IPA: A New Class of Power Attacks

Paul N. Fahn*
and Peter K. Pearson**

Certicom Corp.
25801 Industrial Blvd.
Hayward, CA 94545, USA

Abstract. We present Inferential Power Analysis (IPA), a new class of attacks based on power analysis. An IPA attack has two stages: a profiling stage and a key extraction stage. In the profiling stage, intratrace differencing, averaging, and other statistical operations are performed on a large number of power traces to learn details of the implementation, leading to the location and identification of key bits. In the key extraction stage, the key is obtained from a very few power traces; we have successfully extracted keys from a single trace. Compared to differential power analysis, IPA has the advantages that the attacker does not need either plaintext or ciphertext, and that, in the key extraction stage, a key can be obtained from a small number of traces.

1 Introduction

Recent years have seen significant progress in what are called "power attacks" on cryptographic modules, attacks in which one monitors the power drawn by the module and from these measurements extracts some secret quantity that the module manipulates during some cryptographic operation. In 1998, Kocher et al. [5] described Differential Power Analysis (DPA), in which power measurements from many repeated cryptographic operations are cleverly combined. More recently, Biham and Shamir [1] showed how to derive key information by combining power measurements on different cryptographic modules.

This paper describes a class of attacks called Inferential Power Analysis (IPA) attacks. An IPA attack is characterized by two stages, the first a lengthy profiling stage, and the second a simpler key extraction stage. The profiling step is typically based on comparisons of repeated parts of a selected cryptographic operation, such as the different rounds in a DES encryption. These comparisons can be performed on a single cryptographic module, requiring many measured operations, and result in a profile that can subsequently be used to extract keys from other modules using as little as a single cryptographic operation. Unlike DPA, these attacks do not require knowledge of the operation's inputs or outputs.

* pfahn@certicom.com
** ppearson@certicom.com

Ç.K. Koç and C. Paar (Eds.): CHES'99, LNCS 1717, pp. 173–186, 1999.

More generally, these attacks illustrate a class of attacks in which a one-time effort requiring just one module produces information with which keys can be easily extracted from other modules of the same design. Such attacks can be applied not only by a cardholder against a smartcard in his possession, but also by a terminal owner against smartcards that use his terminal.

Due to the rapidly advancing state of knowledge about power analysis, we cannot make conclusive statements about the effectiveness of specific countermeasures. Nevertheless we suggest several possible defenses in §5 that may make power attacks more difficult and raise the level of effort and expertise required of the attacker.

2 Background

More and more cryptographic systems are embedding keys in portable electronic modules such as smartcards and PC cards. These modules usually provide both storage for the key and processor power sufficient to allow the key to be used *in situ*, so that the key is never exposed to the outside world. When the holder of the module (which we will henceforth assume is a smartcard) has a stake in keeping the key secret, such modules provide strong, convenient, and inexpensive security.

On the other hand, when the cardholder has an incentive to violate the secrecy of the key, protecting the key is a difficult challenge to the system's designer. For example, in the case of stored-value cards, learning the card's key may enable the cardholder to defraud a bank. Since the cardholder has physical possession of the card, many avenues of attack are available:

- The cardholder can subject the card to unusual conditions like out-of-range supply voltage, out-of-range clock frequency, extreme temperatures, radiation, or unusual commands, in order to induce errors. Some errors may directly expose keys, while others may produce incorrect cryptographic results from which keys can be computed [2].
- The cardholder can physically dissect the card and reset protection bits, or directly read electrical charges in memory cells, or measure voltages on bus traces while sensitive data are passing between memory and processor [6].
- While the card is performing cryptographic calculations, the cardholder can measure currents, voltages, electric fields, or execution times [4], any of which might exhibit correlation with the key being used.

