Partitioning Attacks: Or How to Rapidly Clone Some GSM Cards

Josyula R. Rao, Pankaj Rohatgi and Helmut Scherzer
IBM Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598
{jrrao@us, rohatgi@us, scherzer@de}.ibm.com

Stephane Tinguely
Communications Systems Division
Swiss Federal Institute of Technology
1015 Lausanne
Switzerland
stephane.tinguely@epfl.ch

Abstract

In this paper, we introduce a new class of side–channel attacks called partitioning attacks. *We have successfully launched a version of the attack on several implementations of COMP128, the popular GSM authentication algorithm that has been deployed by different service providers in several types of SIM cards, to retrieve the 128 bit key using as few as 8 chosen plaintexts. We show how partitioning attacks can be used effectively to attack implementations that have been equipped with ad hoc and inadequate countermeasures against side–channel attacks. Such ad hoc countermeasures are systemic in implementations of cryptographic algorithms, such as COMP128, which require the use of large tables since there has been a mistaken belief that sound countermeasures require more resources than are available. To address this problem, we describe a new resource–efficient countermeasure for protecting table lookups in cryptographic implementations and justify its correctness rigorously.*
Keywords: *smartcards, authentication, security protocols, commercial and industrial security, side channel attacks, GSM, COMP128.*

1. Introduction

Side–channel attacks [6, 8, 2, 10] have recently gained prominence as an effective and practical means for attacking cryptographic systems. Cryptographic algorithms are traditionally designed to withstand attacks that treat the implementation as a black box, focusing instead on exploiting subtle relationships between inputs and outputs. In practice, the black box model may be unrealistic, since any implementation provides more information to a determined adversary than just the inputs and outputs. For instance, an adversary can obtain sensitive information from side–channels such as timing of operations[5], power consumption [6], electromagnetic emanations [9] etc. In fact, in constrained devices such as chip–cards, straightforward implementations of any cryptographic algorithm can be compromised with minimal work [6, 3].

In view of these exposures, vendors of cryptographic systems employ a variety of software and hardware countermeasures to "harden" their implementations against side–channel attacks. Popular software countermeasures are based on generic principles described outlined [6, 2, 4]. Implementing these principles typically requires either protocol changes [6] or additional resources [2, 4]. In fact, the authors of [2] enunciate a *cardinal principle*, outlined in the next section, that must be followed for implementations to resist first–order differential side–channel attacks. Realizing an implementation that conforms to this principle, especially on devices with resource and cost limitations, is a challenging and error–prone task which several implementors fail to perform correctly. In addition, many implementors erroneously believe resources limitations prevent effective application of this principle and therefore apply some ad–hoc and partly effective countermeasures.

As a consequence, many supposedly "hardened" implementations remain vulnerable. This is especially true for algorithms that employ large table lookups since obtaining side–channel attack resistance for table lookups is tricky and appears to be highly resource intensive. For example, the commonly used authentication/session-key generation algorithm in existing GSM phone networks, i.e, the COMP128 algorithm, requires lookup of five tables of sizes 512, 256, 128, 64 and 32 bytes each.

In this paper, we develop a new class of side–channel attacks, called *partitioning attacks*, which can be used to attack implementations which may otherwise resist some side–channel attacks. We introduce the concepts underlying this new class of attacks by showing how an implementation of the COMP128 algorithm on the SIM cards of a popular GSM network, that is resistant to some side–channel attacks, can be broken[1]. The partitioning attack on the implementation is extremely effective. The entire 128 bit key of COMP128 can be recovered from a SIM card with less than 1000 invocations with random inputs, or 255 chosen inputs, or only 8 adaptively chosen inputs! Thus, an adversary who has possession of a SIM card for a minute can easily extract the key. In contrast, the previously best technique to attack GSM SIM cards was to employ a *cryptanalytic* attack on the COMP128 algorithm with 150,000 chosen inputs. This entails access to a SIM card for around 8 hours and a card reader capable of performing around 6 invocations/second [1].

Next, we generalize the ideas underlying the attack on COMP128 attack to develop a methodology for launching a partitioning attack on any implementation of any algorithm which violates the cardinal principle. Whereas, the specifics and effectiveness of the attack would depend on the particular algorithm, platform and the manner of violation of the cardinal principle, nevertheless the proposed methodology will reduce the amount of entropy in the sensitive information at the very least.

The partitioning attack on COMP128 exploits weaknesses and vulnerabilities in the implementation of table lookups. In general, protecting table lookup operations, while conforming to the cardinal principle, appears to be very tricky, especially in resource constrained devices. Since table lookup is a basic cryptographic primitive for introducing non–linearity, the same problem also exists, to a lesser extent, with many other cryptographic algorithms such as DES and AES. In view of the importance of this primitive, we feel that it is necessary to have a specific protection mechanism for this operation. We, therefore, describe a new and resource efficient table lookup mechanism which can be employed in a wide variety of devices to defend against side–channel attacks. This mechanism is based on a space–time tradeoff: it allows the implementation of table lookups using substantially less RAM than the cumulative sizes of the tables being protected at the cost of additional operations. We believe that with the adoption of this mechanism, table lookups will cease to be avenues for side–channel attacks.

The paper is organized as follows: first, we provide required background on side–channel exposures, attacks and the cardinal principle for countermeasures in Section 2. In Section 3 the attack on the COMP128 implementation on a GSM SIM card is described. The methodology for developing general partitioning attacks is described in Section 4. Finally, in Section 5, we describe a technique which can protect against side–channel attacks on table lookup operations in resource constrained devices.

[1]Partitioning attacks have been validated on SIM cards deployed on several international networks. We have contacted the affected vendors whose identities will not be disclosed here.

2. Background

All equipment leaks information via several side–channels in the course of performing any computation. Examples of side–channels include timing of externally visible operations, instantaneous power consumption, electromagnetic (EM) emissions, etc [5, 6, 7, 9].

