Data Dependent Power Use in Multipliers

Colin D. Walter
Comodo Research Laboratory,
7 Campus Road, Bradford, BD7 1HR, UK
e-mail: Colin.Walter@comodo.com

David Samyde*
FemtoNano,
Paris
e-mail: David.Samyde@FemtoNano.com

Abstract

Recent research has demonstrated the vulnerability of certain smart card architectures to power and electromagnetic analysis when multiplier operations are insufficiently shielded from external monitoring. Here several standard multipliers are investigated in more detail in order to provide the foundation for understanding potential weaknesses and enabling the subsequent successful repair of those systems. A model is built which accurately predicts power use as a function of the Hamming weights of inputs without the combinatorial explosion of exhaustive simulation. This confirms that power use is indeed data dependent at least for those multipliers. Laboratory experiments confirm that EMR also corresponds closely to these power predictions over a wide range of frequencies.

Key Words — *Differential power analysis, DPA, EMA, smart card, multiplication, multiplier, RSA cryptosystem.*

1 Introduction

Security is an increasingly important issue these days, even for computer arithmetic. Typical RSA [11] hardware performs long integer modular multiplications $A \times B \mod M$ using a modification of the standard primary school method for multiplication where modular reductions are interleaved with the additions of digit multiples of the multiplicand. Normally this is done using a k-bit multiplier to compute digit products $a_i \times b_j$ and $q_i \times m_j$. Because switching gates in a circuit requires more power than keeping them in their current states, the amount of power consumed during a multiplication is data dependent. With sufficiently sensitive monitoring equipment, such variations can be observed and perhaps used to deduce properties of the data being processed. This is called *power analysis* [6]. It has serious economic consequences if it can be applied to attack an electronic purse on a smart card, a signature key on a credit card or rights keys on pay-per-view cards. Electro-magnetic radiation (EMR) from the gate switching also leaks substantial information, especially when conducted along power supply lines and network cables [1, 5, 9, 10]. With detectable radiation typically carrying 15 metres or more, this is a security problem not just for smart cards, but particularly also for certificate authorities that are frequently signing with private keys.

Some defences against such side channel leakage are mentioned by Anderson [2], including dual rail logic and bus encryption [3], which may cure the most major source of data dependent power variation, namely the bus. Random transformations of inputs, random register re-location, randomising algorithms, random noise generators, random clock fluctuations and concurrent calculations on another processor are further methods used to blind calculations and data movements so that recovery of secret keys becomes more difficult during their deliberately limited lifespan.

Since they occupy a substantial part of most CPUs and cryptographic co-processors, multipliers now become one of the strongest sources of data leakage which may need protection. On a chip with no security measures and little other than a multiplier, Sommer [7] observed a correlation between power traces and values by looping through a particular cycle of values. Here power variations for several standard multipliers are investigated. This confirms that attacks such as that in [12] should indeed be possible at least for small multipliers. Complexity issues can be used to show that larger multipliers are less vulnerable to such attacks, but different designs also vary in vulnerability.

The investigation was based initially on simulations which computed gate switching activity in software models of some multipliers. Then several smartcard multipliers were tested in order to confirm the predicted connection between power use and gate switching and verify that similar properties held for EMR. This data agreed with the simulations, and so it is reasonable to conclude that the simulations predict what might be measured successfully and non-invasively by EMR probes or RF scanning devices.

*Work partly undertaken while the authors were at UCL Crypto Group (DICE), Université catholique de Louvain, Louvain-la-Neuve, Belgium.

The problem with exhaustive simulations is combinatorial explosion: $O(2^{4k}k^2)$ for a k-bit multiplier as a result of initialisation with two arguments followed by the $O(k^2)$ multiplication of two further arguments. For larger cases which may be met in practice, less accurate random sampling must be performed. This prompts the need for more efficient models to aid the design of less leaky hardware.

Often an attacker just needs information about the Hamming weight of arguments, as in [12], to recover a secret key: the weights of individual digits are sufficient to identify re-use of long integer arguments in an exponentiation, and this may be enough to determine the precise operations in the exponentiation scheme, and hence reveal the secret exponent used. So, for cases when Hamming weights are of interest, a model is constructed which predicts power use in only $O(k^6)$ time. This is tailored to the precise construction of the target multiplier to give very accurate results.

The model has to make a strong assumption to achieve such a large reduction in computational complexity, namely that certain groups of bits generated within the multiplier have independent values. In reality this is far from the case, but the accuracy was still verifiable in two ways. First, the model and simulation were compared for small multipliers. Then the model was compared with multipliers used in practice by obtaining power and EMR traces in the laboratory. In both cases the results matched closely.

The main conclusions are: i) power use in standard multipliers is closely related to input Hamming weights; ii) the simplifications described here enable fairly accurate models of power use to be constructed, so that designs can be tested more easily in the search for less leaky hardware; and iii) that some multiplier designs (such as one with binary Booth re-coding) leak less information about Hamming weights than others (such as the standard add-and-shift multiplier).