The current drawn through the card's power connector (V_{cc}) is easy to measure with a digital oscilloscope, and provides much revealing information. For example, if the smartcard uses a hardware multiplier for modular exponentiations of large integers, each multiplication is visible as a distinct period of increased current consumption. The fastest implementations of modular exponentiation handle ordinary multiplications differently from squarings; but if this technique is used in a smartcard, squarings and nonsquare multiplications can be distinguished in an oscilloscope trace of current consumption. From the order in which these operations occur, one can deduce the exponent, which is often a vital secret.

Almost all processes display a similarly rich variability in current consumption patterns. In a plot of current consumption versus time during the execution of a single high-level command, the eye easily discerns several separate computational phases, distinguished by mean current consumption and by "fuzziness" (short-term variability in current consumption). Because of such variations, the 16 rounds of a DES encryption are generally easy to recognize as a train of 16 identical boxcars. (On closer inspection, one finds that boxcars 1, 2, 9, and 16 are slightly shorter than the rest, due to key schedule idiosyncrasies.)

Since the current drawn by the smartcard is, at constant voltage, proportional to the power consumed by the card, attacks based on current measurements are usually referred to as *power attacks*. We will henceforth refer to power instead of current.

Differential Power Analysis (DPA), developed by Kocher et al. [5], is a powerful extension of these techniques. In a DPA attack on a DES key, the smartcard is repeatedly induced to encrypt various plaintexts with the key to be found, while digitized "traces" of power consumption are recorded along with the plaintexts of the encryptions.[1] When a large number (often on the order of 1000) of traces and plaintexts have been accumulated, averages of subsets of the traces are computed and compared in order to test guesses of various key bits.

Specifically, one sorts the traces into two classes according to conjectured values of a particular bit B computed during the encryption, the conjectured values being computed from the plaintext and a guess at some subset of key bits. If the guessed key bits are correct, the conjectured value of B will match the true value of B in all traces, and the mean $B = 1$ trace will differ significantly from the mean $B = 0$ trace wherever bit B is handled. On the other hand, if the guessed key bits are wrong, the sorting of traces into subsets is (one expects) uniformly random, and no significant difference will be observed.

Two beautiful virtues of DPA are the following: (1) although the attacker makes the assumption that the DES code computes the value B, it is not necessary to know where that computation occurs; and (2) if chip designers add random noise to mask power consumption, the attacker can compensate for the lower signal-to-noise ratio by increasing the number of traces. On the other hand, practical problems in mounting a DPA attack include: (1) protocols may be designed to keep the attacker from seeing the plaintext or ciphertext; and (2) sometimes it is not possible to get enough traces.

More recently, Biham and Shamir [1] described a power attack in which many traces from each of several different cards are compared to identify when key bits are being handled. Instants with large same-card power variations are assumed to be data-handling, not key-handling, instants. Non-data-handling instants with large between-card power variations are assumed to be key-handling instants. Once the attacker has located the instructions handling parts of the key, their power consumptions can be measured to attack the key. Since its "profiling"

[1] Decryptions may be used instead of encryptions; and independently, either plaintexts or ciphertexts can be used.

stage can be separated from its key extraction stage, this attack falls into the class described in the present paper.

3 IPA Attacks

Here we describe an IPA attack. Stage 1, the profiling stage, contains almost all of the effort. Stage 2, the key extraction stage, is then quick and simple. One can think of the profiling stage as a long precomputation, after which one can obtain each subsequent key with only a small incremental effort.

The first steps in the profiling stage are familiar from other sophisticated power attacks, such as DPA: collecting a large number of power traces, and then aligning and averaging the traces; the details of these steps need to be specifically tailored to IPA. Next come the two tasks necessary to finding the key: locating and identifying the key bits. These steps are described in detail below.

As our primary example we will consider an IPA attack on DES, since DES is not only well-known but is perhaps the most widely implemented of all cryptographic algorithms. We have successfully performed IPA attacks on DES smart card implementations and have extracted DES keys from a single power trace in Stage 2 of our IPA attacks.

3.1 Context and Assumptions

We assume the following: we have a smart card containing a known cryptographic algorithm but we do not know the details of the implementation, i.e., we do not possess the source code (knowledge of the source code would enable a far simpler attack). Furthermore, we can cause hundreds of executions of the algorithm with different plaintexts (not necessarily uniformly distributed), and we can record the amount of power (current) used at each step within each such execution.

