Third Order Differential Analysis and A Split Mask Countermeasure For Low Energy Embedded Processors

Catherine H. Gebotys
Department of Electrical and
Computer Engineering,
University of Waterloo
cgebotys@uwaterloo.ca

ABSTRACT

Future wireless embedded devices will be increasingly powerful supporting many more applications including one of the most crucial, security. Although many embedded devices offer more resistance to bus probing attacks due to their compact size, susceptibility to power or electromagnetic analysis attacks must be analyzed. This paper presents a new split mask countermeasure to thwart low order differential power analysis (DPA) and differential EM analysis (DEMA). For the first time real power and EM measurements are used to analyze the difficulty of launching new 3rd order DPA and DEMA attacks on a popular low energy 32-bit embedded ARM processor. Results show that the new split mask countermeasure provides increased security without large overheads of energy dissipation compared to previous research. With the emergence of security applications in PDAs, cellphones, and other embedded devices, low energy countermeasures for resistance to low order DPA/DEMA is crucial for supporting future wireless internet enabled devices.

1. Introduction and Related Research

Wireless devices such as PDAs, cellphones, etc now encompass a wide range of security functions demanding low energy. For example internet purchases, private company network access, authentication, and computations with confidential information will become increasingly common security functions in PDAs. Additionally a wide range of embedded security will proliferate in automobile electronics, security for IP core protection in FPGAs and ASIC technologies, and other areas. The cryptographic algorithms which are essential for these applications are typically run by embedded processors in these wireless devices. Unfortunately cryptographic algorithms are already known to consume significant amounts of energy [2]. Even worse, cryptographic algorithms which are resistant to attacks are known to have latency overheads up to 1.9 times[5]. These attack resistant algorithms have been developed for smartcard applications, where energy dissipation is not viewed as important. As more security applications migrate to the wireless device, resistance to attacks on the PDA or cellphone will become a necessity. These attacks may not only arise from device theft or loss but also during everyday use where unintentional electromagnetic (EM) waves radiated from the wireless device during cryptographic computations may leak confidential data to a nearby attacker. For example an attack may be successful in obtaining the secret keys stored in confidential memory in a wireless device. This attack may be possible through loss or theft of the device, or alternatively through temporary access to the device by monitoring the EM waves emanating from the device while

performing cryptographic computations. In the latter case the attack may be able to extract the encryption keys (typically not even accessible to the user of the PDA), making all wireless communications insecure. Researchers have already demonstrated that this new attack is viable[7,10,29] on a 8-bit processor. Since EM waves are highly correlated with power, ensuring wireless devices are secure from power analysis attacks is important. Nevertheless large overheads in energy to achieve this resistance may not be practical. Outside of smartcard research (which typically has been in the past limited to cheaper 8-bit or 16-bit processors) [6,7], few researchers have examined secure implementations of cryptographic software under the threat of power attacks on 32-bit processors. For example a very popular embedded processor is the ARM which is suitable for portable devices ranging from game devices to PDAs to cryptographic applications [5,16,17]. There is an important need to study energy optimized countermeasures for wireless portable devices, such as PDAs, cellphones, etc.

Currently research in power attacks of smart cards, have utilized 8-bit general purpose processors [5,3,12] and 5V processors running at 4MHz[4,1]with an average current consumption of 10mA[3]. Typically smart card applications are not time critical and energy dissipation is not a major concern since power is attained from the card reader (or ATM machine, etc). The measurement of power while a processor is executing an application (or a power trace) has been used in power-attacks of cryptographic devices, such as smart cards[1]. In particular the analysis of the variation of power, and computations on a number of power traces can be used to detect data and algorithmic dependencies[1]. This research studied the correlation of power variation with data values being manipulated and instruction sequencing. In the former case, known as differential power attacks (DPA), encryption[1] and public key encryption[22] applications were analyzed[1]. Differential power attacks of embedded low power processors have not been reported in the literature. Higher order differential attacks[15] are an extension of the 1^{st} order DPA which involve using joint statistics on multiple points within power traces.

Random sequencing of instructions (desynchronization), randomized power noise (through hardware circuits) and balancing methods[11,13] are some suggested countermeasures against power analysis. In the later technique computations are performed not only on the data but also on the complement of the data (where words contain for each of the data bits, it's complement [11]). Although researchers have suggested the balancing method may fail[13] due to the difference in the two complementary values, few researchers have used real power measurements in conjunction with DPA to verify this claim. Other researchers designed bus complement techniques in conjunction with bus precharging to resist DPA attacks [7]. Only a few researchers [7] have demonstrated this with simulation techniques, however none have verified this countermeasure using real power measurements. Other techniques which combine desynchronization and randomization have also been researched such as clock gating to protect against DPA [6]. However the impact of power randomization on DPA has also not been published with real power measurements.

In addition to the above countermeasures, other approaches have been suggested such as secret splitting[9], duplication method[18], multiplicative masking[19] and random masking[5]. Secret splitting, involves splitting the secret data into smaller pieces and combining them with random data [9]. Then the cryptographic algorithm is run on each word, which is composed of random and secret data. To attack the splitting method, a k^{th} order differential attack is required[9] (where secret data is split into k shares), however no statistics were derived to be used in this attack. The duplication method[18] was used to support secure computations with multiple split variables for input to the cryptographic tables or

Sboxes (*S[x]*). These researchers also used table duplication such that one table contained a randomly-chosen secret transformation on *x*, *A[x]*, and the alternate table contained *A[x]^S[x]*. Multiplicative masking was also defeated by a DPA attack[19]. In the random masking countermeasure, each secret piece of data is exclusive-or'd with a random data value (called a mask). To thwart a DPA attack the random data value must be changed periodically. However this involves remasking the tables (or exclusive-or the complete table data with a mask) within the algorithm. Countermeasures have been reported to run up to 1.9 times slower [5] for Rijndael (or AES, an advanced encryption standard [27]). These overheads are largely due to remasking of tables, so some researchers have investigated storing a limited number of masked tables [14] (referred to as the fixed value masking approach). The authors define the terms key xoring DPA and Sbox DPA. The key xoring DPA was an attack on the result of exclusive-oring the plaintext with the (masked or unmasked) key as shown in figure 1a). The Sbox DPA was an attack on the (masked or unmasked) output of the Sbox table (or output of the table look-up in figure 1a)). In both attacks it is assumed that the attacker has control over the input plaintexts which are exclusive-or'd with the key to index the Sbox table. Using probabilistic DPA the authors[14] proved that the fixed-value masking approach was secure as long as the masks were chosen appropriately (the probability of the i^{th} bit of a fixed mask being equal to zero is ½). However results using real power measurements were not performed to verify the fixed value masking approach. Later a second order DPA attack was developed [15], to thwart the random masking countermeasure (also known as "data whitening") as shown in figure 2a). A heuristic for the 2^{nd} order DPA statistic was developed and applied with real power measurements to a 8-bit processor at 3.57MHz using a sampe rate of 1G samples per second. The 2^{nd} order DPA converged slowly for some bits requiring approximately 2500 power traces. A 2^{nd} order DEMA attack was launched in [10] on a 8-bit processor with 500 EM traces and it was shown to produce better results than the 2^{nd} order DPA. Attacks using power samples of data and correlating them with the hamming weight of the data were studied in [23] for Rijndael, however masking was not considered.

Countermeasures must be energy optimized for wireless device implementation. Furthermore if these countermeasures can be defeated then they must require high order DPA/DEMA in that a large number of power/EM traces must be required making the attack very difficult. The use of real power and real EM measurements of low power embedded 32-bit processors is important to verify these countermeasures for portable applications. This paper presents differential power and EM analysis results for a popular low power embedded processor, the ARM7TDMI. For the first time real power measurements are used to analyze low energy countermeasures such as data complementing, power randomization, and fixed-value masking. New statistics are derived for 2^{nd} order DPA/DEMA and 3^{rd} order DPA/DEMA. The numbers of power and EM traces required in the attacks on a 32-bit processor are computed for both real power and real EM measurements. The next section will provide an introduction to DPA/DEMA followed by the proposed split mask countermeasure and it's application to Rijndael.

1.1 A Brief Introduction to Differential Power Analysis

A brief introduction to the differential analysis attack is presented in this section. Power traces are used to describe the attack however it is also applicable to electromagnetic traces (or the demodulated EM traces as described in [10]).

Consider figure 1 which illustrates a common part of many cryptographic algorithms, especially when executed on 32-bit processors (where table look-ups provide a much faster implementation of bit

level computations). Typically the input data or plaintext and key are involved in some computation whose result is indexed into a table look-up. Assume for now that the plaintext and key are exclusive-or'd together and then indexed into a table, as in the table method of the Rijndael advanced encryption standard[28] or fast implementation of DES. The attacker has control over the plaintext and knows the values in the table (according to some well published encryption algorithm). The attacker wishes to determine the key value.

The measurement of instantaneous current drawn from a device (known as a power trace when multiplied by the supply voltage of the device) while the device is executing an application has been used in power-attacks of cryptographic devices, such as smart cards[1]. In particular the analysis of the variation of power, and computations on a number of power traces can be used to detect data and algorithmic dependencies[1]. For example the DPA attack correlates the measured power signals to data being computed or accessed at the attack point in figure 1.

First the power trace emanating from the device (PDA, etc) is recorded while the device is computing the cryptographic algorithm. The following variables will be used to describe the subsequent processing: $C_i(t)$ represents the instantaneous power signal at time t of the i^{th} power trace, where $i=1,\ldots,n$; and j is the bit of the data resulting from the exclusive-or of the plaintext and key in the i^{th} execution of the algorithm. The attacker partitions the power traces into two groups according to bit j of the plaintext. One group contains all power traces where bit j, or b_j, of the plaintext is equal to 0 and the other group contains all power traces whose bit j of the plaintext is equal to 1.

Let $x_1(t) = \{C_i(t) \mid \forall i = 1,..n1, b_j = 0\}$ and $x_2(t) = \{C_i(t) \mid \forall i = 1,..n2, b_j = 1\}$ be the two groups of power traces. The mean of each group of power traces is computed (to average out the impact of the other non-j bits). The difference of means or differential power trace is obtained by subtracting the two averaged power traces. This subtraction attempts to remove the algorithmic impact on the power trace thus leaving only the j^{th} bit effect on the power trace. The differential power trace is equal to :

$$\overline{x_1(t)} - \overline{x_2(t)}$$

This difference of means is significant if it is much greater than the standard deviation of the difference of means. In this paper we define the DPA characteristic for the guessed key as the maximum value of the absolute differential power trace minus 2 standard deviations (of the difference of means) at that time. In other words the DPA characteristic (for each key guess) is equal to:

$$\max\left[\left|\overline{x_1(t)} - \overline{x_2(t)}\right| - 2\sqrt{\frac{\sigma_1(t)^2}{n1} + \frac{\sigma_2(t)^2}{n2}}\right]$$

where σ_k is the standard deviation of group k and nk is the number of samples in group k (where k=1 or k=2).

In figure 1 the mean of each group is represented by *a:* and *b:* (as an illustration of the means of $x_1(t)$ and $x_2(t)$). In DPA attacks on the exclusive-or operation in cryptographic applications, (key xoring DPA), two peaks will normally be seen. The first represents the effect of loading the plaintext itself (see the plaintext input in figure 1b), where the peak is represented by *a-b = {x…0…x} – {x…1…x})*. The second peak (located at the attack point) will be a similar peak *(a-b = {x…0…x} – {x…1…x})* or a peak in the opposite direction *(a-b = {x…1…x} – {x…0…x}*) depending upon the key bit. This second peak is used to determine whether the j^{th} bit of the secret key (the *?* in *x…?...x* of figure 1b)) was a zero or a one.

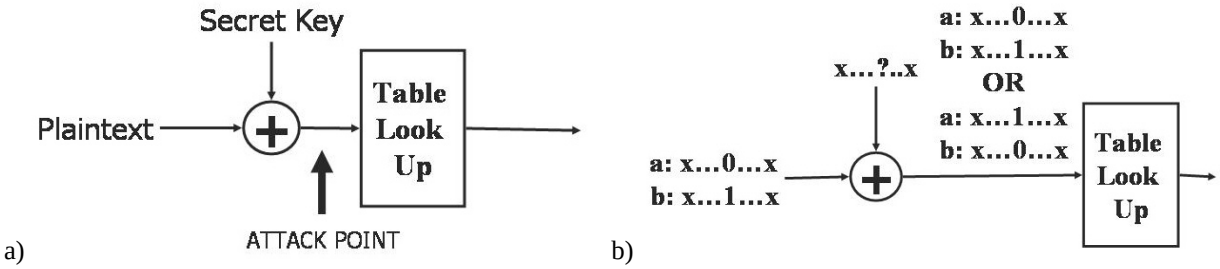


Fig. 1. attack point in a) and grouping of plaintexts in b).

