Side Channel Cryptanalysis of Product Ciphers

John Kelsey[1], Bruce Schneier[1], David Wagner[2], and Chris Hall[3]

[1] Counterpane Internet Security
3031 Tisch Way, 100 Plaza East
San Jose, CA 95128
e-mail: {kelsey,schneier}@counterpane.com

[2] U.C. at Berkeley
Soda Hall
Berkeley, CA 94720-1776
e-mail: daw@cs.berkeley.edu

[3] Princeton University – Mathematics
Fine Hall, Washington Road
Princeton, NJ 08544-1000 USA
e-mail: cjh@math.princeton.edu

Abstract. Building on the work of Kocher[Koc96], Jaffe, and Yun [KJY98], we discuss the notion of *side-channel cryptanalysis*: cryptanalysis using implementation data. We discuss the notion of side-channel attacks and the vulnerabilities they introduce, demonstrate side-channel attacks against three product ciphers—timing attack against IDEA, processor-flag attack against RC5, and Hamming weight attack against DES—and then generalize our research to other cryptosystems.

Keywords: side channels, cryptanalysis, timing attacks, product ciphers.

1 Introduction

Any cryptographic primitive, such as a block cipher or a digital signature algorithm, can be thought of in two very different ways. It can be viewed as a mathematical object; typically, a function taking an input between 0 and $2^N - 1$, and providing an output between 0 and $2^M - 1$. Alternatively, it can be viewed as a concrete implementation of that mathematical object. Traditionally, cryptanalysis has been directed solely against the mathematical object, and the resultant attacks necessarily apply to any concrete implementation. The statistical attacks against block ciphers—differential [BS91,BS93] and linear [Mat93] cryptanalysis—are examples of this; these attacks will work against DES regardless of which implementation of DES is being attacked.

Many of these attacks are more theoretical than operational, and are sometimes called "certificational weaknesses" to indicate that they may not work

against practical implementations. A known-plaintext attack that uses 2^{40} plaintexts, for example, would require an operational cryptanalyst to obtain just under nine terabytes of data (assuming a 64-bit block cipher) encrypted in a single key. While no cryptographer would seriously recommend such an algorithm for operational deployment, it wouldn't be too hard to build a system that denied an attacker access to this magnitude of plaintext.

In the last few years, new kinds of cryptanalytic attack have begun to appear in the literature: attacks that target specific implementation details. Both timing attacks [Koc96] and differential fault analysis [BDL97,BS97] make assumptions about the implementation, and use additional information garnered from attacking certain implementations. Failure analysis [HGS97,Bel96] assumes a one-bit feedback from the implementation—was the message successfully decrypted—in order to break the underlying cryptographic primitive. More recently, differential power analysis [KJY98] (sometimes referred to as "DPA" in the remainder of this article) has been developed and applied to a number of cryptographic implementations.

These attacks don't necessarily generalize—a fault-analysis attack just isn't possible against an implementation that doesn't permit an attacker to create and exploit the required faults—but they can be much more powerful. For example, differential fault analysis of DES requires between 50 and 200 ciphertext blocks (no plaintext) to recover a key [BS97].

In this paper, we consider the general class of *side-channel* attacks against product ciphers. A side-channel attack occurs when an attacker is able to use some additional information leaked from the implementation of a cryptographic function to cryptanalyze the function. Clearly, given enough side-channel information, it is trivial to break a cipher. An attacker who can, for example, learn every input into every S-box in every one of DES's rounds can trivially calculate the key. Our research attempts to show how little side channel information is required to break product block ciphers.

In real-world systems, attackers cheat; prudent engineers of secure systems anticipate this and adapt to it. Exploiting weaknesses in implementations—either by monitoring some "side-channel" of information out of the mechanism implementing the cryptographic primitive (such as timing or power consumption), or by altering some internal data inside that mechanism—may feel like cheating, but that just makes their effects more devastating.

1.1 Focus of this Paper

Our work, while applicable to practical attacks, is essentially theoretical in nature. We investigate what kinds of side-channels will work against product block ciphers, such as DES, IDEA, and RC5. Our goal is to come to a better understanding of what kinds of side-channels can be used to threaten various product block ciphers. We focus on these ciphers, rather than on broader classes of cryptographic mechanism (such as stream ciphers, public-key signature schemes, hash

functions, or pseudorandom number generators) for simplicity, not because they are more susceptible to side-channel attack than these other primitives.

1.2 Summary of Our Results

We develop a number of tools for dealing with side-channel attacks on product block ciphers. In particular, we consider side-channels with relatively low correlation to any single bit of internal state of the cipher, as opposed to the side-channels discussed in preliminary explanations of DPA.

We show the close relationship between side-channel attacks and other well-known attacks on product block ciphers, particularly partitioning attacks and chi-square attacks.

We discuss side-channel attacks on IDEA, RC5, and DES. We emphasize that none of these attacks reflect inherent weaknesses in these ciphers; all are based on the availability of some usable side-channel. In IDEA, the timing side-channel was available in a commonly-used implementation. The attacks on RC5 and DES are more theoretical attacks. All these attacks have the property that they focus not on some single bit of internal state, as in DPA, but on some function of the internal state that is often only very weakly correlated any given bit of internal state.

2 Side Channels and Product Ciphers

Symmetric encryption is most often done with product block ciphers, such as DES [DES81], IDEA [LMM91], and Blowfish [Sch94]. To understand why side channels so often provide devastating attacks against product ciphers, it is necessary to digress a bit into the theory of block cipher design and cryptanalysis.

A block cipher is a cipher that encrypts whole blocks of plaintext at a time; all the ciphers considered in this paper operate on blocks of 64 bits at a time. A product cipher (sometimes called an iterative block cipher) is a block cipher made by iterating a fairly simple round function many times, each time with its own key. Thus, a four-round product cipher would look like:

$$E_K(X) = R_{K_3}(R_{K_2}(R_{K_1}(R_{K_0}(X)))),$$

where $K_{0..3}$ are functions of K.