2 Notation

Most smart card cryptographic processors make use of a single, unpipelined k-bit multiplier, where typically $k = 16$ although $k = 32$ is becoming the norm. Then the long integers used in public-key cryptography are represented using k-bit digits. The RSA crypto-system uses modular exponentiation to encrypt, decrypt, sign and verify [11]. The attacker's aim is to recover the factorisation of the modulus M or, equivalently, the secret decryption exponent.

The exponentiation is comprised of modular multiplications $R \leftarrow (A \times B + C) \bmod M$ which loop round the code:

$$\begin{aligned} R &\leftarrow r \times R + a_i \times B \\ q &\leftarrow R \ \mathrm{div}\ M \\ R &\leftarrow R - q \times M \end{aligned}$$

(or a similar variation such as is given by Montgomery [8]). Here R holds the partial product, a_i is a digit of A in base $r = 2^k$, and q is the digit multiple of M which must be subtracted to keep R within any required bounds. The first assignment breaks down into a number of sequential multiply-accumulate operations on digits with carry digit c:

$$r_j + r \times c \quad \leftarrow \quad r \times r_j + a_i \times b_j + c \qquad (1)$$

The hardware for this leaks data which an attacker can extract by observing variations in current or EMR. The question here is whether or not he can determine the Hamming weight of a_i, say, by averaging over the b_j in (1) since, by [12], this would enable him to determine which multiplier A is used in the exponentiation. With the standard square-and-multiply or m-ary exponentiation methods, this easily leads to reconstruction of the secret exponent.

A significant complication is that power variation from gate switching depends not just on the input data but also on the initial state of the multiplier. Some technique, such as averaging, is needed to remove such dependencies on the initial state and other irrelevant inputs. The repeated re-use of data in (1) provides just the opportunity for this. The main problem faced by implementors is the removal of such handles by which attackers are able to isolate useful subsets of observations over which averaging proves productive: the handle used in [12] was based on the Hamming weight of digits. So part of our interest here is in how averaged data from the multiplier determines Hamming weight.

3 Multiplier Simulation

Two software simulations of k-bit multipliers were built with variable k in order to verify empirical claims by previous authors that gate switching activity was related to the Hamming weights of the inputs. The simulated circuits were constructed from logic gates, 3-bit to 2-bit full adders (Fig. 1) and 2-bit half adders using a standard Wallace tree configuration and carry-propagate adder. They followed the add-and-shift model ([4], Fig. 3.1) and the binary Booth-recoding model ([4], Fig. 3.4). Pipelined multipliers were not considered as they are too large for smart cards and simultaneous processing of several multiplications is likely to mask much of the effect of interest here that designers need to eliminate. (Indeed, pipelining may be an effective counter-measure to power attacks on the multiplier.)

For convenience, we assume that the power consumed by switching AND, OR and XOR gates is the same and identical whichever way they are switched. Then, to obtain a good overall picture, it suffices to count gates which are switched without any weighting for different power use. So the number of gates switched between the registers was recorded, but no term was included for switching within the registers, nor any for the multiplexing of input bits which is required to feed the initial AND gates. Such switching activity simply increases the strength of our conclusions. Furthermore,

the accumulation of the carry digit c was ignored: it makes little difference to the results except that further averaging would be necessary to eliminate the variation caused by it.

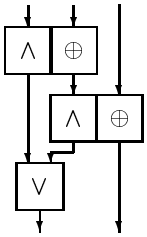

Figure 1. A 3-bit Counter (Full Adder).

Different constructions were tried corresponding to the different groups of three bits which can be chosen for each full adder. Their effect on the simulation results were not usually substantial. So similar results would seem to hold for every multiplier within a major design class, whatever its detailed layout. However, the initial simulation model was over-simplified and gave poor results. Thus,

- the multiplier model used in simulations should match the hardware design as closely as possible.

Ideally, the gates in the simulation would be initialised by executing an initial product, say $c \times d$. For each choice of c and d, the product $a \times b$ would then be performed and the number of gate switches counted. For each a, the average would be computed as b, c and d varied over all values and this used to characterise a. This is only possible for small multipliers, and was done for all the results here with $k \leq 8$. For larger multipliers some simplifications had to be made: only a subset of random values c and d were chosen, and, when necessary, only a random set of values for b were used to derive data supposedly characteristic of a.

For such input sets, the number of gate changes from high to low or low to high were counted. These were averaged for a given first argument a and also for a given second argument b. The first arguments a (the multipliers) and the second arguments b (the multiplicands) were then ordered according to these averages and used to create tables, such as Table 1, and graphs, such as Fig. 2, for different k.

The orders for multipliers and multiplicands are usually slightly different because the inputs cannot be treated symmetrically. Typically this results from using a row of full adders to combine three multiples of the multiplicand into two. Then, in the add-and-shift multiplier, bits of multiplicand b are processed uniformly, but those of multiplier a are not. As a result, we distinguish between the multiplicand (always b here) and the multiplier (always a here). For obvious reasons, the difference between these is more marked for the Booth-recoding case.

4 Gate Count Digit Order

Of interest here is the order of the digits according to the number of gates switched. We write $a_0 <_{gc} a_1$ when the average switched gate count (*gc*) for digit a_0 is less than that for a_1 in a specified situation. Not only was it verified by the simulation that the (averaged) gate count is indeed closely related to the Hamming weight of the argument, but more detailed patterns emerged for the ordering of individual digits according to these gate switching counts.