The timing side–channel is the easiest to describe and protect. The timing side–channel is exploitable if the timing of an externally observable operation is affected by sensitive information. Timing channels are easily addressed by ensuring that sensitive operations take the same amount of time or the timing depends only on other non–sensitive parameters. If that is not possible, then another effective countermeasure is to limit the information leaked via the timing side–channel.

The power and EM side–channels are significantly more powerful and consequently harder to defend against. These side–channels arise due to current flows within a device. The exposure is especially pronounced in CMOS devices, where current flows only when some change occurs in the logic state at each clock cycle: this linkage provides an unobstructed view into the logic state and transitions in the underlying device at each clock cycle via the power and EM side–channels. At each clock cycle, the signals carried by these side–channels are therefore statistically correlated to those bits in the logic state of the device which either determine the events that will occur in the clock cycle or are affected by events occurring within the cycle. These bits are known as *relevant bits* [2]. For example, for a cycle which loads a word of data from a memory location into a register, the relevant bits include the (binary) contents of all bus lines and circuits on which the data will flow, the initial contents of the register to which the data will be transferred and the bits of the specific data in the accessed memory location. While the number of relevant bits appears to be large, it should be noted that they constitute a tiny fraction of the overall state of device. It is also known that each of the relevant bits contributes to the side–channel signals in somewhat different ways; the effect of one relevant bit is not identical to that of another even though the bits may be similar in function (such as two bus lines). Of these two side–channels, the EM side–channel is more powerful as it can provide multiple views of the events unfolding within the device [9] due to different types of emanations from different parts of the device whereas the power consumption channel provides only a single aggregated view of the net current flow into the device.

Power and EM attacks on CMOS devices and countermeasures against them can be derived from this basic understanding of information leakage. Simple power analysis and EM attacks (SPA and SEMA) [6, 7] exploit the fact that if an implementation follows different execution paths based on the value of sensitive information, this dependence shows up as large differences in the side–channel signals since both the relevant bits and their values are quite different for different execution paths. Thus, the side–channel for a single execution could be used to determine the execution path taken and hence obtain sensitive information. For example, in a square and multiply based implementation of RSA exponentiation, the sequence of squares and multiplies performed uniquely determines the secret key; moreover, this sequence is readily visible in the side–channel since the code for squaring big–integers is quite different from the code to multiply big–integers. A countermeasure for these simple attacks is to have all operations take the same execution path.

Differential (or more precisely the first–order differential) side–channel attacks such as DPA and DEMA [6, 7], are a class of powerful attacks which work even when the execution sequence is identical for all inputs and values of the sensitive information. These attacks are based on the fact that even though the execution path is always the same, the values of the relevant bits at intermediate clock cycles are determined by values of inputs and sensitive information, since the device has to compute some function of these quantities. Since the side–channel signals are statistically correlated to the values of these relevant bits and hence depend on the sensitive information, it is possible to extract sensitive information by performing statistical analysis on these signals. For example, an adversary can do hypothesis testing, by forming a hypothesis on the value of some part of the sensitive information and predicting the value of some relevant bit in some cycle based on the hypothesis and known or chosen inputs/outputs. The hypothesis can then be verified by checking whether or not the signals correlate with the predicted value of the relevant bit [6].

The only way to completely eliminate differential attacks is to strictly adhere to the following Cardinal Principle [2]:

Cardinal Principle: *Relevant bits of all intermediate cycles and their values should be* statistically independent *of the inputs, outputs and sensitive information.*

Clearly, these intermediate cycles do not include the initial cycles that manipulate only the input and the final clock cycles that manipulate only the output. Techniques that achieve this while still being able to perform computations have been described in [2]. Whereas incomplete, improper or inadequate implementation of this basic principle may provide partial resistance to hypothesis testing attacks, it will result in susceptibilities to the partitioning attack that is described later in this paper.

An even more powerful class of attacks, known as the higher–order differential side–channel attacks [6], are based on multivariate statistical analysis of multiple signals from multiple sections of the computation. Although, these attacks can be used to overcome countermeasures against differential side–channel attacks, effective countermeasures against these attacks are extensions of the countermeasures against first order attacks[2]. We will not focus on these attacks and countermeasures against them in this paper.

3. Attacks on a GSM SIM card

The main authentication and session key generation mechanism for many GSM networks, specified by the abstract algorithms A3 and A8, are often realized in practice using a single algorithm known as COMP128. While the actual specification of COMP128 was never made public, the algorithm has been reverse engineered and cryptanalyzed [1]. Since, the GSM specification for SIM cards is widely available, all that is needed to clone a SIM card is the 128–bit COMP128 secret key that is embedded in the card.

3.1. COMP128 Algorithm

COMP128 is a keyed hash function. It takes a 16 byte (128 bits) key, K, and 16 byte (128 bits) of data, R, to output a 12 byte (96 bits) hash. The key K, as used in the GSM protocol, is unique to each subscriber and is stored in the SIM card. The input data R is a random challenge supplied by the base station. The first 32 bits of the hash are used as a response to the challenge and sent back to the base station. The remaining 64 bits are used as a session key for voice encryption using the A5 algorithm.

The algorithm first loads K and R in a 32–byte vector $X[]$. K is stored in $X[0..15]$ and R is stored in $X[16..31]$. Then, eight iterative loops are applied on $X[]$. Each iteration starts with a butterfly–structure like compression. The compression consists of five levels of table lookups using $T0[512]$, $T1[256]$, $T2[128]$, $T3[64]$ and $T4[32]$ respectively. In all iterations except the last, a permutation follows the compression. Each Ti contains only (8-i)–bit values. Thus, compression results in 32 4–bit values, that are then assembled into 16 bytes before the permutation is applied. These 16 bytes are stored into $X[16..31]$ and K is loaded into $X[0..15]$ before a new iteration begins. The resulting 128 bits after the eight iterations are further compressed to 12 bytes, which form the output of the algorithm.