Each round function, $R_K()$, is actually a weak block cipher, in the sense that if an attacker knows much about its input (such as simple statistical patterns), he can quickly break the round function and recover its round key. However, if the same attacker is faced with a sequence of random inputs, each encrypted with the round function, he has no way to mount such an attack.

The attacker might be able to immediately identify outputs from one round, given knowledge of the input. Outputs from a second round will be harder to

identify. As successive rounds are applied to the block, it becomes less and less predictable to the attacker. After a certain number of rounds, the attacker cannot distinguish the block's contents from a random permutation applied to the input.

We may consider the entire *internal state* of the block cipher to be all the intermediate results and values that are never output in normal operations. For example, DES has 16 rounds; we can consider the intermediate states, $state[1..15]$, after each round except the last as secret internal state. Side channels typically give information about this internal state, or about the operations used in the transition of this internal state from one round to another.

Existing attacks on product block ciphers depend upon information about this internal state. In a linear attack [Mat93], a subset of bits in the input to the cipher and to the last round don't quite have a balanced parity; in a differential attack [BS91,BS93], the relationship between a pair of texts' inputs to the last round isn't quite random. Partitioning attacks, higher-order differential attacks, differential-linear attacks, and related-key attacks all fit into this pattern. A strong product cipher will have as the input to its last round a random permutation of the input to the cipher: something an attacker can't distinguish from a random number. The attacks typically work by finding some information about the internal state of the cipher which can be learned both by guessing part of the key and checking the value directly, and by some statistical property of the cipher that makes that checkable value slightly nonrandom.

Cryptanalytic attacks like linear and differential cryptanalysis make use of very small statistical imbalances in the internal state of the cipher. Because of this, even side-channel measurements with only a very small correlation to any internal state bit can be used to break a cipher like DES or IDEA. In [Mat93], a rule-of-thumb is provided: If a bit can be predicted with probability $1/2 + b$, that bias can be detected given about 2^{-2b} samples.

2.1 Partitioning and Chi-Square Attacks and Side-Channels

A partitioning attack[HM97] on a product block cipher seeks to find some way to partition the possible values of intermediate states of the cipher such that the intermediate states of the cipher fall into some partitions more often than others, for some set of plaintexts or ciphertexts. Linear cryptanalysis is one instance of a partitioning attack, with only two partitions. A similar kind of attack, known as a "Chi-Square" attack, was independently developed by Vaudenay [Vau96].

In standard partitioning attacks, we can make use of some imbalance in the probability of intermediate states in successive rounds landing in the different partitions to distinguish the input to the last round of a cipher. The side-channel attacks we have seen thus far can always be considered as partitioning attacks, in which the information about the partition in which some intermediate state lies is learned from the side-channel, rather than because of propogation through the cipher. Such properties as whether a given 16-bit subblock is a zero, or the Hamming weight of the block in the input to the last round, simply give us ways

to partition the possible states of the block. We may then do a partial key guess to learn which partition the input to the last round belongs to, and use this to learn that part of the key.

3 Differential Power Analysis and Related Ideas

Differential power analysis (DPA) [KJY98] is method of implementing a side-channel attack using large numbers of power or other measurements. Consider an attack on a chip implementing DES: The attacker divides the time taken by the encryption operation into V intervals. For each of N encryptions under a given key, the attacker takes one power measurement in each of the V intervals. It is expected that some relatively small number of those power measurements will correlate with each bit of internal state used during the encryption. (If the measurement values were uncorrelated with any internal state bits, then this attack would fail.)

The attacker guesses six bits of the first- or last-round subkey, and uses this to compute four bits of internal state. He uses one bit of this state; call this bit b. For a sequence of N ciphertexts, each guess gives the attacker a sequence of N of these bits, $b[0, 1, ..., N-1]$. The attacker wants to determine whether his current guess of that part of the key, and thus his current guess of the sequence $b[0, 1, ..., N-1]$, is right.

Let $X[i][j]$ be the power measurement taken during encryption i, from time interval j, and $X[0..N-1][j]$ be the sequence of measurements taken during time interval j, for each encryption.

For $0 \leq j < V$, the attacker computes

$$T[j] = \sum_{i=0}^{N-1} (1/2 - b[i])X[i][j]$$

If the bit sequence $b[i]$ is uncorrelated with the sequence $X[0..R-1][j]$, then this will average to zero. If there is some correlation, then it will average to either higher or lower than zero, depending on the sign of the correlation coefficient. The guess of part of the key with the largest deviation of $T[j]$ values from zero is most likely to be the right value. (When the bit sequence is uncorrelated with the sequence of power measurements, the distribution is normal with mean $= 0$ and standard deviation $= s/4$, where s is the original standard deviation of the normally-distributed power measurements.)

One really powerful aspect of this attack is that the attacker need not know much about the internals of the implementation. He needs to know only the values of some bits that must be used in the attack. The attack yields both the key, and enough information to derive a model of the device which can be used to extract keys in the future with far fewer measurements and encryptions.

There are a number of obvious extensions to this attack. However, the underlying requirement is that these measurements be correlated with some bit of internal state.

Two of the three attacks described in this paper involve side-channels which would have a relatively low correlation with any individual bit. In this sense, these attacks would be relatively hard to find with DPA, assuming that only the side-channels discussed were present. In particular, in the IDEA attack, we use timing information to determine the number of zero inputs into multiply operations. A side channel that simply gives the number of zero inputs to multiply instructions has a very weak correlation with any given internal state bit, and so would be difficult to exploit directly with DPA. Similarly, Hamming weights, while correlated with individual internal state bits, are not correlated very strongly with any single bit. In both cases, testing that revealed no vulnerability to straightforward DPA given N trial encryptions might easily fail to detect these side-channel attacks.