- **The Hamming Weight Principle:** Let $\text{HW}(a)$ denote the Hamming weight of a digit a, i.e. the sum of its bits. Then $\text{HW}(a_0) < \text{HW}(a_1)$ implies $a_0 <_{gc} a_1$.

For the add-and-shift multiplier, this principle held without exception for both arguments and for all values of k that were investigated, whether by exhaustive simulation as in the cases of $k \leq 8$, or by random approximation as described above for $k > 8$. Indeed, for small k there is a visible jump in the average number of gates switched when the Hamming weight of an argument is increased (*see* Fig. 2).

In the case of the Booth multiplier, the Hamming Weight Principle also held without exception for the *multiplicand* and all tested values of k. However, the *multiplier* argument, namely the input which is re-coded, behaved erratically with respect to Hamming weight. A much better measure was given by using *Booth* weight, which, in a sense, corresponds to the Hamming weight of the recoding:

Definition 1 *The* Booth Weight *of a number with binary representation $b_n \ldots b_1 b_0$ is the sum of the weights of all triples $b_{2i+1} b_{2i} b_{2i-1}$ ($i \in \mathbb{Z}$) where 000 has weight 0, 111 has weight 2 and all other triples have weight 1.*

(Ends are padded with zeros as necessary.) In the Booth multiplier with recoded multiplier, apart from the end conditions, triple 000 leads to a string of 0s being added to the accumulating sum, triple 111 leads to a string of 1s being added, and otherwise a shifted copy of the multiplicand, or its complement, is added. On average, these contribute weights in the ratio $0{:}2{:}1$ respectively, as in the definition.

Here is an illustrative example showing the behaviour:

- For the Booth multiplier with $k = 4$, the average gate count order is 0, 4, 1, 5, 7, 3, 6, 2, 12, 8, 13, 9, 11, 10, 15, 14 for the multiplier a but 0, 2, 4, 1, 8, 6, 5, 10, 3, 12, 9, 7, 14, 13, 11, 15 for the multiplicand b.

The Booth weights of multiplier a are all in ascending order, as are the Hamming weights of multiplicand b.

More detailed ordering of digits with equal Hamming (or Booth) weight follows several rules-of-thumb. These depend on the multiplier type, with exceptions arising from how bits are grouped for feeding into the compressors. The two add-and-shift multiplier "guidelines" below hold for around 75% and 85% respectively of cases with $k \leq 8$:

$a \times b$	a	0	1	2	4	5	3	6	7
b	Averages	6.31	7.31	7.88	8.30	9.80	9.87	10.41	12.27
0	6.31	6.31	6.31	6.31	6.31	6.31	6.31	6.31	6.31
1	7.44	6.31	6.81	7.06	7.31	7.81	7.56	8.06	8.56
2	7.88	6.31	7.06	7.31	7.69	8.44	8.06	8.69	9.44
4	8.17	6.31	7.06	7.69	7.91	8.66	8.44	9.28	10.03
5	9.80	6.31	7.56	8.44	8.91	11.78	9.69	11.03	14.69
3	9.88	6.31	7.56	8.06	8.69	9.94	10.81	12.97	14.72
6	10.39	6.31	7.81	8.69	9.28	10.78	13.16	12.00	15.13
7	12.27	6.31	8.31	9.44	10.28	14.69	14.91	14.94	19.31

Table 1. Average Gate Switching in a 3-bit Add-and-Shift Multiplier, with ordered Arguments.

I. Suppose a_0 and a_1 are *odd* digits with $\text{HW}(a_0) = \text{HW}(a_1)$ and $a_0 < a_1$. Then usually $2^i a_0 >_{gc} 2^j a_1$ for $i, j \geq 0$ in the add-and-shift multiplier.

II. Normally, $a <_{gc} 2^i a$ for $i > 0$.

For the add-and-shift case with k=3, these imply $5 <_{gc} 3 <_{gc} 6$ and $1 <_{gc} 2 <_{gc} 4$, which are readily verified (*see* Table 1). With the Hamming Weight Principle, this gives a total ordering of all digits when k=3. Grouping bits in threes for the 3-to-2 counters means that digits of the same weight can behave quite differently, as is the case for weight 2 (*see* Fig. 2). Hence there are higher order properties that might be exploited to deduce more than just Hamming weight.

When $k = 4$ the order varies slightly between multiplier and multiplicand, but the Hamming Weight rule still holds:

- For the add-and-shift multiplier with $k = 4$, the average gate count order is 0, 1, 2, 4, 8, 3, 9, 5, 10, 6, 12, 11, 7, 13, 14, 15 for the multiplier a but 0, 1, 2, 4, 8, 9, 3, 5, 10, 12, 6, 13, 7, 11, 14, 15 for the multiplicand b.

The second rule-of-thumb always holds also, but the first fails (at least for our multiplier) in a quarter of cases, such as for the multiplicand pairs (3,5), (3,10) and (7,11).