Pseudo–code of the compression in COMP128 is listed below:

```
for $j$ = 0 to 4 do {
    for $k$ = 0 to $2^j$-1 do {
        for $l$=0 to $2^{(4-j)}$-1 do {
            $m = l + k*2^{(5-j)}$;
            $n = m + 2^{(4-j)}$;
```

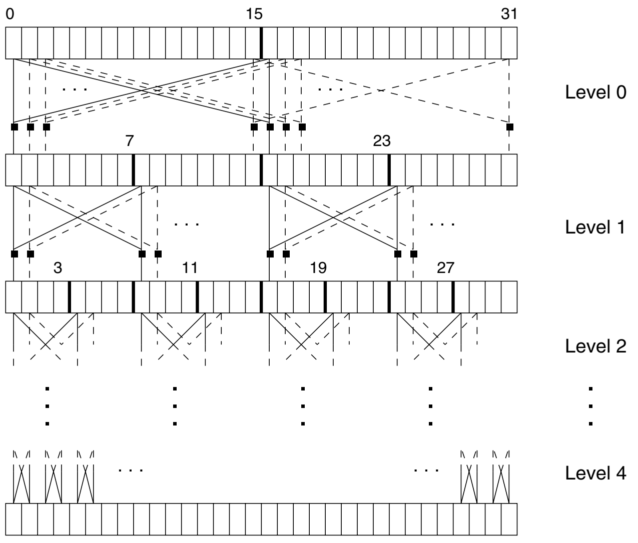

Figure 1. Butterfly structure of a compression in COMP128

```
                y = $(X[m] + 2{*}X[n])$ mod $2^{(9-j)}$;
                z = $(2{*}X[m] + X[n])$ mod $2^{(9-j)}$;
                $X[m] = Tj[y]$;
                $X[n] = Tj[z]$
            }
        }
}
```

For each level, the compression works on pairs of equal sized sections of $X[]$. In level 0, ($j = 0$), $X[]$ is split into two sections $X[0..15]$ and $X[16..31]$. The value of each right element, $X[i{+}16]$, ($i = 0..15$) is combined with that of the left element, $X[i]$, to compute $y{=}(X[i] + 2{*}X[i{+}16])\ mod\ 512$. Similarly, the value of the left element, $X[i]$ is combined with the corresponding right element to compute $z{=}(2{*}X[i] + X[i{+}16])$ mod (512). $X[i]$ and $X[i{+}16]$ are then replaced by $T0[y]$ and $T0[z]$ before the next level starts. This crosswise substitution, as shown in Figure 1, is referred as a butterfly–structure. On every new level, a section gets divided into a pair of sections in which the same scheme is applied. Note that the size of the table decreases in succeeding levels. Accordingly, level 1 computes $y{=}(X[i] + 2{*}X[i{+}8])$ mod 256 and $z{=}(2{*}X[i] + X[i{+}8])$ mod 256 for i=0..7, 16..23. In level 2, $y{=}(X[i] + 2{*}X[i{+}4])$ mod 128 and $z{=}(2{*}X[i] + X[i{+}4])$ mod 128 for i=0..3, 8..11, 16..19, 24..27 and so on.

3.2. Failed DPA Attacks

The GSM SIM card specification defines a command that invokes the COMP128 algorithm with any input data. This invocation can be performed any number of times. This enables us to assemble an experimental setup for invoking COMP128 on any chosen input and collecting the power and EM side–channel signals.

We began our analysis of the card by attempting a differential power analysis (DPA) attack using randomly chosen inputs. From the COMP128 specification, there appears a very simple and obvious avenue for attack.

Initially, $X[0..15] = K$ and $X[16..31] = R$. In the first level of the first compression round, $X[0]$ is replaced by T0[y] where y = (K[0] + 2*R[0]) mod (512). To extract the first byte of the key, $K[0]$, we make a hypothesis on its value and use it to predict the value of $T0[y]$ since the first byte of input data $R[0]$ is known. If the hypothesis is correct, then the predicted value of $T0[y]$ will be identical to the actual value in the computation. Therefore, any bit of the predicted value $T0[y]$ will correlate with the signal. If the hypothesis is incorrect, since the table $T0$ is random, hardly any correlation will be seen. Therefore, of all possible 256 hypotheses for $K[0]$, the hypothesis with the highest absolute correlation with the signal, is most likely to be the key. The same method should work for the remaining key bytes.

For our attack, we collected 1000 signals of the power consumption during the COMP128 encryption with random input data. However, in the attack on $K[0]$, none of the bits of the predicted value $T0[y]$ resulted in a distinguishing correlation value for any hypothesis. Figure 2 summarizes the results obtained for the prediction of the third bit of $T0[y]$ for all the 256 hypotheses of $K[0]$. For each prediction (on the x–axis), the figure shows the absolute value of the maximum difference between the mean signals for a prediction of 0 and a prediction of 1. We call this the *zero–one difference*. This is a measure of the absolute value of the correlation between the predicted bit and the signals. The correlation is more or less the same for all the values of the hypotheses and cannot serve to distinguish the actual key byte from several other wrong hypotheses. The same behavior was observed for all the 8 bits of $T0[y]$ and also for other key bytes.

From these results, it is clear that the card had some countermeasures in place to thwart the obvious DPA attacks. At this stage, we can only speculate on the nature of the countermeasures. One possibility could be that the table values were masked in some deterministic way.

3.3. Input Data Correlation

Since the DPA attack failed, we decided to probe the implementation further for other statistical weaknesses. We started by computing the correlation of the signals with each of the bits of the input. Such an analysis highlights all the places where the input bit affects the computation.

