Examining Smart-Card Security under the Threat of Power Analysis Attacks

Thomas S. Messerges, *Member*, *IEEE*, Ezzat A. Dabbish, *Member*, *IEEE*, and Robert H. Sloan, *Senior Member*, *IEEE*

Abstract—This paper examines how monitoring power consumption signals might breach smart-card security. Both simple power analysis and differential power analysis attacks are investigated. The theory behind these attacks is reviewed. Then, we concentrate on showing how power analysis theory can be applied to attack an actual smart card. We examine the noise characteristics of the power signals and develop an approach to model the signal-to-noise ratio (SNR). We show how this SNR can be significantly improved using a multiple-bit attack. Experimental results against a smart-card implementation of the Data Encryption Standard demonstrate the effectiveness of our multiple-bit attack. Potential countermeasures to these attacks are also discussed.

Index Terms—Cryptography, data encryption standard (DES), security, implementation attack, power analysis attack, smart card.

1 Introduction

Cryptographers have traditionally analyzed the security of ciphers by modeling cryptographic algorithms as ideal mathematical objects. A modern cipher is conventionally modeled as a black box that accepts plaintext inputs and provides ciphertext outputs. Inside this box, the algorithm maps the inputs to the outputs using a predefined function that depends on the value of a secret key. The black box is described mathematically and formal analysis is used to examine the system's security. In a modern cipher, an algorithm's security rests solely on the concealment of the secret key. Thus, attack strategies often reduce to methods that can expose the value of this secret key. Unfortunately, hardware implementations of cryptosystems can leak information about this secret key. Adversaries now have another avenue of attack.

In June of 1998, Kocher et al. [1] announced an attack against smart-card microprocessors. They reported that the secret key of an executing cryptographic algorithm could be extracted by monitoring the power consumption of a smart card. Their initial report [1] and follow-up paper [2] introduced the theory of power analysis attacks. In these reports, Kocher et al. stated that virtually all smart cards were vulnerable to these attacks; however, they omitted experimental data and attack details. Our paper aims to fill in this gap and to extend on previous research. We begin by briefly reviewing traditional cryptographic attacks. Since a power analysis attack is a type of implementation attack, we also review implementation attacks. We discuss why smart cards are a concern. Then, in some detail, we look at both simple power analysis (SPA) and differential power analysis (DPA) attacks against the data encryption standard (DES) algorithm [3]. The noise characteristics of the power signals are examined and an approach to model the signal-to-noise ratio (SNR) is proposed. We show how the SNR of a DPA attack can be significantly improved using a new multiple-bit attack. One of our main contributions is to provide experimental results showing actual attacks against smart-card implementations. We demonstrate that our multiple-bit attack is potentially more powerful than the attacks originally reported by Kocher et al. [1]. Finally, we discuss potential countermeasures that can be used to prevent these attacks. A preliminary version of this paper was published at a USENIX smart-card conference [4] and in [5].

• *T.S. Messerges and E.A. Dabbish are with Motorola Labs, Motorola Inc., 1301 East Algonquin Road., Schaumburg, IL 60196. E-mail: {Tom.Messerges, Ezzy.Dabbish}@motorola.com.*
• *R.H. Sloan is with the Department of Computer Science, University of Illinois at Chicago, 851 S. Morgan Street, Room 1120, Chicago, IL 60607. E-mail: sloan@cs.uic.edu.*

Manuscript received 30 Nov. 2000; accepted 14 Sept. 2001.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 113228.

1.1 Mathematical Attacks

Techniques such as differential [6] and linear [7] cryptanalysis, introduced in the early 1990s, are representative of traditional mathematical attacks. Differential and linear cryptanalysis attacks work by exploiting statistical properties of cryptographic algorithms to uncover potential weaknesses. These techniques are very useful for exploring weaknesses in algorithms represented as mathematical objects. For example, statistical analysis of a cryptosystem's inputs and corresponding outputs can expose the value of the secret key. A traditional mathematical approach often results in attacks that are of considerable theoretical interest. For example, differential cryptanalysis on all 16 rounds of the DES algorithm requires 2^{47} chosen plaintext inputs and analysis of 2^{36} ciphertext outputs in 2^{37} time [8]. This attack is interesting because it provides an improvement over a brute-force attack, where an average of 2^{55} keys need to be searched. Also, this attack, as with others developed using a mathematical approach, does not depend on a particular implementation. Thus, these attacks can be more broadly applied.

Although interesting, a theoretical attack is not necessarily practical. Traditional attacks often require the acquisition and manipulation of extremely large amounts of data.

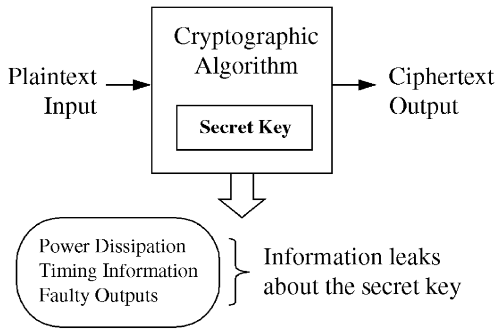

Fig 1. Side-channel information leaks from a cryptographic device. Implementations of cryptographic algorithms are traditionally analyzed as a black box with inputs and outputs. However, a particular implementation may leak unintended information that can lead an attacker to learning the secret key and breach the system's security.

When faced with these difficulties, real attackers will look for easier ways to breach the security of a cryptosystem. Attacks that exploit weaknesses in a particular implementation are an attractive alternative and are often more likely to succeed in practice [9].

1.2 Attacks

The realities of a physical implementation can be extremely difficult to control and often result in the unintended leakage of side-channel information. Fig. 1 illustrates how a cryptosystem can leak information. The leaked information is often correlated to the secret key; thus, adversaries monitoring this information may be able to learn the secret key and breach the security of the cryptosystem.

Techniques developed by Kelsey et al. [10] show that surprisingly little side-channel information is required to break some common ciphers. Whereas traditional mathematical techniques can be difficult to implement, the techniques that monitor information leakage often lead to more practical attacks. Attacks have been proposed that use leaked information, such as timing measurements [11], [12], electromagnetic emissions [13], and faulty hardware [14], [15]. Eliminating side-channel information or preventing it from being used as a means of attack is an active area of research.

1.3 Implementations on Smart Cards

A growing number of researchers are beginning to address the problems presented by implementation attacks. Systems that rely on smart cards [16] to provide security are of particular concern. A smart card is a plastic card, the size of a credit card, which contains an embedded microprocessor capable of running a variety of cryptographic software applications.

The pin configuration for a smart card is dictated by international standard ISO7816-2 [17] There are only six contacts to the outside world: the power supply, the reset signal, the clock input, the ground, the EEPROM programming voltage supply, and the input/output line. All communication to the card is via a single input/output line using a serial communication protocol defined in ISO7816-3 [18]. Higher-level commands and communications are governed by the standard ISO7816-4 [19]. A smart card is used by placing it into a device called a smart-card reader. The reader supplies the necessary inputs to the card and controls a transaction.

Some of the latest smart cards are also being developed with a contactless interface. Power for these contactless cards is provided via a Radio Frequency (RF) signal emitted from the smart-card reader. An antenna on a contactless card inductively couples to this RF signal and a circuit converts the RF power into power that can be used by the smart-card's microprocessor. All communication to and from a contactless card is also via an RF signal. The standard ISO14443 governs the protocols for the contactless interface [20].

Smart cards are often used to store cryptographic keys and execute cryptographic algorithms. The cards often use cryptography for protecting data stored on the card [21] or for securing applications running on the card (e.g., the Java Card [22]). Traditional smart cards have relatively simple microprocessors. For example, many smart-card microprocessors are still 8-bit processors and are often based on classic architectures, such as the Motorola HC05. In 1998, European researchers developed a more advanced RISC-based architecture for a smart-card microprocessor [23]. As of the year 2001, smart-card products with more powerful 16 and 32-bit processors are becoming available. Also, smart-card processors with cryptographic hardware support for computationally intense public-key cryptographic operations are available.

