Data Dependent Power Use in Multipliers

Colin D. Walter*

Comodo Research Laboratory
10 Hey Street, Bradford, BD7 1DQ, UK
colin@comodo.net

Abstract. Recent research has demonstrated the vulnerability of certain smart card architectures to differential power analysis when multiplier operations are insufficiently shielded from external monitoring. Here 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, and bears a very close relationship to the Hamming weights of the inputs.

Key words: Differential power analysis, smart card, multiplication, multiplier, RSA cryptosystem, data dependent power.

1 Introduction

Security is an increasingly important issue these days, even for computer arithmetic. Typical RSA [13] hardware performs long integer modular multiplications $A \times B \bmod M$ using a modification of the standard primary school method for decimal 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*. 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.

For such applications, the required implementations of public-key cryptography need to be exceptionally strong. Moreover, the possibilities for including

* Work done while the author was at UMIST, Manchester, UK.

physical tamper resistance, such as large capacitors to mask current fluctuations, are limited. Recently, work of Kocher [9] has drawn the public's attention to *simple* power analysis (SPA) and *differential* power analysis (DPA) as means of recovering secret keys from embedded cryptographic systems such as smart cards. Much was already known in this regard by government security agencies [1] and it forms a natural part of the on-going Tempest project [7] which concentrated on electro-magnetic radiation (EMR) from computing equipment and cables [4]. Other attacks of interest to readers, but not discussed here, include looking at timing variations arising usually from software branching [8], [14], [16]. A number of defences against such side channel leakage are mentioned by Anderson [1], including bus encryption [2], which cures perhaps the most major source of measurable, data dependent, power variation. Random transformations of inputs, randomising algorithms, random noise generators and concurrent calculations on another processor are some of the methods used to blind the calculations sufficiently for recovery of the secret key to become impossible during a deliberately limited life span.

Although Kocher's work concentrates on variation in the total power consumption of a smart card, Gandolfi *et al.* [6] have shown that it is possible to measure EMR at specific points in a chip. Since multipliers occupy a substantial part of most CPUs and cryptographic co-processors, they now become one of the major sources of possible data leakage which may need protection. On a chip with no security measures and little other than a multiplier, Sommer [10] observed a correlation between power traces and values by looping through a particular cycle of values. Here we investigate multipliers to see what power variations there are. The investigation confirms that attacks such as that presented in [15] 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, of course, new attacks may make the more usual 16- or 32- bit multipliers unsafe in the future.

Contrary to the widely held belief that longer keys mean stronger encryption, in the context of embedded cryptosystems where an attacker can measure side channel leakage directly, it often appears to be the case that longer keys lead to more leaked data per secret bit (in RSA there is on average a long integer square and half a multiplication for each exponent bit) and hence to a *weaker* cryptosystem [15]. One solution is to use a larger multiplier. Results here will enable the advantages of larger multipliers to be evaluated quantitatively.

The investigation is based on simulations which have computed gate switching activity in an accurate software morel of a multiplier. No cards were actually tested; the work of others such as [9] and [10] confirms the connection between power use and gate switching. This data rather than that from cards may be closer to what might be successfully measured non-invasively by EMR probes without unreasonable expense in the near future.

The main problem with exhaustive simulations is the combinatorial explosion: $O(2^{4k}k^2)$ for a k-bit multiplier. So, the simulations can be performed completely only for small multipliers. For the larger cases which are likely to be met in practice, less accurate random sampling has to be performed. This prompts the need for more efficient models. Therefore, for cases when Hamming weights are of interest, a model is constructed which predicts power use in only $O(k^6)$ time. This model is tailored to the precise construction of the target multiplier to give very accurate results.

The simulations enable some theoretical predictions to be made about gate switching activity in a multiplier. Their precision is verified by the accuracy to which our model predicts gate switching. In particular, it is possible to assume independence between various data bits in the multiplier even although they are highly dependent.

Quite apart from applications to the assessment of smartcard security, the following article may be useful as a starting point for taking a more theoretical approach to estimating the maximum (and minimum) power used by a multiplier in a single clock cycle, whether pipelined or not. As multipliers grow in size it becomes more and more difficult to determine the maximum gate switching activity and hence the maximum power needed during each clock cycle. As well as the critical path length, this influences the minimum clock cycle time which is possible without introducing errors due to some gates not having changed state before the end of the clock cycle.

2 Background

Most cryptographic processors make use of a single k-bit multiplier, where typically $k = 16$ in the past and $k = 32$ is becoming the norm. The long integers used in public-key cryptography are then represented using k-bit digits. Suppose M is the modulus for the crypto-system which an attacker wishes to break. Its digits will be denoted m_i. The range of the index i is not important, but typically an attacker needs at least a dozen or so, say $O(2^4)$, digits in the modulus for his methods to succeed, and this is roughly the minimum number which occur in practice.

The attacker's aim is to recover the factorisation of the modulus M or equivalent information. This should remain secret in an RSA cryptosystem. RSA uses modular exponentiation to encrypt, decrypt, sign and verify data [13]. The main cycle of the component modular multiplications $R \leftarrow (A\times B + C) \bmod M$ repeatedly performs the following task:

$$\begin{array}{rcl} R & \leftarrow & r\times R + a_i\times B; \\ q & \leftarrow & R \ \mathbf{div}\ M; \\ R & \leftarrow & R - q\times M; \end{array}$$

(or a similar more efficient variation, such as is given by Montgomery [12]) where R is used for holding the partial product, a_i is a digit of A and q is the digit multiple of M which must be subtracted to keep R within any required bounds. The first and last assignments break down into a number of sequential multiply-accumulate operations

$$r_i + r\times c \ \leftarrow\ r\times r_i + a\times b_i + c \qquad (1)$$

$$r_i + r\times c \quad\leftarrow\quad r_i - q\times m_i + c \tag{2}$$

on digits, where c is a carry digit and $r = 2^k$.