4 Timing Attacks Against IDEA

IDEA [LMM91] is a product block cipher designed by Lai, Massey, and Murphy, to resist all forms of cryptanalysis publicly known at that time. Except for some special "weak-key" conditions, the full 8-round IDEA is apparently very strong against the standard differential, linear, and related- and chosen-key attacks. However, IDEA can be cryptanalyzed with a piece of side-channel information: whether one of the inputs into one of the multiplications is zero.

This side channel can be obtained in several ways. Since the multiplication is done modulo $2^{16}+1$, a zero operand is treated as a special case. Some implementations bypass the multiplication completely and simply patch in the correct value. Timing[1], power consumption, radiation, etc., will all be different if one input is a zero. For the rest of this discussion, we assume that this information is received through timing measurements. However, any side channel that can yield the same information is just as effective.

4.1 The Basic Attack

The basic mechanism used in all of these attacks is to search for one or two 16-bit values visible to the cryptanalyst which indicate a zero input into a multiply somewhere in the cipher.

The time taken to encrypt one block with IDEA may be broken into three parts:

1. Operations which take approximately constant time. These may be assumed to include all operations except the multiplications modulo $2^{16}+1$.

[1] This observation was borne out experimentally using the PGP 2.3 implementation of IDEA, on a 486SX-33.

2. All the multiplications except for the one being targeted. This is probably approximately normally distributed.
3. The multiplication being targeted. If this has a zero input, it will invariably take less time than if it has a nonzero input.

The time taken by the whole encryption process is the sum of these three times. If the multiplication being targeted has a zero input, then the mean of this time will be slightly lower than if the multiplication being targeted does not have a zero input. If we can reliably detect this difference, then we can detect whether or not the multiplication operation being targeted is getting zero inputs.

Timings can be acquired in two simple ways: Either the cryptanalyst makes extremely precise timings of each encryption, or he measures total time to encrypt many similar plaintext blocks at a time. In either case, however, he must then statistically test the hypothesis that one of these sets of times is significantly lower than the others.

A Ciphertext-Only Attack Requiring Precise Block Encryption Timings With precise timing of each encryption, we can recover the key using either plaintext or ciphertext only. An example of a situation where this might be useful is in attacking a cryptographic PRNG implemented using IDEA. IDEA run in counter-mode, or used as the underlying block cipher for the ANSI X9.17 key generator, will not give an attacker direct access to inputs to the block cipher, but gives the attacker access to its outputs.

The attack works as follows:

1. Record precise timings for n encryptions. Also store the resulting ciphertexts. Let $T_{0..n-1}$ be the timings, and $C_{0..n-1}$ be the ciphertext blocks.
2. Group the ciphertexts and timings into 2^{16} subsets, based on the low-order 16 bits of the output.
3. Test the average times of each group against the average times of all the groups statistically, to find whether one of the sets has (with some acceptably high probability) a lower average than the other sets. If so, then the inputs to the last multiply of the output transformation must have been 0 for all inputs in that set. Solve for the last multiplicative subkey. If there is no difference, then either we've chosen some parameters (i.e., n) wrong, or the subkey is a 0.
4. Repeat steps 2-3, above, for the high-order 16 bits and the first multiplicative subkey of the output transformation. We now have 32 bits of expanded key.
5. We now attack the second additive subkey in the output transformation. For each possible value of this subkey, we look at which ciphertexts lead us to a zero value going into the first multiplication of the last round's MA box. For one of these subkey guesses, the average timing should be less than for all the other subkey guesses. This reveals the right subkey. If there is no difference, then either we've chosen some parameters wrong, or the first subkey in the MA-box is zero. We have now recovered 48 bits of expanded key.

6. We now attack the first additive subkey in the output transformation, and the first subkey in the MA-box. We do this as follows:
   (a) Break the ciphertexts and timings up into 2^{16} subsets based on the value of the leftmost (first) input to the MA-box.
   (b) For each possible subkey value for the first additive subkey of the output transformation, break each subset up into 2^{16} sub-subsets, based on what the value of the second MA-box input would be if this were the right subkey.
   (c) For the right subkey, each subset will have one sub-subset which has a smaller timing value than all the other sub-subsets in that subset. We have now found 64 bits of subkey.
   (d) We now choose any three of these sub-subsets, and use them to solve for the first multiplicative subkey of the MA-box. We have now found 80 bits of subkey, and can brute-force the remaining 48. (There are also ways to continue this attack.)

This kind of attack might also be practical for recovering the key from a tamper-resistant box which always encrypts under the same IDEA key. The cryptanalyst does not need to know anything about the plaintext for this attack, but must always know precisely when the encryption started and when it ended. Chaining modes have no real impact on this attack.

4.2 An Adaptive Chosen Plaintext Timing Attack

An attack very similar to the one described above is possible, even if the cryptanalyst doesn't have the ability to measure each encryption time precisely. In this case, we require the ability to choose plaintext batches to send through the encryption algorithm, along with the ability to time these batches.

The attack works as follows:

1. Choose 2^{16} batches of plaintexts large enough that changing the input of an average of one multiply instruction per encryption to zero will be detectable in the timing, with high probability. (We need to specify numbers for the timing statistics here before we can know how many plaintexts must be in each batch.) Every plaintext has the same X_0. Each batch of plaintexts has the same X_2.
2. Determine the time required for each batch to be encrypted.
3. The batch that took the least time to encrypt indicates one equation involving some key material:

$$(X_0 \odot Z_0) = (X_2 \boxplus Z_2)$$

4. Generate another set of 2^{16} plaintexts to be encrypted, this time using a different value for X_0, X_0^*. Follow the steps above, to get a second equation:

$$(X_0^* \odot Z_0) = (X_2 \boxplus Z_2)$$

In most cases, this should allow the cryptanalyst to recover the two key values Z_0 and Z_2. (If $Z_4 = 0$, then this attack will simply not work—there won't be a significant difference in the times.)