The next section will present the split mask countermeasure for the key xoring DPA/DEMA in a general example. The detailed implementation of the split mask countermeasure for Rijndael is then presented in the following section. Finally experimental results are presented using real power and EM measurements for popular ARM 32-bit processor.

2. The Split Mask Countermeasure

Figure 2 illustrates the general differences between the previously proposed random masking[15] in a) and the proposed split masking countermeasure in b). Note that only the attack on the input of the table, or key xoring is considered. In figure 2a) the 'table look up' has masked input where *masked_table(i^mask)=table(i).* Whenever a new random mask is used, a new table must be regenerated. However in figure 2b) a number of split masks are used. When new masking is required, a new set of split masks are generated, however table regeneration is not required since the exclusive-or of the set of random masks does not change. Let $n1_t$ represent the random mask *1* used at time *t*, where the cryptographic application in figure 2b) can be described as : *output = masked_table(plaintext^*$n1_t$ *^…^*nm_t *^key)*, where *masked_table(plaintext^*$n1_t$ *^…^*nm_t *^key)=table(plaintext^key).* After a new set of masks are generated the cryptographic application can be described as : *output = masked_table(plaintext^*$n1_{t+1}$ *^…^*nm_{t+1} *^key),* where $n1_t \text{ ^…^} nm_t = n1_{t+1} \text{ ^…^} nm_{t+1}$.

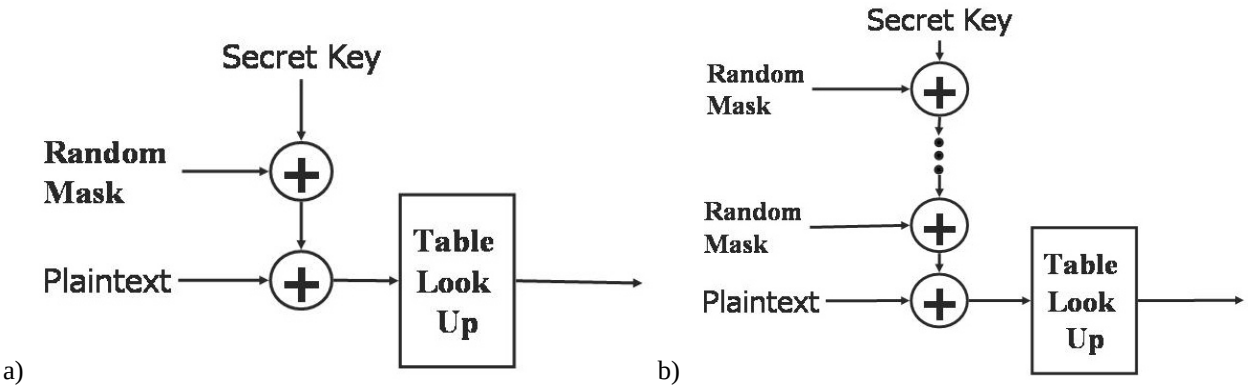


Fig. 2. Previous random masking in a) compared to split mask countermeasure in b).

A 2nd DPA attack [15] on figure 2 a) involves two power samples, specifically a power sample of the random mask and a power sample of the plaintext exclusive-or'd with the key and random mask. In figure 2b) a n^{th} order DPA would involves n power samples, specifically a power sample of each *(n-1)* random masks and a power sample of the plaintext exclusive-or'd with the key and random masks.

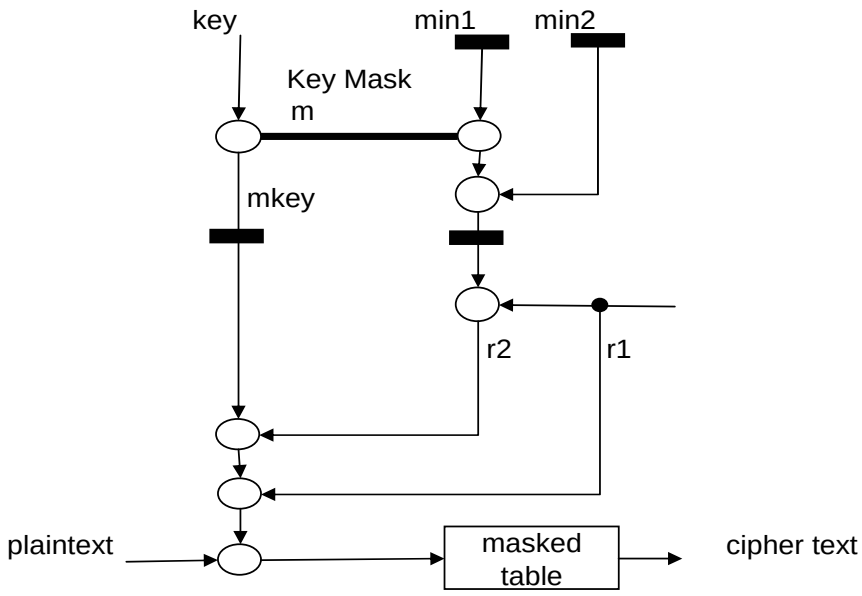


Fig. 3. Key, table, and split masks shown as *m, min1 and min2,* and *r1 and r2.*

The implementation of the split mask countermeasure (using two split masks) is shown in more detail in figure 3 (where circles represent exclusive-or operations). The original secret or master key, *key*, is immediately masked after it is received or derived. Key mask *m* is exclusive-or'd with the master key and the masked key, *mkey*, is stored (the unmasked master key is not stored or ever used again). Stored values are indicated in figure 3 using a black rectangle. The key mask, *m*, is a randomly selected fixed value and it is unchanged for different plaintexts. Key mask *m* is exclusive-or'd with the two table input masks, *min1,min2*, and the result is also stored. The result is exclusive-or'd with *r1*, where *r1* is a random value generated for each new plaintext. The exclusive-or result is defined to be the value *r2* (where mathematically *r2=m^min1^min2^r1)*. The result of the masked key exclusive-or'd with *r1* and *r2* is then exclusive-or'd with the plaintext and input to the masked table. Mathematically the cryptographic application in figure 3 can be described as : *output = masked_table(plaintext^r1^r2^mkey)= masked_table(plaintext^r1^ (m^min1^min2^r1)^ (m^key)) = masked_table(plaintext^min1^min2^key).*

The masks whose names start with a '*m*' (ie. *m, min1,* and *min2*) are generated only once per key. The other masks whose names start with '*r*' (ie. *r1,r2*) are randomly generated with every plaintext. The split masks *r1* and *r2* are exclusive-or'd together to form a final fixed mask for the table input. These two masks can be randomized for each invocation of the cryptographic process. For example, one can generate a random *r1* value and use the computation shown in figure 3) in order to generate a new *r1,r2* pair. However to avoid table regeneration, the final fixed mask for the table input is not changed. The masked table (shown as table look-up in figure 3) is generated from an original unmasked table, *table(i)*, using a two-part mask comprising *min1* and *min2*. The table is defined as : *table(i)=masked_table(i^min1^min2).*

Since the master key is masked and stored immediately after it is derived and during this time *min1* and *min2* are accessed only once per new master key, even a 3rd order DPA (with 3 power samples of *min1, min2* and *plaintext^min1^min2^key*) is difficult to launch since power samples have to be attained immediately after the new key is received. Alternatively a 4th order DPA would involve power samples of *r1, r2, m,* and *r1^r2^m^key^plaintext* which again may be difficult since *m* is only accessed

with each new master key. With each new plaintext and subsequent running of the encryption algorithm alone (for the same master key), an attacker cannot obtain sufficient information to launch a high order DPA since *min1, min2 and m* are never reloaded (unless a new master key is required). Furthermore since *min1^min2* or *r1^r2^m* are never computed on their own, a 2^{nd} order DPA is not possible.

Figure 4 is a further generalization of the split mask countermeasure (using *n* split masks). To provide further security the input table masks are split into *n* masks and there are also *k* random masks used per plaintext. Hence a $(n+1)^{th}$ order DPA would be required in the attack assuming the attacker has access to power samples of all *n* input table masks. In this application, the masked key, *n* input table masks, and the final masked combined value need to be stored. The table does not need to be regenerated.

In all schemes described, it is also possible to employ the proposed split mask countermeasure in conjunction with some previously researched countermeasures, for example where table regeneration is required [5] or where several masked tables [14] are used. The combination of both countermeasures can be used to provide increased security. For example a higher order DPA attack on the random masking countermeasure by extending it with the proposed split mask countermeasure, would be required. For example all masks *r1* through *rk* would be randomly generated (instead of defining *rk* using *r1* through *rk-1*). In this application it would be assumed that the zeroth address of the table regeneration would be protected such that the value *r1^r2^…^rk* is never computed on it's own, otherwise a 2^{nd} order DPA could be launched. For the application of the proposed split mask countermeasure to the *x* stored masked tables, then *x* values of *r^min1^min2* would be stored and the countermeasure would work the same as shown in figure 3 by accessing the proper *r^min1^min2* which corresponds with the current masked table selected.

2.1 Application of Split Mask Countermeasure to Rijndael

An application of the split mask countermeasure will now be presented in detail for Rijndael. In the previous section, the countermeasure was applied before/to the (de)encryption process, however it can also be applied to the key generation process as will be illustrated for Rijndael. In the general application presented in the previous section, the countermeasure provides a initial masking of the master key and during each encryption, only the stored masked key is accessed. However in Rijndael, the key generation algorithm takes as input the master key and produces several other keys, known as round keys, which are then used in the encryption algorithm. When key generation algorithms exist, the countermeasure can be applied before key generation, such that masked round keys are generated, and during each (de) encryption, only the stored masked round keys are accessed. Typically key generation is performed only once per master key, whereas (de)encryption may be performed several times for various plaintexts. Hence by moving the masking to the key generation, the attacker may have a more difficult time obtaining the necessary power samples (hence better security may be attained), since only one power trace is available per master key. Typically in most attacks, many power traces are obtained from encryption runs.

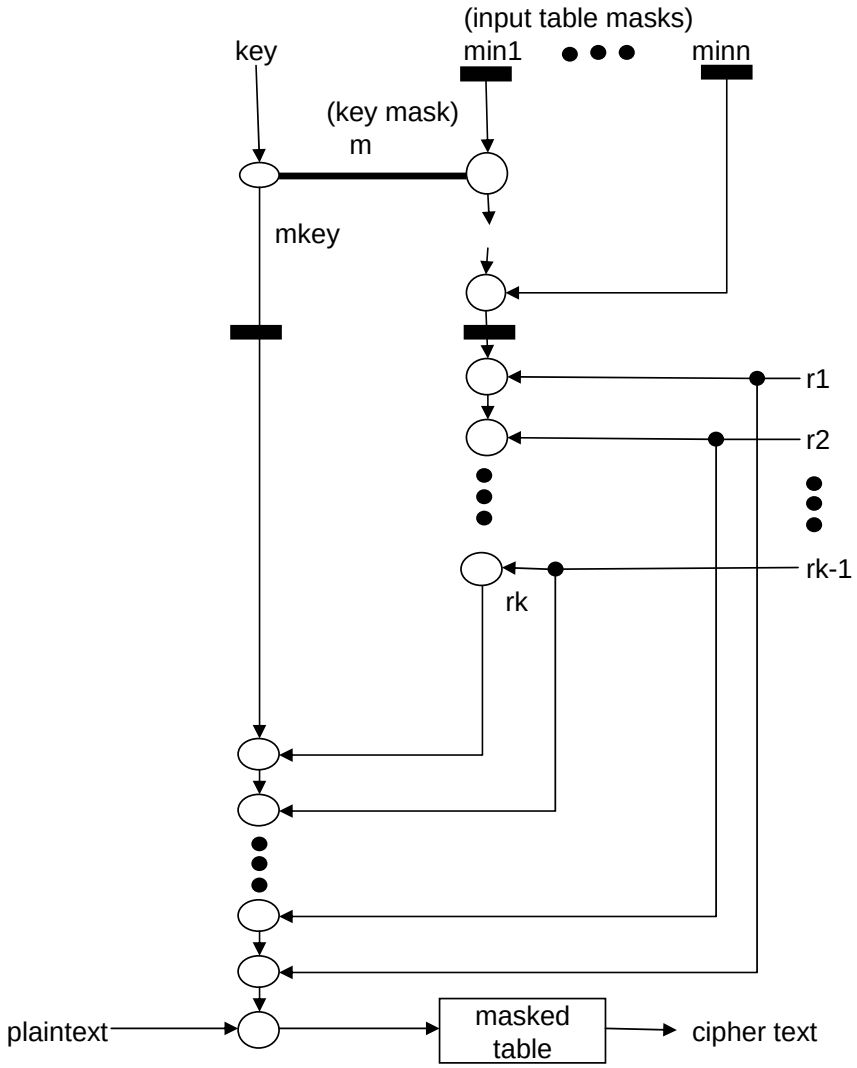


Fig. 4. Key, table, and split masks shown as *m, min1,…, minn,* and *r1,…, rk.*

In Rijndael there is a key generation algorithm which produces 44 32-bit round keys from an initial 128-bit master key. The split mask countermeasure provides split masks to be used in the generation of this set of 44 masked round keys. The initial 128-bit master key is represented in figure 5 as four 32-bit keys, *key0, key1, key2,* and *key3*. The Rijndael key generation algorithm makes use of a function that may be implemented as a table look-up (shown as *mtable0* in figure 5). The table look-up is masked and is defined with respect to the original table in the Rijndael algorithm as : *mtable0(i^M)=table0(i).* For 32-bit processor implementation, the table look up method for encryption [28] requires five 256 by 32-bit tables.