Smart-card processors are often viewed as tamper-resistant devices that are secure against all but the most determined and well-financed attackers. However, Anderson and Kuhn [24] point out that this reliance on the tamper resistance of smart cards needs to be carefully scrutinized. Such scrutiny has become even more prudent in light of the recent attacks using side-channel information. It is often the case that important data stored on smart cards, such as a cryptographic key or an authentication certificate, needs to be kept secret to prevent counterfeiting of cards or the breaking of a system's security (see, e.g., [25]). Such systems are potentially vulnerable because every time the smart card performs a computation using the secret data, side-channel information may be leaked.

Power consumption is a potential source of side-channel information. For a smart card, power is supplied by an external source that can often be directly observed. Ultimately, all calculations performed by a smart card operate on logical 1s or 0s. Current technological constraints result in different power consumptions when manipulating a logical 1 compared to manipulating a logical 0. An attacker of a smart card can monitor such power differences and obtain useful side-channel information. DPA, invented by Kocher et al. [1], is a statistical approach to monitoring such power signals from a smart card. Kocher et al. developed a specific DPA attack against smart cards running the DES algorithm. They warned that DPA is remarkably powerful and can even be used to monitor the actions of a single transistor within a smart card.

One purpose of our paper is to introduce techniques that help maximize the side-channel information. Whereas Kelsey et al. [10] showed how little side-channel information is required by an attacker, our paper takes an alternate approach and shows how such information can be max-

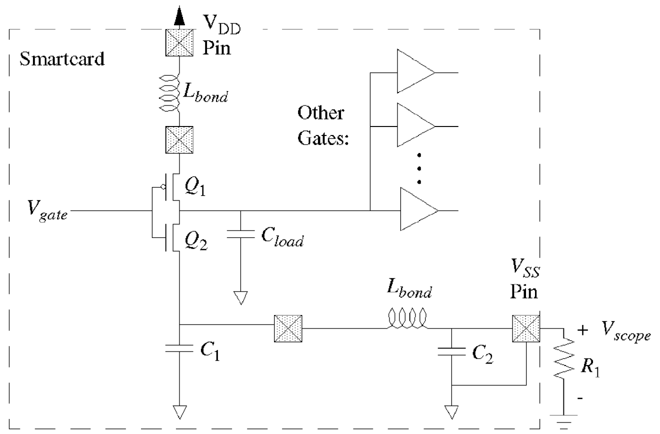

Fig 2. Measuring power consumption of a smart card. All power signals in this report were measured across V_{scope}, where R_1 was a 16 ohm resistor. A number of 8-bit microprocessor-based smart cards were examined and all produced results similar to those reported in this paper.

imized. Adversaries will obviously choose attacks that maximize side-channel information, so it is important that the strongest attacks be considered when designing defensive strategies.

2 Measuring Smart-Card Power Dissipation

The circuit shown in Fig. 2 gives a simple lumped component model that is useful for understanding power dissipation measurements. Power dissipated by the smart card can be monitored at the ground pin (V_{SS}) of the smart card by using a small resistor (R_1) in series between the V_{SS} pin on the card and the true ground. Current moving through R_1 creates a time varying voltage that can be sampled by a digital oscilloscope. This current charges and discharges capacitors C_1, C_2, and C_{load} and flows out of the smart card through a bond wire that acts as an inductor, L_{bond}. The values of the inductor, L_{bond}, and the capacitors are different in every smart card and will determine the shape of the power signal that is observed at V_{scope}. In a CMOS circuit, most power is dissipated when the circuit is clocked. This is known as dynamic power dissipation [26]. As V_{gate} changes from 0 to 5 volts, the transistors Q_1 and Q_2 are both conducting for a brief period, causing current to flow from V_{dd} to ground. Also during this time, the capacitor C_{load} will be discharged (or charged), causing more (or less) current to flow through the V_{SS} pin.

Information useful to a cryptanalyst is leaked. The amount of current being drawn when the circuit is clocked is directly related to the change of state of C_{load} or the current drawn by the other gates attached to C_{load}. On a microprocessor, each clock pulse causes many bit transitions to occur simultaneously. In a typical smart-card microprocessor, a large portion of the power dissipation occurs in the gates attached to internal buses. These buses present large capacitive loads that can readily leak information to potential attackers. Our experiments confirmed that activity on the data and address bus can be a dominant cause of power consumption changes. These changes were observed at V_{scope}. Thus, whenever the secret key or data correlated to this key is manipulated, a microprocessor can leak damaging information that may be observed at V_{scope}.

In our experiments, we observed two types of information leakage from the data bus: Hamming weight leakage and transition count leakage. Hamming weight information leaks when the dominant source of current is due to the discharging of C_{load}. One situation where Hamming weight information can leak is when a design with a precharged bus is used. In this case, the number of 0s driven onto the precharged bus directly determines the amount of current that is being discharged. Transition count information can leak when the dominant source of current is due to the switching of the gates that are driven by the data bus. When the data bus changes state, many of the gates driven by the bus will briefly conduct current. Thus, the more bits that change state, the more power that is dissipated.

2.1 Power Analysis

An SPA attack [1] involves providing an input to a cryptographic device and then visually inspecting the system's power consumption. In this section, we augment the work of Kocher et al. by providing further analysis of SPA attacks and by giving some detailed examples of attacks that we tested in actual smart-card hardware. Different attacks are possible depending on the cryptographic algorithm and the capabilities of the attacker. In some situations, the attacker may be allowed to run only a very small number of encryption or decryption operations. For example, this situation is possible when a person hands over their smart card to a dishonest merchant when making a purchase. Other times an attacker may have unlimited access to the card, such as a person attacking their own smart card trying to learn its internal secrets. The most powerful attackers not only have unlimited access, but also have detailed knowledge of the software and hardware running in the card. In each of these cases, the attacker could measure the power consumption waveform, store the data using a digital oscilloscope, and process the information to learn the secret key.

If an attacker can determine where certain instructions are being executed, it can be relatively simple to extract useful information. For example, in a software implementation of DES the "permuted choice 1" subroutine manipulates all of the key bytes. In one of the HC05-based smart cards, we examined it was possible to determine the Hamming weight of each key byte. This Hamming weight was determined by measuring the pulse height of the power consumption signal at the exact cycle of the instruction that accessed the key byte. Knowledge of the Hamming weight of each key byte can be used to reduce the number of keys that need to be checked during a brute-force attack. A slight reduction of the search space improves the efficiency of a brute-force attack and can make ciphers with small key sizes, such as DES, significantly weaker. We quantify this gain in efficiency in Proposition 1.

Proposition 1. *If the attacker knows the Hamming weight of each of the k words that comprise the secret key, where each word has n bits of key data, then the average brute-force key search space size is reduced from 2^{nk} to*

$$\left[\sum_{m=0}^{n}\binom{n}{m}^2 \Big/ 2^n\right]^k.$$

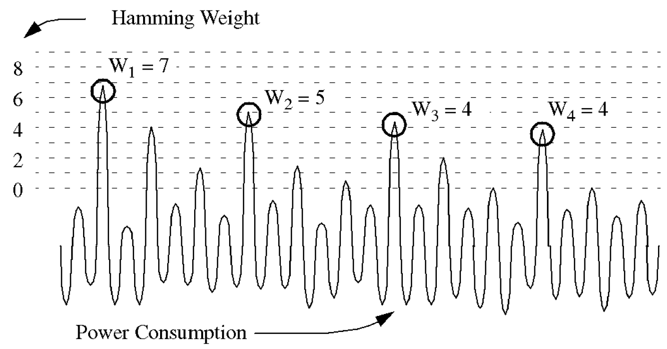

Fig 3. Measuring Hamming weight information from a power consumption signal. The above simulated power consumption signal illustrates how it is possible to extract Hamming weight information about the data being processed. The weight of the byte being processed is proportional to the height of the power consumption pulse. The weights w_i can be measured using a predetermined scale such as the one shown on the left and indicated with the horizontal lines. Depending on how much of this information is available, it might be possible to construct equations and use this information to completely break a cipher.

