Power Analysis Attacks of Modular Exponentiation in Smartcards

Thomas S. Messerges[1], Ezzy A. Dabbish[1], Robert H. Sloan[2,3]

[1]Motorola Labs, Motorola
1301 E. Algonquin Road, Room 2712, Schaumburg, IL 60193
{tomas, dabbish}@ccrl.mot.com

[2]Dept. of EE and Computer Science, University of Illinois at Chicago
851 S. Morgan Street, Room 1120, Chicago, IL 60607
sloan@eecs.uic.edu

Abstract. Three new types of power analysis attacks against smartcard implementations of modular exponentiation algorithms are described. The first attack requires an adversary to exponentiate many random messages with a known and a secret exponent. The second attack assumes that the adversary can make the smartcard exponentiate using exponents of his own choosing. The last attack assumes the adversary knows the modulus and the exponentiation algorithm being used in the hardware. Experiments show that these attacks are successful. Potential countermeasures are suggested.

1 Introduction

Cryptographers have been very successful at designing algorithms that defy traditional mathematical attacks, but sometimes, when these algorithms are actually implemented, problems can occur. The implementation of a cryptographic algorithm can have weaknesses that were unanticipated by the designers of the algorithm. Adversaries can exploit these weaknesses to circumvent the security of the underlying cryptographic algorithm. Attacks on the implementations of cryptographic systems are a great concern to operators and users of secure systems. Implementation attacks include power analysis attacks [1,2], timing attacks [3,4], fault insertion attacks [5,6], and electromagnetic emission attacks [7]. Kelsey et al. [8] review some of these attacks and refer to them as "side-channel" attacks. The term "side-channel" is used to describe the leakage of unintended information from a supposedly tamper-resistant device, such as a smartcard.

In a power analysis attack the side-channel is the device's power consumption. An adversary can monitor the power consumption of a vulnerable device, such as a smartcard, to defeat the tamper-resistance properties and learn the secrets contained inside the device [1]. Although it is preferable to design secure systems that do not rely on secrets contained in the smartcard, there are applications where this may not be possible or is undesirable. In these systems, if the secret, for instance a private key, is compromised, then the entire system's security may be broken.

[3] Partially supported by NSF Grant CCR-9800070

Ç.K. Koç and C. Paar (Eds.): CHES'99, LNCS 1717, pp. 144−157, 1999.

In this paper we examine the vulnerabilities of public-key cryptographic algorithms to power analysis attacks. Specifically, attacks on the modular exponentiation process are described. These attacks are aimed at extracting the secret exponent from tamper-resistant hardware by observing the instantaneous power consumption signals into the device while the exponent is being used for the exponentiation. Experimental results on a smartcard containing a modular exponentiation circuit are provided to confirm the threats posed by these attacks.

Three types of attacks are described that can be mounted by adversaries possessing various degrees of capabilities and sophistication. The first attack requires that, in addition to exponentiating with the secret exponent, the smartcard will also exponentiate with at least one exponent known to the attacker. This attack, referred to as a “Single-Exponent, Multiple-Data” (SEMD) attack, requires the attacker to exponentiate many random messages with both the known and the secret exponent. The SEMD attack is demonstrated to be successful on exponentiations using a small modulus (i.e., 64 bits) with 20,000 trial exponentiations, but with a large modulus might require 20,000 exponentiations per exponent bit. The second attack we introduce requires that the attacker can get the smartcard to exponentiate using exponents of his own choosing. Our experiments showed that this attack, referred to as a “Multiple-Exponent, Single-Data” (MESD) attack, requires the attacker to run about 200 trial exponentiations for each exponent bit of the secret exponent. The last attack that we discovered does not require the adversary to know any exponents, but does assume the attacker can obtain basic knowledge of the exponentiation algorithm being used by the smartcard. With this attack, referred to as a “Zero-Exponent, Multiple-Data” (ZEMD) attack, we can successfully extract a secret exponent with about 200 trial exponentiations for each secret exponent bit.

The organization of this paper is as follows; first, the related work is reviewed and the motivation for research into power analysis attacks is given. Next, implementations of modular exponentiation and the basic principles of power analysis attacks are reviewed. The equipment and software needed for these attacks is described and detailed descriptions of the MESD, SEMD and ZEMD attacks are given. Finally, potential countermeasures are suggested.

1.1 Related Work

Previous papers that describe power analysis attacks mainly examine the security of symmetric key cryptographic algorithms. Kocher, et al. [1] review a Simple Power Analysis (SPA) attack and introduce a Differential Power Analysis (DPA) attack, which uses powerful statistical-based techniques. They describe specific attacks against the Digital Encryption Standard (DES)[9], and their techniques can also be modified for other ciphers. Kelsey, et al. [8] show how even a small amount of side-channel information can be used to break a cryptosystem such as the DES. An alternate approach is taken in [2], where techniques to strengthen the power consumption attack by maximizing the side-channel information are described. The Advanced Encryption Standard (AES) candidate algorithms are analyzed in [10-12] for their vulnerabilities to power analysis attacks. These papers advise that the vulnerabilities of the AES algorithms to power analysis attacks should be considered when choosing the next encryption standard.