The hardware for these multiply-accumulate operations leaks data which an attacker can extract by observing variations in electrical current and electromagnetic radiation. The main question of interest here is whether or not it is possible for him to recover enough data to reconstruct either the secret exponent or the secret factorisation of the modulus. Most current attacks depend on the observation that both (1) and (2) make repeated re-use of the same data. In the case of (1), with the standard square-and-multiply or m-ary exponentiation methods, the multiplicand B is always a small power of the initial text. In the case of (2), the modulus M re-appears every time.

The background to the data dependent power variations of interest in the multiplier is the observation that when a gate output in the combinational logic is switched from low to high or high to low it momentarily uses more power than when remaining in a stable state. The main difficulty is that the power 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) and (2) provides just the opportunity for this. The main problem faced by implementors is the removal of the handles by which an attacker is able to isolate useful subsets of observations over which averaging proves productive: the handle used in [15] was based on the Hamming weight of digits.

With a process for which some data is fixed, it is possible that the average number of switched gates, and hence the mean power used, over a chosen set of clock cycles is characteristic of the fixed data. Then many observations of variations in the power consumption by the process may enable this average to be determined with sufficient accuracy for the value of the fixed data to be deduced correctly. Such averaging may take place over a set of consecutive cycles, over a set of non-adjacent cycles during a single exponentiation, or even over a set of related cycles observed during different exponentiations. Our interest here, then, is in averaged data from the multiplier and what might be deduced from that about its arguments or initial state. For convenience, we will assume that the powers consumed by switching AND, OR and XOR gates are all the same and identical whichever way they are switched, although in practice they will all be slightly different. Thus, to obtain an overall picture, it will suffice to count gates which are switched without any weighting for different power use.

3 Multiplier Simulation

A software simulation of a k-bit multiplier was built with variable k in order to verify claims by previous authors that gate switching activity was related to the Hamming weights of the inputs, i.e. to the number of input bits which are set to 1. The simulated circuit was constructed from a number of AND gates, 3-bit to 2-bit full adders (Figure 1) and 2-bit half adders in the standard way

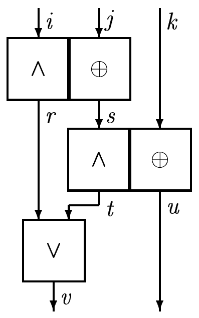

Fig. 1. A 3-bit Counter or Full Adder.

described below in §6. The number of gates switched between the registers was recorded, but no term was included for switching within the registers, or for the multiplexing of input bits which is required to feed the initial AND gates. As we will observe later, 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 this extra parameter.

Different designs were tried corresponding to the different groups of three bits which can be chosen for each full adder. Whilst some differences were noted, these were not substantial in terms of their effect on the simulation results. So similar results would probably hold whatever the construction of the multiplier. However, that said, our initial multiplier model was slightly simplified to make its realisation in software easier: in particular, more gates than are necessary were used. Its behaviour did differ from reality in several significant ways and so had to be re-built with greater accuracy. The final multiplier model used in the simulations accurately matched one that might be realised in hardware. Hence some care is necessary in building the simulation.

Multipliers are very standard and one can assume an attacker knows the detail of the one used well enough to adapt the results appropriately to his particular case. However, it actually suffices for the attacker to treat the target multiplier as a black box and measure power variations for chosen inputs in order to calibrate his processing of data from the target smartcard.

In the ideal simulation, the gates in the multiplier 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. In practice, this is only possible for small multipliers, and was done for all the results here with $k \leq 8$. For larger multipliers some simplifications need to be made. In particular, 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.

$a\times b$	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

Fig. 2. Average Gate Switching for $a\times b$ in a 3-bit multiplier ($k = 3$), sorted according to overall switching activity in each argument.

First, for small k ($k \leq 8$), the average number of gate changes from high to low or low to high were counted during the computation of each of the 2^{2k} digit products $a\times b$ after each of the 2^{2k} possible initialisations of the multiplier by a preceding multiplication. This was repeated for larger k, but only using random initialisation of the multiplier. These gate switch averages were themselves averaged for a given a as the first argument and random second argument, and for a given second argument b with any random first argument. 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 (Fig. 2) and graphs (Fig. 3) for comparison purposes.

Interestingly, these two orders are very slightly different (see Fig. 2 and the Appendix), showing that the multiplier was not quite symmetrical regarding the two inputs. In a typical unsophisticated multiplier, the inputs a and b are used to form k^2 bit products given by ANDing one bit from the multiplicand b with one bit from the multiplier a. As in the primary school method for multiplying decimal numbers, these are treated as k binary numbers by keeping together the bits formed from the product of b by a bit of a. They are then added together. The 3-bit counters are arranged in rows in the multiplier so that sets of 3 such binary numbers (always linear combinations of the multiplicand) are added by one row of counters, reducing them to two binary numbers. The construction processes multiples of the multiplicand b uniformly whilst unavoidably mixing up some of multiples of the multiplier a. As a result of the inevitable lack of symmetry between the arguments of the multiplier, we distinguish between the multiplicand, always the second argument (b here), and the multiplier, which is always the first argument (a here). These effects are less pronounced as k is increased because a smaller proportion of input bits are subject to the re-arrangements needed to obtain three inputs for each full adder. Unfortunately, space limitations make a larger illustration impossible here.

It is the order of the digits according to the number of gates switched which is of interest here. We write $a_0 <_{gc} a_1$ when the average gate count for digit a_0 is less than that for a_1 in a particular, specified situation. Not only was it verified that the gate count is indeed closely related to the Hamming weights of the arguments a and b, but more detailed patterns emerged for the ordering of individual digits according to these gate switching counts.

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

This held without exception for either argument 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 small k there is a clear jump in the average number of gates switched when the Hamming weight of an argument is increased.

