (see[1]), and the algorithm is able to generate key schedules and perform brute force search of the 2^9 possible keys to find a plaintext/ciphertext pair matching one created with the white-box DES. From start to finish, a largely unoptimized version our attack takes approximately three seconds on an 800 MHz G4 processor.

Our statistical bucketing routine returns a DES key when given a complete instance of Chow et al.s DES function and a crippled version that returns the output of the first round of T-boxes. We supplied a crippled function for expediency, but this is not necessary; a function that returns T-box output can be automatically generated by analyzing the encryptors control flow. We have not implemented this portion of a full attack.

[1] The sole exception is s-box 4, for which the preimage sets are the same as those for inputs that were xored an with 101111b prior to entering s-box 4. This property was first reported by Shamir in [Sha85]

5 Implementation Improvements

Because we record preimage sets considering the entire s-box output, we can reduce the hypothesis space for each 6-bit portion of the first round key to a single value. This makes the attack faster than that described by Chow et al., which considered only one bit of s-box output. With minimal algorithmic optimization, the entire process takes about three seconds on our reference platform. The statistical bucketing attack depends upon the separation of the permuted s-box results from the other T-box output bits. The s-boxes are lossy, and so the preimages have multiple elements and may be compared with one another even when the output values are renamed. When T-box output is a permuted 8-bit value, the T-boxes are bijections and preimage comparison is no longer useful.

Chow et al. admitted that their white-box DES implementation is vulnerable to a statistical bucketing attack on the input to the second round of T-boxes, but nevertheless they recommend 4×4 blocking for the matrices making up DES. This exposes their implementation to our more aggressive statistical bucketing attack, which exploits the separation of s-box output and supplemental output produced by the T-boxes.

Using an 8×4 block size rather than a 4×4 block size prevents our new attack. Although it would map well to the input and output of fully-delinearized T-boxes, 8×8 io-block encoding of the M_is unfortunately produces addition tables that map 16 bit values to 8 bit values, for a total of more than 250 MB of tables to implement a single encryption function! Using an 8×4 block size allows the T-box output to produce an eight-bit permuted value from the concatenation of two four bit values, and the multiplication and addition tables that make up the io-block encoded matrix become 8×4 tables. The result is a DES implementation whose T-box output does not leak information needed by the statistical bucketing attack. The new DES is the same size as Chow et al.s, because the addition tables in both are the same size, and these tables make up the bulk of white-box DES.

Further discouraging analysis of the T-box output is possible by completely eliminating it as an accessible intermediate value. Once an M_i has been io-block encoded using an 8×4 block, the initial 8×4 multiplication tables accept the same size output as the T-boxes produce, and the results of an 8×8 T-box and a corresponding 8×4 multiplication table can be precomputed to form a single 8×4 table. Each T-box result is fed into multiple multiplication tables, so ultimately this precomputation produces a replacement for each multiplication table and eliminates the T-boxes. (Chow et al. used the same approach when folding each pair of 4×4 multiplication tables into the subsequent addition table.) After this process, our entire DES implementation consists of 8×4 tables, specifically multiplication tables (for the first AT only), fused T-box/multiplication tables, and addition tables. Chow et al.s implementation used 8×8 T-boxes (split to take two 4-bit input chunks and produce two 4-bit output chunks), and 8×4 vector addition tables.

To prevent the original statistical bucketing attack on the input to the second round of T-boxes, we must disrupt the preimage of each input bit by intermixing the s-box input with L_r and R_r. We mix the 16 replicated bits of R_r with L_r by including a random AT, μ, in the M_is[2], and delinearize the T-box input as a single 8-bit block. Moving to an 8×8 block size for the M_is would allow us to pass the T-boxes 8-bit blocks instead of two concatenated 4-bit blocks, but as we have said this would produce an intolerably large DES implementation. Once the 8×4 io-block encoding has been prepared, however, it is possible to use combined function encoding [CEJv02a] on the final addition tables of each pair of trees (see Figure 2) in the io-block encoding, whose output corresponds to a T-box. The combined function encoding adds twelve 16×8 tables before each round of T-boxes in the place of twice that many 8×4 tables.