5. Generate another set of 2^{16} batches of plaintexts, this time holding (A) constant by use of fixed values for X_0 and X_2, and also holding X_3 constant. For each different batch, X_1 must take on a different value.

4.3 Generalizations

We detailed this attack assuming that the cryptanalyst was able to determine when a multiply by zero occurred by watching the relative encryption time. This is not the only side channel that can yield this information; radiation and power consumption can also leak this multiply-by-zero condition. Moreover, radiation and power consumption might determine the exact round where the multiplication by zero occurred, greatly simplifying the attack.

We note that this side-channel (the number of zero inputs in the block) has very low correlation with any given bit of internal state.

5 A Processor-Flag Attack Against RC5

There may be cases when we can learn the state of the overflow or carry flag during encryption; if so, this can form a useful side channel. While it is unlikely that this would be the only or easiest side-channel available, it demonstrates a useful feature of side-channel cryptanalysis: the side-channel can give information about the operations performed instead of the values used. This can work even when the information about the operation yields only minimal information about the values themselves.

For example, suppose we can determine the state of the carry flag after each half-round of RC5. Recall that an RC5 round looks like

$$R_i = (R_{i-1} \oplus L_{i-1})^{\ll L_{i-1}} + S_{2i-1}$$
$$L_i = (L_{i-1} \oplus R_i)^{\ll R_i} + S_{2i}$$

where S_i denote the round subkeys, (L_0, R_0) is the original plaintext, and (L_r, R_r) is the output of the last round. To make encryption and decryption symmetrical, there is also a final transformation

$$R_{r+1} = (R_r \oplus L_4)^{\ll L_r} + S_{2r+1}$$
$$L_{r+1} = L_r$$

before we obtain the resulting ciphertext (L_{r+1}, R_{r+1}).

RC5 is a variable-width cipher. Usually the width is set so that R_i (resp. L_i) fits into one register. Therefore the addition performed may cause the carry flag

to be set and we denote C_{2i-1} (resp. C_{2i}) the value of the carry flags after the addition computing R_i (resp. L_i).

We will show that with high probability, an attacker can reconstruct S_{2r+1} given several ciphertext pairs $(L_{r+1,j}, R_{r+1,j})$ and the corresponding carry flags $C_{2r-1,j}$. Once an attacker determines S_{2r+1}, they can strip off the last half round and reapply our attack (reversing left and right).

As we will see in the next few sections, each of the two possible values of $\mathrm{msb}(SK_{2r-1})_1$ lead to exactly one value of SK_{2r+1}. Hence an attacker could simply guess the most significant bit of SK_{2i-1} for all $i \leq r$, solve for each of the subkeys, and perform a few trial decryptions. If r is small (say 32), this is certainly feasible. However, in the event that r is too big, we must attempt to find $\mathrm{msb}(SK_{2r-1})_1$ by other means. The next section gives an algorithm for doing exactly that.

5.1 Determining $\mathrm{msb}(SK_{2r-1})_1$

Determining the most significant bit of SK_{2r-1} is rather simple. Consider three ciphertext pairs $(L_{r+1,j}, R_{r+1,j})$, $j = 1, 2, 3$ and their corresponding carry flags $C_{2r-1,j}$. Then we simply take the most significant bit of SK_{2r-1} to be the majority value of the carry flags $C_{2r-1,j}$.

The analysis which shows that three pairs suffices follows. First we must consider a simplified experiment. Let K be a randomly chosen n-bit value and consider the experiment in which we repeatedly choose a random n-bit value A and determine the value C of the carry flag for $A + K$. Then

$$\mathrm{Prob}[C = 1 | K = X] = \mathrm{Prob}[A + X \geq 2^n] = X/2^n.$$

Furthermore, we can generalize this where we only consider the l most significant bits of K. Then

$$\begin{aligned}
&\mathrm{Prob}[C = 1 | \,\mathrm{msb}(K)_l = X] \\
&\quad = \textstyle\sum_{i=0}^{2^{n-l}-1} \mathrm{Prob}[C = 1 | K = X \cdot 2^{n-l} + i] \cdot \mathrm{Prob}[\,\mathrm{lsb}(K)_{n-l} = i] \\
&\quad = \textstyle\sum_{i=0}^{2^{n-l}-1} \frac{X \cdot 2^{n-l} + i}{2^{2n-l}} \\
&\quad = X 2^{-l} + 2^{-l-1} - 2^{-n-1}.
\end{aligned}$$

Let μ' be the observed number of carries and t the number of trials to be performed. Given a set of t observations, we look for the X such that

$$\mu' - t(X 2^{-l} + 2^{-l-1} - 2^{-n-1})$$

is minimal, and guess that $\mathrm{msb}(K)_l = X$. We see that to estimate $\mathrm{msb}(K)_l$ in our attack we simply need to perform sufficiently many trials so that our guessing strategy gives the correct answer with high probability.