1.2 Research Motivation

Tamper-resistant devices, such as smartcards, can be used to store secret data such as a person's private key in a two-key, public-key cryptosystem. Familiar examples of such systems are an RSA cryptosystem [13] and an elliptic-curve cryptosystem [14,15]. In a typical scenario, the owner of a smartcard needs to present the card in order to make a payment, log onto a computer account, or gain access to a secured facility. In order to complete a transaction, the smartcard is tested for authenticity by a hardware device called a reader. The reader is provided by a merchant or some other third party and, in general, may or may not be trusted by the smartcard owner. Thus, it is important that when a user relinquishes control of her smartcard, she is confident that the secrecy of the private key in the card can be maintained. In an RSA cryptosystem, the authenticity of the card is tested by asking the card to use its internally stored private key to modularly exponentiate a random challenge. Since it is possible that the card is being accessed by a malicious reader, the power consumption of the card during the exponentiation process should not reveal the secret key.

2 Review of Modular Exponentiation Implementations

Modular exponentiation is at the root of many two-key, public-key cryptographic implementations. The technique used to implement modular exponentiation is commonly known as the "square-and-multiply" algorithm. Elliptic curve cryptosystems use an analogous routine called the "double-and-add" algorithm. Two versions of the square-and-multiply algorithm are given in Fig. 1. The first routine in Fig. 1, *exp1*, starts at the exponent's most significant nonzero bit and works downward. The second routine, *exp2*, starts at the least significant bit of the exponent e and works upward. Both routines are vulnerable to attack and both return the same result, $M^e \bmod N$. Common techniques to implement modular exponentiation (i.e., particular implementations of the modular square and modular multiply operations) can be found in [16-21]. One pop-

```
exp1(M, e, N)
{  R = M
   for (i = n-2 down to 0)
   {  R = $R^2$ mod N
      if (ith bit of e is a 1)
          R = $R \cdot M$ mod N }
   return R }
```

```
exp2(M, e, N)
{  R = 1
   S = M
   for (i = 0 to n-1)
   {  if (ith bit of e is a 1)
          R = $R \cdot S$ mod N
      S = $S^2$ mod N }
   return R }
```

Fig. 1. Exponentiation Routines Using the Square-and-Multiply Algorithm
Two versions of the square-and-multiply algorithm used for smartcard authentication are given above. The routine *exp1* starts at the most significant bit and works down and the routine *exp2* does the opposite. The routine *exp2* requires extra memory to store the S variable. The exponent, e, has n bits, where the least significant bit is numbered 0 and the most significant nonzero bit is numbered n-1

ular method to speed up the exponentiation is to use Montgomery's modular multiplication algorithm [22] for all the multiplies and squares.

The attacks in this paper are a potential threat to all of these implementations. The MESD and SEMD attacks are against the square-and-multiply method. Every implementation executes the square-and-multiply method in some manner, so all are potentially vulnerable. The ZEMD attack works on intermediate data results and is only possible if an attacker possesses basic knowledge of the implementation. The attacker needs to know which of the types of square-and-multiply algorithms is being used and the technique used for the modular multiplications. Even if the attacker does not know the implementation, there is a fairly small number of likely possibilities. In our attack, all that was necessary to know was that the exponentiation was done using the *exp1* algorithm and Montgomery's method was used for the modular multiplies.

3 Review of Power Analysis Attacks

Power analysis attacks work by exploiting the differences in power consumption between when a tamper-resistant device processes a logical zero and when it processes a logical one. For example, when the secret data on a smartcard is accessed, the power consumption may be different depending on the Hamming weight of the data. If an attacker knows the Hamming weight of the secret key, the brute force search space is reduced and given enough Hamming weights of independent functions of the secret key, the attacker could potentially learn the entire secret key. This type of attack, where the adversary directly uses a power consumption signal to obtain information about the secret key is referred to as a Simple Power Analysis (SPA) attack and is described in [1].

Differential Power Analysis (DPA) is based on the same underlying principle of an SPA attack, but uses statistical analysis techniques to extract very tiny differences in power consumption signals. DPA was first introduced in [1] and a strengthened version was reported in [2].

3.1 Simple Power Analysis (SPA)

SPA[1] on a single-key cryptographic algorithm, such as DES, could be used to learn the Hamming weight of the key bytes. DES uses only a 56-bit key so learning the Hamming weight information alone makes DES vulnerable to a brute-force attack. In fact, depending on the implementation, there are even stronger SPA attacks. A two-key, public-key cryptosystem, such as an RSA or elliptic curve cryptosystem, might also be vulnerable to an SPA attack on the Hamming weight of the individual key bytes, however it is possible an even stronger attack can be made directly against the square-and-multiply algorithm.