More detailed ordering of digits with equal Hamming weight followed several rules-of-thumb for the majority of cases. However, the requirement to group bits in sets of three for the full adders means that the rules were not always followed. For example, the next two "guidelines" held for around 75% and 85% respectively of all cases $k \leq 8$:

– **I.** Suppose a_0 and a_1 are odd digits with $\textsc{hw}(a_0) = \textsc{hw}(a_1)$ and $a_0 < a_1$. Then usually $2^i a_0 >_{gc} 2^j a_1$ for $i, j \geq 0$.

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

For the current example of $k = 3$, these imply the additional relations $5 <_{gc} 3 <_{gc} 6$ and $1 <_{gc} 2 <_{gc} 4$. They are readily verified to hold for the data in Figure 2 and, with the Hamming Weight Principle, they provide a total ordering of all the digits when $k = 3$. Additional data for small k are included in the Appendix, but a comparison with the next case is worthwhile:

– For $k = 4$, the order is 0, 1, 2, 4, 8, 3, 9, 5, 10, 6, 12, 11, 7, 13, 14, 15 for the multiplier argument and 0, 1, 2, 4, 8, 9, 3, 5, 10, 12, 6, 13, 7, 11, 14, 15 for the multiplicand argument.

Here, the order varies slightly between multiplier and multiplicand, but the Hamming Weight rule still holds. The second rule-of-thumb always holds also, but the first fails in a quarter of cases, namely for the pairs (3,5), (3,9), (3,10), (7,13) and (11,13) of multipliers and pairs (3,5), (3,10) and (7,11) of multiplicands. A further principle relates the order of digits as k is increased when the hardware is constructed in the same way:

– **III.** If $a <_{gc} b$ in k-bit arithmetic, then usually $a <_{gc} b$ in $(k+1)$-bit arithmetic.

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

varies. Thus, for $k = 3$, 8 binary numbers must be added together. The first two rows of full adders reduce the first 6 numbers (which are bit multiples of b) to 4 leaving 2 to be added in later rows. However, for $k = 4$ the same process leaves just 1 initial number to be added in later rows.

In conclusion, it is clear that there are general principles for any multiplier which determine a broad ordering of the digits which corresponds to the order given by average gate switching, but the detailed order is fixed only once the multiplier is known quite precisely.

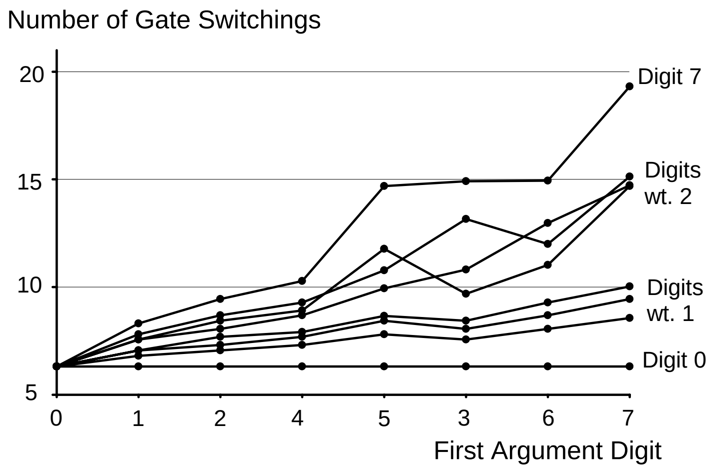

Fig. 3. Gate Switching Frequency Graphs for the Second (Multiplicand) Argument of a 3-Bit Multiplier.

For each second argument digit (the multiplicand) the number of gate changes can be plotted as a function of the first argument (the multiplier). Figure 3 shows this for $k = 3$, with the multiplier digits ordered according to increasing average gate counts. This case is typical. The grouping according to Hamming weight is quite clear, and just as pronounced for all the cases which could be fully computed (i.e. $k \leq 8$). However, within each group, the digits have individual characteristics: this is particularly noticeable here for those of Hamming weight 2. This shows that there are higher order properties than just the average gate counts which might be exploited from the statistics in order to deduce an unknown argument in a digit product.

4 Hamming Weights

The gate count results for the digit products $a\times b$ can be translated easily into statistics which relate Hamming weights simply by averaging over all digits with the same Hamming weight. The table in Figure 4 provides such data for 8-bit arguments. Figure 5 illustrates these data as a surface in 3-dimensional space.

$a\times b$	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

Fig. 4. Gate Switching Activity for $a\times b$ in an 8-bit multiplier, averaged over arguments with the same Hamming Weights.

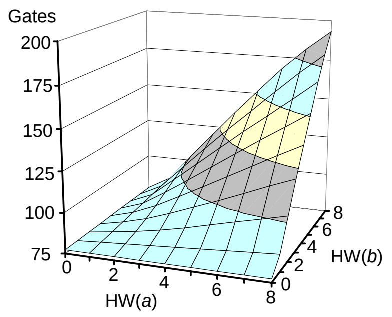

Fig. 5. Surface illustrating Gate Switching Activity in an 8-bit multiplier as a function of the Hamming Weights HW(a) and HW(b) of the inputs.

From these data it is clear that there is a very close 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. 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. Unfortunately for the attacker of an embedded RSA cryptosystem, and fortunately for the security of such a product, 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 k makes life still harder for the attacker.