To approximate the number of trials, we assume that the distribution of μ' is normal. Let $p_l(X)$ denote the probability of a carry when $\mathrm{msb}(K)_l = X$, so

that $p_l(X) = X2^{-l} + 2^{-l-1} - 2^{-n-1}$. After t trials with μ' observed carries, we have

$$\text{Prob}\left[|tp_l(X) - \mu'| < 3\sqrt{tp_l(X)(1-p_l(X))}\right] \approx 0.997.$$

Hence we want to choose t such that

$$p_l(X) - p_l(X-1) > 3/2\sqrt{p_l(X)(1-p_l(X))/t}$$

for all $X > 1$. This gives

$$\begin{aligned} & p_l(X) - p_l(X-1) > 3/2\sqrt{p_l(X)(1-p_l(X))/t} \\ & \quad \Leftrightarrow 2^{-l} > 3/2\sqrt{p_l(X)(1-p_l(X))/t} \\ & \quad \Leftrightarrow 2^{-2l} > (9/4)p_l(X)(1-p_l(X))/t \\ & \quad \Leftrightarrow t > 2^{2l-2}9p_l(X)(1-p_l(X)) \end{aligned}$$

which will be roughly maximal when $X = 2^{l-1}$ and hence

$$\begin{aligned} t &> 2^{2l-2}9p_l(2^{l-1})(1-p_l(2^{l-1})) \\ &= 2^{2l-2}9(2^{-2} - 2^{-2l-2} + 2^{-l-n-1} - 2^{-2n-2}) \\ &= 9(2^{2l-4} - 2^{-4} + 2^{l-n-3} - 2^{2l-2n-4}). \end{aligned}$$

We are interested in the case where $l = 1$ and hence need

$$t > 9(2^{-2} - 2^{-4} + 2^{-n-2} - 2^{-2n-2})$$

pairs. For $n = 32$ this gives $t \approx 1.69$. Therefore we can take three ciphertext pairs and let $\text{msb}(SK_{2r-1})_1$ be the majority value of the carry flags $C_{2r-1,j}$.

5.2 Determining $\text{lsb}(SK_{2r+1})_1$

Once we have determined the most significant bit of SK_{2r-1}, we can use it to determine the least significant bit of SK_{2r+1}. The algorithm is rather simple: we choose plaintext/ciphertext pairs $(L_{r+1,j}, R_{r+1,j})$ such that $L_{r+1,j} \equiv 1 \pmod{32}$ and $C_{2r-1} \neq \text{msb}(SK_{2r-1})_1$.

The former condition ensures that if we know the most significant bit of $R_{r,j}$, then we will know the least significant bit of $R_{r+1,j} \boxminus S_{2r+1}$. The latter condition ensures that we know the most significant bit of $R_{r,j}$. After all, if $\text{msb}(SK_{2r-1})_1 = 1$ and $C_{2r-1} = 0$ we know that $\text{msb}(R_{r,j})_1 = 1$. Similarly, if $\text{msb}(SK_{2r-1})_1 = 0$ and $C_{2r-1} = 1$ we know that $\text{msb}(R_{r,j})_1 = 0$. Hence

$$\text{msb}(R_{r,j})_1 = \text{msb}(SK_{2r-1})_1. \tag{1}$$

From the above and the formula for computing $R_{r+1,j}$ it is easy to see that

$$\text{lsb}(R_{r+1,j} \boxminus S_{2r+1})_1 = \text{msb}(R_{r,j})_1 \oplus \text{msb}(L_{r,j})_1$$

or

$$\text{lsb}(S_{2r+1})_1 = (\text{msb}(SK_{2r-1})_1 \oplus \text{msb}(L_{r+1,j})_1) \boxplus \text{lsb}(R_{r+1,j})_1.$$

Note that we only need one ciphertext pair to determine $\text{lsb}(SK_{2r+1})_1$.

5.3 Determining Remaining Bits of SK_{2r+1}

Determining the remaining bits of SK_{2r+1} comes from generalizing the attack of the previous section. Here we assume that we know the j least significant bits of SK_{2r+1} and wish to determine the $j+1$st bit. Again we choose a ciphertext pair (L_{r+1}, R_{r+1}) such that $C_{2r-1} \neq \text{msb}(SK_{2r-1})_1$. However, this time we choose the pair so that $L_{r+1} \equiv j+1 \pmod{32}$.

The former condition ensures that we know the most significant bit of R_r by the same argument as in the previous section. The latter condition ensures that we know the $j+1$st bit of $R_{r+1} \boxminus S_{2r+1}$. Since we know $\text{lsb}(S_{2r+1})_j$, we can compute $\text{lsb}(R_{r+1} \boxminus S_{2r+1})_j$. However, we also know the $j+1$st bit of the difference as it is a function of $R_{r+1} \boxminus \text{lsb}(S_{2r+1})_j$, $\text{msb}(R_r)_1$ and $\text{msb}(L_r)_1$. Hence

$$\begin{aligned} \text{lsb}((R_r \oplus L_r)^{\lll L_r})_j &= \text{lsb}(R_{r+1})_j \boxminus \text{lsb}(S_{2r+1})_j \\ \Rightarrow \text{lsb}(S_{2r+1})_{j+1} &= \text{lsb}(R_{r+1})_{j+1} \boxminus (\text{lsb}((R_r \oplus L_r)^{\lll L_r})_j \\ &\quad \boxplus 2^j \cdot (\text{msb}(R_r)_1 \oplus \text{msb}(L_r)_1)) \end{aligned}$$

Of course $L_r = L_{r+1}$ so this formula allows us to determine the $j+1$st bit of S_{2r+1} directly.

Repeatedly applying this algorithm for $j = 2, \ldots, 32$ we recover the remaining bits of S_{2r+1}.

5.4 Adaptive Chosen-Plaintext Attack

We can recover S_1 much more directly using a chosen-plaintext attack which requires n chosen texts. Once we determine S_1 we can peel off the first half-round and repeat the attack for the remaining subkeys.

The attack is a very simple binary search on S_1. The crucial observation is that $C_1 = 1$ iff $(R_0 \oplus L_0)^{\lll L_0} \boxplus S_1 \geq 2^n$. If we let $L_0 = 0$, this simplifies to $R_0 \boxplus S_1 \geq 2^n$. Therefore we perform a simple binary search on $2^n \boxminus S_1$: $C_1 = 1$ iff $R_0 > 2^n \boxminus S_1$.

5.5 Generalizations

This is not the only side channel that can be used to cryptanalyze RC5; other side-channel information can also be used to break the cipher. For example, the timing may be different depending on the number of bits rotated. This information can be used to recover the RC5 key [HH98]. Or the power consumed by the rotate mechanism might be different depending on the Hamming weight of the bits rotated.