Similar but more complex results apply to the multiplicand of the Booth multiplier, but other guidelines are needed to characterise the order for the recoded argument.

A further principle relates the order of digits as k is increased when the hardware is constructed in the same way:

III. For add-and-shift multipliers, if $a <_{gc} a'$ in k-bit arithmetic, then usually $a <_{gc} a'$ in $(k+1)$-bit arithmetic.

Thus, the digit order for $k = 3$ is contained within that of $k = 4$ except that the pair (5,3) is reversed. Like the first two, this rule is frequently violated. Re-ordering of digits is inevitable: it is difficult to combine rows of full adders in a consistent manner as k is increased.

It is now clear that there are general principles for any multiplier which determine a broad ordering of the digits under average gate switching, with the detailed order being determined only once the multiplier is defined precisely.

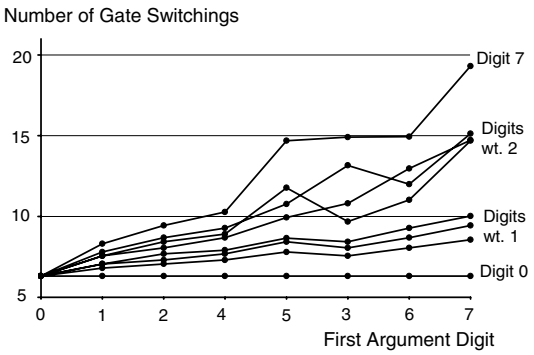

Figure 2. Gate Switching for the Second Argument of a 3-Bit Add-and-Shift Multiplier.

5 Hamming Weights

Gate count results for digit products $a \times b$ can be translated easily into statistics relating Hamming weights by averaging over all digits with the same Hamming weight. Table 2 shows this for the 8-bit add-and-shift multiplier. Fig. 3 illustrates the same data graphically. From this it is clear that there is a close and potentially predictable relationship between Hamming weight and average gate switching activity. This relationship is close to linear in each Hamming weight except for values close to 0. Moreover, from the above, digits of the same Hamming weight behave in a broadly similar way. Thus, if one can obtain a measure of the average gate switching activity for $a \times b$ as b varies uniformly over all digits, or uniformly over all digits of a known Hamming weight, then it should be possible to extract the exact Hamming weight of a. Fortunately for the security of an embedded RSA cryptosystem, this does not lead easily to a deduction of the value of a except in the unusual circumstances of extreme Hamming weights: the variance in average gate counts between individual digits of the same weight is not high, and for the middle weights there are many alternative choices which can be made. Because of this, increasing the digit size k makes life much harder for the attacker. However, just an approximate Hamming weight may be all that the attacker requires [12].

$a\times b$	HW(a)	0	1	2	3	4	5	6	7	8
HW(b)	Av$^{\text{ages}}$	77.30	81.49	88.26	95.88	103.82	111.86	119.87	127.80	135.60
0	77.30	77.30	77.30	77.30	77.30	77.30	77.30	77.30	77.30	77.30
1	81.49	77.30	78.35	79.40	80.45	81.49	82.54	83.59	84.64	85.69
2	88.24	77.30	79.40	81.72	84.47	87.84	91.77	95.54	99.12	102.86
3	95.90	77.30	80.45	84.47	89.78	95.68	101.56	107.76	114.15	120.10
4	103.84	77.30	81.49	87.83	95.69	103.59	111.84	120.02	128.21	136.04
5	111.85	77.30	82.54	91.77	101.56	111.81	121.97	132.06	141.70	151.61
6	119.84	77.30	83.59	95.63	107.69	119.94	132.05	143.64	155.43	167.50
7	127.75	77.30	84.64	99.27	113.91	128.15	141.71	155.46	168.97	181.18
8	135.60	77.30	85.69	102.97	119.62	136.04	152.00	167.58	181.14	193.14

Table 2. Switching in 8-bit Add-and-Shift Multiplier averaged over Arguments of equal Hamming Wt.

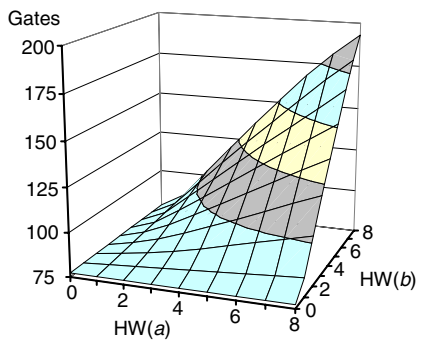

Figure 3. Gate Switching in an 8-bit Multiplier as a Function of input Hamming Weights.

6 Trees of 3-bit Counters

Full adders, i.e. 3-bit counters, form the main body of a typical multiplier. This section studies how the distribution of bits changes as the bits progress down a tree of counters. The changes affect the number of gates that are switched: interesting data-dependent activity occurs near the inputs, but any non-uniformity in the distribution of input bits dissipates lower down the tree so that, by the output, 0s and 1s turn up with roughly equal probability. Multipliers constructed from other compressors behave similarly.

The next theorem describes how a single counter experiences a data-dependent amount of gate switching activity and affects the probabilities of bits as they pass through.