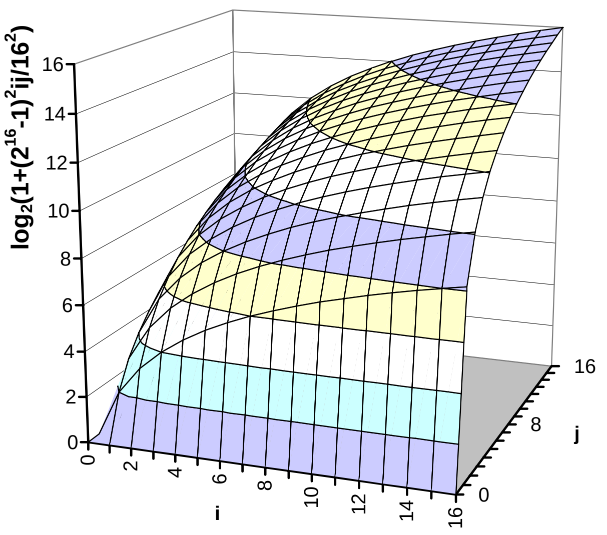

Fig. 6. The function $\frac{1}{2}\log_2(1+(2^k-1)^2 ij/k^2)$ for $0 \le i, j \le k = 16$.

We will see shortly that gate switching activity depends closely on the difference in the number of bits which are set to 1 between input and output. Hence the Hamming weight of a product becomes interesting:

– When averaged over all k-bit values of a and b with given Hamming weight, the Hamming weight of $a \times b$ is close to linear in $\log\{1+\mathrm{HW}(a)\mathrm{HW}(b)\}$ where HW is the Hamming weight function.

This is illustrated graphically in Figure 6. An overview of the justification for it is as follows. The average value for digits of given Hamming weight increases with the weight. So the average value of a product increases as the Hamming weight of its arguments increases. As average Hamming weight increases with increase in value, the average Hamming weight of this product should increase also. In detail, the average value for k-bit digits of Hamming weight i is $(2^k-1)i/k$ (since this is the average for each set of distinct cyclic permutations of a k-tuple of bits for which i bits are 1). Since the average of the products of numbers from two sets is the product of the averages of the two sets, the average value for the product of digits with weights i and j is $(2^k-1)^2ij/k^2$. However, the average Hamming weight of an integer x is approximately $\frac{1}{2}\{\log_2(e)+\log_2(1+x)\}$ (because the integral of this function from $x=0$ to $x=2^k-1$ is the sum of the Hamming weights for such integers x, namely $2^{k-1}k$). Thus,

– we expect the average Hamming weight for the product of digits with weights i and j to be approximately

$$\frac{1}{2}\{\log_2(e)+\log_2(1+(2^k-1)^2ij/k^2)\}\ .$$

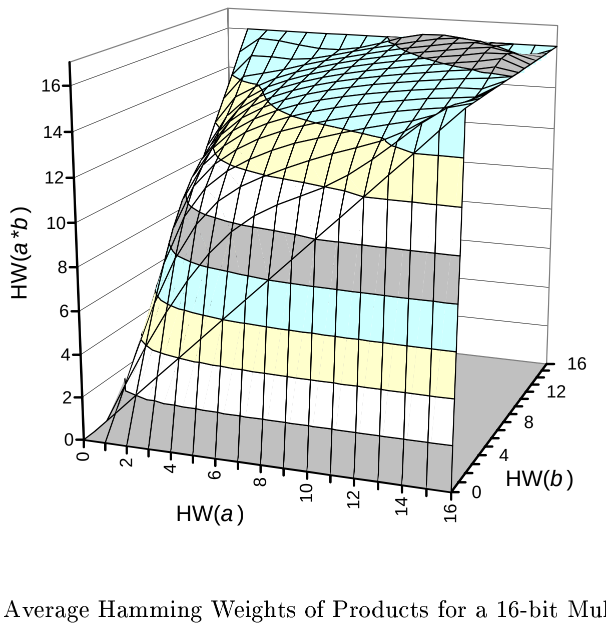

Fig. 7. Average Hamming Weights of Products for a 16-bit Multiplier.

This is essentially what the claim says. For the larger classes of digits with a given Hamming weight, i.e. for the middle range of Hamming weights, it is reasonable to assume random effects to be sufficiently numerous to justify acceptance of this approximation to the average weight of the product. However, for small and large input Hamming weights discrete effects become more noticeable. The discrepancies can be seen when accurate weights are computed.

Figure 7 illustrates the true average Hamming weight of the product of two 16-bit numbers of given Hamming weight. It is typical of the general case for k-bit numbers. There is a reasonable correspondence between it and Figure 6, which illustrates the approximating assumption above. In Figure 7, most digit values give rise to products in the sloping plateau region away from small Hamming weights. The diagram shows that the average Hamming weight of $a{\times}b$ normally increases slowly as the Hamming weight of a or b is increased, and it is normally just below k. However, when a or b has a small Hamming weight, there is a sharp increase in weight of the product as the smaller weight argument increases from weight 0. Furthermore, when both a and b have weights approaching k there is first a little peak with the product weight above k but after that there is a small depression to below k before the final increase to k.

5 3-bit Counters

The above now makes it possible to use the number of input bits set to 1 to estimate the number of output bits which are set to 1 in the multiplier. This can be used to calculate the expected number of 3-bit counters which reduce their number of bits set to 1 from input to output. From that, the approximate number of data-dependent gate switchings can be computed. Consequently, the Hamming weights of the inputs do indeed determine the average Hamming weight of the output in the way expected above and as illustrated in Figures 4 and 5. Unfortunately, this simple overview needs some careful analysis and extra detail before it reflects reality closely enough to be usable.

First we need to look at 3-bit counters in detail. The probability distribution of bits supplied to these full adders varies throughout a multiplier, and this affects the number of gates that are switched. The next theorem describes how a single counter both experiences a data-dependent amount of gate switching activity and also affects the probabilities of bits as they pass through. When such counters are combined in a multiplier, we can see that interesting data-dependent activity will occur near the inputs, but then the effect of any uneven distribution of input bits dissipates lower down in the multiplier so that eventually, near the output, 0s and 1s turn up with roughly equal probability.