The fused T-box and multiplication table could be fused into the 16×8 addition tables, but this would enlarge the implementation once again by duplicating the 16×8 tables a dozen times. We recommend the combined function encoding of each pair of addition tables that produces T-box input and the creation of an intermediate value for T-box input that is obfuscated in 8-bit blocks. Once these modifications are made, each T-box input block includes six s-box input bits and two bits containing information about the left and replicated bits. This makes it more difficult to identify which T-box corresponds

[2] According to [JBF02], Chow et al. used a simple permutation for μ instead of an affine transform

	MiB	Packed MiB
Chow et al. (4×4)	4.54	2.27
8×4	4.49	2.25
8×8	274.75	274.75
8×4, Joined Roots	16.40	14.20
8×4, JR, 3DES	48.83	42.42

MiB is the number of mibibytes (2^{20} bytes) of tables if 4-bit values are stored as 8-bit characters (for efficiency of reference). Packed MiB is the number of mibibytes of tables if two 4-bit values are stored per 8-bit character.

Table 1 Comparison of white-box DES implementations.

to which s-box in the second round and removes the association between preimages and individual bits of input, preventing the attack.

In practice a few optimizations made disproportionate improvements in the time and space required to generate an instance of DES (Table 1). Precomputing constant matrices used in the ATs for DES reduced the AT multiplications that had to be done at generation time. Declaring the types of our matrices to the Lisp compiler allowed elements to be packed more densely and manipulated with unboxed instructions by the compiler. Shrewd preservation and reuse of memory allocated to matrices and permutations reduced the frequency of garbage collection and saved a great deal of time spent initializing those data structures. Finally, a surprising amount of time during generation was devoted to creating and applying small random permutations, so even small performance improvements to these functions made a large difference in the later stages of optimization.

We implemented white-box DES using several approaches, including the 4×4 io-block encoding of Chow et al., to compare the time and space requirements. The test platform was an 800Mhz PPC G4 running Mac OS X 10.2.6 and MCL 5.0b5. We implemented both our modified systems and Chow et al.s original algorithm in Lisp in this environment.

6 Differential Fault Injection Attack

Jacob et al. describe an attack on the white-box DES implementation of Chow et al., in which faults are inserted to enable differential cryptanalysis of the embedded DES s-boxes. The attack is a very efficient one, requiring just dozens of calls to a decryption oracle and a similar number of encryptions using the white-box implementation to recover 48 bits of the embedded key. The other 8 bits can be recovered via brute force attack using a reference implementation of DES. We describe a simplified interpretation of the attack below. Throughout this description we assume the adversary has a white-box DES encryption function, and we will use the following notation:

$\mathcal{E}^k_{d_0,d_1}(l,r)$:	This means to run the white-box encryptor, with embedded key k, for rounds d_0 through d_1 on the plaintext that consists of l in the left block and r in the right. If only d_0 is specified, then only that one round of encryption will be performed at that time. The resulting 96-bit value, σ_{d_1} will be the obfuscated input to the T-boxes of round d_1+1.
$\mathcal{D}^k(l,r)$:	This means to call the decryption oracle with embedded key, k, on the ciphertext consisting of l in the left half, and r in the right. The resulting 64-bit value, l_0, r_0 will be the resulting plaintext.

We will also assume that the attacker is working on the ciphertext at the output of round 16 (denoted by the blocks L_{16} and R_{16}, i.e. before the final DES permutation. This is a valid assumption, as the final permutation is well known and easily invertible.

The fault injection attack proceeds in four stages:

- **Determine the representation of** $L_{15} = f^k_{16}(0), R_{15} = 0$

  This is done by first computing $l_0, r_0 = \mathcal{D}^k(0,0)$, and then computing $\sigma_{15} = \mathcal{E}^k_{1,15}(l_0, r_0)$ (see Figure 4(a)). This is the obfuscated representation of $L_{15} = f^k_{16}(0), R_{15} = 0$ and will be used later in the attack, where we will refer to it as Σ_{15}.

- **Determine which segments of** σ_{15} **affect which bits of** L_{16}

  This is done by first computing $l_i, r_i = \mathcal{D}^k(2^i, 0)$ and then computing $\sigma^i_{15} = \mathcal{E}^k_{1,15}(l_i, r_i)$ for $0 \leq i < 32$ (see Figure 4(b)). The σ^i_{15} can then be compared to Σ_{15} to determine which 4-bit blocks have changed for each i. This also tells the attacker which T-boxes are the "real" T-boxes, as well as to which s-box they correspond.