If exponentiation were performed in software using one of the square-and-multiply algorithms of Fig. 1, there could be a number of potential vulnerabilities. The main problem with both algorithms is that the outcome of the "if statement" might be observable in the power signal. This would directly enable the attacker to learn every bit of the secret exponent. A simple fix is to always perform a multiply and to only save the result if the exponent bit is a one. This solution is very costly for performance and still may be vulnerable if the act of saving the result can be observed in the power signal.

3.2 Differential Power Analysis (DPA)

The problem with an SPA attack is that the information about the secret key is difficult to directly observe. In our experiments, the information about the key was often obscured with noise and modulated by the device's clock signal. DPA can be used to reduce the noise and also to "demodulate" the data. Multiple-bit DPA [2] can be used to attack the DES algorithm by defining a function, say D, based on the guessed key bits entering an S-box lookup table. If the D-function predicts high power consumption for a particular S-box lookup, the power signal is placed into set S_{high}. If low power consumption is predicted, then the signal is placed into an alternate set, S_{low}. If the predicted power consumption is neither high or low, then the power signal is discarded. The result of this partitioning is that when the average signal in set S_{high} is subtracted from the average signal in set S_{low}, the resulting signal is demodulated. Any power biases at the time corresponding to the S-box lookup operation are visible as an obvious spike in the difference signal and much of the noise is eliminated because averaging reduces the noise variance. Correct guesses of the secret key bits into an S-box are verified by trying all 2^6 possibilities and checking which one produces the strongest difference signal.

All of the attacks described in this paper use averaging and subtracting and so are similar to a DPA attack. The averaging reduces the noise and the subtracting demodulates the secret information and enhances the power biases.

4 Power Analysis Equipment

A smartcard with a built-in modular exponentiation circuit was used to evaluate the attacks described in this paper. The exponentiation circuit on this smartcard is a typical implementation of the square-and-multiply algorithm using a Montgomery multiplication circuit to speed up the modular reductions. The exponentiation circuit was accessed via a software program residing in the card's memory. This software executed a simple ISO7816 smartcard protocol [23] which supports a command similar to the standard "internal authenticate" command.

5 Attacking a Secret Exponent

The objective of the attacks described in this paper is to find the value of e, the secret exponent stored in the smartcard's internal memory. The attacker is assumed to have complete control of the smartcard. He can ask the card to exponentiate using e and can monitor all input and output signals. The card will obey all commands of the attacker, except a command to output the secret key. The main command that is needed is the "internal authenticate" command which causes the card to receive an input value, M, and output $M^e \bmod N$. Some smartcard systems require the user to enter a Personal Identification Number (PIN) prior to allowing access to the card. This feature is not considered in our attacks. Also, the number of times the attacker can query the card is assumed unlimited. All of these assumptions are reasonable since smartcard systems have been implemented that allow such access. Other assumptions used for particular attacks are stated in the sections that describe the specific attack details.

5.1 A Simple Correlation Experiment

We performed a correlation experiment to determine if *e* could be revealed by simply cross-correlating the power signal from a single multiply operation with the entire exponentiation's power signal. This attack was designed to see how easy it is to distinguish the multiplies from the squares, thus revealing the bits of *e*. Let the multiply's power signal be $S_m[j]$ and the exponentiation's power signal be $S_e[j]$. The cross-correlation signal, $S_c[j]$ is calculated as

$$S_c[j] = \sum_{\tau = 0}^{W} S_m[\tau] S_e[j + \tau]$$

where W is the number of samples in the multiply's power signal. That is, $W = T_m/T$, where T_m is the time needed for a multiply operation and T is the sampling rate. An attacker can learn the approximate value of W through experimentation or from the smartcard's documentation.

The power signals and cross-correlation signal obtained from running this experiment are shown in Fig. 2. The exponentiation and multiply power signals were obtained by running the smartcard with constant input data and averaging 5,000 power signals to reduce the measurement noise. This experiment was first tested on a known exponent, so the locations of the squares and multiplies are known and are labeled in the Fig. 2. The resulting cross-correlation signal shows peaks at the locations of the individual squares and multiplies, but the height of the peaks are uncorrelated with the type of operation. Thus, this cross-correlation technique is not useful to differentiate between squares and multiplies. However, it is interesting to point out that the time needed for each operation in the square-and-multiply algorithm can be determined from the cross

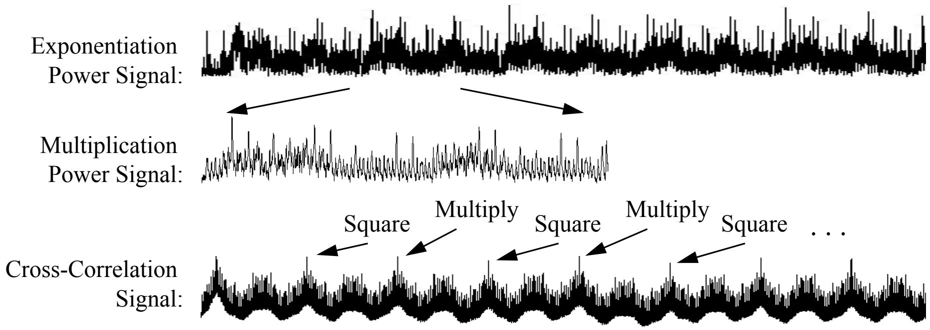


Fig. 2. Cross-Correlation of Multiplication and Exponentiation Power Signals
The above signals were obtained using the power analysis equipment described in Section 4. The signals were averaged for 5,000 exponentiations using a constant input value. The results show an ability to determine the time between the square-and-multiply operations, but cannot be used to distinguish multiply operations from squaring operations