Proof. The attacker would normally need to try all 2^n possible keys for each of the k words, thus this key search size is 2^{nk}. Now, suppose the attacker knows that an n-bit word has a Hamming weight of m. The number of possible n-bit words with Hamming weight m is

$$\binom{n}{m}$$

and the probability of a random n-bit word having this Hamming weight is

$$\binom{n}{m} \Big/ 2^n.$$

Considering all values of Hamming weight m from 0 to n, it is clear that the average key search space for one n-bit word with known Hamming weight is

$$\sum_{m=0}^{n} \binom{n}{m}^2 \Big/ 2^n.$$

There are k of these words that need to be searched and the attacker knows the Hamming weight of each of these k words, hence the final result. □

As an example of Proposition 1, consider an implementation of DES in an 8-bit microprocessor, where there are 7 bytes of key data (i.e., $n = 8$ and $k = 7$). If the attacker knows the Hamming weight of all the key bytes, then the brute-force search space is reduced from 2^{56} to about 2^{40} keys. This is just an example; with algorithms using more key bits than DES or with triple-DES, knowing Hamming weight information alone does not help much with this type of brute-force attack.

2.2 on Simple Power Analysis

We now present a new, more powerful, attack that can be used when slightly more information is available: information about shifted versions of the key bytes. In DES, such information can be leaked when shifting the C and D registers. Recall that, in DES, the C and D registers hold key bits that are rotated left by 1 or 2 bits each round. The C and D register bytes must be accessed every round to determine the subkey for that round. If an attacker can learn the Hamming weight of each byte for eight of the C and D shifts, then there is enough information to solve for the value of every key bit. The attacker can simply solve the following equation for the length 56 vector of key bits:

$$A\vec{k} = \vec{w}, \tag{1}$$

where $\vec{w}$ is a vector of Hamming weights, w_i, and A_{ij} is a 0-1 matrix such that A_{ij} is 1 if and only if weight w_i includes key bit k_j. Fig. 3 shows how Hamming weight information might be collected from a simulated power consumption signal. In this figure, the attacker is assumed to know the location of the pulses that convey the Hamming weight information. The job of the attacker is then very simple. The heights of the pulses directly indicate the Hamming weight of the data being processed at that cycle. With a few characterization tests, the attacker can learn which heights correspond to which weights and then collect and analyze the information needed for solving (1). Even algorithms with more than 56 key bits, such as triple-DES, would be vulnerable to this attack.

Our experiments confirmed that poor implementations of DES are almost always vulnerable to SPA attacks. Shifting the key bytes or the use of conditional branches to test bit values can be especially vulnerable. Also, if portions of the code run in variable time, power analysis could be used to enable a timing attack [11]. Fortunately, implementers of cryptographic algorithms have known about these issues for a number of years. Kocher et al. [1] reported that it is not particularly difficult to build SPA-resistant devices. However, some cryptographers may disagree with this statement.

2.3 Leakage Experiment

Actual experimental results that show transition count leakage from a smart card containing an 8-bit HC05-based microprocessor are given in Fig. 4. In this figure, an 8-bit data byte is transferred from memory into a register. Initially, the bus contains a fixed memory address, but, after a clock pulse, data from memory is put onto the bus. The magnitude of the voltage pulse is directly proportional to the number of bits that changed, which is the Hamming distance between the previous and current values on the data bus. Waveforms for different values of memory data are plotted on top of each other to effectively show this relationship. The difference in voltage between transitions i and $i+1$ is about 6.5 mV. We observed similar plots for smart cards that leak Hamming weight information. The type of information that a particular smart card leaks depends on the circuit design of the microprocessor, the type of operation being performed by the card, or the memory being accessed. For instance, we observed different information leaking from EEPROM versus RAM. Knowing which type of information is leaked will enable an adversary to optimize an attack strategy.

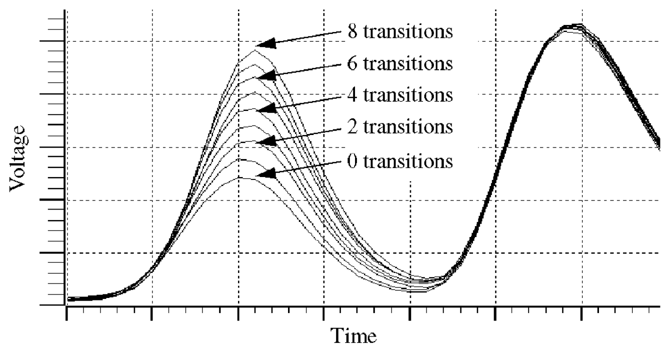

Fig 4. Voltage levels track the number of bit transitions. These results show how the processed data can affect the voltage levels. The nine overlayed waveforms correspond to the power traces when different data is accessed by a load (LDA) instruction. These results were obtained using an HC05-based smart card and by averaging the power signals across 500 samples in order to reduce the noise content. The difference in voltage between transitions i and $i+1$ transitions is about 6.5 mV.

2.4 Power Analysis

A DPA attack, invented by Kocher et al. [1] and reviewed here, is more powerful than an SPA attack because the attacker does not need to know as many details about how the algorithm was implemented. The technique also gains strength by using statistical analysis to help recover side-channel information. The objective of the DPA attacks described in this paper is to determine the secret key used by a smart card running the DES algorithm. These techniques can also be generalized to attack other cryptographic algorithms. In this section we expand on Kocher et al.'s work by providing a theoretical basis for the DPA attack. We use these theoretical results to create more powerful DPA attacks, which we describe in Section 4.

A DPA attack begins by running the encryption algorithm for N random values of plaintext input. For each of the N plaintext inputs, PTI_i, a discrete time power signal, $S_i[j]$, is collected and the corresponding ciphertext output, CTO_i, may also be collected. The power signal $S_i[j]$ is a sampled version of the power consumed during the portion of the algorithm that is being attacked. The i index corresponds to the PTI_i that produced the signal and the j index corresponds to the time of the sample. The $S_i[j]$ are split into two sets using a partitioning function, $D(\cdot)$:

$$\begin{aligned} S_0 &= \{S_i[j] | D(\cdot) = 0\} \\ S_1 &= \{S_i[j] | D(\cdot) = 1\}. \end{aligned} \tag{2}$$

The next step is to compute the average power signal for each set:

$$\begin{aligned} A_0[j] &= \frac{1}{|S_0|} \sum_{S_i[j]\in S_0} S_i[j] \\ A_1[j] &= \frac{1}{|S_1|} \sum_{S_i[j]\in S_1} S_i[j], \end{aligned} \tag{3}$$

where $|S_0| + |S_1| = N$. By subtracting the two averages, a discrete time DPA bias signal, $T[j]$, is obtained:

$$T[j] = A_0[j] - A_1[j]. \tag{4}$$

Selecting an appropriate D function results in a DPA bias signal that can be used to verify guessed portions of the secret key. An example of such a D function for the DES algorithm is

$$D(C_1, C_6, K_{16}) = C_1 \oplus \mathrm{SBOX1}(C_6 \oplus K_{16}), \tag{5}$$

where C_1 represents the one bit of CTO_i that is XORed with bit #1 out of S-box #1, C_6 represents the six bits of CTO_i that are XORed with last round's subkey, K_{16} represents the six bits of the last round's subkey that feed into S-box #1, and $\mathrm{SBOX1}(x)$ is a function returning bit #1 resulting from looking up x in S-box #1.

This particular D function is chosen because, at some point during a DES implementation, the software needs to compute the value of this bit. When this occurs or any time data containing this bit is manipulated, there will be a slight difference in the amount of power dissipated depending on whether this bit is a 0 or a 1. If this difference in the power signal is ε and the instructions manipulating the D bit occur at times j', the following expected difference equation results when $j = j'$:

$$E\{S_i[j] \mid D(\cdot) = 0\} - E\{S_i[j] \mid D(\cdot) = 1\} = \varepsilon. \tag{6}$$

When j is not equal to j', the smart card is manipulating bits other than the D bit and the power dissipation is independent of the D bit; thus, when $j \neq j'$:

$$E\{S_i[j]|D(\cdot) = 0\} = E\{S_i[j]|D(\cdot) = 1\} = E\{S_i[j]\}$$

and

$$E\{S_i[j]|D(\cdot) = 0\} - E\{S_i[j]|D(\cdot) = 1\} = 0. \tag{7}$$

As the number N of PTI inputs is increased, (4) converges to the expectation equation:

$$\lim_{N\to\infty} T[j] = E\{S_i\cdot[j]|D(\cdot) = 0\} - E\{S_i\cdot[j]|D(\cdot) = 1\}. \tag{8}$$

Thus, a review of (6), (7), and (8) shows that if enough PTI samples are used, $T[j]$ will show power biases of ε at times j' and will converge to 0 all other times. However, due to small statistical biases in the S-box outputs, the assumption of independence in (7) for DES is not completely true. In reality, $T[j]$ will not always converge to 0; however, the largest biases will occur at times j'.