For the purposes of this exposition, we will assume that the algorithm has been implemented in a straightforward manner, without introducing elements designed to thwart power attacks; in particular, the algorithm execution is a deterministic function of the plaintext and key. If the card did employ defenses, modified versions of IPA might still work, depending on which defenses were used; space does not permit discussion of these modified versions. In §5, we discuss some defensive measures that can be used by system implementers.

These assumptions appear to represent the standard context facing someone wishing to attack a smart card. In practice we have successfully performed IPA attacks in this context using only moderate resources, e.g., only a few hundred power traces and a low sample rate (3.57 million samples per second, or one per clock cycle) on an oscilloscope.

3.2 Stage 1: Profiling

The goal of the profiling stage is to locate and identify the key bits as they are used during the computation. To do this, we often need to learn about other

aspects of the implementation we are facing: the order of operations, how key bits are handled, and other details that were decided by the programmer.

The attacker starts by facing an unknown implementation and then gradually learns more and more during the course of the profiling stage, until he finds when, and often how and where, the program engages the key bits. At the end of the profiling stage, the attacker knows many of the decisions that were made by the implementer.

The next three sections describe the important steps in the profiling stage.

3.3 Profiling: First Steps

As the attacker, we first cause the smart card (or other device) to execute its cryptographic algorithm a large number of times to obtain a large number of traces. In practice, we have found that a few hundred traces usually suffices; as lower and upper bounds, we estimate that in general the number of traces needed will be between 100 and 1000.

For simplicity of exposition, we will suppose that the executions are all with the same key and with varying plaintexts. This is not essential, as the attack will work even if the keys vary. Of course, if both the key and the plaintext are constant, then we are merely resampling the same data point and the multiple traces are practically useless. The plaintext needs not be either random or uniformly distributed, but merely non-constant.

The traces are then averaged together, in order to remove the effects of the varying data bits while keeping the effects of the constant key bits. Before averaging, we must first align the traces so that the power consumptions of every operation are matched across all the traces. In practice, we have found that each implementation of each algorithm requires a slightly different alignment technique, and the alignment effort ranges from quite simple to quite cumbersome.

We now have a single "average" trace containing the average power consumed throughout the execution. This represents the average, over all plaintexts, of the power consumed using the constant, unknown key.[2]

3.4 Profiling: Key Location

The next major task is to locate the key bits within the average trace. We first describe the basic procedure, followed by a more mathematical description, and then some mention some implementation notes.

Basic Description

Almost all cryptographic algorithms contain repetitive structures, used with

[2] Even if the plaintext is not uniformly distributed, the "data" bits soon become uniform, due to the randomizing effect of the rounds; for example, even if the plaintext into DES has its top 50 bits set to one fixed value, the bits in the L and R registers become uniform after the first couple of rounds.

changing pieces of the overall key. For example, in most symmetric ciphers, repeating rounds use differing subkeys. Public key algorithms such as RSA use different key bits while repeatedly performing modular multiplications. We call these repeating structures "rounds", and the corresponding key bits "subkeys", with the understanding that in an algorithm such as RSA, the modular multiplications play the role of rounds.

Suppose there are n rounds, and let K_i denote the subkey used in round i.

Due to code space limitations on smart cards and other devices, the repeating structures are generated by the same source code being executed over and over; therefore, each round's subkey is handled identically.

The key location proceeds as follows: we chop the average trace into rounds to obtain traces $R_1, R_2, \ldots, R_n$, representing "average round 1", "average round 2", and so on. Then we average these together to obtain a single "super-average round" $\mathcal{R}$, i.e., the average of the average rounds.

Next, we take the difference, for each round i, between R_i and $\mathcal{R}$ to obtain the "round i difference trace" Δ_i. Finally, we square and then average together the Δ_i's. These last few steps are equivalent to computing the variances of the instruction offsets in the R_i's. The final trace, the mean square of the Δ_i's, contains peaks that reveal the key bit locations.