6 A Hamming-Weight Attack Against DES

Next we introduce Hamming-weight cryptanalysis. This assumes we have a side channel that gives information on the Hamming weight of intermediate encryption values. We claim that this model is not implausible: for instance, in some hardware implementations total power consumption is correlated to the total Hamming weight of all intermediate values, and in software timing data may leak information about the Hamming weight of internal variables when certain operations (e.g. integer multiplication) are used.

We show here that Hamming weight side channels can enable powerful ciphertext-only attacks in many cases. We will concentrate throughout on cryptanalysis of DES for concreteness, but it should become clear that these attacks will apply directly to any product cipher.

Let's start first with a very simple scenario. Assume that we are provided with a side channel that discloses the Hamming weight of the block after 15 rounds of encryption (i.e. the input to the 16th and final round). Then attacks to recover the last-round subkey abound. For instance, we could simply guess the 48-bit last-round subkey, and verify correct guesses on a few known ciphertexts. This attack will require about $48 \,/\, \log_2 \sqrt{32\pi} \approx 15$ ciphertexts and offline work equivalent to 2^{44} trial encryptions; the computational complexity could be reduced to about 2^{22} with meet-in-the-middle techniques.

For reasons that will become clear later, we wish to describe a different statistical attack which uses the Hamming weight of the input to the last round. Let $(R, L \oplus F(L))$ be the last-round input corresponding to the ith ciphertext (L, R) where $F(L)$ is the output of the Feistel F function under the last-round subkey. We write $W = \operatorname{wt} F(L)$ for the Hamming weight of the output of the last-round Feistel function; W is easily determined from the side channel. Also, we write S for the Hamming weight of the 4-bit output of the first S-box in the computation of $F(L)$, and N for the total Hamming weight of the other S-box outputs, so that $W = S + N$. Then guessing the 6 key bits entering the first S-box (in the 16th round) gives us S if the guess was correct, or (say) T otherwise (where T is a random variable with the same distribution as S, but T is independent of S). The core idea of the statistical attack is that S is strongly correlated with the known values $W = S + N$, but T is not, so we can detect correct guesses at those 6 key bits by the statistically-significant correlation that results.

The statistical attack thus proceeds as follows. Suppose we have n ciphertexts where the side channel discloses $W[1], \ldots, W[n]$. Guess the 6 key bits k entering the first S-box and compute the Hamming weight of that S-box's output, calling it (say) $U_k[1], \ldots, U_k[n]$. Calculate a measure of correlation[2] $c_k = \sum_{i=1}^{n} U_k[i]W[i]/n$, which is an estimate at the expected value $\mathbb{E}[U_k \cdot W]$ of $U_k \cdot W$. When k is correct, we get $U_k[i] = S[i]$, and thus c_k is an estimate of

[2] It is in fact equivalent in power to calculating the observed correlation coefficient ρ, or the covariance for that matter.

$\mathbb{E}[SW] = \mathbb{E}[S(S+N)] = \mathbb{E}[S^2] + \mathbb{E}[S]\,\mathbb{E}[N]$; when k is false, c_k is an estimate of $\mathbb{E}[TW] = (\mathbb{E}[S])^2 + \mathbb{E}[S]\,\mathbb{E}[N]$. Therefore, the counter c_k is expected to be noticeably larger when k is correct. If n is sufficiently large, we should be able to pick out the correct value of k from the largest of the 64 counters.

Of course, once we have determined the correct values of the 6 key bits entering the first S-box, we can repeat the attack for each of the other S-boxes in turn, until we have recovered the entire last-round subkey. In practice one could make do with significantly fewer ciphertexts by treating several S-boxes at once; here we focus on giving the essence of the central ideas of the attack, with an eye towards applying them to a wider class of Hamming-weight-based side channels.

We now analyze the complexity of this algorithm. The random variables S, T are independent symmetric binomial random variables with mean 2; also N is a symmetric binomial with mean 14. We see that $\mathbb{E}[SW - TW] = \mathbb{E}[S^2] - (\mathbb{E}[S])^2 = 1$. Also, a bit of computation shows that $\mathrm{Var}[SW - TW] = 23^2$. Letting k be a (generic) incorrect guess and K be the correct guess, we see that $c_K - c_k$ will be approximately Gaussian with mean 1 and standard deviation $23/\sqrt{n}$ when n is not too small. We expect success when c_K is larger than all the incorrect c_k counter values, and we can roughly estimate the probability of this as $\mathrm{Prob}[c_K > c_k]^{63} = \Phi(\sqrt{n}/23)^{63}$. Therefore, with $n = (3 \cdot 23)^2 \approx 4800$ ciphertexts, we expect to succeed with probability about $(1 - .0013)^{63} \approx .92$.

To recap, we have expressed the known Hamming weight W as a sum $W = S + N$ of "signal" S and "noise" N. Each guess at the 6 key bits entering the first S-box gives us a list of candidate observations for S. This reduces the problem of recognizing correct guesses to the problem of filtering out the noise from a noisy signal; this, in turn, is made possible because S is strongly correlated with $S + N$.

This technique can be generalized to scenarios with just about any Hamming-weight-based side channel imaginable.

For instance, imagine a side channel which gives us W, the total weight of all of the DES F function outputs summed over all 16 rounds. This is a good deal more plausible than our previous model, where we obtained the Hamming weight of only the last-round F function output; however, it may not be immediately obvious how to use such a side channel to break DES. We note that the statistical techniques developed above will apply immediately. As before, we let S stand for the total Hamming weight of the output of the first S-box in the last round. The total weight W can be expressed as $W = S + N$, where N is a sum of 127 other S-box outputs. Thus N can be treated as a symmetric binomial random variable of mean 254 which is independent of S. The attack proceeds as before, though we will of course need significantly more known ciphertexts. We estimate that $n = 2^{20}$ known ciphertexts and offline work comparable to 2^{19} trial encryptions suffices to recover the entire DES key with very high probability. This figure could be reduced to about 2^{18} known ciphertexts and 2^{37} offline work by simultaneously guessing 24 key bits entering 4 S-boxes.