One input to the D function is K_{16}, 6 bits of the subkey. The attacker does not know these bits, but can use brute force and try all 2^6 possible values. For each guess, the attacker can construct a new partition for the power signatures and get a new bias signal, $T[j]$. If the proper value for K_{16} was chosen, then the bias signal will show spikes whenever the D bit was manipulated. If the value for K_{16} was not chosen correctly (i.e., the wrong subkey bits were guessed), then the resulting $T[j]$ will not show any significant biases. Using this approach, an attacker can determine the six subkey inputs to S-box #1 at round 16 of DES. Repeating this approach for the seven other DES S-boxes allows the attacker to learn all 48 bits of the round 16 subkey. The remaining 8 bits can be found by brute force or by successively applying this approach backward to previous rounds. Kocher et al. tested this attack and claimed it was successful on all smart cards they examined.

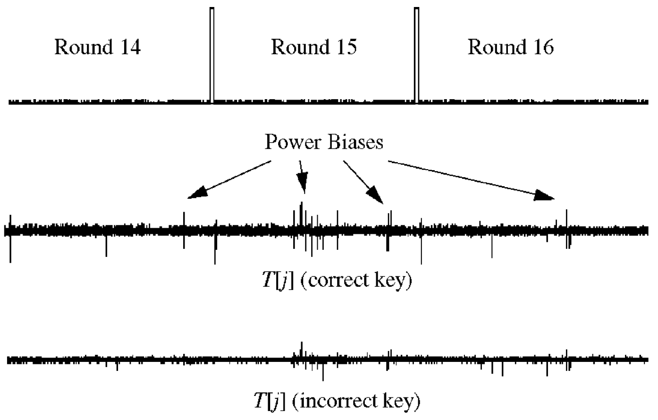

Fig 5. DPA results show power bias signals for correct key and incorrect key. The power biases are about 6.5 mV for the correct key and about half the size for the incorrect key. The voltage SNR for the correct key is 17.3. A total of $N = 1,300$ power signals were used to generate the above plots.

2.5 Power Analysis Attack on DES

We set up an experiment to perform a conventional DPA attack on the smart card characterized by the power trace shown in Fig. 4. The smart card that we tested implements the DES algorithm in software. The power consumption signals for the last three DES rounds were collected and analyzed. Fig. 5 shows the results of an attack. The result of Fig. 5 was first documented in the preliminary conference version of this paper [4] and, to the best of our knowledge, is the first openly published result of an actual DPA attack on the DES algorithm. In Fig. 5, the bias signal for a correctly chosen key is compared to that of an incorrectly chosen key. The signal for the incorrect key shows a few small biases, but the correct key has biases that are twice as large, so it is easily recognized.

We created the signals in Fig. 5 using $N = 1,300$. This attack reveals 6 bits of the DES secret key for S-box #1. However, these same 1,300 signals can also be used seven more times with the remaining S-boxes. Thus, a total of 48 bits can be revealed using DPA and the remaining eight DES key bits can be easily found by using a brute-force attack. The time to collect the 1,300 signals mainly depends on how fast the signals can be acquired with an oscilloscope. In our case, a PC software program was used to control an oscilloscope. This program could acquire enough signals for an attack in less than an hour. The attack equipment, which included a PC, a general-purpose interface bus (GPIB) card, and a digital oscilloscope, can be acquired for less than US 10K.

3 Filtering and Modeling Power Analysis Noise

One way to strengthen a power analysis attack is to reduce the unwanted noise in the power signal by using filtering strategies. Noise reduction is particularly important when trying to measure side-channel emanations because the magnitude of these signals can be extremely small. Kocher et al.'s DPA uses averaging to reduce noise; here, we study additional strategies that can be combined with averaging. When DPA attacks were first announced, experts reported that one way to prevent a power analysis attack is to mask the side-channel information with random calculations that increase the measurement noise [27]. A good understanding of the achievable noise levels is needed to evaluate how much noise needs to be added. In this section, we describe the noise present during a power analysis attack and develop a model for the DPA signal-to-noise ratio (SNR).

3.1 Noise Characteristics

There are at least five potential types of noise present when performing power analysis of smart cards: external, intrinsic, quantization, sampling, and algorithmic. External noise is generated by external sources and is coupled into the smart card. Intrinsic noise is due to the random movement of charge carriers within conductors. Quantization noise is due to the quantizer in the Analog-to-Digital (A/D) converter that is used to sample the power signal. Sampling noise arises from the discrete time digitalization of the power signal. Lastly, algorithmic noise is due to the variations of the data bytes being processed by the smart card.

Intrinsic and quantization noise are small when compared to present day side-channel signals. External noise can be reduced through careful use of measuring equipment, good circuit design practices, and filtering. Sampling noise can be minimized by using a faster sampling clock or by synchronizing the sampling clock to the microprocessor's clock. Algorithmic noise can be reduced if an attack strategy uses unbiased random data that can be averaged out.

3.2 Filtering the Noise

Filtering can be used to reduce noise, but care needs to be taken so that the components necessary to create a power bias signal are not affected. For instance, filtering will not work on algorithmic noise because any filter would also reduce the desired bias signal. If the noise is assumed to be white noise, then the optimal filter that maximizes the SNR is the matched filter. We designed and tested such a filter. The original voltage SNR in Fig. 5 was 17.3; after using the matched filter, it increased to 23.2. This gives some indication of how much an attacker with a perfectly matched filter can improve the SNR.

3.3 Modeling the DPA Signal-to-Noise Ratio

The DPA bias signal contains two distinct parts, a signal portion and a noise portion. The signal portion contains the information (i.e., bias spikes) used by an attacker to judge the correctness of guessed key bits. The noise portion is the remaining part of the signal. A successful DPA attack requires that an attacker can detect the signal over the noise. This situation is analogous to the classic communications problem of receiving a signal that was transmitted over a noisy channel. In communication theory, the SNR determines the probability that accurate information can be communicated across a channel. Likewise, in a DPA attack the SNR determines the effective amount of information that is leaked, thus setting the probability that an attack is successful.

For a DPA attack, there are two problems that an attacker faces. The first is trying to detect the bias spikes in a single DPA bias signal. The second is to determine which DPA bias signal has the largest bias spikes. These two problems result in two SNRs that affect the probability of a successful attack. We refer to the SNR corresponding to the first problem, which concerns a single DPA bias signal, as the *intrasignal* SNR. We refer to the SNR of the second problem, which concerns all of the DPA bias signals, as the *intersignal* SNR. Referring back to (6), the intrasignal SNR characterizes the level of difficulty an attacker will have in finding the spikes corresponding to the times j' in the DPA bias signal $T[j]$. The intersignal SNR characterizes the level of difficulty an attacker will have in finding the $T[j]$ with the largest spikes.

DPA attacks are so often successful because averaging reduces noise energy. The reduction of noise consequently reveals spikes or concentrations of signal energy that occur at constant positions in time. To see the effect of averaging on the intrasignal SNR, it is helpful to develop a noise model. In developing this noise model, the external, intrinsic, sampling, and quantization noise are lumped together and referred to as *nonalgorithmic noise*. In a DPA bias signal, the average variance of the nonalgorithmic noise can be measured. The average variance, σ^2, of the nonalgorithmic noise signal $N[j]$ with M samples is given by

$$\sigma^2 = \frac{1}{M}\sum_{j=0}^{M-1}(N[j] + E\{N[j]\})^2. \tag{9}$$

The *algorithmic noise* of the intrasignal SNR will occur only in the parts of the DPA bias signal that correspond to data-dependent portions of the algorithm. Data-dependent portions of the power signal vary when the key or plaintext input to the algorithm change. The percentage of the signal that contains algorithmic noise is denoted as α. The value of α is typically small because most of the processor's clock cycles are used for nondata-dependent functions, such as opcode fetching, loop overhead, or address decoding.

In the original DPA attack [1], the attacker needs to detect bias spikes (i.e., the signal) in a DPA bias signal. The size of these spikes is determined by the voltage difference that is observed in the power signal when manipulating two data elements with Hamming distance equal to 1. This voltage difference, which is also the signal size, is denoted as ε. The value of ε can be determined using characterization plots such as the one that was given in Fig. 4. The magnitude of ε will determine the size of the bias spikes and will also affect the magnitude of algorithmic noise.