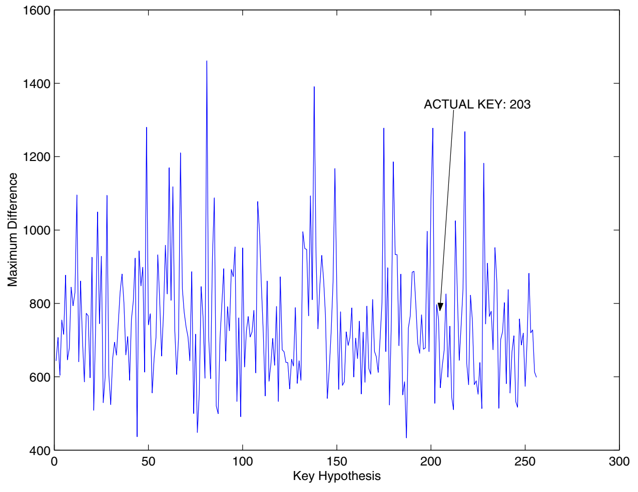

Figure 2. DPA results for bit 3

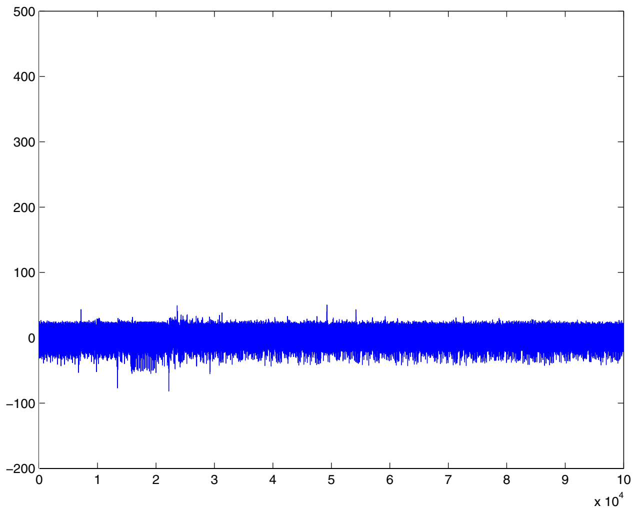

Figure 3. Input correlation for LSB of R[0]

First, we computed the input correlation of the least significant bit of $R[0]$ and found small correlations in a few places presumably where the bit was accessed, as shown in Figure 3, where the zero–one difference is plotted against time. The same was observed for the next few bits. Surprisingly, for the most significant bit (MSB) we observed a huge correlation in the beginning of the first compression, as shown in Figure 4, where once again the zer–one difference is plotted against time. A similar, phenomenon was also observed for all bytes of R; in some cases, the second most significant bit also had a somewhat high correlation at another place in the first compression.

These experiments indicated a statistical anomaly with respect to the MSB of the data bytes, which could potentially be exploited as an avenue for attack.

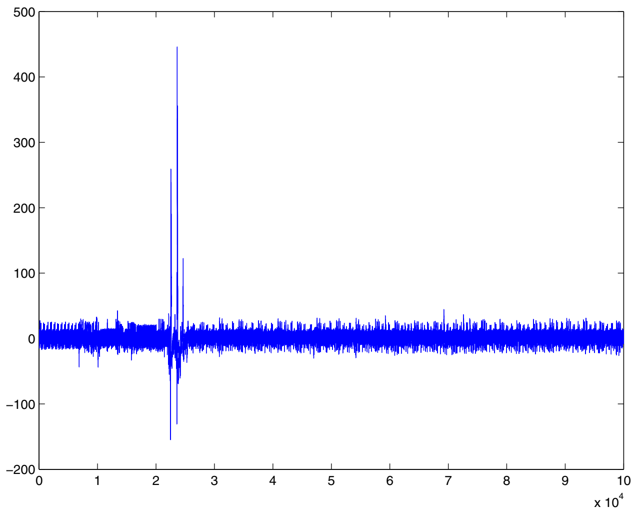

Figure 4. Input correlation for MSB of R[0]

3.4. Formulating a Partitioning Attack

As a prerequisite for converting an observed statistical anomaly into a viable attack, one needs to formulate an explanation for the anomaly, consistent with the observations. This requires knowledge of the algorithm, typical implementation techniques used by programmers for such platforms and some ingenuity on the part of the attacker.

As can be observed from the specification of COMP128, its usage of 9-bit quantities (in table indices, etc) makes its implementation on an 8-bit SIM card challenging even for an experienced programmer. Notice that, in the first level of compression, the quantities y and z are 9–bit values and the corresponding table $T0$ has a 9–bit index. Since it is not possible to directly address such a table on an 8-bit addressing architecture, it is highly likely that a programmer will split $T0$ into two tables $T00$ and $T01$, of sizes 256 each, which can be addressed by 8–bit indices. Furthermore, the easiest way to split $T0$ is to store the first 256 elements of $T0$ in $T00$ and the last 256 elements in $T01$.

Assuming that this type of table split was done by the implementors, we now have to explain why the observed anomaly arises in this case. Since the relevant bits and values are different for accessing different parts of memory, looking up a random element of $T00$ will result in a somewhat different power signal than looking up a random element of $T01$. Let s0(t) be the average power signal of the encryption when $T00$ is looked up at a random index and let s1(t) be the average signal in the case of a random lookup in $T01$. Let m(i) be the number of distinct values of $R[i]$ with MSB = 1, for which y will be in $T01$, i.e. ($K[i]$ + 2*$R[i]$) mod (512) will be in the range [256,511]. Correspondingly let n(i) be the number of bytes $R[i]$ with MSB = 0, for which y will be in $T01$.

We now claim that the observed correlation of the signals with the MSB of $R[i]$ will be proportional to (m(i)-n(i))*(s0(t)-s1(t)), i.e., the observed correlation will be proportional to the quantity (m(i)-n(i)). This claim is quite easy to prove, once the reader observes a symmetry property of $T0$ lookups in COMP128: independent of the key $K[i]$, exactly half of the $R[i]$ result in values of y which lie in $T00$ and the other half in $T01$. Moreover, the values of $R[i]$ which fall into $T00$ and $T01$ are contiguous modulo 256. The results of these two different contiguous equipartitions of the space of 256 possible values of $R[i]$, based on MSB and table accessed is depicted in Figure 5. From this it can be observed that the average signal for MSB=0 should be proportional to (m(i)*s0(t) + n(i)*s1(t)). The average signal for MSB=1 should be proportional to (m(i)*s1(t) + n(i)*s0(t)). This indicated that the input correlation to the MSB should be proportional to the differences of the average signals, i.e., (m(i)-n(i))*(s0(t)-s1(t)).