In some environments, we may not be able to obtain the total Hamming weight of all the F function outputs, but we may be able to obtain a "noisy" version of it. (As an example, imagine a hardware implementation where the power consumption reveals the total Hamming weight of all intermediate values computed during a single DES encryption, or even something correlated to this weight. This would yield a side channel which is correlated to the total Hamming weight of all F function outputs.) In these situations, it is clear that we can filter out the noise as before: the same statistical technique still works, though more ciphertexts will be needed to compensate for the stronger "noise" contribution to the side channel value.

So far we have concentrated on ciphertext-only attacks. In fact, by symmetry, one can just as easily apply the attacks to scenarios where only the plaintext is known, and not the ciphertext! We are not aware of any other cryptanalytic technique where this is possible.

In summary, we see that even a side channel which is only mildly correlated to values of interest (such as the input to the last round) can be used to mount a powerful statistical attack against just about any product cipher. These attacks require only known-ciphertexts, and so should be quite practical to mount in practice. This shows that cryptosystem implementors must take great care to avoid even the slightest correlation between observable outputs and internal values.

7 Conclusions and Further Directions

The purpose of this paper was to demonstrate the power of side-channel cryptanalysis against product ciphers. Our attacks are by no means exhaustive; the algorithms discussed have other possible side channels and other attacks are possible given other side channel information. And other product ciphers are vulnerable to similar attacks. We believe attacks based on cache hit ratio in large S-box ciphers like Blowfish [Sch94], CAST [Ada97], and Khufu [Mer91] are possible.

Stream ciphers with irregular clocking features can be especially vulnerable to side-channel attacks. A5 [XHW94], PIKE [And95], Gollman cascades [CG88], shrinking and self-shrinking generators [CKM94,MS94], and any of the family of alternating stop-and-go generators [Sch96] are easily breakable if an attacker knows how many LFSRs clocked for each output. Implementations of RC4 [Sch96] that have different side-channel characteristics when $i = j$ are also vulnerable.

Public-key algorithms are vulnerable. Kocher showed how to mount at timing attack against several public-key algorithms [Koc96]. These results can be generalized to any primitive that uses multiplication modulo large numbers for security, e.g. stream ciphers like Blum-Blum-Shub [BBS86] and hash functions like IBC Hash.

The open questions lead in two different directions: how to obtain more detailed side-channel information, and how little side-channel information is required to break a cryptographic primitive. With regards to the former, we leave that to the electronic engineers. We have collected side-channel information using both timing and power channels, and have often been surprised by how easy it is. We can only speculate how much more information a well-equipped hardware laboratory can collect.

As to the second question, the surprising result from this research is that the amount of side-channel information necessary to break a product cipher is very small. Ciphers have not been designed with side channels in mind; hence, they are often very vulnerable to analysis using them.

It is our belief that most operational cryptanalysis makes use of side-channel information. Sound as a side-channel—listening to the rotation of electromechanical rotor machines—was alluded to in Kahn [Kah67]. Van Eck radiation—another side channel—has been demonstrated as a way to get plaintext [vEc85]. And Peter Wright discussed data leaking onto a transmission line as a side channel used to break a French cryptographic device [Wri87].

The (unclassified) military literature provides many examples of real-world side channels. TEMPEST shielding is just an attempt to stop the side channel that can result from van Eck radiation. Peter Wright's crosstalk anecdote is probably what the HIJACK codeword refers to [USAF98]. Along similar lines, [USAF98] alludes to the possibility that crosstalk from sensitive hardware near a tape player might modulate the signal on the tape; [USAF98] recommends that tapes played in a classified facility be degaussed before they are removed, presumably to prevent side channels from leaking. Finally, one last example from the military literature is the NONSTOP attack [USAF98, Chapters 3–4]: after a careful reading of unclassified sources, we believe this refers to the side channel that results when cryptographic hardware is illuminated by a nearby radio transmitter (e.g. a cellphone), thereby modulating the return signal with information about what the crypto gear is doing [AK98].

Using side channels to break cryptographic primitives is such a powerful notion that it is reasonable to expect intelligence organizations to have built on the successes alluded to in the previous paragraph. By continuing to research both the collection of side-channel data and the vulnerabilities to specific primitives to side-channel data, it is our hope that we can begin to build mathematical algorithms that are more resistant to side-channel cryptanalysis as well as implementations that leak less side-channel data.

8 Acknowledgments

The authors wish to thank Pawel Pogoda, Peter Pearson, Paul Kocher, Joshua Jaffe, and Niels Ferguson for useful conversations, and the anonymous reviewers for several helpful comments.

References

[Ada97] C. Adams, "Constructing Symmetric Ciphers Using the CAST Design Procedure", *Designs, Codes and Cryptography*, v.12, n.3, Nov 1997, pp. 71–104.

[And95] R. Anderson, "On Fibonacci Keystream Generators," *Fast Software Encryption, 2nd International Workshop Proceedings*, Springer-Verlag, 1995, pp. 346–352.

[AK98] R. Anderson and M. Kuhn, "Soft Tempest: Hidden Data Transmission Using Electromagnetic Emanations," *Proc. 2nd Workshop on Information Hiding*, Springer, 1998.

[Bel96] S. Bellovin, "Problem Areas for the IP Security Protocols," *Proceedings of the Sixth Usenix Unix Security Symposium*, Jul 1996, pp. 1–16.

[BS91] E. Biham and A. Shamir, "Differential Cryptanalysis of DES-like Cryptosystems," *Journal of Cryptology*, Vol. 4, No. 1, 1991, pp. 3–72.