Why does this work? At an intuitive level, note that the first averaging (of different traces) removed the effects of the data bits but left the effects of the key bits: the only differences between the R_i's are due to the differences between the subkeys (see Figure 1 for an example). The second averaging (of different rounds) removed the effects of the key bits as well, leaving only "code" features. When we then take the difference between average round R_i and the super-average round $\mathcal{R}$, the code features cancel out, leaving only the effects of the specific subkey K_i. The subsequent squaring and averaging produces clean peaks at all subkey bit locations.

Since we know the algorithm, we know the number of key bits comprising each subkey, and therefore we know how many peaks to look for. In DES, we look for 48 peaks in the average of the squared Δ_i's. An example is shown in Figure 2.

It is a good idea at this point to observe the distribution of power levels at the detected peaks, in order to verify that the peaks represent key bits and to determine the power threshold that separates a 0 from a 1 bit value. Since the actual key bits are presumably 0 or 1 with probability $\frac{1}{2}$, an instruction that handles a single key bit should exhibit a bimodal distribution, with half the probability in each mode. An instruction that handles more than one key bit should exhibit the appropriate binomial distribution. Thus the shape of the power distribution can reveal the way the key bits are being handled.

In practice, the straightforward cryptographic implementations we have seen most commonly on smart cards handle key bits individually, and we have therefore seen bimodal distributions at the peaks. If key bits are not handled individually, one can use the resulting binomial distributions to learn the Hamming

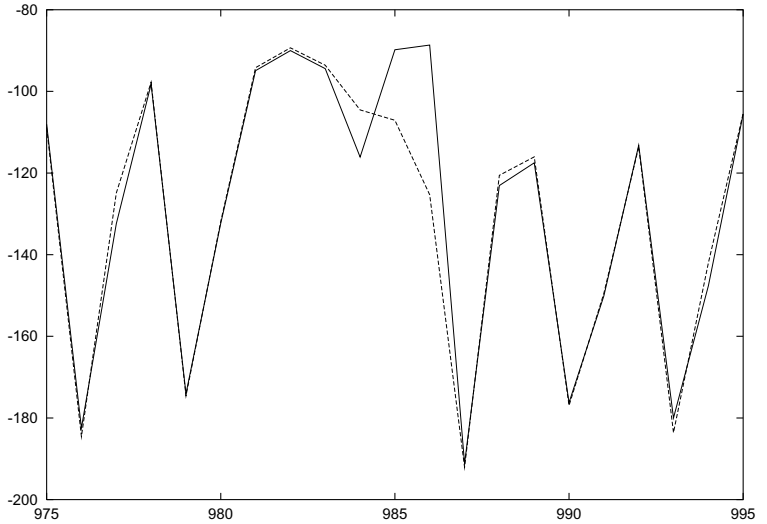

Fig. 1. A section of two round traces R_i and R_j. They differ only where a key bit is being handled, at positions 984 through 986.

weights of key bit groupings; one way to proceed in this case is discussed by Biham and Shamir [1].

Mathematical Description

Let $T_{i,j}(t)$ denote the power consumed at time t within the i-th round in power trace j. In general, the power consumed at any time t is a function f_t of some key bits $\boldsymbol{k} = \langle k_1, \ldots, k_r \rangle$, and some data bits $\boldsymbol{d} = \langle d_1, \ldots, d_s \rangle$. If we write $\boldsymbol{d}_{i,j}$ and $\boldsymbol{k}_i$ for the actual bit values of $\boldsymbol{d}$ and $\boldsymbol{k}$ in round i in the j-th trace ($\boldsymbol{k}$ depends only on the round, not the trace), we have[3]

$$T_{i,j}(t) = f_t\left(\boldsymbol{d}_{i,j}, \boldsymbol{k}_i\right). \tag{1}$$

Assuming that the number of traces m is large enough and the numbers of dependent key bits r and data bits s are small enough,[4] we will find that the power $R_i(t)$ in the average round trace is approximately the same as if we had

[3] In all equations we ignore the random power fluctuations due to internal and external noise. In practice, these effects disappear once we have done the first averaging.