correlation signal. This information could lead to a combined power analysis and timing attack in implementations where the time to multiply is slightly different than the time to square. Such an attack would be more powerful than previously documented timing attacks because the cross-correlation signal would yield the timing of all intermediate operations. Fortunately, the smartcards we examined do not have this problem.

5.2 Single-Exponent, Multiple-Data (SEMD) Attack

The SEMD attack assumes that the smartcard is willing to exponentiate an arbitrary number of random values with two exponents; the secret exponent and a public exponent. Such a situation could occur in a smartcard system that supports the ISO7816 [23] standard “external authenticate” command. Whereas the “internal authenticate” command causes the smartcard to use its secret key, the “external authenticate” command can be used to make the smartcard use the public key associated with a particular smartcard reader. It is assumed that the exponent bits of this public key would be known to the attacker.

The basic premise of this attack is that by comparing the power signal of an exponentiation using a known exponent to a power signal using an unknown exponent, the adversary can learn where the two exponents differ, thus learn the secret exponent. In reality, the comparison is nontrivial because the intermediate data results of the square-and-multiply algorithm cause widely varying changes in the power signals, thereby making direct comparisons unreliable. The solution to this problem is to use averaging and subtraction. This simple DPA technique begins by using the secret exponent to exponentiate L random values and collects their associated power signals, $S_i[j]$. Likewise, L power signals, $P_i[j]$, are collected using the known exponent. The average signals are then calculated and subtracted to form $D[j]$, the DPA bias signal,

$$D[j] = \frac{1}{L}\sum_{i=1}^{L} S_i[j] - \frac{1}{L}\sum_{i=1}^{L} P_i[j] = \bar{S}[j] - \bar{P}[j]$$

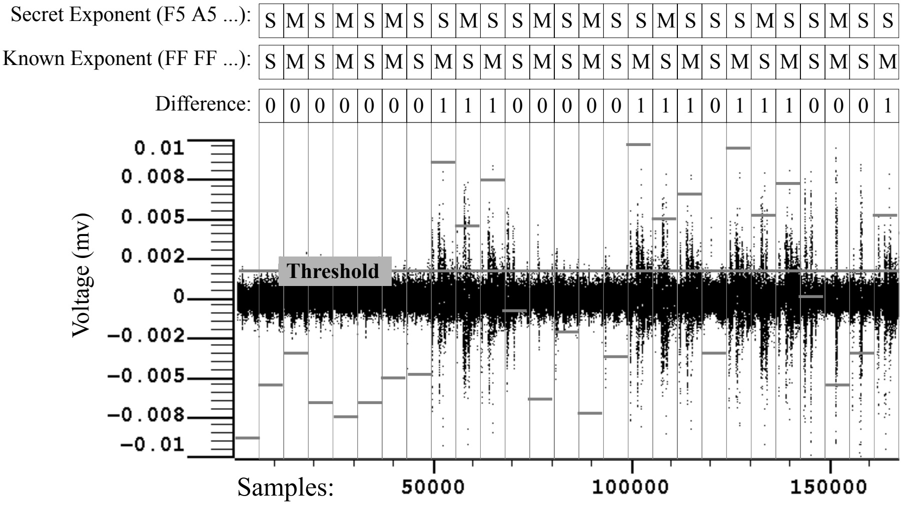
The portions of the signals $\bar{S}[j]$ and $\bar{P}[j]$ that are dependent on the intermediate data will average out to the same constant mean μ, thus:

$$\bar{S}[j] \approx \bar{P}[j] \approx \mu \qquad \text{if} \qquad j = \text{a data dependent sample point}$$

The portion of the signals $\bar{S}[j]$ and $\bar{P}[j]$ that are dependent on the exponent bits will average out to different values, μ_{s} or μ_{m}, depending on whether a square or multiply operation is performed. Thus, if μ_s and μ_m are not equal, then their difference will be nonzero and the DPA bias signal, $D[j]$, can be used to determine the exact location of the squares and multiplies in the secret exponent:

$$D[j] \approx \begin{cases} 0 & \text{if } j = \text{data dependent point or exponentiation operations agree} \\ \text{nonzero} & \text{if } j = \text{point where the exponentiation operations differ} \end{cases}$$

The SEMD attack was performed on a smartcard and the result is shown in Fig. 3. For this experiment the exponentiation was simplified by using a modulus and data with only 64 bits. This simplification was done only for illustrative purposes. Using smaller