The master key in Rijndael is masked using a key mask comprising four random 32-bit values *n0,n1,n2,n3*. The masked master key is stored (*mkey0, mkey1, mkey2, mkey3*). The masked master key is then exclusive-or'd with random values *r0, r1, r2, r3* to produce *rk0, rk1, rk2, rk3* for input to the Rijndael key generation algorithm. If key generation is invoked many times on the same master key, new random values of *r0, r1, r2, r3* will be generated, and exclusive-or'd with the original stored master key, *mkey0, mkey1, mkey2, mkey3.* The table look-up is shown as a separate step in figure 5 in order to illustrate the use of intermediate masks (*mi()*). The intermediate mask for each round key is given in table 1. In general the mask of the round key is generated from the exclusive-or of other masked round

keys within Rijndael algorithm. The intermediate mask is generated to be the equivalent of the additional masks that, when combined by an exclusive-or with the generated mask for the round key, produces a resultant mask that is equal to the table input mask. The table input mask is *M* and therefore the intermediate mask for each round key value is defined such that the combination of the mask applied to generate the round key value and the intermediate mask will be the exclusive-or product. By applying the intermediate masks defined in this way, the input to the masked table, *mtable0*, will effectively be *M*. However it is not limited to *M*, and further split mask representation for *M* could also be used as in the previous section. The value *r3* is generated as *r3= n0^r0^n1^r1^n2^r2^n3^M.*

As an example, the Rijndael key generation algorithm computes the fourth to seventh round keys using round key 0 to round key 3 using the following algorithm (where *table0 r*epresents a key rotation):

rk4 = table0(rk3) ^ rcon(0) ^ rk0
rk5 = rk1 ^ rk4
rk6 = rk2 ^ rk5
rk7 = rk3 ^ rk6
…
rk40 = table0(rk39) ^ rcon(9) ^ rk36
rk41 = rk37 ^ rk40
rk42 = rk38 ^ rk41
rk43 = rk39 ^ rk42

In the split mask countermeasure, since *rk0* through *rk3* are masked with *n0^r0, n1^r1, n2^r2, n3^r3* respectively, masks will be produced automatically on round keys rk4 through rk7. These masks will be *n0^r0, n0^r0^n1^r1, n0^r0^n1^r1^n2^r2,* and *n0^r0^n1^r1^n2^r2^n3^r3* respectively. These generated masks for round keys are given in the second column of table 1, where for simplicity m_x represents *nx^rx,* and $m_x m_y$ represents *nx^rx^ny^ry, etc*. A pattern of masks on the round keys will emerge according to the Rijndael and the proposed masking produces a pattern which repeats every 16 round keys as shown in table 1. The third column represents the intermediate masks (*mi()* in figure 5) which are exclusive-or'd with the masked round keys to provide a final set of round keys all having a mask of $m_0 m_1 m_2 m_3$ *or n0^r0^n1^r1^n2^r2^n3^r3=M*. As round keys are generated, intermediate masks are loaded and used for any round keys that are defined by a look-up access of *mtable0* (in the example of figure 5, these round keys *rki* are for *i*= 3,7,11,15,19,23,27,31,35,39). Round keys are automatically masked when they are defined with different values to provide a countermeasure for power analysis. As part of AES key generation, *rk3,* for example is further exclusive-or'd with the intermediate mask *mi(3), mi(3)=n0^r0^n1^r1^n2^r2* (see table 1), since it accesses table *mtable0*. The round key *rk4* is then defined as follows: *rk4=mtable0(rk3)^rcon(0)^rk0,* where *rcon(0)* is a constant defined for Rijndael and *mtable0* is the masked table value (*table0(i)=mtable(M^i)*, for mask *M*). All round keys, except those which were input to *mtable0* (which are already fully masked), are exclusive-or'd with their intermediate masks according to table 1. These masked round keys are then stored and available to the encryption algorithm. Each round key has a final mask of *M*, the input table mask. The use of split or multiple masks in the masking that was carried out makes the Rijndael key generation more secure from DPA attacks.

The Rijndael encryption steps shown in figure 6 make use of the masked round keys in conjunction with a set of defined mask tables to encrypt plaintext data. The set of masked tables that are used with masked round keys are defined to have an input mask with a value of *n0^r0^n1^r1^n2^r2^n3^r3=M.* As shown in figure 6, the plaintext is exclusive-or'd with masked round keys in accordance with the Rijndael process. The result is used as input for the appropriate table look-up in masked tables. The result of the Rijndael encryption process carried out using masked round keys, plaintext and masked tables is a set of four values that are shown as *s0,s1,s2,s3* in figure 6. The output

values *s0,s1,s2,s3* are masked . To increase the security of Rijndael encryption, the output values are unmasked in a two-step process. Initially, the output values are each exclusive-or'd with the value *n0^r0^n1^r1* (an intermediate mask, *mi(15)*). A second exclusive-or is then carried out on the result, using the value *n2^r2^n3^r3* (an intermediate mask, *mi(13)*). The split mask approach which provides for increased security for key generation in the Rijndael process and the encryption step, using the masked round keys, is itself made more secure. The unmasking step, carried out after the masked encryption tables have been accessed, is done using what is effectively a split mask, adding to the security of the encryption of the plaintext.

In Rijndael, the split mask countermeasure requires additional storage of nine unique intermediate masks, and the mask for generating *r3* (ie. *n0^n1^n2^n3^M^r0^r1^r2*, the rectangle in figure 5 shown above the first exclusive-or of *r0*). The number of tables required for storage does not increase over the normal Rijndael algorithm, unlike [14], since only one set of masked tables needs to be generated before the algorithm is downloaded to the portable device. After the code is downloaded to the device the tables never need to be regenerated unlike [5]. For each new master key, five exclusive-or operations are required in order to generate the masked master key and *r3* generating mask. For each new set of random *r0-r2* values , the key expansion performs an additional 47 exclusive-or operations (the three exclusive-ors, or xors, with *r1-r3* and one per masked round key). In addition to this eight exclusive-or operations are required to generate the nine intermediate masks (*mi()*).

TABLE 1. Key Generation Intermediate Masking.

Round key (rk#)	**Generated Mask of round key**	**Intermediate mask, mi(#)**
0, 16, 32	m_0	m_1 m_2 m_3 *
1, 17, 33	m_1	m_0 m_2 m_3 *
2, 18, 34	m_2	m_0 m_1 m_3 *
3, 19, 35	m_3	m_0 m_1 m_2 *
4, 20, 36	m_0	m_1 m_2 m_3 *
5, 21, 37	m_0 m_1	m_2 m_3 *
6, 22, 38	m_0 m_1 m_2	m_3 *
7, 23, 39	m_0 m_1 m_2 m_3	-
8, 24, 40	m_0	m_1 m_2 m_3 *
9, 25, 41	m_1	m_0 m_2 m_3 *
10, 26, 42	m_0 m_2	m_1 m_3
11, 27, 43	m_1 m_3	m_0 m_2
12, 28	m_0	m_1 m_2 m_3 *
13, 29	m_0 m_1	m_2 m_3*
14, 30	m_1 m_2	m_0 m_3 *
15, 31	m_2 m_3	m_0 m_1*
		* updated with ^w1and/or ^w2

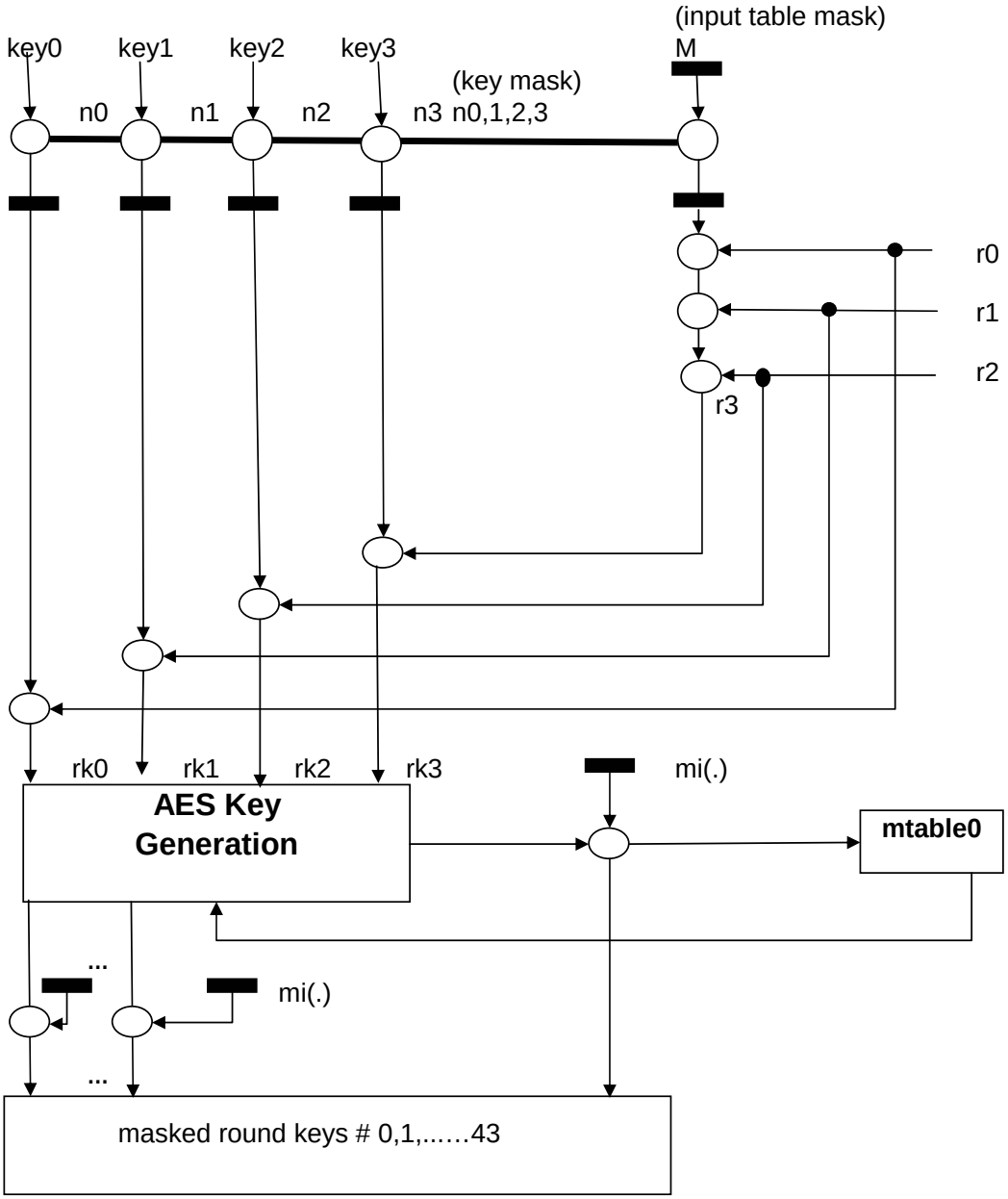


Figure 5. The split mask countermeasure applied to Rijndeal key generation.