[4] We would like to see $m > \max\{2^r, 2^s\}$. This is reasonable since typical values might be $m \approx 300$, and $r, s \leq 4$.

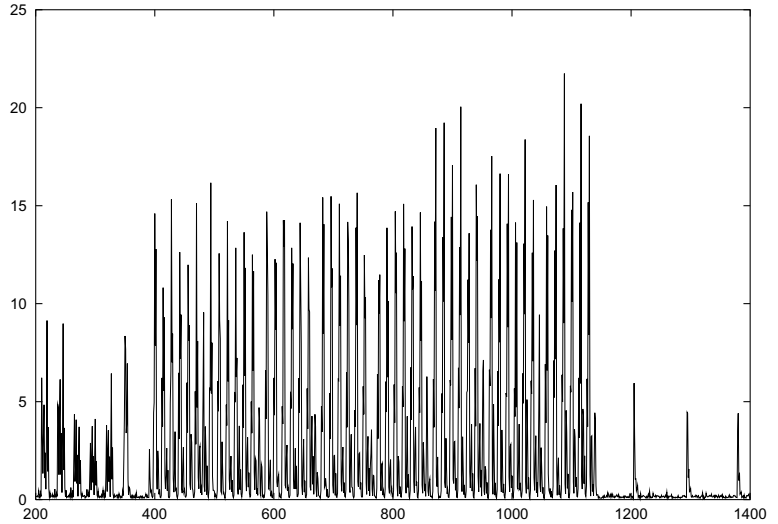

Fig. 2. Peaks in Δ_i^2 in a DES implementation. In this implementation, there are 48 peaks, one for each subkey bit, and they are clustered in 8 groups of 6, for the 8 S-boxes.

averaged over all 2^s possible values of $\boldsymbol{d}$:

$$\begin{aligned} R_i(t) &= \frac{1}{m} \sum_{j=1}^{m} T_{i,j}(t) \\ &= \frac{1}{m} \sum_{j=1}^{m} f_t(\boldsymbol{d}_{i,j}, \boldsymbol{k}_i) \\ &\approx \frac{1}{2^s} \sum_{\boldsymbol{d}} f_t(\boldsymbol{d}, \boldsymbol{k}_i). \end{aligned} \tag{2}$$

Furthermore, taking the average of the R_i's to get the super-average trace $\mathcal{R}$ will have the effect of averaging over all 2^r values of $\boldsymbol{k}$, so that $\mathcal{R}(t)$ is:

$$\begin{aligned} \mathcal{R}(t) &= \frac{1}{n} \sum_{i=1}^{n} R_i(t) \\ &\approx \frac{1}{n} \sum_{i=1}^{n} \frac{1}{2^s} \sum_{\boldsymbol{d}} f_t(\boldsymbol{d}, \boldsymbol{k}_i) \quad \text{from Equation 2} \\ &\approx \frac{1}{2^{r+s}} \sum_{\boldsymbol{k},\boldsymbol{d}} f_t(\boldsymbol{d}, \boldsymbol{k}). \end{aligned} \tag{3}$$

Consider an instruction, at, say, time t_1, that does not involve any key bits; then the power consumption function f_1 depends only on the data bits $\boldsymbol{d}$, and

the value of $R_i(t_1)$ from Equation 2 becomes

$$R_i(t_1) \approx \frac{1}{2^s} \sum_{\boldsymbol{d}} f_1(\boldsymbol{d}). \tag{4}$$