Theorem 1 *In a 3-bit full adder, let δ be the number of input bits set to 1 minus the resulting number of output bits set to 1.*

i) *δ is either 0 or 1. If the input bits are independent with probability p of being 1 and $q = 1-p$, then $\delta = 0$ with probability $3pq^2+q^3$ and $\delta = 1$ with probability p^3+3p^2q;*

ii) *If all inputs are independently and uniformly distributed then, on average, two gates are switched if $\delta = 0$, and two and a half gates are switched if $\delta = 1$;*

iii) *Suppose the full adder has been initialised with input bits which are independent with probability p of being 1, and the following input bits are 1 independently with probability p'. Define $q = 1-p$ and $q' = 1-p'$. Then the average number of gates switched by the latter input is*

$$p(3+2q) + \tfrac{p'}{1+2p'}(q-p)^2(5q-p)$$

when $\delta = 0$ and, when $\delta = 1$, it is

$$(3+2p+4p^2)q + \tfrac{q'}{1+2q'}(3p-q)^2(p-q)\ ;$$

iv) *For the situation in* (iii)*, the average difference in the number of gates switched between cases $\delta = 0$ and $\delta = 1$ is:* a) *at least $\frac{1}{2}$ for all $p \le \frac{1}{2}$ with equality for $p = \frac{1}{2}$,*
b) *a decreasing function of p for most values of p', and*
c) *positive for all p less than approximately 0.6.*

Theorem 2 i) *For inputs to a 3-bit counter which are independent and equal to 1 with probability p, an (unspecified) output bit is 1 with probability $p^3+\frac{3}{2}pq$ where $q = 1-p$.*

ii) *Assume the output bits from 3-bit counters are independent. If bits are fed in a random order from one row to the next in a tree of 3-bit counters, then the probability of an output bit from the final row being 1 tends monotonically towards $\frac{1}{2}$ as the number of rows is increased.*

Proof. These are easy exercises for the reader. □

Theorem 2 provides probabilities of 1s down a tree of adders and then, using Theorem 1(iii) with $p' = p$, one can obtain the average gate switching difference g_j at depth j. These are tabulated in Table 3 starting at $p = \frac{1}{4}$. Dependency on the initial distribution of data clearly diminishes as distance increases down the multiplier. Over three successive counters, the position dependent differences are approximately halved e.g. from 1.5417–0.5 to 1.0461–0.5 (where 0.5 is the limiting value).

j	0	1	2	3	4	5
p_j	0.2500	0.2969	0.3393	0.3753	0.4045	0.4275
g_j	1.5417	1.3635	1.1950	1.0461	0.9218	0.8221
j	6	7	8	9	10	11
p_j	0.4453	0.4588	0.4690	0.4767	0.4825	0.4869
g_j	0.7442	0.6843	0.6388	0.6044	0.5784	0.5589

Table 3. Probability p_j of a 1 and the extra average Gate Switching g_j when a Bit is removed by a counter at depth j in the Tree.

7 The Add-and-Shift Multiplier Model

The add-and-shift hardware multiplier was simulated without pipelining or bit recoding. It is best described via bit slices where the ith slice processes all bits corresponding to 2^i. In order for the multiplier to compute $a\times b$, multiplexors first produce k copies of each bit from its two arguments. These are then ANDed in pairs to create k^2 bits a_ib_j.

Bit slice 0 just outputs bit a_0b_0 from its AND gate. For bit slice i with $0<i<k$, there are $i+1$ input bits from AND gates and $i-1$ carry bits from slice $i-1$. These are added using $(i-1)$ 3-bit counters and a single 2-bit counter, generating one output bit and i carries. Slice k has $k-1$ inputs from the AND gates and $k-1$ carries in from slice $k-1$. These are added using $(k-2)$ 3-bit counters and a 2-bit counter which generate $k-1$ carries up to slice $k+1$ and one output bit. Lastly, for bit slice $2k-i$ where $0<i<k$, there are $i-1$ input bits from AND gates and i carries in from the previous slice. These are added using $(i-1)$ 3-bit counters which generate $i-1$ carries to the next slice and one output bit.

To minimise critical path lengths a Wallace tree was built for each bit slice, i.e. the full adders were arranged so that bits with the shortest critical path were always added together first. The multiplier contains $k(k-2)$ full adders, and the first row in the Wallace tree has about one third of these. Subsequent rows have about $\frac{2}{3}$rds of the number in the previous row. So, although the two theorems indicate a decrease in the data-dependent gate switching as one descends the tree, the decrease is of less importance because so many of the 3-bit counters are at the top of the tree.

8 The Hamming Weight Multiplier

The simulation of Section 3 uses $O(2^{4k}k^2)$ time because of the 2^k choices for each of a, b, c and d. This makes it infeasible to model 16-bit or larger multipliers. To investigate behaviour in terms of Hamming weights, it is necessary to build a multiplier which computes (average) gate switching activity just from the weights of a, b, c and d. Ideally this should have polynomial, not exponential, time complexity so that any expected size for a multiplier can be processed.