The proposed split mask countermeasure requires a 3rd order DPA to attack the key xoring during encryption. For example an attacker may be able to obtain a power sample of m_0m_1 *(mi(15))* and m_2m_3 *(mi(13))* to be used along with the power sample of *rk0^m0^m1^m2m3^plaintext* over many power traces and use joint conditional probabilities (as will be derived in Appendix A) to predict the value of the key *rk0*. A 1st order DPA is not possible since only the masked key can be obtained using this attack and it is assumed that the attacker does not have access to the mask value *M*. A 2nd order DPA is only possible if during the master key processing a power sample of *M* is obtained. Since this master key processing is only performed once immediately after the master key is received, it is highly unlikely that

this power sample could be obtained. However *M* could also be split into multiple masks whose exclusive-or'd value is never computed. A 3rd order DPA may be possible if the user can obtain proper power samples during key generation or encryption. For example the attacker would have to know where to obtain a power sample of *m0m1* and *m2m3* (the mi()'s in figure 5), either during key generation or during the last part of the encryption process (last two xor's before the cipher output in figure 6). To further increase the security, the masks could be randomized during each invocation of the encryption process. For example new values of *r0,r1,r2* could be randomly generated before each encryption and used to compute *r3* and the new intermediate masks. For example, the masks can be changed (ie. *ra rb rc rd* in place of *r0 r1 r2 r3*) as long as the intermediate masks are also recomputed and the final mask is the same (*ra^rb^rc^rd = r0^r1^r2^r3*), so that the masked tables do not need to be regenerated. Alternatively *r0* and *r2* could be randomized through exclusive-or'ing with a random value, *w1*, and *r1* and *r3* randomized with *w2* (ie. given random values *w1,w2, r0^=w1, r2^=w1, r1^=w2, r3^=w2*). For example the two intermediate masks , *mi(13)* and *mi(15)*, would be updated as *mi(15)^w1^w2* and *mi(13)^w1^w2*. The intermediate masks shown with a * in table 1 indicates that during randomization of the masks they must be exclusive-or'd with *w1 or w2*. Using this scheme the masked tables again do not have to be regenerated. Using this fixed masking scheme, the final mask on input to the table is never computed (hence an attacker can never get the power sample of this mask). The table *mtable0()* has an input mask of *m0^m1^m2^m3* and no output mask for this scheme in figure 1. However other masking schemes are also possible including masking of the table output and adding additional exclusive-or'ing before the cipher output is produced, both requiring higher order DPA.

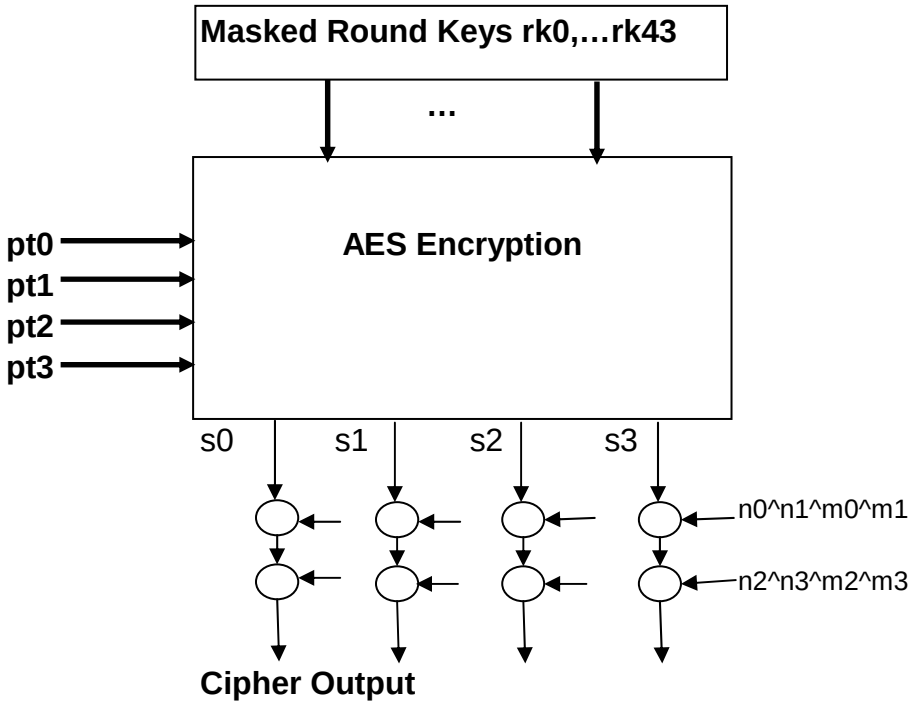


Figure 6. The split mask countermeasure applied to Rijndeal encryption.

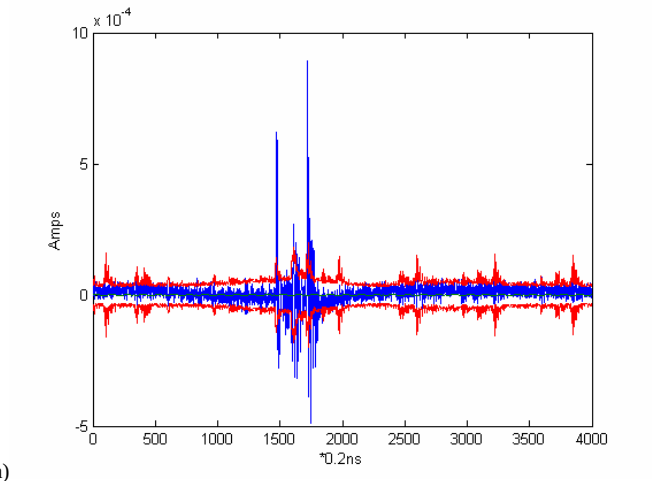
In Rijndael the split mask countermeasure could be used in conjunction with other countermeasures as well. For example to be used with the previous countermeasure which stores a number of masked Sbox tables[14], a different value of *M^n0^n1^n2^n3* would be accessed depending

upon which set of masked Sbox tables was going to be used. This would be the only change to the split mask countermeasure for use with the countermeasure in [14]. Additionally the random masking countermeasure which regenerates tables[5,15], could be used in conjunction with the split mask countermeasure as well. In this case *r3* would not be generated as shown in figure 5 from *r0,r1,r2*, and *M^n0^n1^n2^n3*. Instead *r3* would be a completely new random value similar to *r0,r1,r2*. No other changes would be necessary to the implementation of the split mask countermeasure with the countermeasure in [5,15].

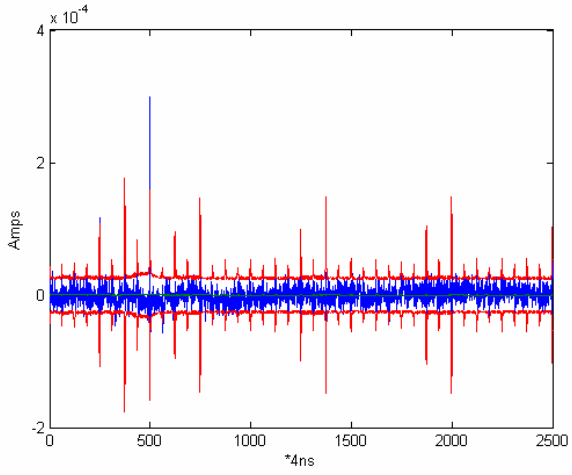
The next section will present the experimental setup and the experimental results using real power and EM measurements to quantify security and energy dissipation. Analysis of possible hamming weight attacks on the round key fixed masking (see Appendix C) is also presented.

3. Experimental Setup

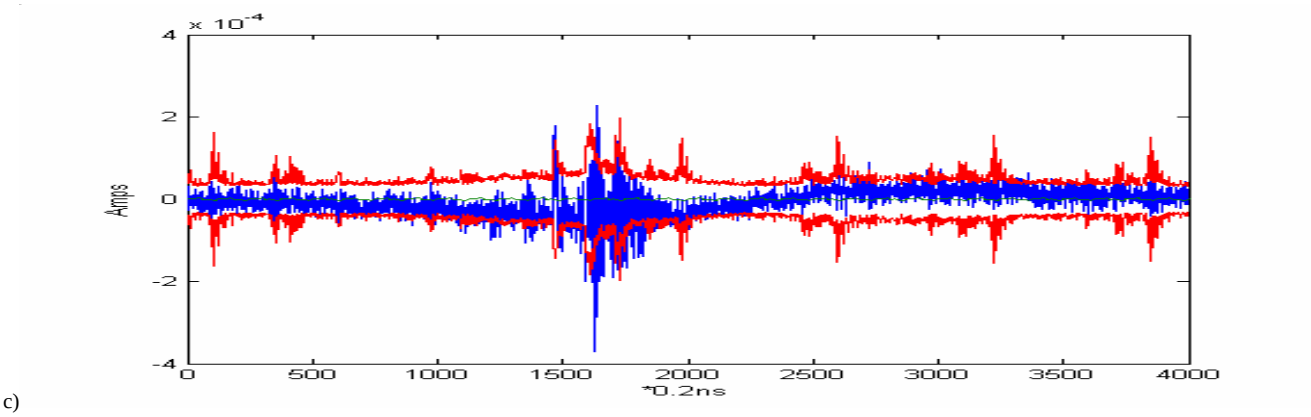
A high sample rate oscilloscope, a trigger probe, an EM probe, an inductive probe and an ARM7TDMI evaluation board (providing access to the core's power supply) were used to acquire power and EM traces. The inductive probe measured the instantaneous current drawn on the 3.3V ARM7TDMI processor core power supply line. In this paper we will refer to the processor current as the power consumed (since the supply voltage was assumed to be stable at 3.3V). The EM probe (consisting of a 1cm loop passive magnetic field probe) was placed on top of the ARM7TDMI packaged chip. A broadband preamplifier was also used to boost the signal to noise ratio of the EM signals. The trigger signal was controlled by software and measured with a probe in order to synchronize the measurements. The scope sampled at rates up to 2.5Gsamples/sec, and allowed many power and EM traces to be acquired automatically. The evaluation board had the 16/32-bit ARM7TDMI RISC processor core on one chip separate from the memory. Hence all the power measurements reflect the processor core power consumption only and not the input/output buffer power



a)



b)



c)

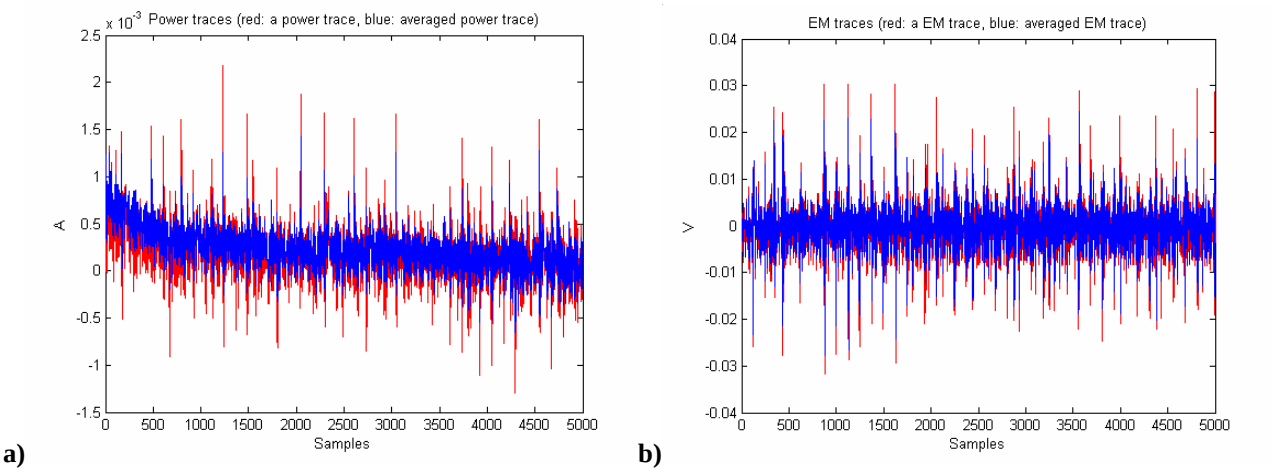
Figure 7. DPA of bit 0 at 40MHz, 2MHz and bit 9 at 40MHz, in a)b)c) respectively.

or memory power. The ARM7TDMI could be set to different clock frequencies (up to 56MHz) and utilized a three stage pipeline. This ARM processor core is often referred to as a low power processor consuming on average 0.6mA/MHz at 3V.

Each scope run acquired up to 6000 power or EM traces (referred to as a frameset) and acquisitions were repeated three times to ensure high reliability and reproducibility in the results. The ARM7TDMI processor was programmed to use a 40MHz clock for all experiments. All results were repeatable and where possible the same set of plaintexts (or input data to the cryptographic algorithm) was used to compare results of different DPAs/DEMAs and countermeasures. All DPAs/DEMAs were verified as significant using two standard deviations.

To acquire the multiple samples in each power/EM trace for the 2nd and 3rd order DPA/DEMA experiments, the cryptographic programs were first characterized. Specifically DPAs/DEMAs were run to identify the best power sample points within the specific load and stores within the cryptographic program. Figures 7a) and c) illustrate two DPAs on different bits for the 32-bit processor (where the red lines indicate +/- two standard deviations of the difference of means). The highest point of the positive DPA characteristic was used to identify the sample point in the power trace which would be used as the power samples ($b1_k$, $b2_k$, c_k from Appendix A,B) in the high order DPAs. Each frameset of power/EM traces acquired from the scope was characterized wth DPAs to determine the best sample point. Those sample points were then fixed for all experiments utilizing that cryptographic program. The next section will present the DPA and statistic results utilizing real power measurements obtained with the ARM7TDMI and high sampling rate scope. Unlike [10] no demodulation of the EM signals were performed.