To summarize, a DPA attack requires that an attacker detect a signal of size ε embedded in noise. The noise is composed of both algorithmic and nonalgorithmic noise. For our models, we assume that the signal and noise can be described as random variables R_{signal} and R_{noise} having probability distributions D_{signal} and D_{noise}, respectively. The variable R_{signal} has a mean equal to ε and the variable R_{noise} has a mean equal to 0. Both random variables have variances that depend on the algorithmic and nonalgorithmic noise. For a DPA attack, the variance of the nonalgorithmic noise is based on σ^2 and the variance of the algorithmic noise is dependent on ε and α. The job of the attacker is to acquire power consumption data and decide whether this data has probability distribution D_{signal} or D_{noise}. We now develop a model that uses the means and variances of these distributions to describe the intrasignal SNR.

Theorem 2. *A DPA attack using N signals on an n-bit processor, with signal size ε, average nonalgorithmic noise variance σ^2, and percentage of algorithmic noise α, has a voltage intrasignal SNR that can be modeled by*

$$SNR = \frac{\varepsilon\sqrt{N}}{\sqrt{8\sigma^2 + \varepsilon^2(\alpha n + n - 1)}}. \tag{10}$$

Proof. First, consider the noise portion of the DPA bias signal, which has probability distribution D_{noise}. The average nonalgorithmic noise variance of $S_i[j]$ is given as σ^2. We assume that $S_i[j]$ is independent of $S_k[j]$ when $i \neq k$, so if the N signals are equally split between sets S_0 and S_1, then the average variance of A_0 and A_1 (as defined in (3)) is $2\sigma^2/N$. Therefore, when A_0 and A_1 are combined, the points with distribution D_{noise} have an average nonalgorithmic noise variance of $4\sigma^2/N$. To find the contribution of the algorithmic noise, consider that a data word is n bits wide. If the data words being manipulated are assumed to be random, then the Hamming weight of a data word is a random variable with binomial distribution. This random variable has mean $n/2$ and variance (i.e., algorithmic noise) $n/4$. Averaging over N samples has a similar effect on the algorithmic noise variance as it did on the nonalgorithmic noise variance. Thus, the algorithmic noise is $(n/4)(4/N) = n/N$. The voltage difference corresponding to a Hamming distance of 1 is ε and the percentage of algorithmic noise is α, so the noise contribution due to the algorithmic noise is $\alpha\varepsilon^2 n/N$. Assuming that the algorithmic and nonalgorithmic noise are independent, the total noise (i.e., variance of D_{noise}) is $4\sigma^2 + \alpha\varepsilon^2 n/N$. The mean of this noise is 0.

The bias spike corresponding to the signal portion of the SNR has distribution D_{signal}. Similar to before, the noise portion, or variance, of the bias spike can be divided into algorithmic and nonalgorithmic noise. The nonalgorithmic noise is the same as before, $4\sigma^2/N$ and the algorithmic noise can be determined by noticing that the D function proposed in (5) has the effect of fixing one bit of the data being processed. If a data word is n bits wide, then the remaining unfixed bits in the word have average Hamming weight $(n-1)/2$ and variance (i.e., algorithmic noise) $(n-1)/4$. So, the signal has an algorithmic noise equal to $(n-1)\varepsilon^2/N$. Again, we assume that algorithmic and nonalgorithmic noise are independent, so the total noise variance is $(4\sigma^2 + (n-1)\varepsilon^2)/N$. The signal portion, or mean, of the bias spike is the signal size ε. A summary of these results yields the following statistical parameters for D_{noise}:

$$\begin{aligned} &E\{T[j] \mid j \neq j'\} = 0 \\ &\mathrm{var}\{T[j] \mid j \neq j'\} = \frac{4\sigma^2 + \alpha n\varepsilon^2}{N} \end{aligned} \tag{11a}$$

and the following statistical properties for D_{signal}:

$$\begin{aligned} &E\{T[j']\} = \varepsilon \\ &\mathrm{var}\{T[j']\} = \frac{4\sigma^2 + (n-1)\varepsilon^2}{N}. \end{aligned} \tag{11b}$$

Hence, the final voltage intrasignal SNR can be found by dividing the signal level ε by the square-root of the sum of the two variances in (11a) and (11b) , resulting in (10). □

Equation (10) can be checked with our previously measured experimental results. Setting $\sigma = 7.5$ mV (from our experiments), $\varepsilon = 6.5$ mV (from Fig. 4), $n = 8$, $N = 1,000$, and $\alpha = 0$, yields $\mathrm{SNR} = 7.5$. Experimental results using these parameters showed an SNR between 7 and 10, confirming that (10) is a valid approximation.

Theorem 2 gives a model for the intrasignal SNR, but does not explain the intersignal SNR. The intersignal SNR is dependent on the algorithm and biases that might be present in the particular *D* function that is being used for an attack. For instance, the biases of a DES S-box function might result in some key guesses that create stronger bias spikes than others. A noise model for the intersignal SNR is not developed in this paper because, in our experiments, whenever the intrasignal SNR was large enough to observe the DPA bias spikes, the intersignal SNR was also large enough to determine which DPA bias signal had the largest spikes.

4 Maximizing DPA Bias Signal

One way we discovered to improve the SNR in (10) is to increase the signal magnitude. We observed that the magnitude of the signal is dependent on the number of bits output by the partitioning function *D*. In (5), the *D* function outputs only one bit, but, in general, a *D* function could output *d* bits. Here, we extend Kocher et al.'s [1] work by examining DPA attacks that partition on more than one bit. A *d-bit DPA attack* is an attack where the *D* function outputs *d* bits. In this more general type of attack, the signal size is $\beta = d\varepsilon$, where ε is the 1-bit DPA signal size corresponding to the voltage difference seen between two data words with Hamming weight i and $i+1$. As was shown in the results given in Fig. 4 and also in experiments we performed on other smart cards, these differences appear to be approximately equal, for all i.

When performing the DPA attack there are now three sets used for partitioning:

$$\begin{aligned} S_0 &= \{S_i[j] \mid D(\cdot) = 0^d\} \\ S_1 &= \{S_i[j] \mid D(\cdot) = 1^d\} \\ S_2 &= \{S_i[j] \mid S_i[j] \notin S_0, S_1\}. \end{aligned} \tag{12}$$

If the *D* function outputs *d* 0s or *d* 1s, then the signal is placed into set S_0 and S_1, respectively. Otherwise, the signal is placed into set S_2 and is not used. Without loss of generality, we assume that the *D* function is chosen to be unbiased so that similar numbers of signals are placed in partitions S_0 and S_1. We refer to this new DPA attack as an "all-or-nothing *d*-bit DPA attack" because signals are used only when the corresponding *D* function outputs are all 0s or all 1s.

We use an approach similar to Theorem 2 to model the intrasignal SNR for a *d*-bit DPA attack. The resulting model confirms that a *d*-bit DPA attack is potentially more powerful than a 1-bit DPA attack.

Theorem 3. *An all-or-nothing d-bit DPA attack using N signals on an n-bit processor with 1-bit DPA signal size ε, average nonalgorithmic noise variance σ^2, and percentage of algorithmic noise α, has a intrasignal voltage SNR that can be modeled by*

$$SNR = \frac{d\varepsilon\sqrt{N}}{\sqrt{8\sigma^2 + \varepsilon^2(\alpha n + n - d)}}. \tag{13}$$

Proof. In a *d*-bit DPA, attack the noise portion of the DPA bias signal is unchanged from a 1-bit DPA attack, so the average variance and mean of the noise distribution D_{noise} are the same as summarized in (11a).