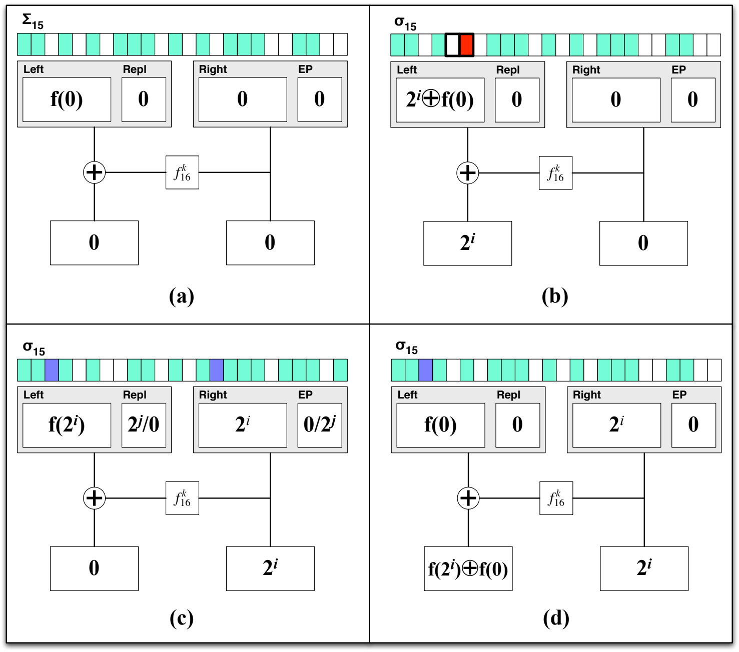

Figure 4 The attack proceeds by selecting ciphertexts, consulting a decryption oracle, recording the next to final results from encryption, and selectively overwriting these intermediate results prior to completing the encryption.

- **Determine $f^k_{16}(0) \oplus f^k_{16}(2^i)$**

  First compute $l_i, r_i = \mathcal{D}^k(0, 2^i)$, then compute $\sigma^i_{15} = \mathcal{E}^k_{1,15}(l_i, r_i)$ for $0 \leq i < 32$ (see Figure 4(c)). In the second step, it can be determined which T-box is affected by the bit that is set in the right half of the ciphertext, and in particular, which 4-bit block(s) in σ^i_{15}. The attacker can now swap in all of the blocks of Σ_{15} that will not overwrite the 4-bit block that is affected by 2^i, call this new value σ'^i_{15}.[3] This will have the effect of changing L_{15} from $f^k_{16}(2^i)$ back to $f^k_{16}(0)$ (see Figure 4(d)). Now, computing $\mathcal{E}^k_{16}(\sigma'^i_{15})$ results in $L_{16} = f^k_{16}(0) \oplus f^k_{16}(2^i)$, which will be used in the final step.

- **Perform differential cryptanalysis on the s-boxes**

  Given up to six different $f^k_{16}(0) \oplus f^k_{16}(2^i)$ for each s-box that the various 2^i will fall into, the attacker now performs differential cryptanalysis on each s-box to recover the 6-bits of the final round sub-key that goes into each s-box. Once the 48-bit sub-key is recovered, use brute force on a reference implementation of DES to recover the other 8 bits of the key. Note that the technique described in the previous step is guaranteed to produce at least four valid values of $f^k_{16}(0) \oplus f^k_{16}(2^i)$, which may result in the attacker recovering as few as 40 of the 48 bits of the final round sub-key, in which case the brute-force portion of the attack will require up to 2^{16} trial encryptions. Slightly more involved techniques in step three, however, can retrieve all 48 bits of the sub-key.