Figure 8 illustrates the noise in the power and EM trace acquisitions. The averaged power/EM trace (averaged over 1487 traces) is shown in blue, whereas one power/EM trace is shown in red. It is interesting to note that there is slightly less noise in the EM acquisition than in the power.



a) b)

Figure 8. Power and averaged power trace in a) and EM and averaged EM trace in b) respectively.

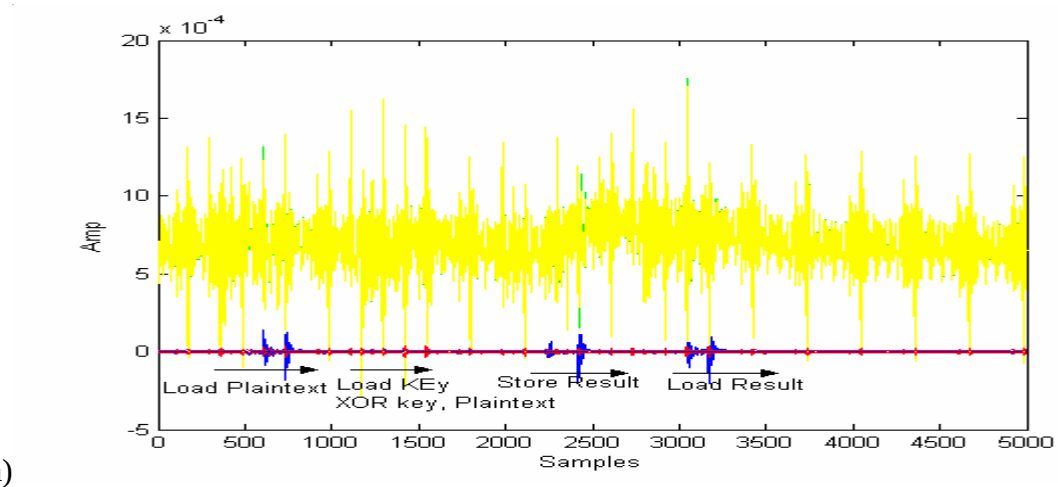
4. Experimental Results

This section will illustrate results of the proposed split mask countermeasure and high order DPA/DEMA attacks using both real power/EM measurements. The assumptions made to support a higher order DPA/DEMA derived in Appendix A and Appendix B were also evaluated with real power measurements. The statistics derived in Appendix A and B in conjunction with the new low energy countermeasure are used with both real power and EM measurements. All probabilities referred to in this section are actually estimated probabilities using the statistics from Appendix A and B.

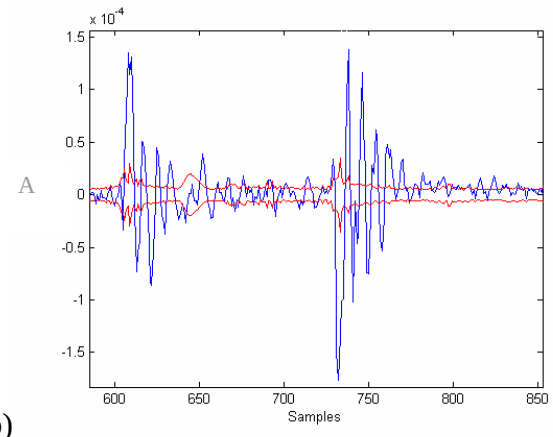
Figure 7 illustrates a DPA attack on the load data instruction on from the ARM7TDMI. Random data was loaded from memory using the same memory address and surrounded by no-instruction (or nop) instructions. Over 2000 power traces were acquired for each run of the experiment. These power traces were split into two groups according to the value of bit *i* of the 32-bit random words. The difference of means was computed and is illustrated in the figures along with plus or minus two standard deviations (shown in red). The power analysis was performed at 40MHz, in figure 7a), as well as 2MHz, in figure 7b), both for the same bit 0. Figure 7c) illustrates the DPA at 40MHz, for bit 9. The DPA peak was negative in this case for bit 9 compared to the generally higher positive peaks for bit 0 at 40MHz in a) and bit 0 at 2MHz in b). The sample resolution at 40MHz was scaled down proportionately for the 2MHz case (so for one clock cycle, the same number of samples was captured). The DPA in figure 7b) is much narrower. The DPA peaks at this sample resolution for 2MHz processor clock were often missed due to the need for both higher resolution and wider time frame required due to the slower clock speed. The narrow or missed DPA peak occurs because the processor's logic delays do not change, thus high sampling rate is still required to see the DPA even using a lower clock frequency for the processor.

The previously researched bus complement and power randomization countermeasures [11,13] were found to be very ineffective on the ARM7TDMI processor. In both cases the DPAs were still evident. For example in the bus complement experiment 16-bit data values, $(d)_{15..0}$, were utilized on the 32-bit bus in different arrangements including $(d)_{0..15}||(d)_{15..0}$. The power randomization technique loaded random 32-bit data before and after the sensitive data (such as key value). Although the countermeasures were successful in reducing the DPA peaks by up to 60%, the DPAs were still significant (or greater than two standard deviations).

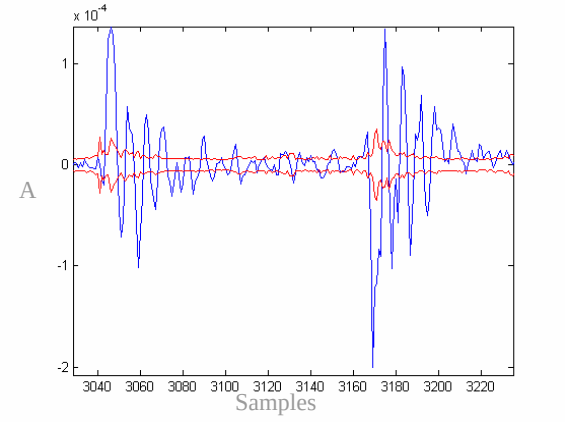
The key xoring DPA attack is illustrated with real power and EM measurements in figure 9. In the ARM7TDMI processor, although both a DPA and DEMA attack on the xor operation was not successful (likely due to the lower average power dissipation of a xor instruction compared to the load or store instruction), DPA and DEMA attacks on all load and store instructions were very successful. Hence to demonstrate a key xor DPA/DEMA attack, the xor result was stored to memory and loaded back again. The whole DPA trace and power traces for the sequence of loading plaintext, (loading key and) computing the xor (of the key and the plaintext), storing the xor and loading the xor is shown in figure 9a). The dark blue (and red) line is the differential (and two times standard deviation) trace and the larger yellow (and hidden green) traces are the two averaged power traces. The nop instructions were used to separate the instructions for clarity. Figure 9b) illustrates a close up view of the DPA of the plaintext load. Figure 9c) and 9d) illustrate the



a)



b)



c)

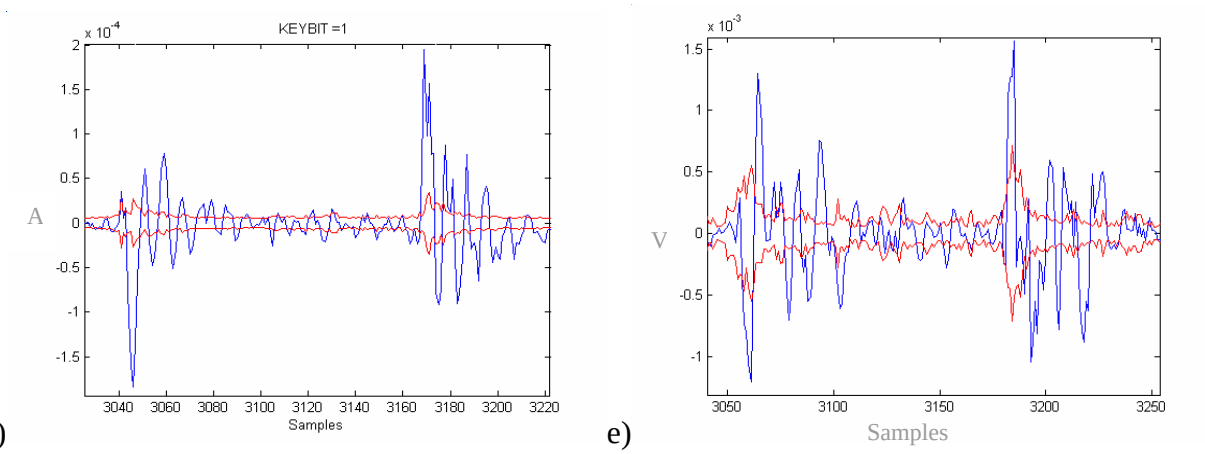


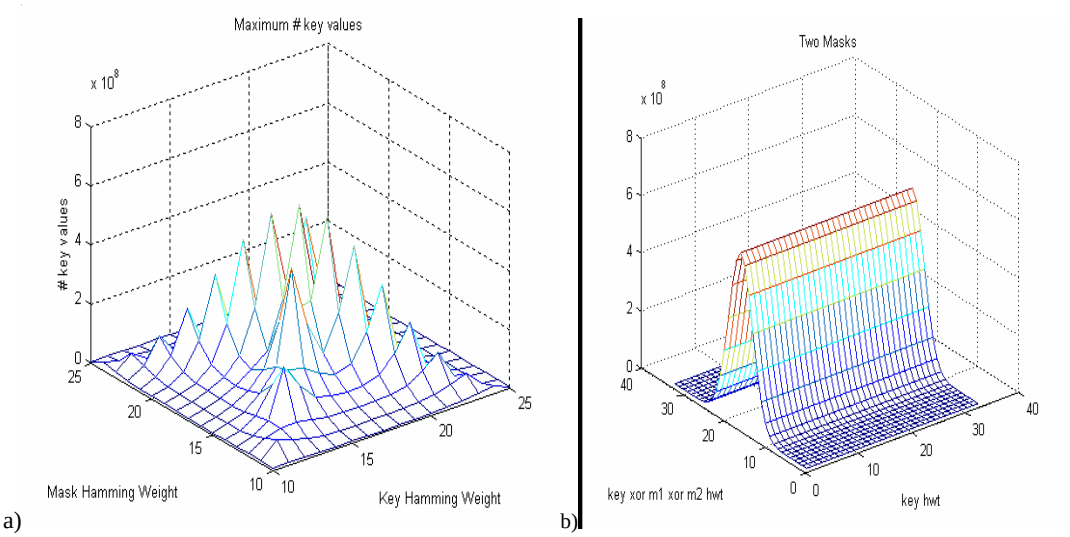
Figure 9. Key xoring DPAs and power traces in a), DPA of load plaintext b), DPA of xor result in c) and d) for keybit of 0 and 1 respectively, and DEMA of xor with keybit of 1 in e).

DPA of the xor result for a key bit of 0 and 1 respectively. The first spike on loading plaintext is clearly a positive rise which is also apparent for the key bit=0 load. The first spike for keybit=1 is a negative rise as expected. It is interesting to note that as opposed to a clearly positive or negative DPA characteristic, we see both positive and negative dual spikes. The dual peaks appear likely due to the capacitance of the processor causing the effect to last over several clock cycles. Hence the attacker can examine the DPAs to see if they initially rise similar to the plaintext load (indicating a key bit of zero) or if they rise in opposite direction to the plaintext load (indicating a key bit of one). The EM result, DEMA, in Figure 9e) is very similar to the DPA in figure 9d) as expected.

The energy dissipation overhead and memory overhead of the split mask countermeasure was also evaluated and compared to previous research overheads. Using the current meter connected to the core power of the ARM7TDMI chip, real averaged power dissipation measurements were made. The energy dissipation of Rijndael key generation for a 128-bit key with no countermeasures on the ARM7TDMI is 0.354uJ. With the proposed split mask countermeasure, where the masked round keys are updated with intermediate masks during key generation, the total energy is 0.509uJ per master key (13% higher energy dissipation than without any countermeasure for key expansion alone). In this scheme the AES encryption is run with no additional operations and does not require any further memory storage (since only one set of masked tables is required). In this scheme all round keys will have the same fixed mask. To increase security of the encryption algorithm and by using different random values for *r0-r2*, the key generation and encryption could be run together on each set of plaintexts. This would require 16% more energy dissipation (0.572mJ per 8KB of plaintext) compared to key generation and encryption without any countermeasures. There is a 20% overhead in energy dissipation if masked round keys are exclusive-or'd with *mi()* during the encryption process compared to regular unmasked encryption. The previously researched random masking technique [15], where the Sbox tables are regenerated for each new random mask, required 8 times overhead in energy dissipation for the unmasked encryption algorithm alone using the same ARM7TDMI processor. In [14] several sets of tables have to be stored for each new mask used with the round keys, where each set has 5 tables, each of 256x32-bit size using the table method in [28].