Clearly, the values of m(i) and n(i) depend on the key byte $K[i]$. The exact values of the absolute difference of m(i) and n(i) for all possible key bytes $K[i]$ is shown in Figure 6. It is clear from this figure that (m(i)-n(i)) is large for almost all values of $K[i]$, with m(i) and n(i) being equal only in the special cases where $K[i]$ is 128 or 129. This explains the observed large correlations seen with the MSBs of the input bytes. In addition, the explanation based on table splitting, is also consistent with observations about the other bits. For the other bits, the indexing of a split table with y = ($K[i]$ + 2*$R[i]$) mod (512), does not introduce any correlation. However, indexing with z=(2*$K[i]$ + $R[i]$) mod (512), introduces correlation which diminishes exponentially as the bit position moves from MSB to LSB.

We therefore accept the table split explanation for the anomaly and focus on how this knowledge can be use to formulate an attack (that we term as a "partitioning" attack) on this and similar implementations of COMP128.

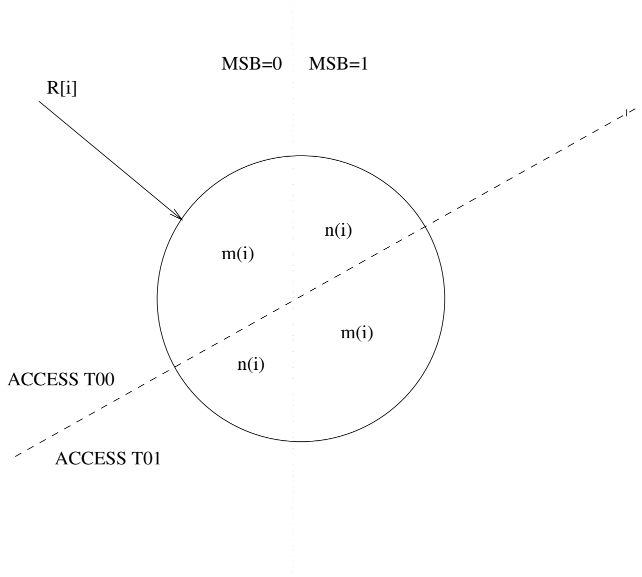

Figure 5. Partitioning of $R[i]$

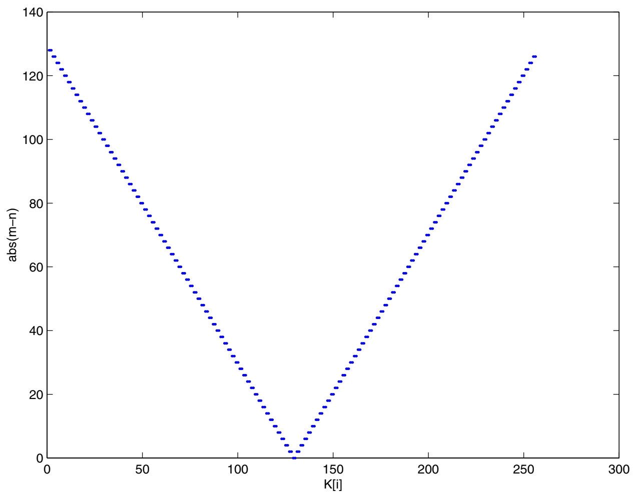

Figure 6. Absolute value of (m(i)-n(i)) for various values of key byte $K[i]$

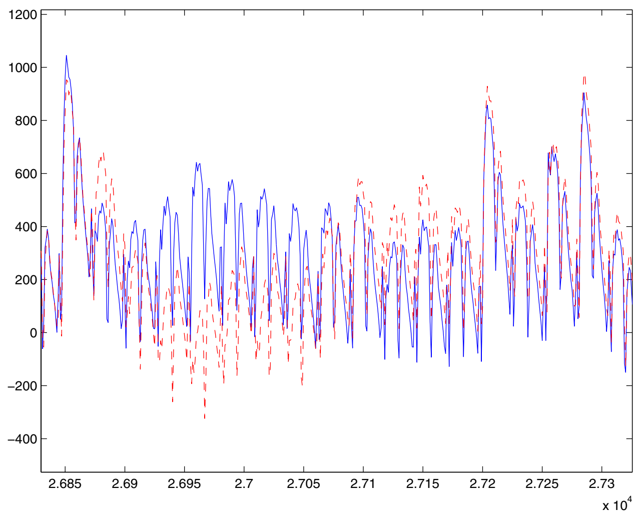

Figure 7. Power signals: $T00$ access versus $T01$ access

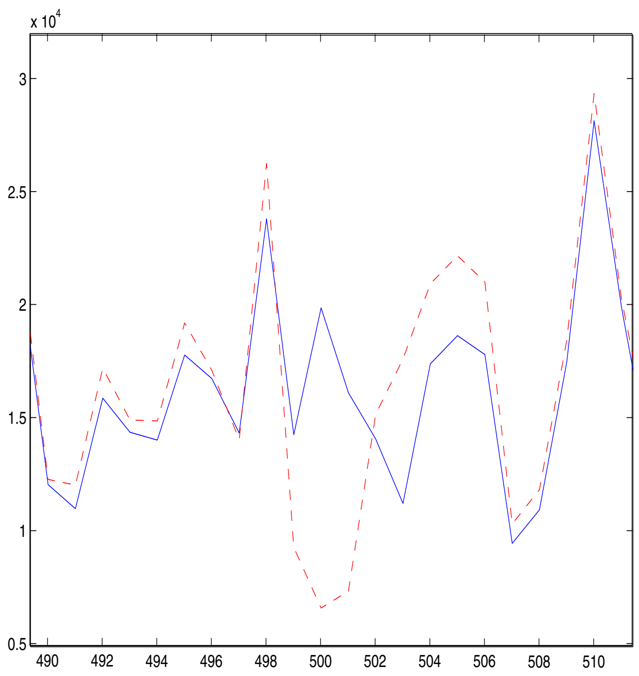

Figure 8. Processed power signals: $T00$ access versus $T01$ access

3.5. Partitioning Attack on COMP128

Assuming that the table split mechanism was employed, we started comparing signals to see whether it was possible to distinguish between the use of tables $T00$ and $T01$. By focusing on the regions where input correlation was found, we discovered that in each such region, one could directly classify each power signal into one of two categories, based on its shape in the region. We assumed that these two categories correspond to accessing $T00$ vs accessing $T01$, even though we could not decide which category corresponded to which table. Figure 7, which plots strengths of two signals drawn from two different categories against time, shows the distinctions between a signal in one category and a signal in the other. This distinguishability could be further enhanced by signal processing, as shown in Figure 8 which plots strengths of the processed signals against time. Similar results were also obtained for signals obtained from electromagnetic emanations from the card.

Moreover, there were 32 regions during the first level of compression where such distinctions existed, which in turn were organized into 16 pairs. This is consistent with the fact that the first level of compression in COMP128 requires two table lookups into $T0$ with indices y and z, for each of the 16 bytes of input.

We then focussed on the first region in the first pair and classified 1000 signals we had earlier collected with random inputs into the two categories. This yielded some very interesting but not entirely unexpected results; all signals with $R[0]$ in the range [0,..,26] and [155,...,255] fell in one category and all signals with $R[0]$ in the range [27,..,154] fell into the other. This contiguous equipartitioning (modulo 256) is consistent with the behavior expected while performing a lookup of $T0[y]$ for y =($K[0]$ + 2*$R[0]$) mod (512), using split tables $T00$ and $T01$. When $R[0]$ is 0, $T00$ will be accessed independent of $K[0]$. The transition when $R[0]$ goes from 26 to 27 has to be caused by the value of y crossing 256 for the first time. Similarly the transition when $R[0]$ goes from 154 to 155 must be caused by the value ($K[0]$ + 2*$R[0]$) crossing 512 for the first time. From this it follows that $K[0]$ can only be 202 or 203. Next, the same classification was performed with the second region in the first pair, which would correspond to accessing $T0$ at index z = (2*$K[0]$+$R[0]$) mod (512). Values of $R[0]$ in the range [0,..,105] were in one category and values in the range [106,..,255] in the other. Since $K[0]$ is either 202 or 203, the transition when $R[0]$ goes from 105 to 106 must be caused by (2*$K[0]$+$R[0]$) crossing 512 for the first time. This only occurs for $K[0]$=203 and hence we obtained the first byte of the key.