This fault injection attack takes advantage of two of the design decisions made by Chow et al. in order to expose the final round subkey: the splitting of the 8-bit T-box into two 4-bit values, and the specification of μ as a straight permutation. This is done as follows:

1. The four bit blocking ensures that the "swapping in" of the representation of $f^k_{16}(0)$ will harm at most two right half bits in the "real" T-boxes (the only T-boxes we care about in any case), thus guaranteeing at least four valid $f^k_{16}(0) \oplus f^k_{16}(2^i)$ data points per s-box – more than enough to carry out a differential attack on the

[3] This step will need to be repeated twice for those bits that are replicated by the expansion permutation, once for each 4-bit block that will be affected.

s-boxes.

2. Because μ is a straight permutation of bits, a one bit change to L_{16} results in a single bit change in L_{15} (and vice versa) This means that only a small subset of the 4-bit blocks that represent $f_{16}^k(0)$ need to be "swapped in" in order to learn the value of $f_{16}^k(0) \oplus f_{16}^k(2^i)$ for any single s-box. Acting in concert with the note above, this makes it likely that an attacker can gain more than 4 valid $f_{16}^k(0) \oplus f_{16}^k(2^i)$ data points for each s-box, improving the efficiency of the attack.

Our improvements to white-box DES remove both of those leverage points. The first is affected by our introduction of 8-bit blocking at the inputs to the T-boxes. If a bit is flipped in the right half of the ciphertext, this change will be destroyed if the bit is anywhere in one of the T-boxes that are overwritten when $f_{16}^k(0)$ is swapped back into L_{15}. With the 4-bit blocking of Chow et al., four input bits are guaranteed to never be overwritten, thus four valid $f_n^k(0) \oplus f_n^k(2^i)$ data points are guaranteed to be computed in every DES s-box. In our implementation, there are no such guarantees.

The second leverage point is removed by replacing the simple permutation μ with a random affine transform. Any single bit change to L_{16} will result in an expected change in half of the bits of L_{15} (as well as the replicated bits of the right half). Since each real T-box contains two of these bits, we can expect that 6 of the 8 real T-boxes (and 10 of the 12 total T-boxes) will be affected by a single bit change in L_{16}. What this implies is that in order to reset L_{15} so it contains $f_{16}^k(0)$ in at least the four bit positions that contain the s-box we are attacking, we will likely have to swap in all twelve 8-bit blocks of Σ_{15}.[4]

Unfortunately, the 8-bit blocking does not completely eliminate the threat of the fault injection attack. This is because each real T-box contains only two of the mixed left-half and replicated right-half bits. And so if the attacker always swaps in the eleven 8-bit blocks of Σ_{15} that do not overwrite the bit of interest set in the right half, there is still a chance that those two bits, which correspond to two bits of the representation of $f_{16}^k(2^i)$ may just be the same as the equivalent bits of $f_{16}^k(0)$. This event occurs with probability 0.25.

If this happens, L_{15} will be reset to $f_{16}^k(0)$ after completing the encryption, L_{16} will contain $f_{16}^k(0) \oplus f_{16}^k(2^i)$ for the s-box of interest, and the attacker will have gained a valid data point to use in differential cryptanalysis. The attacker can determine if this has happened because there will be at most two non-zero 4-bit blocks in the left half[5]. If this has not happened, most if not all of the 4-bit blocks of the left half will be non-zero with overwhelming probability.

In our improved implementation of white-box DES, an attacker still has a 0.25 probability of obtaining a valid data point to use in the differential cryptanalysis, with up to six data points per s-box. The amount of information gained varies according to the s-box being attacked, the position of the injected fault within the s-box, and the number of data points obtained. As an example, this information is fully described for s-box 1 in Table ??. However, the attacker will gain an expected 1.22 effective bits[6] of the final round sub-key per s-box. Based on this measurement, we have improved the resiliency of white-box DES against this attack from surrendering a guaranteed 40 bits of the final round key (and all 48 with high probability), to surrendering an expected 9.8 bits of the final round key. While this is still not as strong as a black-box implementation, it is strong enough to make an attack on the triple-DES variant of our implementation infeasible.

7 Application to 3DES

When extending the technique to 3DES, the final matrix from each DES portion ($M_3 M_2$) is combined with the first matrix from the following DES (M_1). This eliminates intermediate values that would enable the implementation to be compromised in thirds. Using $M_1 M_3 M_2$ as a single io-block encoded matrix between the three DES portions requires the entire 3DES be attacked as a whole. In addition it saves both space and time, causing the whole of 3DES to be less than three times as large and take

[4] The probability of this not occurring is approximately $2^{-8.7}$

[5] The block corresponding to the s-box of interest plus one if the bit set in the right half was copied by EP. The adversary can tell which is which based on which s-boxes affect which bits in the final output.