Secret Exponent (F5 A5 ...):	S	M	S	M	S	M	S	S	M	S	S	M	S	M	S	S	M	S	S	S	M	S	S	M	S	S
Known Exponent (FF FF ...):	S	M	S	M	S	M	S	M	S	M	S	M	S	M	S	M	S	M	S	M	S	M	S	M	S	M
Difference:	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	1	1	1	0	1	1	1	0	0	0	1

Fig. 3. Single-Exponent, Multiple-Data (SEMD) Attack Results
The above plot is the DPA signal comparing the exponentiation power signal produced with a known exponent and an unknown exponent. The energy in the DPA signal is greater when the two exponent operations are different. The shaded horizontal bars show the output of an integrate-and-dump filter indicating the energy associated with each interval of time. The above signal was obtained using 20,000 trial exponentiations

data made it possible to store more of the exponentiation's power signal in the digital oscilloscope, so many more exponent bits could be attacked with each test. In an actual attack against full-sized data, one would likely be able to attack only a small portion of the exponentiation at a time. The number of bits attacked at one time depends on the size of the memory in the attacker's digital oscilloscope. The DPA signal in Fig. 3 shows an attack on about 16 exponent bits and was obtained with L=10,000; thus 20,000 trial exponentiations were needed. In a real attack, using full-sized data, this attack might need 20,000 exponentiations for each exponent bit. In this case a sliding window approach is needed, where only a windowed portion of the power trace is attacked at one time.

The DPA bias signal in Fig. 3 is labeled to show the squares (S) and multiplies (M) associated with the secret and known exponents. The regions where these operations differ exhibit a corresponding increase in the amplitude of the DPA bias signal. An integrate-and-dump filter was used to compute the signal energy associated with each region and the output of the filter is graphed as shaded horizontal line segments in Fig. 3. The output of the integrate-and-dump filter is given as:

$$I_{out}[j] = \sum_{i = jT_m}^{(j+1)T_m} F_{clip}(D[i]) \qquad \text{where, } F_{clip}(x) = \max(x^2, V_{clip})$$

In this equation, V_{clip} is chosen to eliminate the overwhelming influence of spurious spikes in the bias signal. The final result shows that the output of the integrate-and-dump filter is good at indicating where the secret and known exponent operations differ. Thus, the SEMD attack is an important attack that implementors of smartcard systems need to consider when designing a secure system.

5.3 Multiple-Exponent, Single-Data (MESD) Attack

The MESD attack is more powerful than the SEMD attack, but requires a few more assumptions about the smartcard. The previously described SEMD attack is a very simple attack requiring little sophistication on the part of the adversary, but the resulting DPA bias signal is sometimes difficult to interpret. The Signal-to-Noise Ratio (SNR) can be improved using the MESD attack. The assumption for the MESD attack is that the smartcard will exponentiate a constant value[1] using exponents chosen by the attacker. Again, such an assumption is not unreasonable since some situations might allow the smartcard to accept new exponents that can be supplied by an untrusted entity. Also, the smartcard does not have unlimited memory, so it is impossible for it to keep a history of previous values it has exponentiated. Thus, the card cannot know if it is being repeatedly asked to exponentiate a constant value.

The algorithm for the MESD attack is given in Fig. 4. The first steps of the algorithm are to choose an arbitrary value, *M*, exponentiate *M* using the secret exponent *e*, and then collect the corresponding average power signal $S_M[j]$. Next, the algorithm progresses by successively attacking each secret exponent bit starting with the first bit used in the square-and-multiply algorithm and moving towards the last. To attack the *i*th secret exponent bit, the adversary guesses the *i*th bit is a 0 and then a 1 and asks the card to exponentiate using both guesses. It is assumed that the adversary already knows the first through (*i*-1)st exponent bits so the intermediate results of the exponentiation up to the (*i*-1)st exponent bit will be the same for the guessed exponent and the secret exponent. If the adversary guesses the *i*th bit correctly, then the intermediate results will also agree at the *i*th position. If the guess is wrong, then the results will differ. This difference can be seen in the corresponding power traces. Let e_g be the current guess for the exponent. The average power signals for exponentiating *M* using an e_g with the *i*th

```
M = arbitrary value and $e_g$ = 0
Collect $S_M$[j]
for (i = n-1 to 0)
{  guess (ith bit of $e_g$ is a 1) and collect $S_1$[j]
   guess (ith bit of $e_g$ is a 0) and collect $S_0$[j]
   Calculate two DPA bias signal:
      $D_1$[j] = $S_M$[j] - $S_1$[j] and $D_0$[j] = $S_M$[j] - $S_0$[j]
   Decide which guess was correct using DPA result
   update $e_g$ }
$e_g$ is now equal to e (the secret exponent)
```

Fig. 4. Algorithm for the Multiple-Exponent, Single Data (MESD) Attack
This algorithm gradually makes e_g equal to the secret exponent bit, by using the DPA signal to decide which guess is correct at the *i*th iteration

[1] This value may or may not be known to the attacker.

bit equal to 1 is $S_1[j]$ and an e_g with the ith bit equal to 0 is $S_0[j]$. Two DPA bias signals can be calculated:

$$D_1[j] = S_M[j] - S_1[j] \qquad \text{and} \qquad D_0[j] = S_M[j] - S_0[j]$$

Whichever exponent bit was correct produces a power signal that agrees with the secret exponent's power signal for the larger amount of time. Thus, whichever bias signal is zero for a longer time corresponds to the correct guess.