Similar analysis performed on the remaining pairs of regions yielded the remaining the key bytes [2]. In fact, the key byte is always uniquely determined from partitioning infor-

[2]This was confirmed by comparing the output of the card with the output of COMP128 code [1] with the obtained key.

mation about accesses with indices y and z. Even when the transition from one partition to the other is approximately known due to noisy signals, this technique cuts down the possible values of each key byte to a very small number, permitting feasible brute force search.

This attack was performed with 1000 samples with random inputs. This ensured that with high probability, each input byte adequately covered the range of values [0,..,255] so that the transitions in the partitions could be determined with good accuracy. In practice, one could work with fewer random inputs (say 500). This problem also disappears if the inputs could be chosen, in which case 256 different inputs with each input byte covering all the 256 possibilities suffices. If one is satisfied with 2 possibilities per key byte, i.e., 2^{16} possibilities for a 16-byte key, only 128 inputs with covering byte values [1,128] suffice, since these can determine the first y transitions. If in addition, one could adaptively choose the inputs, then exploiting the fact that for each input byte, the partitions consist of contiguous values, one can perform a binary search to determine the transition from one partition to another. This binary search could be performed simultaneously for all 16 input bytes and thus entire 16-byte key can be obtained using only 8 adaptively chosen inputs. In fact, seven adaptively chosen inputs are sufficient to determine all the first y transitions thus restricting each key byte to two possibilities. A final carefully chosen input targeted towards all the z accesses then uniquely determines the key.

4. General Partitioning Attacks

We now generalize the ideas used in the preceding COMP128 attack to show how Partitioning Attacks can be formulated on any implementation of any algorithm in which countermeasures against differential side-channel analysis have not been properly applied. Clearly, the actual attack would be very dependent on the algorithm being implemented, the architecture being used and would require some guesswork on the part of the attacker as to the types of software countermeasures being used. Therefore the approach can only be described at an abstract level.

Assume there is an implementation that violates the Cardinal Principle, that is, the relevant bits or their values thereof in some intermediate cycle are not *statistically independent* of the input, output and sensitive information. Since the side–channel signals obtained during that intermediate cycle are affected by and correlated to these relevant bits and the contribution of each of these relevant bits is somewhat different, this means that the side–channel signals will also not be *statistically independent* of inputs, outputs and sensitive information. General Partitioning attacks try to exploit this *statistical dependence* in the signals to extract the sensitive information.

Without loss of generality, assume that the values of the relevant bits at that intermediate cycle depend on some parts of the input and the sensitive information [3]. Since sensitive information such as keys is usually fixed for a device, this dependence will show up as a dependence on the inputs. The statistical distribution of the side channel signal for each of the intermediate cycles for any input can be estimated with reasonable accuracy by sampling. This can be done by repeatedly invoking the operation on the same input and extracting the resulting signals. By performing these estimates on several inputs, the intermediate cycles where the statistical distribution is input dependent can be identified. In practice, estimation of the statistical distribution is likely to be unnecessary; an estimation of a parameter of the distribution (such as the mean signal) should suffice.