Hamming weight attacks (for a non-randomized implementation) were also analyzed on the 32-bit processor. Although our real power measurements were unable to correlate the power to hamming weights, assuming it may be possible with other processors, the security was evaluated. Figure 10a) illustrates the number of possible key guesses for a single mask case where three hamming weights are

available, specifically the hamming weights of the key, the mask and the exclusive-or result, *key^mask*. This illustrates an attack possible using the previously researched countermeasure illustrated in figure 2a). In the proposed split mask countermeasure case, the number of key guesses possible given the four hamming weights, that of the first mask (*mA*), the second mask (*mB*), the key, and the exclusive-or result, (*mA^mB^key*) is illustrated in figure 10b). For example to illustrate this attack consider the Rijndael implementation in section 2, where round key *rk0* has a final fixed mask of *m0m1m2m3*. Hence we assume the attacker may be able to calculate the four hamming weights from the four power samples of *rk0, mi(15) (* where *mi(15)=m0m1* from table 1, to represent *mA*), *mi(13) (*where *mi(13)= m2m3* from table 1, to represent *mB*), and the final masked round key. In both cases the maximum number of possible solutions is 601 million solutions (for a number of hamming weight combinations). The average number and the sum of the number of key guesses increased by 8 and 280 times respectively in b) over a). Hence there are more combinations of masks which require larger number of key guesses for the split mask countermeasure compared to the single mask case. Appendix C derives the explicit equations used to obtain these results.



a) b)

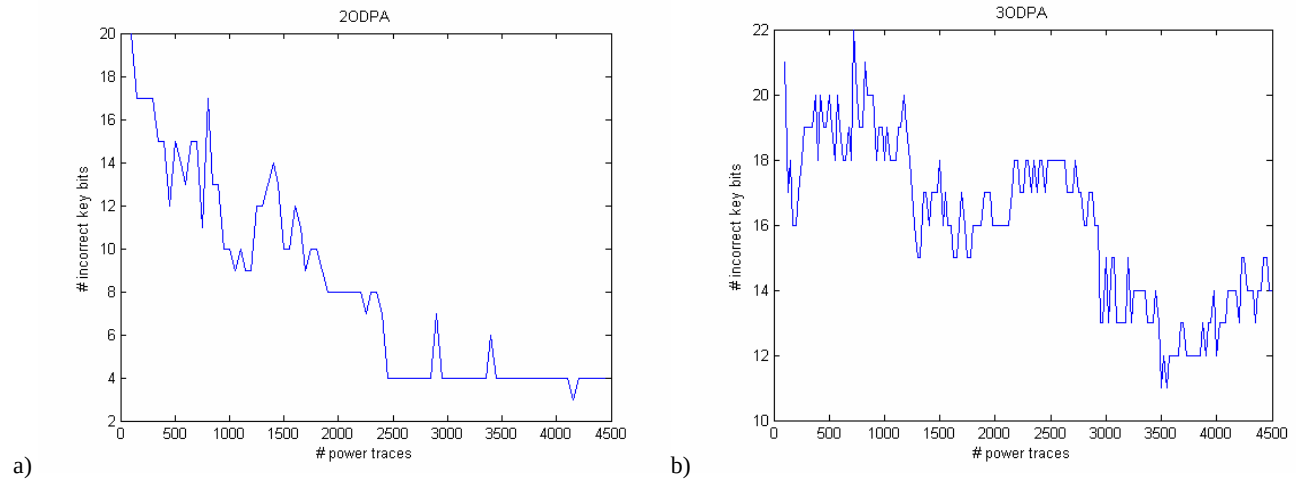
Figure 10. Number of key guesses for one a), and two split masks b).

In some power trace acquisitions, the means and standard deviations assumptions made after normalization and detailed in Appendix A and Appendix B, (for example $f_{b1}^{-} = f_{b1}(b1_k \mid r1_i = 0) \approx N(-\frac{\varepsilon}{2}, \sigma)$ and $f_{b1}^{-} = f_{b1}(b1_k \mid r1_i = 1) \approx N(+\frac{\varepsilon}{2}, \sigma)$) did not always hold. Although in most cases they were found to be valid, we did obtain acquisitions such as: $f_{b1}^{-} = f_{b1}(b1_k \mid r1_i = 0) = N(-0.037, 0.99)$, $f_{b1}^{-} = f_{b1}(b1_k \mid r1_i = 1) = N(+0.038, 1.00)$ and $f_c^{-} = f_c(c_k \mid mres_i = 0) = N(0.0577, 0.99)$, $f_c^{-} = f_c(c_k \mid mres_i = 0) = N(-0.055, 1.00)$ for a power acquisition of 4000 power traces.

To illustrate the increased difficulty of mounting a 3rd order DPA/DEMA attack , as possible in the proposed split mask countermeasure, results were compared to a 2nd order DPA/DEMA attack (such as proposed with the random masking countermeasure in [15]). The number of incorrect key bit guesses versus the number of traces was plotted in figures 11 and 12 for power and EM signals respectively. The same ARM7TDMI processor and experimental setup (including sample rate) was used in both

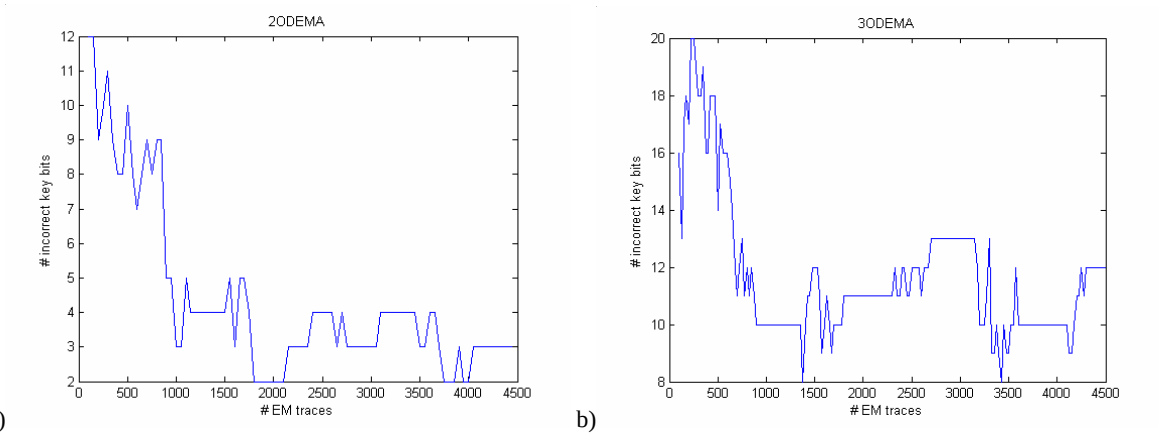
cases. The statistics Λ_{key_i} from Appendix A and Appendix B were used to predict the key bits for the 3rd order and 2nd order analysis respectively. In the 2nd order differential analysis, power and EM samples of the 32-bit random data (*r*), and the 32-bit exclusive-or result (*r^key^plaintext*) were used. In the 3rd order differential analysis the power and EM samples of the 32-bit random data (*r1*), the second 32-bit random data (*r2*), and the 32-bit exclusive-or result (*r1^r2^key^plaintext)* were used. It was assumed that the split masks were randomized as *r1^=r, r2^=r,* where *r is* a random value, as discussed in section 2.1. For example in section 2, the round key *rk0* has a final fixed mask of *mi(13)^mi(15),* and through randomizing the split masks, the split masks are updated as *mi(15)^=r, mi(13)^=r, where r=w1^w2 (*as indicated by * in table 1). To show how the number of incorrect key bits decreased over time, all 32 key bits were predicted from an initial 100 power and EM traces using the statistic Λ_{key_i}. Then after every 25 power/EM traces another prediction of all 32 key bits was calculated using the same statistic but using all cumulative power/EM traces.

The third order DPA results (using the split mask countermeasure), figure 11b) was compared to a 2nd order DPA (using random masking technique as described in [15]), figure 11a). The DEMA results are shown in figure 12, for a second order and third order DEMA in a) b) respectively. The same set of plaintexts, and key, were used for both sets of power and EM traces and 2nd and 3rd order analysis. For each power and EM figure, 9000 power traces and 9000 EM traces were acquired using several framesets (or scope acquisitions). The x-axis refers to the total number of power or EM traces in a group, where the traces are partitioned into two groups according to the plaintext bit (as detailed in Appendix A and B). For example 4500 on the x-axis refers to 9000 (x-axis value multiplied by two) total traces used in the prediction of the number of incorrect key bits. Clearly the 2nd order DPA and DEMA results converge much faster than the 3rd order DPA and DEMA results. It is also interesting to note that the DEMA results converge much faster than the DPA results for the same order differential analysis. This could be due to the preamplifier used helping to improve signal to noise ratios as evident in figure 8. After 9000 power traces, the 3rd order analysis still predicts 14 key bits incorrectly, whereas the 2nd order analysis predicts 4 key bits incorrectly. After 9000 EM traces, the 3rd order analysis predicts 12 key bits incorrectly, whereas the 2nd order analysis predicts 3 key bits incorrectly.



a) b)

Figure 11. In a) The 2nd order DPA, and b) the 3rd order (split mask) DPA, showing number of incorrect key bits versus number of real power traces.



a) b)

Figure 12. In a) The 2nd order DEMA, and b) the 3rd order (split mask) DEMA, showing number of incorrect key bits versus number of real EM traces.

5. Discussion and Conclusions

Employing masking before and/or within key generation provides more complexity for the attacker in finding where the masks are loaded in the power trace of the key generation and/or encryption, in order to launch a high order DPA or DEMA attack. However it is also possible to perform the split mask exclusive-oring explicitly at the beginning of the Rijndael encryption algorithm. This alternative approach requires more exclusive-or computations than the proposed split mask countermeasure approach described in section 2.1. The split mask countermeasure is similar to the secret splitting method [18] in that a higher order differential analysis is required, however unlike [18] the key is not split into shares. Hamming weight attacks as described in [23] which do consider attacks on the key generation stage for Rijndael are also made more difficult using this split masking technique since unlike other countermeasures, most round keys (all but the initial secret master key) are automatically masked upon definition and hence the hamming weight of each round key is never available nor is the round key ever unmasked during the key generation or encryption stages. Other countermeasures [25] only consider the encryption algorithm and not the key generation stage. Whereas other studies of DPA have been researched for the Sbox or table look-up attacks[26], and do not consider attacks on the key generation stage. Although the split mask countermeasure was demonstrated for Rijndael, it can easily be applied in general to other encryption algorithms including key generation of other ciphers such as DES. The attack at the output of the Sbox or table look-up was not addressed in this paper, however this split mask countermeasure could be used in conjunction with other countermeasures which address the Sbox attack such as [14,5,15] as described in section 2. Previously these countermeasures [5,15] could be defeated with a 2nd order DPA or DEMA, however in conjunction with the proposed split mask countermeasure a higher 3rd order DPA or DEMA would be required, making the attack more difficult to mount. Since the DPA/DEMA attacks using this embedded processor were only possible during load or stores similar to research findings in [14], it may be more secure to exclusive-or the masked round keys with their intermediate masks during encryption (which typically is executed more frequently than key expansion), to avoid loading all round keys with the same fixed masked value. In this way the hamming weight of the final masked round keys would not be made available for a possible SPA attack. As evident from the experimental results in section 4, the larger number of traces required for the 3rd order DPA or DEMA indicates that the proposed split mask countermeasure increases the security. The faster convergence of 2nd order DEMA compared to DPA as demonstrated in section 4, is also supported by

previous research findings on a 8-bit processor [10]. Results with real power and EM measurements for the ARM7TDMI processor core showed that finding all bits of the secret key required more than 9000 power traces (unlike previous research which investigated only a few bits on an 8-bit processor[10,15]). Furthermore results showed that the DPA assumptions (for example $f_c^+ \approx N(+\frac{\varepsilon}{2}, \sigma)$) made in Appendix A and B as well as in [15] did not always hold.

This paper presented for the first time 3rd order key xoring DPA and DEMA results for a 32-bit low power embedded processor using real power and EM measurements with a new split mask countermeasure. This is unlike previous research which has investigated only 2nd order DPA [15] or 2nd order DEMA [10], both on 8-bit processors, or previous research which investigated theoretical variants of 2nd order analysis [30] without real measurements. Results of the new 3rd order DPA/DEMA statistics on the new proposed split mask countermeasure found that a larger number of power and EM traces are required, making the split mask countermeasure very effective. The countermeasure requires negligible additional energy dissipation compared to previous countermeasures [5,15] and smaller storage overhead compared to [14]. Hamming weight attacks are still difficult due to the split masks. The split mask countermeasure can trade off memory for security, thus supporting a higher n^{th} order DPA with additional memory for storing the additional split masks. This research is crucial for supporting low energy security for embedded systems which will be prevalent in wireless IP-enabled devices designed with nanometer technologies of the future.