As in Theorem 1, let undashed characters such as p denote probabilities during initialisation and let dashed characters such as p' denote probabilities during execution. The probability of a bit of input a being 1 is $p'=\frac{1}{k}\mathrm{HW}(a)$. Let us denote these bit probabilities for a, b, c and d by p', q', p and q, respectively. Then the multiplier model is initialised by computing $c\times d$ and then executing $a\times b$ during which gate switching is calculated probabilistically.

8.1 The Multiplexors and AND Gates

The contribution from the multiplexors and AND gates is easy to assess exactly. Each of the initial $2k$ input bits is first multiplexed to k AND gates at the top of the multiplier. Those corresponding to the first argument (c then a) are changed from 0 to 1 with probability $(1-p)p'$ and from 1 to 0 with probability $p(1-p')$. Thus, taking both arguments into account, the multiplexors introduce a power variation proportional to $p+p'-2pp'+q+q'-2qq'$. As this is linear in p and q, the average over all initial conditions is given by taking $p=q=\frac{1}{2}$. The formula is then constant, showing the power variation resulting from multiplexors is constant. So this contribution is of no further concern.

On average, the AND gates will be initialised with their outputs set to 1 with probability pq and, when $a\times b$ is computed, the AND gates output a 1 with probability $p'q'$. So $(1-pq)p'q'$ is the probability that an AND gate will be switched from 0 to 1 and $pq(1-p'q')$ is the probability that it will be switched in the opposite direction. So it will be switched with probability $pq+p'q'-2pqp'q'$. This must be multiplied by k^2 to obtain the expected number of AND gates which will change state: $\mathrm{HW}(c)\mathrm{HW}(d)+\mathrm{HW}(a)\mathrm{HW}(b)-2k^{-2}\mathrm{HW}(c)\mathrm{HW}(d)\mathrm{HW}(a)\mathrm{HW}(b)$. Averaging over all initialising inputs c and d yields $\frac{1}{4}k^2+\frac{1}{2}\mathrm{HW}(a)\mathrm{HW}(b)$.

This is a very reliable component of the total variation in gate counts, almost independent of the multiplier construction. It is a substantial part of the total data-dependent variation of the multiplier. There are $\mathrm{HW}(a)\mathrm{HW}(b)$ input 1s to the counter tree, and this is an upper bound on the number of 1s removed by the adder tree. Each removal of a 1 causes a data dependent increase of about 1.2 gates switching for an average initialisation (by Table 3 and an estimate of the number of counters in each row). Comparing the resulting total of $1.2\mathrm{HW}(a)\mathrm{HW}(b)$ gate switches with the total from the AND gates, we expect AND gate switching to account for about 0.5/(1.2+0.5), i.e. 30%, of the total data dependent variation, which is what we find.

8.2 The Compressor Tree

To prevent combinatorial explosion in the counter tree, some simplification is necessary.

$\mathrm{HW}(a)\times\mathrm{HW}(b)$	0	1	2	3	4	5	6	7	8
0	73.54	73.54	73.54	73.54	73.54	73.54	73.54	73.54	73.54
1	73.54	74.90	76.32	77.82	79.39	81.05	82.79	84.60	86.48
2	73.54	76.32	79.39	82.79	86.48	90.35	94.32	98.30	102.25
3	73.54	77.82	82.79	88.40	94.32	100.28	106.14	111.82	117.28
4	73.54	79.39	86.48	94.32	102.25	109.95	117.28	124.21	130.76
5	73.54	81.05	90.35	100.28	109.95	119.05	127.53	135.46	143.00
6	73.54	82.79	94.32	106.14	117.28	127.53	136.99	145.95	154.92
7	73.54	84.60	98.30	111.82	124.21	135.46	145.95	156.46	168.34
8	73.54	86.48	102.25	117.28	130.76	143.00	154.92	168.34	193.46

Table 4. Gate Switching Activity for the 8-bit Hamming Weight Add-and-Shift Multiplier.

- *Independence Assumption for Counters:* The inputs to each full adder and each half adder are independent.

In practice this assumption is absurd because there are only $2k$ bits fed into the multiplier but k^2 bits which are generated from them and processed by the counters. Moreover, strong dependencies can persist if related bits are grouped together for input into counter sub-trees.

It is necessary just to know the probability of input bits to a counter being 1 in order to apply Thm. 2 to compute the probability of output bits being 1. Then Thm. 1 with the assumption enables the average number of gate changes to be computed for the whole tree once input bit probabilities are chosen for the initialisation and execution phases. After averaging over all initial states, expected gate switching activity was obtained, as in Table 4.

8.3 Evaluation

This model matches the actual values very well: the maximum relative error between entries in Tables 2 and 4 is in the region of 7.5%. Moreover, for all small k the forms of the discrepancies are similar, and therefore predictable. For example, model values for input weights $(0, i)$ are consistently marginally smaller than true values by about 5%. So it seems reasonable to assume that extrapolating to large cases will yield fairly accurate results and that a standard correction could be applied to remove most of the error.

There is a rather beautiful connection between Hamming weights of inputs to a product and the average Hamming weight of the result. This is illustrated in Fig. 4. The same features appear for all word lengths k. The difference between these weights is, of course, the number of bits removed by the counters, which is proportional to the average number of gates switched in the multiplier. So a useful health check for the model is to construct a table of output weights and compare it with these expected values.