The important point about Equation 4 is that $R_i(t_1)$ does not depend on i at all, and is therefore constant among all the rounds and in the super-average round $\mathcal{R}$, i.e., $\mathcal{R}(t_1) = R_i(t_1)$. So when we take the difference between the average round i trace, R_i, and the super-average round trace, $\mathcal{R}$, we find, at position t_1,

$$\Delta_i(t_1) = \mathcal{R}(t_1) - R_i(t_1) = 0. \tag{5}$$

Now consider another instruction, at time t_2, which handles some key bits $\boldsymbol{k}$ as well as data bits $\boldsymbol{d}$; its power consumption function f_2 then looks like that in Equation 1, and the values of $R_i(t_2)$ and $\mathcal{R}(t_2)$ look like Equations 2 and 3, respectively. In the round i difference trace Δ_i we then find, at position t_2,

$$\begin{aligned} \Delta_i(t_2) &= \mathcal{R}(t_2) - R_i(t_2) \\ &= \frac{1}{2^{r+s}} \sum_{\boldsymbol{k},\boldsymbol{d}} f_2(\boldsymbol{d}, \boldsymbol{k}) - \frac{1}{2^s} \sum_{\boldsymbol{d}} f_2(\boldsymbol{d}, \boldsymbol{k}_i) \end{aligned} \tag{6}$$

which in general is *not* 0, but some function that depends on the specific values of the subkey bits $\boldsymbol{k}_i \subseteq K_i$.

The end result is that the difference traces Δ_i will be close to zero whenever key bits are *not* being handled. Therefore, when we square and average the Δ_i's, we find peaks exactly at those times (like t_2 in our example) when key bits *are* being handled.

Implementation Notes

An actual implementation of an IPA attack may encounter difficulties; here we mention a couple of the most common.

The super-average $\mathcal{R}$ will work as planned only if the average rounds R_i are aligned at use of the key bits. Therefore, in practice, one may need to do some alignment of the R_i's before averaging. For example, in DES, some rounds contain more shifts of the key registers than others; therefore the offsets of instructions at which the key bits are used may differ from round to round, and this difference must be accounted for before averaging. Depending on the algorithm, this may mean identifying other features within the round traces; this is one instance in which the "profiling" can involve learning more than just the key usage patterns.

Another important point is that the number of spikes in Δ_i may differ from the size of the subkey, depending on exactly how the key bits are being handled in the (unknown) implementation. However, the number of spikes and the number of subkey bits should have a simple mathematical relation.

At the end of the key location step, we know the times when the power consumption depends on the key bits, i.e., we know where to find the key.

3.5 Profiling: Key Bit Identification

Given the list of key bit locations, we still can't read off the key, because we don't know which location corresponds to which key bit. This process of determining the identity of the key bits is the final step in the profiling stage of an IPA attack.

The actual method of key bit identification varies greatly with the algorithm and the implementation. So rather than give overall rules, we restrict ourselves to some general comments on algorithmic features, followed by specific remarks about some of the more common algorithms.

An algorithm's specification may (or may not) restrict the order in which the key bits are used. In RSA, for example, the secret key is used as a modular exponent, and the exponentiation uses the bits of the key sequentially. However, the programmer's decisions still affect the order in which the key bits are used: one can start from either the most significant bit or the least significant bit of the exponent, and the bits can be used one at a time or two at a time. But these are a relatively small number of choices, and therefore the key identification step for RSA is usually fairly simple.

In DES, on the other hand, key identification can be much more difficult, since there are fewer restrictions on key bit order. A DES subkey consists of 48 bits used as inputs to 8 S-boxes. The S-box operations within a round do not depend on one another, and thus all 8! orderings of S-boxes are possible. Furthermore, there is no restriction on the order in which the 6 key bits used in an S-box are loaded from the key registers. A DES programmer may in fact choose a key bit order not based on any S-box order, but based on, say, the key bit locations inside the key registers.

Because of the large subkey size in DES, we have also run into difficulties when we have found less than 48 key locations. Suppose we find only 32 key locations. Then the key location step must determine, first, which of the possible $\binom{48}{32}$ subsets, and, second, which of the 32! permutations of that subset, we are seeing. Of course, in a straightforward implementation, some key bit orderings are much more common than others. The S-box order is more likely to be 12345678 or 87654321 than 53821467, for example. Still, we caution against assuming any particular ordering, and after the most obvious guesses have failed, it may be unclear where to turn next.