REFERENCES

[1] P.Kocher "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems", LNCS 1109, 1996.

[2] P.Kocher, J.Jaffe, B.Jun "Differential Power Analysis" Crypto'99, LNCS 1666, 1999

[3] J.Coron, P.Kocher, D.Naccache "Statistics and secret leakage" LNCS 1962, 2001, pp.157-173.

[4] T.Messerges, etal. "Investigations of Power analysis attacks on Smartcards" USENIX workshop on Smartcard Technology, 1999.

[5] T.Messerges, "Securing the Rijndael finalists against power analysis attacks" LNCS 1978, 2001, pp.150-164.

[6] H.Saputra, etal. "Masking the energy behavior of DES encryption", Proceedings of DATE 2003.

[7] D.Agrawal et al. "The EM side-channel(s)" CHES 2002, 2002, pp.29-45.

[8] E.Biham, A.Shamir "Power analysis of the key scheduling of the Rijndael candidates", 2nd Rijndael conference, 1999.

[9] S.Chari, etal. "Towards sound approaches to counteract power-analysis attacks", LNCS 1666, 1999, pp.398-412.

[10] D.Agrawal, etal. "The EM side-channel…methodologies" at http ://www.research.ibm.com/intsec/emf.html

[11] J.Daemen, V.Rijmen "Resistance against implementation attacks", 2nd Rijndael conference, 1999.

[12] M.Akkar, etal. "Power analysis, what is now possible…", LNCS 1976, 2000, pp489-502.

[13] S.Chari etal. "A cautionary note regarding evaluation of Rijndael candidates on smart-cards", 2nd Encryptn. Std. Cand. Conf, March 1999.

[14] K.Itoh M.Takenaka, N.Torii "DPA countermeasure based on the masking method", LNCS 2288, 2002,pp.440-456.

[15] T.Messerges "Using 2nd Order Power Analysis to attack DPA resistant software", LNCS 1965, 2000 pp.238-251.

[16] M.Aydos, T.Yanik, C.K.Koc "An high speed ECC-based wireless authentication protocol on an ARM Microprocessor", Proc of 16th Annual Comp. Sec. Appl. Conf. 2000.

[17] G.Hachez, F.Koeune, J.-J. Quisquater "cRijndaelar results: Implementation of four Rijndael candidates on two smart cards", 2nd Rijndael conference, 1999.

[18] L.Goubin, J.Patarin "DES and Differential power analysis- the duplication method" CHES 2001.

[19] J.Golic "Multiplicative Masking and power analysis of Rijndael", CHES 2002.

[20] J.Daemen M.Peeters, G.V.Assche "Bitslice ciphers and power analysis attacks" Proc of Fast software encryption workshop, 2000.

[21] J.Coron, L.Goubin "On Boolean and arithmetic masking against differential power analysis", CHES 2000.

[22] J.Coron “Resistance against differential power analysis for ECC” CHES 1999.

[23] S.Mangaard, “A Simple Power-Analysis Attack on Implementations of the AES Key Expansion”, ICICS 2002 , LNCS 2587, pp.343-358, 2003.

[25] E.Trichina, L.Korkishko “Secure and efficient AES software implementations for smart cards”, Cryptology ePrint Archive, 2004/149 http://eprint.iacr.org/2004/149.pdf , 2004.

[26] E.Brier, C.Clavier, F.Olivier “Correlation power analysis with a leakage model” CHES 2004, LNCS 3156, pp16-29, 2004.

[27] J.Daemen, V.Rijmen “AES Proposal: Rijndael” http://csrc.nist.gov/encryption/aes, 1999.

[28] B. Gladman, “A Specification for Rijndael, the AES Algorithm”, at fp.gladman.plus.com /cryptography_technology /rijndael /aes.spec.311.pdf, 15 April 2003 (pages 18-19).

[29] K.Gandolfi etal. “Electromagnetic analysis: concrete results” , CHES 2001, LNCS 2162, pp.251-261, 2001.

[30] J.Waddle, D.Wagner “Towards efficient second-order power analysis” CHES 2004, LNCS 3156, pp1-15, 2004.

Appendix A: Derivation of 3rd order DPA Statistic

Let the masked round key, and two masks be represented by *mres*, and *r1, r2* respectively. In our example of table 1, *r1=m1m3* (assumes the attacker can get a power sample of accessing mi(10) of table 1 or during unmasking of cipher text), *r2=m0m2*, (assumes the attacker has complete knowledge of the algorithm and can get a power sample of accessing *mi(11)* of table 1 or during unmasking of cipher text), and the attacker is trying to determine *rki*, *mres* (which has a resultant mask of *m0m1m2m3*). For example consider the following lines of code:

Line 1. load *r1*

Line 2. *key1 = key ^r1*

Line 3. load *r2*

Line 4. *key2 = key1^r2*

Line 5. load *p*

Line 6. *mres = key2^p=key^r1^r2^p*

Let the k[th] normalized power consumption of the random mask *r1 (line 1.)*, the random mask *r2 (line 3.)*, and the masked round key *mres (line 6.)*, be represented by $b1_k$, $b2_k$ and c_k respectively. The distribution for these values is assumed to be Gaussian with mean of zero and standard deviation of one (due to normalization, $N(\mu,\sigma) = N(0,1)$) $f_{b1}(b1_k) \approx N(0,1)$, $f_{b2}(b2_k) \approx N(0,1)$, and $f_c(c_k) \approx N(0,1)$.

Let $f_{b1}^{-} = f_{b1}(b1_k \mid r1_i = 0) \approx N(-\frac{\varepsilon}{2},\sigma)$ represent the distribution of the power consumption values of $b1_k$ such that the i[th] bit of *r1* is zero. Assume it is also a Gaussian distribution. Similarly let the following distributions of power consumption exist (similar to [15]) $f_{b2}^{-} = f_{b2}(b2_k \mid r2_i = 0) \approx N(-\frac{\varepsilon}{2},\sigma)$, $f_c^{-} = f_c(c_k \mid mres_i = 0) \approx N(-\frac{\varepsilon}{2},\sigma)$. Let $f_{b1}^{+} = f_{b1}(b1_k \mid r1_i = 1) \approx N(+\frac{\varepsilon}{2},\sigma)$ represent the distribution of the power consumption values of $b1_k$ such that the i[th] bit of *r1* is one. Similarly let the following distributions of power consumption exist $f_{b2}^{+} = f_{b2}(b2_k \mid r2_i = 1) \approx N(+\frac{\varepsilon}{2},\sigma)$, $f_c^{+} = f_c(c_k \mid mres_i = 1) \approx N(+\frac{\varepsilon}{2},\sigma)$. Next one can calculate the following joint conditional probability distributions of $b1_k$, $b2_k$ and c_k (and $b1_k$, $b2_k$ are equally likely to be a 0 or 1):

$$f_{b1,b2,c}(b1_k, b2_k, c_k \mid key_i \oplus p_i = 1) = \frac{1}{4}(f_{b1}^- f_{b2}^- f_c^+ + f_{b1}^+ f_{b2}^+ f_c^+) + \frac{1}{4}(f_{b1}^- f_{b2}^+ f_c^- + f_{b1}^+ f_{b2}^- f_c^-)$$

$$f_{b1,b2,c}(b1_k, b2_k, c_k \mid key_i \oplus p_i = 0) = \frac{1}{4}(f_{b1}^- f_{b2}^- f_c^- + f_{b1}^+ f_{b2}^+ f_c^-) + \frac{1}{4}(f_{b1}^- f_{b2}^+ f_c^+ + f_{b1}^+ f_{b2}^- f_c^+)$$

next one can substitute using the normal distribution : $\eta(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$ and factor out the constant $\frac{1}{\sigma\sqrt{2\pi}}$ to obtain the following expression for the probability of $key_i \oplus p_i = 1$ given all power traces Ψ :

$$\Pr(\Psi \mid key_i \oplus p_i = 1)$$

$$\approx \prod_{k=0}^{N-1} \left[\begin{array}{l} \left(e^{-\frac{1}{2\sigma^2}\left((b1_k+\frac{\varepsilon}{2})^2+(b2_k+\frac{\varepsilon}{2})^2+(c_k-\frac{\varepsilon}{2})^2\right)} \right) + \left(e^{-\frac{1}{2\sigma^2}\left((b1_k-\frac{\varepsilon}{2})^2+(b2_k-\frac{\varepsilon}{2})^2+(c_k-\frac{\varepsilon}{2})^2\right)} \right) \\ + \left(e^{-\frac{1}{2\sigma^2}\left((b1_k+\frac{\varepsilon}{2})^2+(b2_k-\frac{\varepsilon}{2})^2+(c_k+\frac{\varepsilon}{2})^2\right)} \right) + \left(e^{-\frac{1}{2\sigma^2}\left((b1_k-\frac{\varepsilon}{2})^2+(b2_k+\frac{\varepsilon}{2})^2+(c_k+\frac{\varepsilon}{2})^2\right)} \right) \end{array} \right]$$

For the rest of the analysis let ε represent $\frac{\varepsilon}{2\sigma^2}$

$$\Pr(\Psi \mid key_i \oplus p_i = 1)$$

$$\approx \prod_{k=0}^{N-1} \left(e^{-(\varepsilon b1_k + \varepsilon b2_k - \varepsilon c_k)} \right) + \left(e^{-(-\varepsilon b1_k - \varepsilon b2_k - \varepsilon c_k)} \right) + \left(e^{-(\varepsilon b1_k - \varepsilon b2_k + \varepsilon c_k)} \right) + \left(e^{-(-\varepsilon b1_k + \varepsilon b2_k + \varepsilon c_k)} \right)$$

$$\approx \prod_{k=0}^{N-1} \left[\left(e^{-\varepsilon c_k} \left(e^{-(\varepsilon b1_k - \varepsilon b2_k)} + e^{-(-\varepsilon b1_k + \varepsilon b2_k)} \right) \right) + \left(e^{+\varepsilon c_k} \left(e^{-(\varepsilon b1_k + \varepsilon b2_k)} + e^{-(-\varepsilon b1_k - \varepsilon b2_k)} \right) \right) \right]$$

Next we substitute using the following trigonometric identity $\cosh(x) = \left(\frac{e^x + e^{-x}}{2} \right)$ to obtain the following:

$$\Pr(\Psi \mid key_i \oplus p_i = 1)$$

$$\approx \prod_{k=0}^{N-1} \left(e^{-\varepsilon c_k} \left(e^{(-\varepsilon b1_k + \varepsilon b2_k)} + e^{-(-\varepsilon b1_k + \varepsilon b2_k)} \right) \right) + \left(e^{+\varepsilon c_k} \left(e^{-(\varepsilon b1_k + \varepsilon b2_k)} + e^{(\varepsilon b1_k + \varepsilon b2_k)} \right) \right)$$

$$\approx \prod_{k=0}^{N-1} (e^{-\varepsilon c_k} \cosh(\varepsilon(-b1_k + b2_k)) + e^{\varepsilon c_k} \cosh(\varepsilon(b1_k + b2_k)))$$

$$\Pr(\Psi \mid key_i \oplus p_i = 0)$$

$$\approx \prod_{k=0}^{N-1} (e^{-\varepsilon c_k} \cosh(\varepsilon(b_k 1 + b2_k)) + e^{+\varepsilon c_k} \cosh(\varepsilon(-b1_k + b2_k)))$$