[BS93] E. Biham and A. Shamir, *Differential Cryptanalysis of the Data Encryption Standard,* Springer-Verlag, 1993.

[Bih94] E. Biham, "New Types of Cryptanalytic Attacks Using Related Keys," *Journal of Cryptology*, v. 7, n. 4, 1994, pp. 229–246.

[BS97] E. Biham and A. Shamir, "Differential Fault Analysis of Secret Key Cryptosystems," *Advances in Cryptology—CRYPTO '97 Proceedings*, Springer-Verlag, 1997, pp. 513–525.

[BBS86] L. Blum, M. Blum, and M. Shub, "A Simple Unpredictable Pseudo-Random Number Generator," *SIAM Journal of Computing*, v. 15, n. 2, 1986, pp. 364–383.

[BDL97] D. Boneh, R.A. Demillo, R.J. Lipton, "On the Importance of Checking Cryptographic Protocols for Faults," *Advances in Cryptology—EUROCRYPT '97 Proceedings*, Springer-Verlag, 1997, pp. 37–51.

[CG88] W.G. Chambers and D. Gollmann, "Generators for Sequences with Near-Maximal Linear Equivalence," *IEE Proceedings*, v. 135, pt. E, n. 1, Jan 1988, pp. 331–343.

[CKM94] D. Coppersmith, H. Krawczyk, and Y. Mansour, "The Shinking Generator," *Advances in Cryptology—CRYPTO '93 Proceedings*, Springer-Verlag, 1994, pp. 22–39.

[DES81] ANSI X3.92, "American National Standard for Data Encryption Algorithm (DEA)," American National Standards Institute, 1981.

[HGS97] C. Hall, I. Goldberg, B. Schneier, "Reaction Attacks Against Several Public-Key Cryptosystems," 1998, in preparation.

[HH98] H. Handschuh and H. Heys, "A Timing Attack on RC5," *Selected Areas of Cryptography–SAC '98, Procedings from 5th International Workshop*, Springer, 1998.

[HM97] C. Harpes and J. Massey, "Partitioning Cryptanalysis," *Fast Software Encryption, 4th International Workshop Proceedings*, Springer-Verlag, 1997, pp. 13–27.

[Kah67] D. Kahn, *The Codebreakers*, The MacMillan Company, 1967.

[KJY98] P. Kocher, J. Jaffe, B. Yun, "Introduction to Differential Power Analysis and Related Attacks", *http://www.cryptography.com/dpa/technical/index.html*, December 1998.

[KSW96] J. Kelsey, B. Schneier, and D. Wagner, "Key-Schedule Cryptanalysis of IDEA, G-DES, GOST, SAFER, and Triple-DES," *Advances in Cryptology — CRYPTO '96 Proceedings*, Springer-Verlag, 1996, pp. 237–251.

[KSW97] J. Kelsey, B. Schneier, and D. Wagner, "Related-Key Cryptanalysis of 3-WAY, Biham-DES, CAST, DES-X, NewDES, RC2, and TEA," *Information and Communications Security, First International Conference Proceedings*, Springer-Verlag, 1997, pp. 203–207.

[KSWH98] J. Kelsey, B. Schneier, D. Wagner, C. Hall, "Side Channel Cryptanalysis of Product Ciphers,' *ESORICS '98 Proceedings*, Springer-Verlag, September 1998, pp. 97-110

[Koc96] P. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," *Advances in Cryptology—CRYPTO '96 Proceedings*, Springer-Verlag, 1996, pp. 104–113.

[Koc98] P. Kocher, personal communication, 1998.

[LM90] X. Lai, J.L. Massey, "A Proposal for a New Block Encryption Standard," *Advances in Cryptology—EUROCRYPT '90 Proceedings*, Springer-Verlag, pp. 389–404.

[LMM91] X. Lai, J.L. Massey, and S. Murphy, "Markov Ciphers and Differential Cryptanalysis," *Advances in Cryptology—EUROCRYPT '91 Proceedings*, Springer-Verlag, pp. 17–38.

[Mat93] M. Matsui, "Linear Cryptanalysis Method for DES Cipher," *Advances in Cryptology—EUROCRYPT '93 Proceedings,* Springer-Verlag, 1994, pp. 386–397.

[MS94] W. Meier and O. Steffelbach, "The Self-Shrinking Generator," *Communications and Cryptography: Two Sides of One Tapestry*, R.E. Blahut et al, eds., Kluwer Academic Publishers, 1994, pp. 287–295.

[Mer91] R. Merkle, "A Fast Software Encryption Function," *Advances in Cryptology—CRYPTO '90 Proceedings*, Springer-Verlag, 1991, pp. 476–501.

[Sch94] B. Schneier, "Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish)," *Fast Software Encryption, Cambridge Security Workshop Proceedings*, Springer-Verlag, 1994, pp. 191–204.

[Sch96] B. Schneier, *Applied Cryptography, 2nd Edition,* John Wiley & Sons, 1996.

[USAF98] US Air Force, *Air Force Systems Security Memorandum 7011—Emission Security Countermeasures Review*, 1 May 1998.

[Vau96] S. Vaudenay, "An Experiment on DES Statistical Cryptanalysis," *3rd ACM Conference on Computer and Communications Security*, ACM Press, 1996, pp. 139–147.

[vEc85] W. van Eck, "Electromagnetic Radiation from Video Display Units: An Eavesdropping Risk," *Comptuers & Security*, v. 4, 1985, pp. 269–286.

[Wri87] P. Wright, *Spycatcher*, Viking Penguin Inc., 1987.

[XHW94] S.B. Xu, D.K. He, and X.M. Wang, "An Impelementation of the GSM General Data Encryption Algorithm A5," *CHIANCRYPT '94*, 11-15 Nov 1994, pp. 287–291.