Theorem 1. *In a* 3*-bit full adder, let* δ *be the difference between the number of bits set to* 1 *at input and the number of bits set to* 1 *at output. Then,*

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

ii) *On average, if all inputs are independently and uniformly distributed then 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*. If the next inputs are* 1 *independently with* *probability p' then the average number of gates switched when $\delta = 0$ is*

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

and the average number of gates switched when $\delta = 1$ is

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

iv) *For the situation described in* (iii)*, the average difference in the number of gates switched between the 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, and* c) *positive for all p less than approximately* 0.6*.*

Proof. i) The adder inputs three bits representing three units and outputs the sum as a 2-bit binary number. The Hamming weight of the input is the value of the output: the 3-bit adder counts the number of input bits which are set to 1. So inputs with Hamming weights 0, 1, 2 and 3 yield outputs with Hamming weights 0, 1, 1 and 2 respectively. For the first two, there is no difference in Hamming weights between input and output, but for the second two there is a difference of 1. The difference in Hamming weights occurs exactly when the output bit from the OR gate (Figure 1) is set: it performs the combination of two bits.

Let p be the probability of an input 1 and $q = 1{-}p$ that of 0. Then, for independent inputs, the probability of a reduction in Hamming weight from input to output is the probability of the input being 2 or 3, namely $p^3{+}3p^2q$. This is the probability of an output 1 from line v. The probability of an output 1 from line u is the probability of the input being 1 or 3, namely $p^3{+}3pq^2$.

ii) We will write full adder inputs ijk following the left-to-right order presented in Figure 1. Interchanging the values of i and j makes no difference to the switching of any gates or to their outputs. This means we can combine some cases, namely the pairs of inputs $01x$ and $10x$ for $x = 0$ and $x = 1$. The state of the adder will be represented by the outputs of the five gates taken in the order r, s, t, u, v as defined in Figure 1. So inputs 000, 001, 010/100, 011/101, 110 and 111 respectively generate the following six states s_i $(0 \le i \le 5)$ with probabilities p_i:

$$\begin{array}{lll}
s_0 = 00000 & & p_0 = q^3 \\
s_1 = 00010 & & p_1 = pq^2 \\
s_2 = 01010 & & p_2 = 2pq^2 \\
s_3 = 01101 & & p_3 = 2p^2q \\
s_4 = 10001 & & p_4 = p^2q \\
s_5 = 10011 & & p_5 = p^3
\end{array}$$

The 3-bit counter will always be in one of these six states. However, the above probabilities are for the state of the adder *after* the input of data for which 1 has probability p. Previous and future states may be determined by different values of p derived from other data.

To obtain the number of gates changed, the current 5-bit state vector has to be XORred with the previous state vector, and the bits counted in the result. For a given new state s'_i, this is averaged over all immediately preceding states s_j to obtain the average number g_i of gates changed for a given input. All cases are computed in the same way: $g_i = \sum_{j=0}^{5} \mathrm{HW}(s'_i \oplus s_j) p_j$ is the average number of gates changed given that state s'_i is the final state and the p_j are the initial probabilities for the states s_j.

One of these, g_2, is calculated here for illustration. The reader is invited to check the others if he so wishes. Input 010/100 creates state 01010 and so causes 2, 1, 0, 3, 4 and 3 gate changes respectively from the six possibilities for the preceding state. The average number of gate changes for input 010/100 is therefore $2{\times}q^3 + 1{\times}pq^2 + 0{\times}2pq^2 + 3{\times}2p^2q + 4{\times}p^2q + 3{\times}p^3 = 2q^3{+}pq^2{+}10p^2q{+}3p^3$. The numbers for all states are, in order,

$$\begin{array}{rcl} g_0 &=& 5pq^2{+}8p^2q{+}3p^3 \\ g_1 &=& q^3{+}2pq^2{+}11p^2q{+}2p^3 \\ g_2 &=& 2q^3{+}pq^2{+}10p^2q{+}3p^3 \\ g_3 &=& 3q^3{+}10pq^2{+}3p^2q{+}4p^3 \\ g_4 &=& 2q^3{+}11pq^2{+}6p^2q{+}p^3 \\ g_5 &=& 3q^3{+}8pq^2{+}9p^2q \end{array}$$

In the case of $p = q = \frac{1}{2}$ when 0s and 1s are equally likely, this gives 2, 2, 2, 2.5, 2.5 and 2.5 gates respectively for the six states. The first three of these represent inputs with Hamming weights of 0 or 1 and therefore there is no change in Hamming weight from input to output. The last three represent the inputs with Hamming weights of 2 or 3 and therefore there is a change of 1 in the Hamming weight from input to output. Hence the number of gates switched *does* indicate a reduction in Hamming weight by the counter.

iii) Suppose now that the full adder has been initialised with data for which 1 occurs with probability p and that data is supplied for which 1 occurs with probability p'. Define quantities p'_i for p' analogously to the p_i above. In general, the average number of gate changes when there is no reduction in 1 bits from input to output is then $(\sum_{j=0}^{2} p'_j g_j)/(\sum_{j=0}^{2} p'_j) =$

$$\frac{(5pq^2{+}8p^2q{+}3p^3)q'^3{+}(q^3{+}2pq^2{+}11p^2q{+}2p^3)p'q'^2{+}2(2q^3{+}pq^2{+}10p^2q{+}3p^3)p'q'^2}{q'^3 + 3p'q'^2}$$

$$= \frac{(5pq^2{+}8p^2q{+}3p^3)q' + (5q^3{+}4pq^2{+}31p^2q{+}8p^3)p'}{q' + 3p'}$$

$$= p(5q{+}3p)(q{+}p) + \frac{p'}{q'{+}3p'}(q{-}p)^2(5q{-}p)$$

which simplifies to the expression given in the statement of the theorem. Similarly, the average number of gate changes when there is a reduction in 1 bits from input to output is $(\sum_{j=3}^{5} p'_j g_j)/(\sum_{j=3}^{5} p'_j)$

$$= \frac{(8q^3{+}31pq^2{+}12p^2q{+}9p^3)q'{+}(3q^3{+}8pq^2{+}9p^2q)p'}{3q' + p'}$$