Now one can further use the following trigonometric identities $\cosh(x+y)=\cosh(x)\cosh(y)+\sinh(x)\sinh(y)$, $\sinh(-x)=-\sinh(x), \cosh(-x)=\cosh(x)$, and $\sinh(x)=\left(\frac{e^{x}-e^{-x}}{2}\right)$, so the equation can be rewritten as:

$$\Pr(\Psi \mid key_i \oplus p_i = 1)$$
$$\approx \prod_{k=0}^{N-1}\left[e^{-\varepsilon c_k}\left(\cosh(\varepsilon b1_k)\cosh(\varepsilon b2_k)-\sinh(\varepsilon b1_k)\sinh(\varepsilon b2_k)\right)+e^{\varepsilon c_k}\left(\cosh(\varepsilon b1_k)\cosh(\varepsilon b2_k)+\sinh(\varepsilon b1_k)\sinh(\varepsilon b2_k)\right)\right]$$
$$\approx \prod_{k=0}^{N-1}\left[(e^{-\varepsilon c_k}+e^{\varepsilon c_k})\cosh(\varepsilon b1_k)\cosh(\varepsilon b2_k)+(-e^{-\varepsilon c_k}+e^{\varepsilon c_k})\sinh(\varepsilon b1_k)\sinh(\varepsilon b2_k)\right]$$
$$\approx \prod_{k=0}^{N-1}\cosh(\varepsilon c_k)\cosh(\varepsilon b1_k)\cosh(\varepsilon b2_k)+\sinh(\varepsilon c_k)\sinh(\varepsilon b1_k)\sinh(\varepsilon b2_k)$$

$$\Pr(\Psi \mid key_i \oplus p_i = 0)$$
$$\approx \prod_{k=0}^{N-1}\left[\cosh(\varepsilon b1_k)\cosh(\varepsilon b2_k)\cosh(\varepsilon c_k)-\sinh(\varepsilon b1_k)\sinh(\varepsilon b2_k)\sinh(\varepsilon c_k)\right]$$

Next we can further extend the expressions to directly solve for probabilities. Thus the probability of the key bit being equal to zero is a product of two probabilities:

$$\Pr ob(key_i = 0) = f_{b1,b2,c}(b1_k, b2_k, c_k \mid key_i \oplus p_i = 0, p_i = 0) \times f_{b1,b2,c}(b1_k, b2_k, c_k \mid key_i \oplus p_i = 1, p_i = 1)$$
$$\Pr ob(key_i = 1) = f_{b1,b2,c}(b1_k, b2_k, c_k \mid key_i \oplus p_i = 0, p_i = 1) \times f_{b1,b2,c}(b1_k, b2_k, c_k \mid key_i \oplus p_i = 1, p_i = 0)$$
$$if\ (\Pr ob(key_i = 0) > \Pr ob(key_i = 1)), (key_i = 0), else (key_i = 1)$$

The following notation is used to indicate partitioning of the power traces. Specifically let $\delta 1 = \{k \mid p_i = 1\}, \delta 0 = \{k \mid p_i = 0\}$. The statistic, $\Delta_{key_i=0}$, represents the probability that the key bit is a zero:

$$\Delta_{key_i=1} \approx \prod_{k|k\in\delta 1}(\cosh(\varepsilon b1_k)\cosh(\varepsilon b2_k)\cosh(\varepsilon c_k)-\sinh(\varepsilon b1_k)\sinh(\varepsilon b2_k)\sinh(\varepsilon c_k))) \times$$
$$\prod_{k|k\in\delta 0}(\cosh(\varepsilon b1_k)\cosh(\varepsilon b2_k)\cosh(\varepsilon c_k)+\sinh(\varepsilon b1_k)\sinh(\varepsilon b2_k)\sinh(\varepsilon c_k)))$$
$$\Delta_{k=0} \approx \prod_{k|k\in\delta 0}(\cosh(\varepsilon b1_k)\cosh(\varepsilon b2_k)\cosh(\varepsilon c_k)-\sinh(\varepsilon b1_k)\sinh(\varepsilon b2_k)\sinh(\varepsilon c_k))) \times$$
$$\prod_{k|k\in\delta 1}(\cosh(\varepsilon b1_k)\cosh(\varepsilon b2_k)\cosh(\varepsilon c_k)+\sinh(\varepsilon b1_k)\sinh(\varepsilon b2_k)\sinh(\varepsilon c_k)))$$

Alternatively one can represent the statistic Λ as given below, where the statistic is positive or negative to indicate the key value is a 1 or 0 respectively.

$$\Lambda_{key_i} = \Delta_{key_i=1} - \Delta_{key_i=0}$$
$$= \prod_{k|k\in\delta 1}(\cosh(\varepsilon b1_k)\cosh(\varepsilon b2_k)\cosh(\varepsilon c_k) - \sinh(\varepsilon b1_k)\sinh(\varepsilon b2_k)\sinh(\varepsilon c_k))) \times$$
$$\prod_{k|k\in\delta 0}(\cosh(\varepsilon b1_k)\cosh(\varepsilon b2_k)\cosh(\varepsilon c_k) + \sinh(\varepsilon b1_k)\sinh(\varepsilon b2_k)\sinh(\varepsilon c_k)))$$
$$- \prod_{k|k\in\delta 0}(\cosh(\varepsilon b1_k)\cosh(\varepsilon b2_k)\cosh(\varepsilon c_k) - \sinh(\varepsilon b1_k)\sinh(\varepsilon b2_k)\sinh(\varepsilon c_k))) \times$$
$$\prod_{k|k\in\delta 1}(\cosh(\varepsilon b1_k)\cosh(\varepsilon b2_k)\cosh(\varepsilon c_k) + \sinh(\varepsilon b1_k)\sinh(\varepsilon b2_k)\sinh(\varepsilon c_k)))$$

Appendix B: 2nd Order DPA derivation

Let the masked round key, and masks be represented by *mres*, and *r1*, respectively. For example consider the following lines of code:

Line 1. load *r1*

Line 2. *key1 = key ^r1*

Line 3. load plaintext

Line 4. *mres = key1^p=key^r1 ^p*

Let the k[th] normalized power consumption of the random mask *r1 (line 1.)*, and the masked round key *mres (line 4.)*, be represented by $b1_k$, and c_k respectively. The distribution for these values is assumed to be Gaussian with mean of zero and standard deviation of one (due to normalization, $N(\mu,\sigma) = N(0,1)$) $f_{b1}(b1_k) \approx N(0,1)$, and $f_c(c_k) \approx N(0,1)$. Let $f_{b1}^- = f_{b1}(b1_k \mid r1_i = 0) \approx N(-\frac{\varepsilon}{2},\sigma)$ represent the distribution of the power consumption values of $b1_k$ such that the i[th] bit of *r1* is zero. Assume it is also a Gaussian distribution. Similarly let the following distributions of power consumption exist (similar to [15]) $f_c^- = f_c(c_k \mid mres_i = 0) \approx N(-\frac{\varepsilon}{2},\sigma)$. Let $f_{b1}^+ = f_{b1}(b1_k \mid r1_i = 1) \approx N(+\frac{\varepsilon}{2},\sigma)$ represent the distribution of the power consumption values of $b1_k$ such that the i[th] bit of *r1* is one. Similarly let the following distributions of power consumption exist , $f_c^+ = f_c(c_k \mid mres_i = 1) \approx N(+\frac{\varepsilon}{2},\sigma)$. Next one can calculate the following joint conditional probability distributions of $b1_k$, and c_k (and $b1_k$ is equally likely to be a 0 or 1):

$$f_{b1,c}(b1_k, c_k \mid key_i \oplus p_i = 1) = \frac{1}{2}(f_{b1}^+ f_c^+) + \frac{1}{2}(f_{b1}^+ f_c^-)$$
$$f_{b1,c}(b1_k, c_k \mid key_i \oplus p_i = 0) = \frac{1}{2}(f_{b1}^+ f_c^-) + \frac{1}{2}(f_{b1}^+ f_c^+)$$

next one can substitute using the normal distribution : $\eta(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$ and factor out the constant $\frac{1}{\sigma\sqrt{2\pi}}$ to obtain the following expression for the probability of $key_i \oplus p_i = 1$ given all power traces Ψ :

$$\Pr(\Psi \mid key_i \oplus p_i = 1)$$
$$\approx \prod_{k=0}^{N-1}\left[\left(e^{-\frac{1}{2\sigma^2}\left((b1_k+\frac{\varepsilon}{2})^2+(c_k-\frac{\varepsilon}{2})^2\right)}\right)+\left(e^{-\frac{1}{2\sigma^2}\left((b1_k-\frac{\varepsilon}{2})^2+(c_k+\frac{\varepsilon}{2})^2\right)}\right)\right]$$

For the rest of the analysis let ε represent $\frac{\varepsilon}{2\sigma^2}$

$$\Pr(\Psi \mid key_i \oplus p_i = 1)$$
$$\approx \prod_{k=0}^{N-1}\left(e^{-(\varepsilon b1_k-\varepsilon c_k)}\right)+\left(e^{-(-\varepsilon b1_k+\varepsilon c_k)}\right)$$

Next we substitute using the following trigonometric identity $\cosh(x)=\left(\frac{e^x+e^{-x}}{2}\right)$ to obtain the following:

$$\Pr(\Psi \mid key_i \oplus p_i = 1) \approx \prod_{k=0}^{N-1}\cosh(\varepsilon(b_k 1-c_k))$$
$$\Pr(\Psi \mid key_i \oplus p_i = 0) \approx \prod_{k=0}^{N-1}\cosh(\varepsilon(b_k 1+c_k))$$

So far these results are identical to those found in [15]. They derive a theorem which states the decision problem as $\prod_{k=0}^{N-1}\cosh(b_k+c_k) < or > \prod_{k=0}^{N-1}\cosh(b_k-c_k)$. Then the previous research suggests using a heuristic approximation, $\sum_{k=0}^{N-1}|b_k-c_k|$, to use in the attack. Unlike the research in [15], we will derive a different statistic to use in the attack.

We can further extend the expressions to directly solve for probabilities. Thus the probability of the key bit being equal to zero is a product of two probabilities:

$$\Pr ob(key_i = 0) = f_{b1,c}(b1_k, c_k \mid key_i \oplus p_i = 0, p_i = 0) \times f_{b1,c}(b1_k, c_k \mid key_i \oplus p_i = 1, p_i = 1)$$
$$\Pr ob(key_i = 1) = f_{b1,c}(b1_k, c_k \mid key_i \oplus p_i = 0, p_i = 1) \times f_{b1,c}(b1_k, c_k \mid key_i \oplus p_i = 1, p_i = 0)$$
$$if\,(\Pr ob(key_i = 0) > \Pr ob(key_i = 1)), (key_i = 0), else(key_i = 1)$$

The following notation is used to indicate partitioning of the power traces. Specifically let $\delta 1 = \{k \mid p_i = 1\}, \delta 0 = \{k \mid p_i = 0\}$. The statistic, $\Delta_{key_i=0}$, represents the probability that the key bit *i* is a zero:

$$\Delta_{key_i=0} \approx \prod_{k|k\in\delta 0}(\cosh(\varepsilon(b_k 1+c_k))) \times \prod_{k|k\in\delta 1}(\cosh(\varepsilon(b_k 1-c_k)))$$
$$\Delta_{key_i=1} \approx \prod_{k|k\in\delta 1}(\cosh(\varepsilon(b_k 1+c_k))) \times \prod_{k|k\in\delta 0}(\cosh(\varepsilon(b_k 1-c_k)))$$

Alternatively one can represent the statistic Λ as given below, where the statistic is positive or negative to indicate the key value is a 1 or 0 respectively.

$$\Lambda_{key_i} = \Delta_{key_i=1} - \Delta_{key_i=0}$$
$$= \prod_{k|k\in\delta 1}(\cosh(\varepsilon(b_k 1 + c_k))) \times \prod_{k|k\in\delta 0}(\cosh(\varepsilon(b_k 1 - c_k))) - \prod_{k|k\in\delta 0}(\cosh(\varepsilon(b_k 1 + c_k))) \times \prod_{k|k\in\delta 1}(\cosh(\varepsilon(b_k 1 - c_k)))$$

Appendix C: Hamming Weight Attack on Fixed Masks

Attack Problem 1. Given the hamming weight of a mask, the hamming weight of the key and the value of the mask^key (obtained from 1st order DPA), how many possible key values exist?

Attack Problem 2. Given the hamming weight of mask 1, *m1*, the hamming weight of mask 2, *m2*, the hamming weight of the key and the value of the *m1^m2^*key (obtained from 1st order DPA), how many possible key values exist?

Solution to Attack problem 1:
Let the hamming weight of a value *x* be represented by *hwt(x)*. Let *i* be the number of '1's in *k* which combine with '1' in *m* to produce a '0' in *m^k*. It can be formulated as:
i=(hwt(m) + hwt(k) –hwt(m^k))/2
The number of possible key values =
[(word_size – hwt(m^k)) choose i] × [hwt(m^k) choose (hwt(k) – i)]

Solution to Attack problem 2:
Let *min_hwt(m1,m2)* be the minimum hamming weight value possible of the exclusive-or of *m1* and *m2*. Similary let *max_hwt(m1,m2)* be the maximum hamming weight value possible by the exclusive-or of *m1* and *m2*.
The number of possible key values =

$$\sum_{i=\min\_hwt(m1,m2)\,to\,\max\_hwt(m1,m2)} [hwt(m1\hat{}\,m2\hat{}\,k)\ \text{choose}\ (hwt(k) - i)] \times [(\text{word\_size} - hwt(m1\hat{}\,m2\hat{}\,k))\ \text{choose}\ i]$$