Once the input dependent intermediate cycles are identified, attention should be focussed on the first such cycle; the others can be revisited after the analysis of the first such cycle is complete. From the known information about the abstract algorithm being implemented, the characteristics of the device, guesswork about the implementation approach and by experiments with different inputs, it should be possible to isolate a small valued function of the input which affects this intermediate cycle. This is because, the cycle can be dealing with at most one machine word of information. In some cases this could directly be a word of the input or a word derived from the input and possibly some secret information during the initial processing stage of the algorithm. A good heuristic to isolate this function would be to vary a few bits of the input while keeping the rest of the bits the same, to see whether or not the distribution is affected. Another heuristic is to compute correlations between the input bits and the signal to identify how far these input bits are directly manipulated in the computation and whether or not these bits correlate at this cycle. If there is no input correlation at this cycle, then one can still estimate how far this cycle is from the other cycles which manipulated inputs. Some algorithms perform some form of input mixing operation before performing sensitive operations; in those cases, the heuristic should compute correlations with the bits resulting from the mixing operation rather than input bits.

After the small valued function has been reasonably well isolated, the next step is to perform statistical characterization of the signal distributions for all possible values of the function, using chosen or known inputs that affect the function. For example, if the function is the first byte of input, statistical characterization should be performed with all possible values for the first byte of input (with other bytes

[3]It is easy to attack implementation in which the signals depend on just the sensitive information; after some initial training on signals with different values of the sensitive information, a "best match" approach can be used to recover sensitive information used in a specific device.

kept the same). Based on this analysis, it should be possible to cluster the various values of the function into different partitions based on statistical similarities of the side–channel signals created by these values.

The gist of the partitioning attack is the following: Knowledge of the partitions and the values that fall into each partition provides information as to the processing that has occurred thus far. If this processing involves secret information then given knowledge of the algorithm being implemented, guesswork on how the implementation has been done and limitations of the device, one can come up with an hypothesis for the observed partitioning behavior. This hypothesis together with the actual values that fall into each partition puts constraints on the sensitive information was involved in the computation thus far; which in turn implies a leakage of sensitive information in information theoretic terms.

Similarly, each intermediate cycle where a different partitioning behavior is observed, provides an avenue for information leakage about the sensitive information employed in the algorithm. If enough such cycles can be found and exploited then the sensitive information may be completely disclosed. Even if this is not the case, it is quite possible that the entropy of the sensitive information could be so reduced that exhaustive search based attacks become feasible.

5. Countermeasures

Table lookup is a fundamental primitive used by several cryptographic algorithms such as DES, AES and COMP128. Implementing this primitive in a side–channel attack resistant manner on constrained devices poses special challenges since straightforward application of countermeasures seem to require the creation and use of large random tables which may take up much more RAM than is available. In addition, the use of the tables requiring large indices poses problems for devices with limited addressing capabilities, e.g., many chip–card devices only support an 8–bit index into a byte table. Due to these complications, several implementations of table lookup based algorithms on limited devices remain susceptible to side–channel attacks.

We now propose a methodology to create efficient, first–order differential side–channel attack resistant implementation of table lookups using substantially less RAM than the cumulative size of the tables being accessed. In addition, for most practical cases, this solution also takes care of the problem of large indices. This methodology is based on using a combination of three basic building blocks, the "Table Mask" operation, the "Table Split" operation and the "Table Aggregate" operation. The Table Mask operation provides the main defense against side–channel analysis but does not address the problem of RAM or addressing constraints. However, in combination with the Table Split and Table Aggregate operations the problem can be solved for most situations arising in practice. For illustrative purposes, in this paper, we focus on table lookups for tables with indices a power of 2.

5.1. Table Mask Operation

Let $T : 0,1^n \rightarrow 0,1^b$ be an n–bit to b–bit table to be accessed, i.e., T consists of 2^n elements of size b–bits each. The size of T is $b * 2^n$ bits. Any implementation which directly looks up T is vulnerable to differential side–channel attacks since the side–channel signals at the time of table lookup will correlate with each bit of the index accessed and with each bit of the value retrieved. To remove this correlation, the Table Mask operation is performed as follows:

Definition 1 *Let IP (index permutation) be a class of n–bit to n–bit permutations with the property that for any n–bit value i, if a permutation ip is chosen uniformly at random from the class IP, then the value $ip(i)$ is* statistically independent *of i.*

A simple example for IP would be the class of n–bit XOR permutations. For any fixed n–bit number r and input i, the function $i \oplus r$ is a n–bit to n–bit permutation, which we term as an XOR–permutation. The class of n–bit XOR permutations is the collection of XOR–permutations for all possible values of r.

Definition 2 *Let OP (output permutation) be a class of b–bit to b–bit permutations with the property that for any b–bit value o, if a permutation op is chosen uniformly at random from the class OP, then the value $op(o)$ is* statistically independent *of o.*

A simple example for OP would be the class of b–bit XOR permutations.

For each instance of a cryptographic operation requiring one or multiple lookups of a table T, a fresh random looking "masked" table T' is computed and placed in RAM. This is done as follows:

- Select a permutation $ip \in IP$ uniformly at random.
- Select a permutation $op \in OP$ uniformly at random.
- Define the masked table T' as: $\forall i, 0 <= i < 2n, T'[ip(i)] = op(T[i])$.

The table lookups work with a masked values of the index (ip(i) instead of i) and result in masked values of the output. By working throughout with appropriately masked values, one can perform the entire algorithm without ever exposing any intermediate result in the clear and preventing a differential side–channel attack [2, 4]. The properties of IP and

OP ensure that the side–channel during table lookups has a statistical distribution which is *independent* of the index and the corresponding value of T (not T') that is being accessed.

The main problem with the Table Mask operation described above is that it requires the table T' to be in RAM and the size of T' is the same as that of T. Moreover, if many such tables have to be examined multiple times within the algorithm, then for efficiency purposes, masked versions of all tables must be simultaneously kept in RAM. Also if n is large, then T' can have the same indexing problem as T. The solution to these problems, is to use the Table Mask operation in conjunction with Table Split and Table Aggregate operations described below.