The resulting bias signal for a correct and incorrect guess are shown in Fig. 5. It is clear that the SNR in Fig. 5 is much improved over the SEMD attack. The higher SNR of the MESD attack means that fewer trial exponentiations are needed for a successful attack. Also, an experienced attacker really needs to calculate only one DPA bias signal. For example, the attacker could always guess the exponent bit is a 1. If the guess is correct, the DPA bias signal will remain zero for the duration of a multiply and a square operation. If the guess is wrong, then the bias signal will only remain zero for the duration of the square operation. This technique effectively cuts the running time of the algorithm of Fig. 4 in half. Our experiments showed that as few as 100 exponentiations were needed per exponent bit. Memory limitations in a digital oscilloscope also might require a moving window approach to collect the secret exponent's power signal, thus resulting in 200 exponentiations per exponent bit. The circumstances allowing an MESD attack definitely need to be addressed by implementors concerned with power analysis attacks.

5.4 Zero-Exponent, Multiple-Data (ZEMD) Attack

The ZEMD attack is similar to the MESD attack, but has a different set of assumptions. One assumption for the ZEMD attack is that the smartcard will exponentiate many random messages using the secret exponent. This attack does not require the adversary know any exponents, hence the zero-exponent nomenclature. Instead, the adversary needs to be able to predict the intermediate results of the square-and-multiply algorithm using an off-line simulation. This usually requires that the adversary know the algorithm being used by the exponentiation hardware and the modulus used for the exponentiation. There are only a few common approaches to implementing modular exponentiation algorithms, so it is likely an adversary can determine this information. It is also likely that the adversary can learn the modulus because this information is usually public.

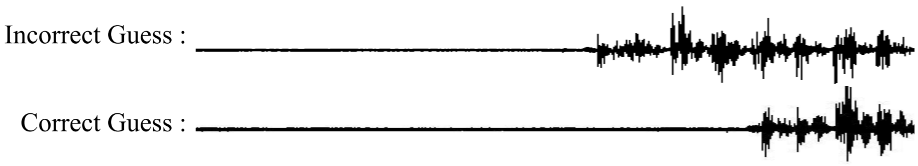


Fig. 5. Multiple-Exponent, Single-Data (MESD) Attack Results
The above plot is the DPA signal obtained when the next bit is guessed correctly compared to when the next bit guess is wrong. The correct guess is clearly seen to be the signal that remains zero the longest. This signal was obtained using 1,000 trial exponentiations

The algorithm for the ZEMD attack is given in Fig. 6. The ZEMD attack starts by attacking the first bit used during the exponentiation and proceeds by attacking each successive bit. In this algorithm, the variable e_g gradually becomes equal to the secret exponent. After each iteration of the attack, another exponent bit is learned and e_g is subsequently updated. At the *i*th iteration of the algorithm, it is assumed that the correct exponent, e_g, is correct up to the (*i*-1)st bit. The algorithm then guesses that the *i*th bit of the secret exponent is a 1 and a DPA bias signal is created to verify the guess. The DPA bias signal is created by choosing a random input, *M*, and running a simulation to determine the power consumption after the multiply in the *i*th step of the square-and-multiply algorithm. This simulation is possible because the exponent e_g is known up to the *i*th bit and the power consumption can be estimated using the Hamming weight of a particular byte in the multiplication result. Previous power analysis experiments showed that a higher number of ones correspond to higher power consumption.

If the multiply at the *i*th step actually occurred, then the power analysis signals can be accurately partitioned into two sets, thereby creating biases (or spikes) in the DPA bias signal when the average signals in each partition are subtracted. If the guess is incorrect, then the partitioning will not be accurate and the power biases will not occur. A natural error-correcting feature of this algorithm is that if there is ever a mistake, all subsequent steps will fail to show any power biases.

The ZEMD attack was implemented and an example of the DPA bias signals for a correct and an incorrect guess are given in Fig. 7. The DPA bias signals in Fig. 7 were generated using an 8-bit partitioning function based on the Hamming weight of the multiplication result. Power signals corresponding to results with Hamming weight eight were subtracted from power signals corresponding to results with Hamming weight zero. This partitioning technique creates a larger SNR and is further described in [2].

```
$e_g$ = 0
for ($i$ = $n$-1 to 0)
{  guess ($i$th bit of $e_g$ is a 1)
   for ($k$ = 1 to $L$)
   {  choose a random value: $M$
      simulate to the ith set the calculation of $M^{e_g} \bmod N$
      if (multiplication result has high Hamming weight)
         run smartcard and collect power signal: $S[j]$
         add $S[j]$ to set $S_{high}$
      if (multiplication result has low Hamming weight)
         run smartcard and collect power signal: $S[j]$
         add $S[j]$ to set $S_{low}$ }
   Average the power signals and get DPA bias signal:
      $D[j] = \overline{S}_{low}[j] - \overline{S}_{high}[j]$
   if DPA bias signal has spikes
      the guess was correct: make $i$th bit of $e_g$ equal to 1
   else
      the guess was wrong: make $i$th bit of $e_g$ equal to 0 }
$e_g$ is now equal to $e$ (the secret exponent)
```