[6] The attacker may not necessarily determine any actual bits of the sub-key, but is still able to reduce the search space of possible keys that need to be checked during brute force search.

		Bits 4-6							
		000	100	010	001	110	101	011	111
Bits 1-3	000	0.00 0.00 0.00	2.68 3.31 5.00	2.42 3.32 5.00	2.42 3.24 5.00	3.68 5.42 6.00	3.68 5.51 6.00	3.42 5.40 6.00	5.00 5.88 6.00
	100	2.42 3.08 5.00	3.68 5.14 6.00	4.00 5.43 6.00	3.42 5.33 6.00	5.00 5.84 6.00	4.00 5.88 6.00	5.00 5.91 6.00	5.00 5.97 6.00
	010	2.19 2.99 5.00	4.42 5.40 6.00	3.68 5.37 6.00	3.42 5.14 6.00	5.00 5.91 6.00	4.42 5.93 6.00	4.00 5.84 6.00	6.00 6.00 6.00
	001	2.42 3.24 5.00	3.42 5.15 6.00	3.68 5.43 6.00	3.42 5.40 6.00	5.00 5.94 6.00	4.42 5.86 6.00	4.00 5.81 6.00	5.00 5.97 6.00
	110	3.68 5.39 6.00	5.00 5.78 6.00	5.00 5.94 6.00	4.00 5.84 6.00	5.00 5.97 6.00	5.00 5.97 6.00	6.00 6.00 6.00	6.00 6.00 6.00
	101	3.68 5.43 6.00	4.00 5.81 6.00	5.00 5.91 6.00	4.41 5.86 6.00	5.00 5.97 6.00	5.00 5.97 6.00	5.00 5.97 6.00	6.00 6.00 6.00
	011	3.68 5.38 6.00	5.00 5.88 6.00	5.00 5.97 6.00	5.00 5.91 6.00	6.00 6.00 6.00	6.00 6.00 6.00	5.00 5.97 6.00	6.00 6.00 6.00
	111	5.00 5.97 6.00	6.00 6.00 6.00	6.00 6.00 6.00	6.00 6.00 6.00	6.00 6.00 6.00	6.00 6.00 6.00	6.00 6.00 6.00	6.00 6.00 6.00

Cell entries show the minimum, mean, and maximum information gain given successful fault injection attacks on bits of s-box 1 input bits 1 through 6 as indicated by the row and column.

Table 2 Comparison of white-box DES implementations.

less than three times the time as the encoded DES.

Our statistical bucketing attack applies to a 3DES with split-output T-boxes. Once we extract 47 bits of the first key, leaving 9 bits of uncertainty (for the remaining 8 bits and the two possible 6-bit key portions associated with s-box 4), we have 512 hypotheses for what the first round encryption result could be. This equates to 512 hypotheses for what would result (implicitly) in a zero result after the first DES (or a zero result xored with a 6-bit first-round subkey for the second DES), giving 512 possible 48-bit first round keys (at most; it may be possible to eliminate some of these). Ultimately this leads to $2^1$8 hypotheses for DES keys 1 and 2 when extracting a round key from the final iteration of DES. The resulting search space is $2^2$7 to recover the full key. It is also worth noting that once the initial 512 hypotheses are acquired, they can be investigated independently, parallelizing the attack.

We are inclined to recommend 3DES rather than DES for any conceivable use case, because the malicious host is able to generate as many plaintext-ciphertext pairs as needed.

8 Future Work

Further improvements in the size and security of white-box DES may be possible. Joining the roots of the vector addition trees to prevent statistical bucketing attacks on the t-boxes may not be necessary for any but the first and final few T-boxes, for example. Avoiding it in the interior of DES would greatly reduce the size of the implementation, and would lead to even greater savings in a 3DES implementation. It is also possible to eliminate the dummy T-boxes in favor of eight 12-bit T-boxes. This would improve the security of the system further against the differential cryptanalysis attack of Jacob et al., reducing the adversarys expected gain to 0.05 effective bits per T-box, totaling 0.40 effective bits. This security comes at a substantial increase in size if 12-bit T-boxes are used throughout DES, so it is worth evaluating the potential of reverting to 8-bit, split-input T-boxes for the interior operations to compensate.