Similarly, the nonalgorithmic noise portion of the signal distribution D_{signal} is the same as before, $4\sigma^2/N$; however, the algorithmic portion is changed. The new *D* function has the effect of fixing *d* bits of the data being processed. If a data word is *n* bits wide, then the remaining unfixed bits in the word have average Hamming weight $(n-d)/2$ and variance (i.e., algorithmic noise) $(n-d)/4$. Thus, the signal has an algorithmic noise equal to $(n-d)\varepsilon^2/N$. Again, assuming that algorithmic and nonalgorithmic noise are independent, the total noise is $(4\sigma^2 + (n-d)\varepsilon^2)/N$. The signal portion of the bias spike is increased d times, so the resulting signal size is $d\varepsilon$. A summary of these results yields the following statistical parameters for D_{noise}:

$$\begin{aligned} E\{T[j] \mid j \neq j'\} &= 0 \\ \mathrm{var}\{T[j] \mid j \neq j'\} &= \frac{4\sigma^2 + \alpha n\varepsilon^2}{N} \end{aligned} \tag{14a}$$

and the following statistical properties for D_{signal}:

$$\begin{aligned} E\{T[j']\} &= d\varepsilon \\ \mathrm{var}\{T[j']\} &= \frac{4\sigma^2 + (n-d)\varepsilon^2}{N}. \end{aligned} \tag{14b}$$

Hence, the final voltage intrasignal SNR can be found by dividing the signal level $d\varepsilon$ by the square-root of the sum of the two variances of (14a) and (14b), thus resulting in (13).□

Notice that the resulting intrasignal SNR given by (13) is a multiplicative factor of *d* better than the conventional DPA SNR given by (10).

In a hardware or software implementation of a cryptographic function, the *d* output bits of the *D* function are actually processed by logical gates or stored in some type of memory device. During each clock cycle of the hardware, several bits are processed. For instance, during each clock cycle, a simple *n*-bit processor operates on *n* bits. The *d* output bits of the *D* function might correspond to these *n* bits.

We refer to a *d*-bit DPA attack as an "all-or-nothing *d*-bit DPA attack" when the outputs of the *D* function need to be all 1s or all 0s for the corresponding power signal to be used. We now give a "generalized *d*-bit DPA attack,", which is defined by the partitioning scheme:

$$\begin{aligned} S_0 &= \{S_i[j] \mid \mathrm{wt}[D(\cdot)] \leq n-d\} \\ S_1 &= \{S_i[j] \mid \mathrm{wt}[D(\cdot)] \geq d\} \\ S_2 &= \{S_i[j] \mid S_i[j] \notin S_0, S_1\}. \end{aligned} \tag{15}$$

In (15), $\mathrm{wt}(x)$ denotes the Hamming weight of x or the Hamming distance between x and a previous state. The value of n is the total number of bits output by the D function. As before, n might be the number of bits that are processed at a given clock cycle, such as in an n-bit processor. A special case of the partitioning scheme given by (15) is when $d = n$. In this case, the generalized DPA attack reduces to an all-or-nothing DPA attack. An interesting condition exists when $d \leq n/2$. In this case, a signal $S_i[j]$ can qualify for membership into both sets S_0 and S_1. This condition, however, will not pose a problem because, when the average signals of set S_0 and S_1 are subtracted, the duplicated signals will cancel each other out. Thus, there is no effect on the DPA bias signal $T[j]$.

4.1 Choosing a Partitioning Function

The D partitioning function in a d-bit DPA attack should be a function whose result is related to a certain state of the processor being attacked. For example, the D function could be an intermediate result that must be calculated during the running of the cryptographic algorithm. If the D function is chosen properly, then, at some point in time, the data being manipulated by the processor or cryptographic hardware will be precisely the data output by the D function. A successful attack is possible when the output of the D function is correlated to the secret key data. In general, there will be a number of D functions. Each D function will correspond to a disjoint set of secret key bits that an attacker wishes to learn. For example, in the previous 1-bit DPA attack on DES, there would be eight different D functions, one for each S-box function. The number of key bits that are correlated with the outputs of the individual D functions determines the number of DPA bias signals that an attacker will need to generate.

Proposition 4. *If each D function is correlated to a disjoint set of k key bits and there are a total of L key bits, where L/k is an integer, then the attacker will need to compute 2^k L/k DPA bias signals to learn all L key bits.*

Proof. Each D function will result in the attacker learning k distinct key bits, so it is clear that the attacker will need L/k such functions in order to learn all L key bits. When the attacker is using a particular D function to learn a group of k key bits, all possible guesses for those k key bits will need to be checked using a DPA bias signal. Thus, 2^k bias signals will need to be generated for each of the D function. Hence, the total number of bias signals will be 2^k L/k. □

For an attack on DES where each D function corresponds to an S-box function and the attacker is trying to determine the 48-bit subkey of the last round, $k = 6$ and $N = 48$. Thus, according to Proposition 4, a total of 512 bias signals must be calculated. Of course, to calculate each bias signal the attacker will need to average a large number of power signals. A designer can use the result of Theorem 3 to estimate the number of signals N that are required to get an adequate SNR.

4.2 Specific Multiple Bit DPA Attacks on DES

In DES, a 4-bit D function, based on the four bits output from an S-box function can be used to partition the sets S_0 and S_1. Equation (5) can be modified to output four bits:

$$D(C_6, K_{16}) = \mathrm{SBOX1}_4(C_6 \oplus K_{16}). \tag{16}$$

where $\mathrm{SBOX1}_4(x)$ returns all four bits resulting from looking up x in S-box #1. Similar D functions would be used for the other S-boxes to enable all the key bits to be eventually determined.

In general, other multiple-bit D functions can be defined. In software implementations, the S-box lookups are performed using a load instruction from a table storing all the S-box data. To compress the size of this table, the 4-bit S-box data is frequently stored in pairs of two or more, depending on the word size of the processor. If the attacker knows how the S-box data is packed into this table, it may be possible to increase the bit biases even further. For example, power signals with S-box lookups that yield many 0s can be separated from signals with S-box lookups containing many 1s. The number of bits that can be biased will proportionally increase the SNR.

Another way to mount an attack would be to partition S_0 and S_1 based on the addresses used for the S-box lookups rather than the data. The attacker forms two partitions, one that maximizes the number of address bus transitions and one that minimizes the number of address bus transitions. Power signatures from both partitions can be averaged and then subtracted. If the address bus is larger than the data bus, this method could achieve even larger biases.

4.3 Effectiveness of Multiple Bit DPA Attacks

The question arises as to whether partitioning schemes, such as the one in (16), will result in partitions that are unbiased. Recall that incorrect choices of guessed key bits should result in random power signals that average to 0. In the case of the partitioning given by (16), we ran experiments on a software implementation of DES to see what type of biases could actually be expected. The results for a generalized 8-bit, an all-or-nothing 4-bit, and a 1-bit DPA attack using $N = 1,000$ random PTI inputs and a typical key are graphed in Fig. 6. The graph shows the maximum value of the spikes in the bias signals for each of the 64 possible key guesses. The arrow in Fig. 6 indicates the correct key guess. The correct key exhibits the largest bias for each type of attack. Furthermore, the size of the bias increases as more bits are used.

We also ran experiments on a smart card to confirm the effective signal size obtained during the following attacks: an all-or-nothing 1-bit and 4-bit DPA attack, a generalized 8-bit DPA attack, and a generalized address-bit DPA attack. Due to characteristics of the implementation we examined, the average Hamming distance of the D function outputs was 6 bits for the 8-bit DPA attack and 6.5 bits for the address-bit DPA attack. The resulting signal levels, given in Table 1, show that multiple-bit DPA attacks are clearly superior to the conventional 1-bit DPA attack.

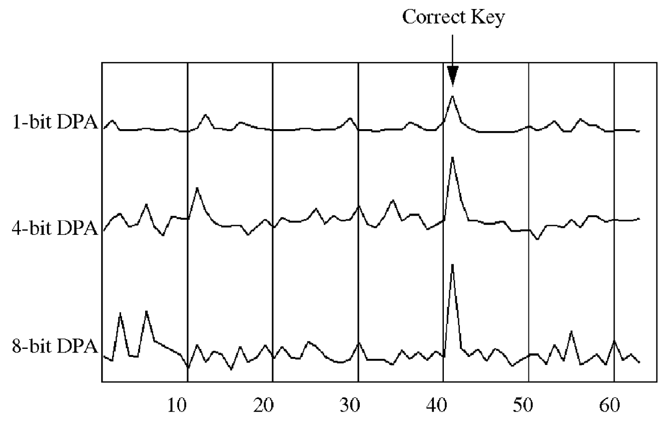

Fig 6. Experimental DPA biases generated for DES key = "8e 30 4f d1 ed ee 3b 0a." The horizontal axis indicates each 6-bit DES key guess and the vertical axis indicates the maximum level of the bias signal for each of the key guesses generated during a DPA attack. The correct key generates the most bias and the other keys show smaller biases. These results were obtained using an HC05-based smart card with $N = 1,000$ random PTI inputs.