$$= \quad (3q^3{+}8pq^2{+}9p^2q) + \frac{q'}{3q'{+}p'}(q{-}3p)^2(p{-}q)$$

iv) These two expressions have been written as a sum of two terms, one of which is independent of p' and the other of which has factors such as $p{-}q$ and a denominator which make it relatively small. Thus, these two expressions have only small dependency on the value of p' except for values close to 0 or 1. By taking $p' = q' = \frac{1}{2}$, a quite reasonable approximation is obtained for the two functions at most points. For this choice of p', the difference between the expressions for $\delta = 0$ and $\delta = 1$ is quite close to being linear in p, taking the values 1.5 gates for $p = 0$, 0.5 gates for $p = 0.5$ and -0.5 gates for $p = 1$. As is easily verified from graphing software (Figure 9), the greatest discrepancies occur for p and p' both large or both small. Such software demonstrates the claims in (iv).

Using the binary cut method, it is straight-forward to locate the point at which the difference between the two functions is zero. This turns out to be at approximately $p = 0.6$ for all values of p'. □

We will be interested mainly in cases with $0.25 \leq p, p' \leq 0.5$ for which the gate difference is clearly positive. Specifically, the initial AND gates in a multiplier, which provide bit products to a tree of full adders, generate bits with average only $\frac{1}{4}$ and this average increases towards $\frac{1}{2}$ further down the tree. Thus, when data flows through the tree of full adders in a multiplier, its distinguishing characteristics tend to disappear:

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

ii) *Take any initial value for* p *and assume the output bits from* 3*-bit counters are independent. If bits are repeatedly fed randomly from one row to another 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. i) The previous proof contained the probabilities p_i of each final state s_i. (These are entirely independent of the previous state of the counter.) The probabilities of a 1 on each of the output lines u and v are then $p_1{+}p_2{+}p_5$ and $p_3{+}p_4{+}p_5$ respectively, i.e. $p^3{+}3pq^2$ and $p^3{+}3p^2q$. Because it is unknown whether the bit comes from u or v, these are averaged with equal weighting here to give $p^3{+}\frac{3}{2}p^2q + \frac{3}{2}pq^2$. The expression is then simplified using $p{+}q = 1$.

ii) For p in the interval $[0,1]$, $p < p^3{+}\frac{3}{2}pq \iff p^2{+}\frac{3}{2}(1{-}p){-}1 > 0 \iff (1{-}p)(\frac{1}{2}{-}p) > 0 \iff p < \frac{1}{2}$. So $p < p^3{+}\frac{3}{2}pq$ for $p < \frac{1}{2}$ and $p > p^3{+}\frac{3}{2}pq$ for $p > \frac{1}{2}$. Similarly, $p^3{+}\frac{3}{2}pq < \frac{1}{2} \iff p^3{+}\frac{3}{2}p(1{-}p){-}\frac{1}{2} < 0 \iff (p{-}\frac{1}{2})((p{-}\frac{1}{2})^2{+}\frac{3}{4}) < 0 \iff p < \frac{1}{2}$. So $p^3{+}\frac{3}{2}pq < \frac{1}{2}$ for $p < \frac{1}{2}$ and $p^3{+}\frac{3}{2}pq > \frac{1}{2}$ for $p > \frac{1}{2}$. Let p_i be the probability of an output 1 from the ith row of counters and let $p_0 = p$. Then, using the assumptions, $p_{i+1} = {p_i}^3{+}\frac{3}{2}p_iq_i$ for $q_i = 1{-}p_i$ and $i \geq 0$. The inequalities establish that the sequence p_i $(i = 0, 1, 2, ...)$ will be monotonic and, because such sequences increase from below $\frac{1}{2}$ but decrease from above $\frac{1}{2}$, it must have limit equal to $\frac{1}{2}$. □

Table 8 shows how the average difference in the number g of gates switched between the cases $\delta = 0$ and $\delta = 1$ decreases down a branch in the tree of full adders when starting at $p = \frac{1}{4}$. The lemma provides the sequence of values for p through applying part (i) repeatedly to obtain the probabilities for the data supplied to successive full adders. For each counter, previous data is assumed to have the same value of p as incoming data. The gate differences are then computed by taking $p' = p$ in part (iii) of the theorem.

j	0	1	2	3	4	6
$p^{(j)}$	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

Fig. 8. Average Probability of a 1 and the extra Average Gate Switching when a Bit is removed from jth counter in a sequence of 3-bit Adders.

These values clearly illustrate the diminution of data dependency as distance from the inputs increases down the multiplier. For each sequence of 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).

Although Theorem 1 can be used to perform direct numerical calculations for the size of standard multipliers encountered in practice, the following approximations make some general trends easier to see as well as providing upper and lower bounds for what happens:

Theorem 2. *Suppose a tree of* 3*-bit full adders is constructed so that the outputs from each row are fed randomly into the next row. Suppose also that the tree is given random, independent input bits which are* 1 *with probability* p*. Then, for a full adder at depth* $i{+}1$ *in the tree,*

i) *the probability of the input data bits being* 1 *is further away from* $\frac{1}{2}$ *than* $\frac{1}{2} - (\frac{3}{4})^i(\frac{1}{2}-p)$ *always; but is nearer than* $\frac{1}{2} - (\frac{13}{16})^i(\frac{1}{2}-p)$ *when* $|\frac{1}{2}-p| < \frac{1}{4}$;

ii) *the probability of the counter reducing the number of bits set to* 1 *is at least* $\frac{1}{2}-\frac{3}{4}\alpha^i(1-2p)+\frac{1}{4}\alpha^{3i}(1-2p)^3$ *where* $\alpha = \frac{3}{4}$ *for* $p \geq \frac{1}{2}$, *and* $\alpha = \frac{13}{16}$ *for* $\frac{1}{4} \leq p \leq \frac{1}{2}$; *the probability is also bounded above by this expression with* $\alpha = \frac{3}{4}$ *when* $p \leq \frac{1}{2}$, *and* $\alpha = \frac{13}{16}$ *when* $\frac{1}{2} \leq p \leq \frac{3}{4}$;

iii) *over a distance of one full adder in the tree, the difference between the average gate switching activity for the adder and that for input with* $p = \frac{1}{2}$ *drops by approximately* 25%*.*