5.2. Table Split Operation

This operation is useful in cases where the size of a table is greater than the available RAM or the index is too large for the addressing capabilities of the device. The basic idea is that an unmasked n–bit to b–bit table T can be split into multiple tables $T_1, T_2, \ldots, T_k$ such that each table T_i takes an n–bit index and produces a b_i bit output where the the b_i's (for i=1 to k) sum up to b. For example, a n–bit to b–bit table T could be split into b, n–bit to 1–bit tables $T_1, T_2, \ldots, T_b$ where T_i gives the i'th bit of the output of T. By packing multiple entries into a single addressable memory unit (such as a byte or word), the actual index needed to access some value into any one of these split tables could be less than n–bits and also the size of each of these tables is smaller. For example, an n–bit to b–bit table T is split into b, n–bit to 1–bit tables, $T_1, \ldots, T_b$ as described above by packing 8 output bits into a single byte, the size of the index into any table T_i becomes $n-3$ bits and the individual table sizes are smaller by a factor of b.

5.3. Table Aggregate Operation

Suppose we have k, equal sized tables $T_1, T_2, \ldots, T_k$ each of size no more than M bytes. By combining the tables in various algebraic ways, one can create an aggregate table T of size M bytes, with the property that given T and all but one of the tables T_i, one can recover the table T_i.

For example, T could be created by padding all tables with 0–valued bytes to make their sizes exactly M bytes each and then doing a byte–wise XOR[4] of the corresponding bytes in all the K padded tables. Clearly, any table T_i can be recovered from T and $T_1, \ldots, T_{i-1}, T_{i+1}, \ldots, T_k$ by XORing all these tables. An advantage of using the byte–wise XOR[5] to create the aggregate table T is that, if the tables $T_1, \ldots, T_k$ can be created one or a few bytes at a time, then the table T can be created using only slightly more than M bytes of memory.

[4] Addition modulo 256 could also be used instead of XOR

[5] or addition modulo 256

5.4. Combining Building Blocks for a Solution

Suppose one only had M bytes of RAM to spare for table lookup on a device and a restriction on the memory addressing mechanism which limited any index to within n–bits. Suppose one had several tables to be looked up. In the first stage, one would use the Table Split mechanism on all tables whose size is more than M and/or whose index requirements exceed the bounds. One can now rewrite the algorithm to work with these split tables and these split tables could be in ROM. At this stage, we have eliminated the large index problem but are still vulnerable to differential side–channel attacks. In the next stage we would "conceptually" apply the "Table Mask" operation on all the tables we now have and then apply the "Table Aggregate" operation on all the resulting masked tables. In practice, one would not actually create the full masked tables, since this would take too much RAM, one would create one or a few bytes of these tables at a time and update the Aggregate table being computed. Thus at this stage our main RAM usage would be the M bytes needed to store the Aggregate table. We also put copies of all the original tables in ROM.

We then modify the algorithm code we wrote for split tables to work with masked values and use the Aggregated Masked table for the table lookup for any table. The problem with this approach is that table lookup into the Aggregated Masked table gives us not just the Masked entry of the table we are interested in but instead some combination of the Masked entry of interest and contributions of entries from other Masked tables. However since the masking information, i.e., the index and output permutations are available for all the tables, one can use direct ROM lookups in the actual tables to remove the contributions from the *other* entries so that we are left with only the masked entry we are interested in. Moreover, given the way masking works, i.e., the masked index being independent of the real index, the indices looked up in the other ROM tables would be random, i.e., *statistically independent*, of the real index in the table of interest. Thus, we get a differential side–channel resistant implementation involving table lookups within the resource/addressing bounds.

6 Acknowledgements

We would like to thank Arvind Krishna for challenging us to devise power and EM side–channel attacks on GSM cards (in the short span of four days!) and Charles Palmer for his encouragement and support during that time.

References

[1] Mark Briceno, Ian Goldberg and David Wagner. See `http://www.isaac.cs.berkeley.edu/isaac/gsm--faq.html`

[2] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao and Pankaj Rohatgi. Towards Sound Countermeasures to Counteract Power–Analysis Attacks. Advances in Cryptology — Proceedings of Crypto '99, Springer–Verlag, LNCS 1666, August 1999, pages 398–412.

[3] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao and Pankaj Rohatgi. A Cautionary Note Regarding the Evaluation of AES Candidates on Smart Cards. Proceedings of the Second Advanced Encryption Standard Candidate Conference, Rome, Italy, March 1999.

[4] L. Goubin and J. Patarin. DES and Differential Power Analysis. Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems, CHES '99, August 12–13, 1999, Worcester, MA, pages 158–172.

[5] P. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS and Other Systems. Advances in Cryptology-Crypto '96, Lecture Notes in Computer Science # 1109, pp 104–113.

[6] P. Kocher, J. Jaffe and B. Jun. Differential Power Analysis: Leaking Secrets. Advances in Cryptology — Proceedings of Crypto '99, Springer Verlag, LNCS 1666, pages 388–397. One version of the paper is available online at `http://www.cryptography.com/dpa/technical/index.html`.

[7] Jean–Jacques Quisquater and David Samyde. Simple electromagnetic analysis for smart cards: new results. Rump session talk at Cyrpto 2000.

[8] J. Kelsey, Bruce Schneier, D. Wagner and C. Hall. Side Channel Cryptanalysis of Product Ciphers. Journal of Computer Security, Volume 8, Number 2–3, 2000, pages 141–158.

[9] Josyula R. Rao and Pankaj Rohatgi. EMpowering Side–Channel Attacks. IACR Crypto e-print archive at http://eprint.iacr.org, paper 2001/037.

[10] Dawn Xiaodong Song, David Wagner and Xuqing Tim. Timing Analysis of Keystrokes and Timing Attacks on SSH. Proceedings of the 10th USENIX Security Symposium, Washington, DC, August 2001, pages 337–352.