4.4 Diminishing Returns for Multiple-Bit DPA

Sometimes the attacker cannot control all of the inputs to the partitioning function. In our specific example, the D function of (16) depends on the ciphertext output data. Since the attacker does not know the key, he cannot control this data. Therefore, when our specific d-bit attack is executed, the outputs of the D function will be unpredictable. Thus, many of the power signals will be placed in partition S_2 and go unused. However, if our D function is dependent on controllable plaintext inputs, then we could strategically choose inputs to avoid placing signals into partition S_2. Therefore, to avoid collecting unused signals, attackers will try to use a D function that has all of its nonsecret key inputs controllable. When this is not possible, a fraction of the power signals will not be usable and the effectiveness of multiple-bit DPA will diminish.

Theorem 5. *In a generalized d-bit DPA attack on an n-bit processor, where n is even, the average fraction of power signals that are usable is*

$$\begin{array}{ll} \left[2\sum_{k=d}^{n}\binom{n}{k}\right]\Big/2^n & \text{if} \quad d > n/2 \\ \quad 1 & \text{if} \quad 0 < d \leq n/2. \end{array}$$

Proof. It is clear that if $d \leq n/2$ then there are at least d 0s or d 1s, so all signals will be usable. If $d > n/2$ and n-bit data is processed with d or more 0s, then, according to (15), the corresponding power signal will be placed into set S_0. Likewise, if n-bit data is processed with d or more 1s, the corresponding power signal will be placed into set S_1. The number of n-bit data words with d or more 0s is

$$\sum_{k=d}^{n}\binom{n}{k}.$$

TABLE 1
Relative Signal Levels for Multiple-Bit DPA

Attack Type	Signal Level
1-bit DPA	9.3 mV
4-bit DPA	38.5 mV
8-bit DPA	79.5 mV
Address DPA	74.4 mV

Similarly, the number of n-bit data words with d or more 1s is

$$\sum_{k=d}^{n}\binom{n}{k}.$$

Assuming that the n-bit data being manipulated by the processor is uniformly distributed, then, when $d > n/2$, only

$$2\sum_{k=d}^{n}\binom{n}{k}$$

out of possible 2^n words correspond to power signals that are usable. □

The generalized d-bit DPA attack reduces to an all-or-nothing d-bit DPA attack when $d = n$. In this case, the average Hamming difference between the D function outputs of sets S_0 and S_1 is maximized. A maximal Hamming difference means that the size of the DPA bias spike will also be maximal; however, this also means that the attacker will have the smallest number of usable power signals.

Corollary 6. *In an all-or-nothing d-bit DPA attack, the average fraction of power signals that are usable is* $2/2^d$.

Proof. A D function with d output bits has 2^d possible outputs. By (12), only two outputs are usable: the all 0 case and the all 1 case. □

All of the above results indicate that the fraction of usable signals decreases exponentially as d is increased. Thus, an attacker might not be able to increase d too much without requiring an excessive number of power signals. The number of signals an attacker might need when running an all-or-nothing d-bit DPA attack is given in Corollary 7.

Corollary 7. *A sufficient condition to keep the intrasignal SNR of an all-or-nothing d-bit DPA attack the same as the intrasignal SNR of a 1-bit DPA attack is that the attacker obtain, on average,* $2^{d-1}/d^2$ *times the number of power signals used in the 1-bit DPA attack.*

Proof. Assume that, in the 1-bit DPA attack, N_1 signals were used to obtain a desired SNR. To obtain the same SNR in the all-or-nothing d-bit DPA attack, N_d signals will be needed. Then, by (13), N_d can be found by solving the inequality

$$\frac{d\varepsilon\sqrt{N_d}}{\sqrt{8\sigma^2 + \varepsilon^2(\alpha n + n - d)}} \geq \frac{\varepsilon\sqrt{N_1}}{\sqrt{8\sigma^2 + \varepsilon^2(\alpha n + n - 1)}}.$$

TABLE 2
Number of Samples Needed for *d*-Bit DPA

d	N_d
1	N_1
2	$0.5N_1$
3	$0.44N_1$
4	$0.5N_1$
5	$0.64N_1$
6	$0.88N_1$
7	$1.3N_1$
8	$2N_1$

Isolating N_d in this inequality yields

$$N_d \geq \frac{N_1\lfloor 8\sigma^2 + \varepsilon^2(an+n-d)\rfloor}{d^2\lfloor 8\sigma^2 + \varepsilon^2(an+n-1)\rfloor}. \tag{17}$$

Clearly, if $N_d \geq N_1/d^2$, then (17) is satisfied. Thus, a *d*-bit DPA attack needs $1/d^2$ fewer usable power signals to maintain the same SNR. The result of Corollary 6, however, states that, on average, only $2/2^d$ of the power signals will be usable in a *d*-bit DPA attack. Hence, $2^d/2$ times as many signals will be needed in the *d*-bit DPA attack, resulting in an overall average of $2^d/(2d^2)$ times as many signals being needed to maintain the same SNR. □

Table 2 shows that, for small values of *d*, fewer power signals are actually necessary and, with 8-bit DPA, twice as many signals are needed.

4.5 Countermeasures Against Power Analysis Attacks

Implementations of cryptographic algorithms on smart cards are widely realized using software. Software enables flexible implementations that can be easily developed, maintained, and modified. Thus, initial methods to fight power analysis attacks are being implemented in software. Two types of software countermeasures include the addition of random delays and data masking. Power analysis attacks often are possible because the operations being attacked occur at constant positions in time. If the operations are randomly shifted in time, then statistical analysis of the power consumption signal can be made more difficult. Many researchers (e.g., [28] and [29]) have noted the benefits of adding random delays as a countermeasure, but they have also cautioned that attackers might be able to remove such randomization.

One of the more attractive software countermeasures is data masking. The basic premise behind this strategy is that a random mask is generated and used to hide sensitive information. As an example, suppose *d* represents the data that needs to be kept secret. In a data masking scheme, every time the cryptographic algorithm is executed, a new random *r* is generated. The mask *r* is then combined with *d* using a reversible operation. The XOR or the modular addition operations are two possible approaches. Thus, the masked version of *d* might equal $(d+r) \bmod n$ or $d \oplus r$, depending on which method of masking is preferable. Goubin and Patarin applied a masking technique to protect the DES algorithm [30] and Chari et al. looked at using multiple masks $(r_1, r_2, \ldots, r_k)$ to protect sensitive data [31]. Chari et al. also proved that the task of analyzing power signals grows exponentially with regard to *k*. Finally, results showing the impact of masking the Advanced Encryption Standard (AES) finalist algorithms were reported by Messerges [32].

The underlying threats posed by power analysis attacks all stem from the fact that physical implementations of cryptographic algorithms leak information. Thus, a fundamental countermeasure is to implement cryptographic algorithms using hardware that does not leak information or to use hardware that dissipates power in a random manner. Unfortunately, some amount of energy must be consumed when processing information [33]. Hardware solutions can be difficult to design, analyze, and test. Also, hardware methods can lead to more elaborate circuits, which raises the cost of the devices. As of August 2001, there have been only a few published reports regarding hardware methods to prevent power analysis. These reports suggest noise generation hardware [27], power signal filtering [34], or the use of novel circuit design methodologies [35]. It is still not clear which, if any, of these solutions will evolve to completely solve the problem. Designers may need to consider multiple solutions to provide the best possible defense.

5 Conclusions

Attacks that monitor side-channel information and, in particular, power analysis attacks have recently been receiving much attention from experts in the smart-card industry. The results presented in this paper confirm that power analysis attacks can be quite powerful and need to be addressed. Ways an attacker might maximize the side-channel signals have been investigated and were found to be very effective. Solutions to prevent DPA attacks need to consider these advanced attacks in order to provide an adequate amount of security. Understanding the noise characteristics of the power signals is also very important. The experimental and theoretical results presented in this paper hopefully will help with modeling the problem and designing solutions. Unless computer hardware is designed to not leak sensitive information, cryptographers may always need to be concerned about the power analysis attack.