Fig. 6. Algorithm for the Zero-Exponent, Multiple Data (ZEMD) Attack
This algorithm gradually makes e_g equal to the secret exponent bit, by using the DPA signal to decide if the guess of the *i*th bit being a one is correct

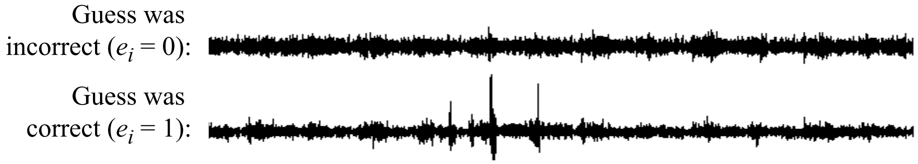


Fig. 7. Zero-Exponent, Multiple-Data (ZEMD) Attack Results
The above plot is the DPA signal comparing the DPA bias signal produced when the guess of the *i*th exponent bit is correct compared to when it is incorrect. The spikes in the correct signal can be used to confirm the correct guess. This signal was obtained using 500 trial exponentiations

The signals in Fig. 7 were obtained by averaging 500 random power signals, but we have also been able to mount this attack with only 100 power signals per exponent bit.

In the attack we implemented it was necessary to collect power signals using a windowing approach. This meant it was necessary to collect new power signals for each exponent bit being attacked. With optimizations to the equipment and algorithm, more exponent bits could be attacked simultaneously requiring even fewer trial exponentiations. The exact number of trial exponentiations necessary is dependent on the equipment of the adversary, the size of the power biases, and the noise in the signals. Implementors need to keep the ZEMD attack in mind when designing modular exponentiation hardware and software.

6 Countermeasures

Potential countermeasures to the attacks described in this paper include many of the same techniques described to prevent timing attacks on exponentiation. Kocher's [3] suggestion for adapting the techniques used for blinding signatures [24] can also be applied to prevent power analysis attacks. Prior to exponentiation, the message could be blinded with a random value, v_i and unblinded after exponentiation with $v_f = (v_i^{-1})^e \mathrm{mod} N$. Kocher suggests an efficient way to calculate and maintain (v_i, v_f) pairs.

Message blinding would prevent the MESD and ZESD attacks, but since the same exponent is being used, the SEMD attack would still be effective. To prevent the SEMD attack, exponent blinding, also described in [3], would be necessary. In an RSA cryptosystem, the exponent can be blinded by adding a random multiple of $\phi(N)$, where $\phi(N) = (p-1)(q-1)$ and N=pq. In summary, the exponentiation process would go as follows:

1. Blind the message M: $\hat{M} = (v_i M) \bmod N$
2. Blind the exponent e: $\hat{e} = e + r\phi(N)$
3. exponentiate: $\hat{S} = (\hat{M}^{\hat{e}}) \bmod N$
4. unblind the result: $S = (v_f \hat{S}) \bmod N$

Another way to protect against power analysis attack is to randomize the exponentiation algorithm. One way this can be accomplished is to combine the two square-and-multiply algorithms of Fig. 1. A randomized exponentiation algorithm could begin by selecting a random starting point in the exponent. Exponentiation would proceed from this random starting point towards the most significant bit using *exp2* of Fig. 1. Then, the algorithm would return to the starting point and finish the exponentiation using *exp1* and moving towards the least significant bit. It would be difficult for an attacker to determine the random starting point from just one power trace (an SPA attack), so this algorithm would effectively randomize the exponentiation. The amount of randomization that is possible depends on the number of bits in the exponent. For large exponents this randomization might be enough to make power analysis attacks impractical to all but the most sophisticated adversaries. All the attacks presented in this paper would be significantly diminished by randomizing the exponentiation.

7 Conclusions

The potential threat of monitoring power consumption signals to learn the private key in a two-key, public-key cryptosystem has been investigated. A variety of vulnerabilities have been documented and three new attacks were developed. The practicality of all three attacks was confirmed by testing on actual smartcard hardware. Table 1 summarizes the attacks and some of the assumptions and possible solutions.

The goal of this research is to point out the potential vulnerabilities and to provide guidance towards the design of more secure tamper-resistant devices. Hopefully the results of this paper will encourage the design and development of solutions to the problems posed by power analysis attacks.