When the key identification becomes non-trivial, one can turn to the key scheduling section of the algorithm's specification, and select patterns that can be sought for in the empirical key location data. In DES, for example, the key schedule specifies the patterns in which individual key bits move from one S-box position to another in consecutive rounds. This allows a key identification hypothosis to be tested by comparing the movement from round to round of 1's and 0's in our observed key locations against the movement of fixed key bits in the DES key schedule.

At the end of a successful key identification step, and thus at the end of the profiling stage of the attack, we have a table of locations (inside the round traces) and the corresponding key bit identity. If we are attacking DES, for example, our table might look something like that in Table 1, where the numbers in the

location column are offsets into the aligned, averaged round traces (R_i) and the key indices refer to bits within the round subkeys (K_i).

Table 1. The final result of the profiling stage of an IPA attack: the key table.

Location	Subkey Bit
380	k_4
672	k_1
1022	k_9
⋮	⋮

3.6 Stage 2: Fast Key Extraction

Armed with the key location table, we can easily find the subkey bits and then the master key bits from the traces we have. But the profiling data depend only on the software implementation that we are attacking, and not at all on the key that was used in the traces we processed. Therefore the information in the table will be equally valid for all other instances of the same software running on identical hardware, and so we can easily find the key in any such instance, not just the instance whose power traces we have already recorded.

In short, the profiling stage needs only be done once, and then key extraction can be done quickly and efficiently from new instances with unknown keys. One can think of the profiling stage as a long precomputation of the key location table; after the precomputation we can then quickly solve any similar instance of the same problem. For example, given a second smart card, identical to the first except for a different key, the data in our key location table immediately point us to the new key, without taking hundreds of new traces involving the new key.

To extract the key from a new instance of the same implementation, we take a single power trace, chop it into rounds, and measure the power consumed at the locations specified in the key location table. Using our knowledge of the key bit power distribution, which we obtained during the profiling stage, we can tell whether the key bit is a 0 or a 1.

Due to the particularities of the algorithm and the implementation, a single power trace may not suffice, in which case we would take, say, 5 traces, average them together, and then measure the power levels at the key locations. The issue of whether one trace will suffice depends, among other things, on whether there are instructions that handle key bits separately from data bits. In most of the implementations that we have seen, each key bit is handled by itself in at least one instruction (for example, the bit is loaded into a register) without the interference of data bits; therefore we have been able to extract keys using a single trace.

In any case, the number of traces needed in the key extraction stage is far less than the number needed during profiling, where we needed enough traces to average away any effects of the data bits and to ensure that any peaks we found were due to key bits only. For key extraction, on the other hand, we already know where the key bits are, and only care about whether data bits may affect readings at the key bit locations, and can safely ignore the effects of data bits elsewhere in the trace.

4 Strengths of IPA

There are several aspects of IPA attacks that make them effective in situations where DPA attacks would be difficult to mount.

First we note that in order to mount a DPA attack an attacker needs the plaintext (or ciphertext) associated with every trace; but this is not required for an IPA attack. Thus one of the important defenses against DPA — protocol designs that hide plaintext and ciphertext when master keys are used — is useless against IPA.

Also, a DPA attack is restricted to points in the algorithm where the plaintext (or ciphertext) interacts directly with the key; this is because the differential traces are based on a "selection function" that predicts a bit value based on a small number of plaintext bits and a small number of key bits. In practice this usually means that a DPA attack is restricted to the beginning (if plaintext) or ending (if ciphertext) of a cryptographic algorithm; for example, DPA attacks on DES generally concentrate on either the first or last couple of rounds.

In contrast, an IPA attack is as capable of looking at the middle of an algorithm as at the beginning or end. This can be an important advantage in cases where some intervening processing is applied to the plaintext before the key is directly applied. For example, in a recent DPA attack on the AES candidate cipher Twofish [3], the attackers could only extract a certain "whitening key," after which significantly more analysis (including an exhaustive search of 9^8 possibilities) was needed to derive the master key. An IPA attack, on the other hand, can focus its attention on any (or all) of the intervening rounds, and thus extract the round keys without any further analysis.