In the model, the output digit is a vector of probabilities associated with its bits. The sum of these is the average Hamming weight predicted by the model for the output. The table of these output weights was created for various

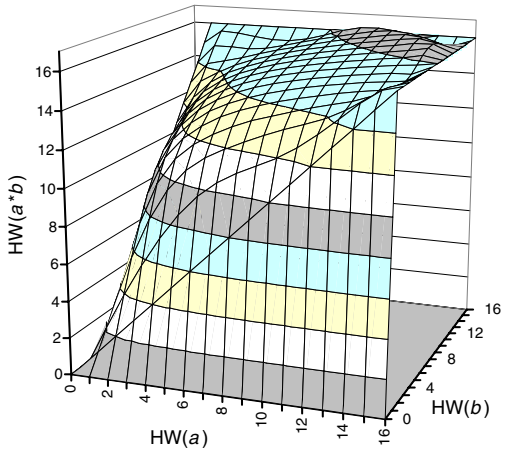

Figure 4. Average Hamming Weights of Products for a 16-bit Multiplier.

k up to $k = 32$. The same overall features were found between the surface for the model and that for the true values: steep sides when one Hamming weight is small, a large plateau region for the majority of values, and a small peak just before the value (k, k). The only main feature missing from the model is the slight depression near (k, k).

8.4 Extrapolated Results

It is now apparent that the Hamming weight model for the add-and-shift multiplier describes an overall level of data-dependent power consumption which has essentially the same properties for multipliers of any size. Total gate switching for weights $(0, 0)$ and (k, k) is listed in Table 5. In all cases where $k \leq 12$ the model gives values for $(0, 0)$ which are almost exactly 95% of the correct values, and so it is possible to predict the true values for $(0, 0)$ when $k > 12$. For (k, k) the values for the model seem to increase at a slightly faster rate than the actual values, thereby providing an upper bound on actual values above $k = 8$.

Switching for weights $(0, 0)$ always gives the minimal

k	4	5	6	7	8	12	16	24	32	48	64
Actual$(0,0)$	13.81	24.47	38.71	56.46	77.30	194.8	*353.5*	*822.8*	*1486*	*3395*	*6081*
Model$(0,0)$	13.13	23.24	36.70	53.66	73.54	186.9	353.1	842.0	1540	3563	6422
Actual(k,k)	40.12	68.23	103.05	144.40	193.14	456.3	*824.5*	*1887*	*3380*	*7662*	*13669*
Model(k,k)	39.57	67.65	102.65	144.40	193.46	458.6	834.7	1921	3453	7851	14030

Table 5. Minimum and Maximum Gate Switching Activity, namely that for Hamming Weights $(0,0)$ and (k,k). (*Predicted Values in italics*).

activity, and that for (k,k) gives the maximal activity. For all small k this coincides with the minimum and maximum for digit-by-digit averages. So we can assume that the table also accurately predicts minimum and maximum gate switching activity during any multiplication performed by a multiplier built according to the specification in Section 7 *when averaged over all possible initialisations*. Without such averaging the extremes are greater: for example, $a \times b$ followed by $a \times b$ should result in no gate switching at all.

As the total number of gates in the model multiplier is quadratic in k, namely $6k^2-8k$, it seems worth trying to fit a quadratic curve to the values in Table 5: gate switching for $(0,0)$ is approximated by $1.52k^2-1.99k-2.87$ in the actual multiplier, compared with $1.58k^2-3.45k+1.37$ in the Hamming weight model; and that for (k,k) is approximated by $3.37k^2-2.03k-5.79$ in the actual multiplier, compared with $3.46k^2-3.01k-3.88$ in the model. These functions fit within the bounds imposed by the hardware. As they also provide a very good fit to all ten (resp. 15) points used rather than just three (with about 3% relative error in the worst case), they were used to suggest approximations for the true values which were too costly to compute in Table 5. Thus, on average between about a quarter and just over a half of the gates are switched during each multiplication.

9 The Booth-Hamming Weight Multiplier

Following the above simplifications, a binary Booth multiplier was also built for simulating power consumption as a function of the Hamming weight of one input, and the Booth weight of the other, recoded input. However, the simplifications used for the add-and-shift multiplier failed to produce a sufficiently accurate model of the behaviour here. The problem seems to lie in the independence assumption: a sizeable proportion of the addends are not the random bits of the multiplicand (or its complement) but strings of all 0s or all 1s. Further research is merited here.

For the Booth-recoding multiplier behaviour is much more consistent over the re-coded input when measured using Booth weight instead of Hamming weight. Moreover, average gate switching for any pair of digits shows much less variance than for the add-and-shift multiplier. Consequently there is much less variation between the behaviour of different Hammming/Booth weight classes here than for the add-and-shift multiplier. So this multiplier should be more resistant to a weight-based side channel attack.