Chow et al. speculates the possibility of matrix analysis enabled by sparse tables multiplication tables containing only a few of the 2^b possible output values. We have not yet implemented our solution to this, but it should be possible to give individual values in the table many obscuring names instead of only one, and compensate for this when computing the contents of the following tables. Analysis of the utility in attacking the matrix before and after such modifications is necessary. Some analysis is also necessary to confirm that the new system is resilient to other forms of cryptanalysis. While we believe that the four 4-bit values passed to the joined root in our current implementation are not as susceptible to cryptanalysis as the original split T-box inputs were, we have not yet demonstrated this.

This additional work should ultimately result in a white-box implementation of 3DES that is as compact as possible and provides black-box level security.

References

[AF90] Martín Abadi, Joan Feigenbaum (1990). Secure circuit evaluation: A protocol based on hiding information from an oracle. *Journal of Cryptology*, 2(1):1–12.

[ACCK00] Joy Algesheimer, Christian Cachin, Jan Camenisch, Günter Karjoth (2000). Cryptographic security for mobile code. Technical Report,

[BGI+01] Boaz Barak, Oded Goldreich, Rusell Impagliazzo, Steven Rudich, Amit Sahai, et. al. (2001). On the (im)possibility of obfuscating programs. *Lecture Notes in Computer Science*, 2139:1–??.

[CEJv02a] Stanley Chow, Phil Eisen, Harold Johnson, Paul van Oorschot (2002a). A white-box DES implementation for DRM applications. In *Proceedings of DRM 2002 - 2nd ACM Workshop on Digital Rights Management*.

[CEJv02b] Stanley Chow, Phil Eisen, Harold Johnson, Paul van Oorschot (2002b). A white-box cryptography and an AES implementation. In *Proceedings of SAC 2002 - 9th Annual Workshop on Selected Areas in Cryptography*, number 2595 in Lecture Notes in Computer Science, pages 250–270. Springer.

[CTL97] Christian Collberg, Clark Thomborson, Douglas Low (1997). A taxonomy of obfuscating transformations.

[CTL98] Christian Collberg, Clark Thomborson, Douglas Low (1998). Manufacturing cheap, resilient, and stealthy opaque constructs. In *Principles of Programming Languages 1998, POPL'98*, pages 184–196.

[Hoh98] Fritz Hohl (1998). Time limited blackbox security: Protecting mobile agents from malicious hosts. *Lecture Notes in Computer Science*, 1419:90–111.

[JBF02] Matthias Jacob, Dan Boneh, Edward Felten (2002). Attacking an obfuscated cipher by injecting faults. In *ACM CCS-9 Workshop DRM.*

[Mit99] W.P.R. Mitchell (1999). Protecting secret keys in a compromised computational system. In Andreas Pfitzmann, editor, *Information Hiding*, number 1768 in Lecture Notes In Computer Science, pages 448–462. Springer.

[ST98a] Tomas Sander, Christian F. Tschudin (1998a). Protecting mobile agents against malicious hosts. *Lecture Notes in Computer Science*, 1419:44–??.

[ST98b] T. Sander, C. Tschudin (1998b). Towards mobile cryptography. In *Proceedings of the IEEE Symposium on Security and Privacy.* IEEE Computer Society Press.

[Sha85] Adi Shamir (1985). On the security of DES. In [Wil86], pages 280–281.

[WDHK01] C. Wang, J. Davidson, J. Hill, J. Knight (2001). Protection of software-based survivability mechanisms.

[Yao86] Andrew Chi-Chih Yao (1986). How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science*, pages 162–167, Toronto, Ontario, Canada. IEEE.

[css] http://www.dvdcca.org/css.

[Wil86] Hugh C. Williams, editor (1986). *Advances in Cryptology - CRYPTO '85, Santa Barbara, California, USA, August 18-22, 1985, Proceedings*, number 218 in Lecture Notes in Computer Science. , Springer.