TABLE 1: Summary of Power Analysis Attacks on Exponentiation

Attack Name	Number of trial exponentiations	Assumptions	Possible Solution
SEMD	20,000	attacker knows one exponent	exponent blinding
MESD	200	attacker can choose exponent	message blinding
ZEMD	200	attacker knows algorithm and modulus	message blinding

References

1. P. Kocher, J. Jaffe, and B. Jun, "Introduction to Differential Power Analysis and Related Attacks," http://www.cryptography.com/dpa/technical, 1998.
2. T. S. Messerges, E. A. Dabbish and R. H. Sloan, "Investigations of Power Analysis Attacks on Smartcards," *Proceedings of USENIX Workshop on Smartcard Technology*, May 1999, pp. 151-61.
3. P. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," in *Proceedings of Advances in Cryptology–CRYPTO '96*, Springer-Verlag, 1996, pp. 104-13.

4. J. F. Dhem, F. Koeune, P. A. Leroux, P. Mestré, J. J. Quisquater and J. L. Willems, "A Practical Implementation of the Timing Attack," in *Proceedings of CARDIS 1998*, Sept. 1998.
5. D. Boneh and R. A. Demillo and R. J. Lipton, "On the Importance of Checking Cryptographic Protocols for Faults," in *Proceedings of Advances in Cryptology–Eurocrypt '97*, Springer-Verlag, 1997, pp. 37-51.
6. E. Biham and A. Shamir, "Differential Fault Analysis of Secret Key Cryptosystems," in *Proceedings of Advances in Cryptology–CRYPTO '97*, Springer-Verlag, 1997, pp. 513-25.
7. W. van Eck, "Electromagnetic Radiation from Video Display Units: An Eavesdropping Risk," *Computers and Security*, v. 4, 1985, pp. 269-86.
8. J. Kelsey, B. Schneier, D. Wagner, and C. Hall, "Side Channel Cryptanalysis of Product Ciphers," in *Proceedings of ESORICS '98*, Springer-Verlag, September 1998, pp. 97-110.
9. ANSI X.392, "American National Standard for Data Encryption Algorithm (DEA)," American Standards Institute, 1981.
10. J. Daemen, V. Rijmen, "Resistance Against Implementation Attacks: A Comparative Study of the AES Proposals," *Second Advanced Encryption Standard (AES) Candidate Conference*, http://csrc.nist.gov/encryption/aes/round1/conf2/aes2conf.htm, March 1999.
11. E. Biham, A. Shamir, "Power Analysis of the Key Scheduling of the AES Candidates," *Second Advanced Encryption Standard (AES) Candidate Conference*, http://csrc.nist.gov/encryption/aes/round1/conf2/aes2conf.htm, March 1999.
12. S. Chari, C. Jutla, J.R. Rao, P. Rohatgi, "A Cautionary Note Regarding Evaluation of AES Candidates on Smart-Cards," *Second Advanced Encryption Standard (AES) Candidate Conference*, http://csrc.nist.gov/encryption/aes/round1/conf2/aes2conf.htm, March 1999.
13. R. L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Comm. ACM*, vol. 21, 1978, pp. 120-126.
14. N. Koblitz, "Elliptic Curve Cryptosystems," *Mathematics of Computation*, vol. 48, 1987, pp. 203-9.
15. V. S. Miller, "Uses of Elliptic Curves in Cryptography," in *Proceedings of Advances in Cryptology–CRYPTO '85*, Springer-Verlag, 1986, pp. 417-26.
16. E. F. Brickel, "A Survey of Hardware Implementations of RSA," in *Proceedings of Advances in Cryptology–CRYPTO '89*, Springer-Verlag, 1990, pp. 368-70.
17. A. Selby and C. Mitchel, "Algorithms for Software Implementations of RSA," *IEE Proceedings*, vol. 136E, 1989, pp. 166-70.
18. S. E. Eldridge and C. D. Walter, "Hardware Implementations of Montgomery's Modular Multiplication Algorithm," *IEEE Transactions on Computers*, vol. 42, No. 6, June 1993, pp. 693-9.
19. S. R. Dussé and B. S. Kaliski Jr., "A Cryptographic Library for the Motorola 56000," in *Proceedings of Advances in Cryptology–Eurocrypt '90*, Springer-Verlag, 1991, pp. 230-44.
20. G. Monier, "Method for the Implementation of Modular Multiplication According to the Montgomery Method," *United States Patent*, No. 5,745, 398, April 28, 1998.
21. C. D. Gressel, D. Hendel, I. Dror, I. Hadad and B. Arazi, "Compact Microelectronic Device for Performing Modular Multiplication and Exponentiation over Large Numbers," *United States Patent*, No. 5,742,530, April 21, 1998.
22. P. L. Montgomery, "Modular Multiplication Without Trial Division," *Mathematics of Computation*, vol. 44, 1985, pp. 519-21.
23. ISO7816, "Identification Cards-Integrated Circuit(s) Cards with Contacts," International Organization for Standardization.
24. D. Chaum, "Blind Signatures for Untraceable Payments," in *Proceedings of Advances in Cryptology–CRYPTO '82*, Plenum Press, 1983, pp. 199-203.