Another set of advantages of IPA derives from its ability to do fast key extraction after a single lengthy profiling stage. This means that the cost of the profiling stage can be amortized over many key extractions, thus making an IPA attack economically feasible even if the cost of obtaining hundreds of power traces is large. In a DPA attack, by contrast, the attacker must collect a large number of traces for every key to be broken. If the cost of obtaining those traces is greater than the benefit of the key itself, the DPA attack is rendered impractical, whereas an IPA attack remains viable.

Fast key extraction also overcomes another defense against DPA: a protocol may disable a card after only a small number of operations, if the operator does not know the secret. Such a protocol can block DPA, but does not block IPA, where the many profiling traces can be obtained using a "friendly" card (one

where we are the legitimate owner and know the secret), and only a small number of traces are needed for each "unfriendly" card being attacked.

5 Defenses

Here are some suggestions to system implementers trying to protect against this class of attacks.

First, avoid handling key bits one at a time. Some ciphers are more amenable to this approach than others. Still, even a bit-oriented cipher like DES can sometimes be effectively protected: when a DES master key is inserted, its key schedule can be computed once for all time, and stored as six bits in each of 16×8 bytes, ready to be used with no further bit manipulation. For further protection, unused bits in any byte may be filled with irrelevant values instead of being set to zero.

Randomize the execution of the code. Where the order of operations is unimportant, such as S-box evaluation in DES, vary the order instead of using a fixed order. Insert random delays, even if only one instruction-time in duration.

Randomize the representation of data. Sometimes a quantity can be "blinded" by combining it with a randomly chosen constant; for example, value A may be maintained as $A \oplus K_1$, value B as $B \oplus K_2$, and $A \oplus B$ computed as $(A \oplus K_1) \oplus (B \oplus K_2) \oplus (K_1 \oplus K_2)$. When a single bit must be handled, consider representing the bits 0 and 1 as "01" and "10".

Limit the number of times a key can be used without confirmation of legitimacy, while simultaneously reducing the attacker's signal-to-noise ratio with filters or generators of random noise. The addition of noise will prevent key extraction from a captured trace of a legitimate transaction, and the limit on key probes will discourage key extraction from a stolen card.

Although no single defense makes a system impervious to IPA, and new attacks can be expected in the future, adding a variety of these countermeasures will likely increase the difficulty of IPA attacks, reducing, one would hope, both the number of potential attackers and the probability of any given attacker's succeeding.

References

1. Eli Biham and Adi Shamir. "Power Analysis of the Key Scheduling of the AES Candidates," *Second Advanced Encryption Standard Candidate Conference*, Rome, March 1999.
2. Eli Biham and Adi Shamir. "Differential Fault Analysis of Secret Key Cryptosystems," in *Advances in Cryptology — Crypto '97*, Lecture Notes in Computer Science Vol. 1294, p. 513–525. 1997.
3. Suresh Chari, Charanjit Jutla, Josyula R. Rao, and Pankaj Rohatgi. "A Cautionary Note Regarding Evaluation of AES Candidates on Smart-Cards," *Second Advanced Encryption Standard Candidate Conference*, Rome, March 1999.

4. Paul Kocher. "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS and Other Systems," in *Advances in Cryptology — Crypto '96*, Lecture Notes in Computer Science Vol. 1109, p. 104–113. 1996.
5. Paul Kocher, Joshua Jaffe, and Benjamin Jun. "Introduction to Differential Power Analysis and Related Attacks". `http://www.cryptography.com/dpa/technical/index.html`. 1998.
6. Oliver Kömmerling and Markus G. Kuhn. "Design Principles for Tamper-Resistant Smartcard Processors," in *Proceedings of the USENIX Workshop on Smartcard Technology (Smartcard '99)*, USENIX Association, p. 9–20. 1999.