Acknowledgments

The authors would like to thank the reviewers for their helpful comments and advice. Thomas S. Messerges also thanks his wife Beata for her encouragement and support during the completion of this work. R.E. Sloan's work was partially supported by US National Science Foundation Grant CCR-9800070.

REFERENCES

[1] P. Kocher, J. Jaffe, and B. Jun, "Introduction to Differential Power Analysis and Related Attacks," 1998. Available at http://www.cryptography.com/dpa/technical.

[2] P. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," *Proc. Advances in Cryptology (CRYPTO '99),* pp. 388-397, 1999.

[3] *ANSI X9.32—American National Standard for Data Encryption Algorithm (DEA).* Am. Standards Inst., 1981.

[4] T.S. Messerges, E.A. Dabbish, and R.H. Sloan, "Investigations of Power Analysis Attacks on Smartcards," *Proc. USENIX Workshop Smartcard Technology,* pp. 151-161, May 1999.

[5] T.S. Messerges, "Power Analysis Attacks and Countermeasures for Cryptographic Algorithms," PhD dissertation, Univ. of Illinois at Chicago, 2000.

[6] E. Bihamand and A. Shamir, "Differential Cryptanalysis of DES-Like Cryptosystems," *J. Cryptology,* vol. 4, no. 1, pp. 3-72, 1991.

[7] M. Matsui, "Linear Cryptanalysis Method for DES Cipher," *Proc. Advances in Cryptology (Eurocrypt '93),* pp. 386-397, 1994.

[8] E. Biham and A. Shamir, "Differential Cryptanalysis of the Full 16-Round DES," *Proc. Advances in Cryptology (CRYPTO '92),* pp. 487-496, 1993.

[9] R. Anderson, "Why Cryptosystems Fail," *Proc. First ACM Conf. Computer and Comm. Security,* pp. 215-227, Nov. 1993.

[10] J. Kelsey, B. Schneier, D. Wagner, and C. Hall, "Side Channel Cryptanalysis of Product Ciphers," *Proc. European Symp. Research in Computer Security (ESORICS '98),* pp. 97-110, Sept. 1998.

[11] P. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," *Proc. Advances in Cryptology (CRYPTO '96),* pp. 104-113, 1996.

[12] J.F. Dhem, F. Koeune, P.A. Leroux, P. Mestré, J.J. Quisquater, and J.L. Willems, "A Practical Implementation of the Timing Attack," *Proc. Smart Card Research and Advanced Applicatioin Conf. (CARDIS) 1998,* Sept. 1998.

[13] W. van Eck, "Electromagnetic Radiation from Video Display Units: An Eavesdropping Risk," *Computers and Security,* vol. 4, pp. 269-286, 1985.

[14] D. Boneh, R.A. Demillo, and R.J. Lipton, "On the Importance of Checking Cryptographic Protocols for Faults," *Proc. Advances in Cryptology (Eurocrypt '97),* pp. 37-51, 1997.

[15] E. Biham and A. Shamir, "Differential Fault Analysis of Secret Key Cryptosystems," *Proc. Advances in Cryptology (CRYPTO '97),* pp. 513-525, 1997.

[16] D. Naccache and D. M'Raihi, "Cryptographic Smart Cards," *IEEE Micro,* vol. 16, no. 3, pp. 14-24, June 1996.

[17] *ISO7816-2—Identification Cards—Integrated Circuit(s) Cards with Contacts—Part 2: Dimensions and Location of the Contacts.* Int'l Organization for Standardization, 1999.

[18] *ISO7816-3—Identification Cards—Iintegrated Circuit(s) Cards with Contacts—Part 3: Electronic Signals and Transmission Protocols.* Int'l Organization for Standardization, 1997.

[19] *ISO7816-4—Identification Cards—Integrated Circuit(s) Cards with Contacts—Part 4: Inter-Industry Commands for Interchange.* Int'l Organization for Standardization, 1995.

[20] *ISO/IEC 14443—Identification Cards—Contactless Integrated Circuit(s) Cards—Proximity Cards—Part 4: Transmission Protocol.* Int'l Organization for Standardization, 2001.

[21] *ISO7816-7—Identification Cards—Integrated Circuit(s) Cards with Contacts—Part 7: Inter-Industry Commands for Structured Card Query Language (SCQL).* Int'l Organization for Standardization, 1999.

[22] S.B. Guthery, "Java Card: Internet Computing on a Smart Card," *IEEE Internet Computing,* vol. 1, no. 1, pp. 57-59, 1997.

[23] "CASCADE: Chip Architecture for Smartcard and Intelligent Devices,"European ESPRIT Project (EP8670), 1998. Available at http://www.dice.ucl.ac.be/crypto/cascade/.

[24] R. Anderson and M. Kuhn, "Tamper Resistance—A Cautionary Note," *Proc. Second USENIX Workshop Electronic Commerce Proc.,* pp. 1-11, 1996.

[25] *Cardtech/Securtech '98—Conf. Proc., Volume II: Applications,* 1998.

[26] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design.* Addison-Wesley, 1993.

[27] P. Wayner, "Code Breaker Cracks Smart Cards' Digital Safe," *New York Times,* pp. C1, 22 June 1998.

[28] J. Daemen and V. Rijmen, "Resistance against Implementation Attacks: A Comparative Study of the AES Proposals," *Proc. Second Advanced Encryption Standard Candidate Conf.,* Mar. 1999. Available at http://www.nist.gov/aes.

[29] S. Chari, C. Jutla, J.R. Rao, P. Rohatgi, "A Cautionary Note Regarding Evaluation of AES Candidates on Smart-Cards," *Proc. Second Advanced Encryption Standard Candidate Conf.,* Mar. 1999. Available at http://www.nist.gov/aes.

[30] L. Goubin and J. Patarin, "DES and Differential Power Analysis—The Duplication Method," *Proc. Workshop Cryptographic Hardware and Embedded Systems,* pp. 158-172, Aug. 1999.

[31] S. Chari, C.S. Jutla, J.R. Rao, and P.J. Rohatgi, "Towards Sound Approaches to Counteract Power-Analysis Attacks," *Proc. Advances in Cryptology (CRYPTO '99),* pp. 398-412, 1999.

[32] T.S. Messerges, "Securing the AES Finalists against Power Analysis Attacks," *Proc. Fast Software Encryption Workshop 2000,* Apr. 2000.

[33] C.H. Bennett and R. Landauer, "The Fundamental Physical Limits of Computation," *Scientific Am.,* vol. 253, no. 1, pp. 48-56, 1985.

[34] P. Rakers, L. Connell, T. Collins, and D. Russell, "Secure Contactless Smartcard ASIC with DPA Protection," *Proc. IEEE Custom Integrated Circuits Conf.,* May 2000.

[35] J. Kessels, "Applying Asynchronous Circuits in Contactless Smartcards," *Proc. ACiD-WG Workshop,* Feb. 2000.

Thomas S. Messerges received the BS and the MS degrees in electrical and computer engineering from the University of Illinois at Urbana-Champaign in 1987 and 1989 and the PhD degree in electrical engineering and computer science from the University of Illinois at Chicago in 2000. He has worked at Motorola for 12 years and is currently a principal staff engineer with the Security Technology Research Laboratory, which is a part of Motorola Labs, Schaumburg, Illinois. His research interests include digital rights management, cryptography, security, and embedded hardware and software design. He is a member of the IEEE and the IEEE Computer Society.

Ezzat (Ezzy) Dabbish received the BS degree in electrical engineering from Alexandria University, Egypt, and the MS degree in electrical engineering from Michigan Technological University, Houghton, in 1964 and 1973, respectively. He has been with Motorola corporate research since 1973 and is currently a member of the technical staff in the Security Technology Research Laboratory, which is part of Motorola Labs, Schaumburg, Illinois. He holds 13 issued US patents to date. His research interests include cryptography, security, cryptographic hardware embedded systems, and digital rights management. He is a member of the IEEE.

Robert H. Sloan received the BS degree in mathematics from Yale University in 1983, the MS degree in electrical engineering and computer science in 1986, and the PhD degree in computer science from Massachusetts Institute of Technology in 1989. He is an associate professor in the Computer Science Department of the University of Illinois at Chicago, but currently away serving as the program director of the Theory of Computing Program at the National Science Foundation. His research interests are in design and analysis of algorithms, computational learning theory, machine learning, and cryptography. He is a senior member of the IEEE.