Proof. i) We have already seen from Lemma 1 that data going into a gate with probability p of being 1 will emerge with probability $p^3+\frac{3}{2}pq$ of being 1. It is convenient to rewrite this in terms of $\pi = \frac{1}{2}-p$: input data which are 1 with probability $\frac{1}{2}-\pi$ yield output data which are 1 with probability $\frac{1}{2}-\frac{3}{4}\pi-\pi^3$. So, after i iterations corresponding to passing through i full adders in succession, the data are 1 with probability $\frac{1}{2}-(\frac{3}{4})^i\pi-\pi f^{(i)}(\pi^2)$ for some polynomial $f^{(i)}$ which has non-negative coefficients. Since the terms in π all have the same sign, deleting any terms will result in a value closer to the constant term $\frac{1}{2}$. In particular, $\frac{1}{2}-(\frac{3}{4})^i\pi$ is closer. Hence the first claim in (i) holds.

By choosing κ such that $\frac{1}{2}-\frac{3}{4}\pi-\pi^3$ is between $\frac{1}{2}-\kappa\pi$ and $\frac{1}{2}$ and noting that applying the function $p \mapsto p^3+\frac{3}{2}pq$ to each of these three values preserves their relative order, the second claim of (i) can be satisfied. This requires $\kappa \geq \frac{3}{4}+\pi^2$. Since $|\pi| \leq \frac{1}{4}$, we choose $\kappa = \frac{3}{4}+(\frac{1}{4})^2$.

ii) From Theorem 1(i), the probability of reducing the number of bits in a gate whose inputs are 1 with probability p is p^3+3p^2q, i.e. $3p^2-2p^3$. Its derivative $6p(1-p)$ is non-negative over the whole interval $[0,1]$. Hence, an under-estimate for the value of p will lead to an under-estimate for the probability of a reduction in the number of 1 bits, and similarly for over-estimates.

When p has the form $\frac{1}{2} - \alpha^i(\frac{1}{2}-p)$, the expression $3p^2-2p^3$ is equal to $\frac{1}{2}-\frac{3}{4}\alpha^i(1-2p)+\frac{1}{4}\alpha^{3i}(1-2p)^3$. Part (i) provides a lower bound by taking $\alpha = \frac{3}{4}$ for $p > \frac{1}{2}$, i.e. for negative π, and an upper bound by taking $\alpha = \frac{3}{4}$ for $p < \frac{1}{2}$, i.e. for positive π. These roles are reversed by taking $\alpha = \frac{13}{16}$ instead in the region $\frac{1}{4} \leq p \leq \frac{3}{4}$.

iii) Theorem 1(iii) enables the gate switching activity to be computed. If the expressions there are written in terms of π with $p' = p$ and the denominators written as power series, then the gate switching activity is obtained as a power series in π for which the constant is the gate switching activity at $\pi = 0$, i.e. at $p = \frac{1}{2}$, and the coefficient of π is non-zero. Over one full adder, the value of π is reduced by a factor of approximately $\frac{3}{4}$. For smaller π, this will be increasingly close to the factor by which the power series, less the constant, is reduced. For larger π, Table 8 shows that the other terms do not dominate the calculation, as well as showing the accuracy of the result for small π. □

Remark 5.1 Another valuable insight into gate switching activity is provided by graphing the difference between the functions given in Theorem 1 (iii). This is done in Figure 9. Except for extreme values, the dependence on p' is minimal; the greatest dependence is on p. Thus, when the number of bits set to 1 is reduced, the number of gates switched in a counter will depend mostly on the previous data, not the current data. The power variation due to the current data will therefore reside mostly in the number of counters whose bit count is reduced.

Conversely, although the multiplier may demonstrate a power variation due to the arguments it is currently using, it might also have a useful and interesting data dependent variation as a result of the inputs to the immediately previous multiplication which initialised it. Specifically, we should also consider the average gate switching activity during a random multiplication which takes place immediately after the multiplier is initialised by a product $a\times b$ where a is now

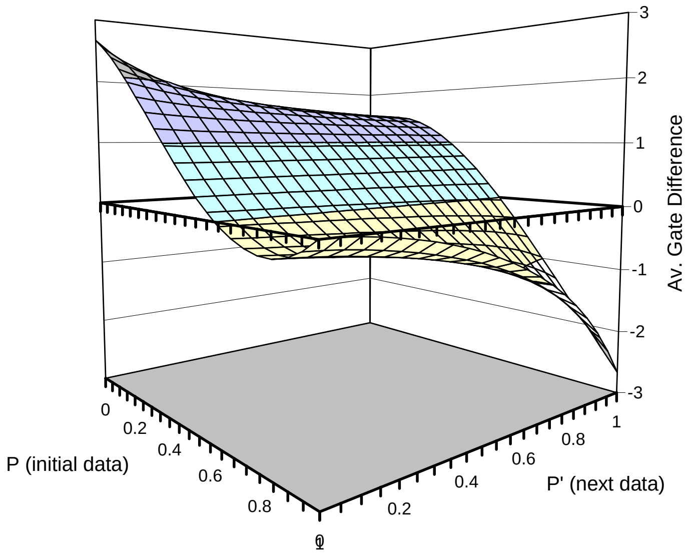

Fig. 9. Average Difference in Gate Switching when the 3-bit Counter removes a Bit, given as a Function of Initial and Following Data.

fixed and b fixed or random. To study this, the roles of the current and previous products just need to be interchanged. This was done in all appropriate simulations. However, identical results to those in Section 3 were obtained: obviously, the gates switched in going from a computation of $a\times b$ to a computation of $a'\times b'$ are exactly the ones which are switched in going from $a'\times b'$ back to $a\times b$ – the gates are simply switched in the opposite direction. So identical results were inevitable.