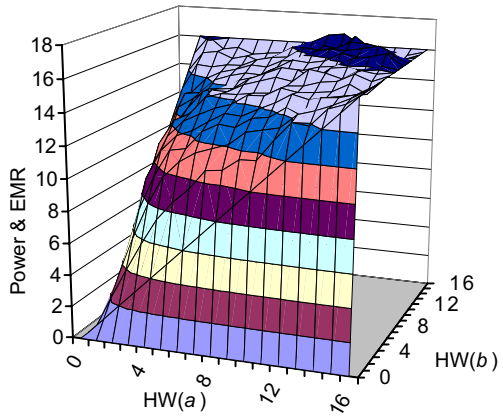

Figure 5. Combined Power & EMR Trace Results for 16-Bit Multiplier.

10 Laboratory Results

To verify the accuracy of our models in practice rather than compared with a simulation, power and EMR from several hardware multipliers were measured following practices described in [9, 10]. We had access to unpipelined add-and-shift 16- and 32- and 64-bit multipliers. Data was loaded in one cycle, the product computed in the next, and the output written to memory in the third. Our model applies to the middle cycle.

For each pair of Hamming weights, five random inputs a and b were chosen with those weights, and the current and electromagnetic traces of the multiplication cycle obtained after an unknown initialisation which we assumed was random. Thus there were 5^2 traces to be averaged for each Hamming weight pair. The traces consisted of 12-bit current and EMR measurements made at 35 times the (internal) clock frequency of the multiplier. They included information about the execution address and the last instruction ex-

ecuted, so this was removed using an accurate template. As the traces still included further noise from other sources, an average of two traces for each input pair was taken. Thus, the average of 2×35 points was taken as the data dependent output for each multiplication.

The results from the 16-bit case are illustrated in Fig. 5. As expected from the theory, there is a remarkable correspondence between this and the average Hamming weight of the products as illustrated in Fig. 4. An attacker with a side channel value can therefore predict a small set of possible Hamming weight pairs for the arguments using the Hamming weight multiplier above or perhaps reference values from the target multiplier itself.

11 Conclusion

By simulation, theoretical modelling and laboratory experimentation, it has been confirmed that the variation in power used by an add-and-shift multiplier and the EMR it generates are closely related to the Hamming weights of the inputs when averaged over all possible initial states. This is a potential weak point which attackers of embedded cryptosystems may try to exploit to extract secret keys.

A model was derived which determines power usage accurately from the Hamming weights of inputs. It executes in polynomial time with respect to the number of bits, contrasting markedly with the exponential time required to consider all input bit patterns in a full simulation. Hence it can be used to predict behaviour accurately in larger multipliers for which a full simulation is computationally infeasible.

The same construction principles may be applicable to build Hamming (and Booth) weight multipliers which will predict side channel leakage from any multiplier design. Research is on-going for a binary Booth recoding multiplier, but it was clear that gate switching exhibited less variation than in the add-and-shift case. Although more complex in construction, this seems to make such multipliers more resistant to side channel attack.

References

[1] D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi. The EM side-channels. In B. Kaliski, Ç. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 29–45. Springer-Verlag, 2002.

[2] R. Anderson and M. Kuhn. Tamper resistance - a cautionary note. *Proc 2nd USENIX Workshop on Electronic Commerce, Oakland, California*, pages 1–11, 18th-21st November 1996.

[3] R. M. Best. Microprocessor for executing enciphered programs. *US Patent 4,168,396*, 8th September 1979.

[4] M. Flynn and S. Oberman. *Advanced Computer Arithmetic Design*. John Wiley & Sons Inc., New York, 2001. ISBN 0-471-41209-0.

[5] K. Gandolfi, C. Mourtel, and F. Olivier. Electromagnetic analysis: Concrete results. In Ç. Koç, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 251–261. Springer-Verlag, 2001.

[6] P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In M. Wiener, editor, *Advances in Cryptology – Crypto '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer-Verlag, 1999.

[7] R. Mayer-Sommer. Smartly analyzing the simplicity and the power of simple power analysis on smartcards. In C. Paar and Ç. Koç, editors, *Cryptographic Hardware and Embedded Systems (Proc CHES 2000)*, volume 1965 of *Lecture Notes in Computer Science*, pages 78–92. Springer-Verlag, 2000.

[8] P. L. Montgomery. Modular multiplication without trial division. *Math. Computation*, 44:519–521, 1985.

[9] J.-J. Quisquater and D. Samyde. Electromagnetic analysis (EMA): Measures and counter-measures for smart cards. In *Smart Card Programming and Security (E-smart 2001)*, volume 2140 of *Lecture Notes in Computer Science*, pages 200–210. Springer-Verlag, 2001.

[10] J.-J. Quisquater and D. Samyde. Eddy current for magnetic analysis with active sensor. *Proc. e-Smart 2002, Nice*, pages 183–194, September 2002.

[11] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Comm. ACM*, 21:120–126, 1978.

[12] C. D. Walter. Sliding Windows succumbs to Big Mac attack. In Ç. Koç, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 286–299. Springer-Verlag, 2001.

[13] C. D. Walter and S. Thompson. Distinguishing exponent digits by observing modular subtractions. In D. Naccache, editor, *Topics in Cryptology – CT-RSA 2001*, volume 2020 of *Lecture Notes in Computer Science*, pages 192–207. Springer-Verlag, 2001.