Of course, if we were to assume different power consumption figures for switching gates from high to low than vice versa then power would vary between the two cases according to the difference between the number of gates switching in each direction. This line of investigation is not pursued any further here, but it may provide an attacker with additional opportunities to extract useful data.

6 The Multiplier Model

We will assume a simple implementation for hardware multipliers which has no special requirements such as pipelining or shortened critical path. So, in order for the multiplier to compute $a\times b$, bits from the two arguments are first ANDed together to create k binary integers which are the product of the multiplicand b by a multiplier bit a_i. This requires some initial multiplexors to generate k

copies of each input bit. The k^2 bit products are then sent to a number of rows of 3-bit counters which add together bits representing the same power of 2. Each row takes a set of bits which, in effect, represents three binary integers and transforms it into a set of bits which represents two binary integers. After at most $k-2$ rows of counters, the initial k binary integers have been reduced to just two bits for each power of 2. Since one of the outputs from each full adder is a carry up, carry propagation is taken care of automatically, leaving just one or two output bits from each digit slice. If two such bits are generated, a 2-bit "half adder" is used to reduce this to a single bit plus a carry, yielding the required $2k$-bit output of the multiplier.

The horizontal view of the multiplier just presented is useful for structuring purposes. A vertical view enables one to count the components more easily. The recognisable diamond shape of a multiplier comes from aligning the $i+1$ or $2k-(i+1)$ initial bit products which are coefficients of 2^i. They form a bit slice which performs all the computations associated with 2^i. Each bit slice has inputs from the AND gates, generates one output result bit, but also receives carries from the previous bit slice and sends carries up to the next bit slice.

For bit slice 0, there are no gates after the AND gate. For the ith bit slice, with $0 < i < k$, the initial $i+1$ AND gates form products from pairs of input bits. There are a further $i-1$ inputs to the bit slice which are the carry bits from bit slice $i-1$. These bits are added together using $(i-1)$ 3-bit counters and a single 2-bit counter which together generate i carries which are output to the $i+1$st bit slice. Each 3-bit counter reduces the number of bits by 1, so all the counters reduce the total number of bits being processed by the slice from $2i$ to $i+1$. As i of these are carries up, a single output bit remains, which is part of the answer.

At the slice for 2^k there are only $k-1$ inputs from the AND gates and $k-1$ carries in from the previous slice. So these are added by $(k-2)$ 3-bit counters and a 2-bit counter which generate $k-1$ carries up to the next slice and one output bit.

Then, for bit slices $2k-i$ where $0 < i < k$, an initial $i-1$ AND gates form products from pairs of input bits. There are a further i inputs to the bit slice which are the carry bits from bit slice $2k-i-1$. These bits are added together using $(i-1)$ 3-bit counters. This generates $i-1$ carries which are output to the $2k-i+1$st bit slice, and a single output digit which is part of the result.

To minimise critical path lengths, as many bits as possible are added together in each row of the adder tree. For a digit slice with i inputs from the AND gates there are ideally $i/3$ full adders in the first row. These generate $i/3$ carries up to the next digit slice and $i/3$ inputs to the next row of full adders. The previous digit slice, with almost the same numbers of inputs and outputs, generates $i/3$ carries from its first row, and these provide a further $i/3$ inputs to the second row. So there are $2i/3$ inputs to $2i/9$ full adders in the second row. The process continues in the same way with $i2^j/3^{j+1}$ adders in the jth row ($j = 0, 1, ...$) of that bit slice. Of course, minor adjustments need to be made to this because there must be a whole number of adders at each level and also the number of carries in or out does not quite match the number of inputs from the AND gates. However, the first row, which contains about one third of all the adders, receives inputs only from the AND gates. Thereafter, on the whole, subsequent rows receive inputs which are all from the immediately preceding row, half being from the current slice and half being carries from the preceding slice.

7 An Unsuccessful Hamming Weight Multiplier

The multiplier 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, and because each multiplication has time order $O(k^2)$. This makes it infeasible to model 16-bit or larger multipliers. To investigate behaviour in terms of Hamming weights, it is necessary to built a multiplier which computes gate switching activity just from the Hamming weights of a, b, c and d. This should have time complexity $O(k^6)$ so that any expected size for a multiplier can be processed.

The first attempt at such a Hamming weight multiplier ignored the specific construction details which placed an integral number of full adders in each row within the adder tree of each bit slice. The following approximation was made for simplicity:

– *Non-Integrality Simplification*: For $i \geq 0$ assume the ith row in the multiplier contains exactly $k^2 2^i/3^{i+1}$ full adders.

In addition, many arguments were made easier by assuming the pairwise independence of the bits which are input to each full adder. That is, it was helpful to assume:

– *Indpendence Assumption*: All inputs to a row of full adders are independent with the same probability of being 1.

This is the key simplification that was made in order to overcome the combinatorial explosion that occurs when looking at larger, unstructured multipliers. In practice the assumption is absurd because there are only $2k$ bits fed into the multiplier but k^2 bits which are generated from them and have to be processed by the $O(k^2)$ full adders. Moreover, strong dependencies can persist because outputs from the AND gates are usually fed to sets of full adders in groups which correspond to the multiplicand times a multiplier bit, and such grouping continues down the hardware circuit. However, whether or not this simplifying assumption of independence is reasonable was determined by comparing the model which uses it with the original digit simulation.

7.1 The AND Gates

As earlier in Theorem 1, let undashed characters such as p to denote probabilities during the initialisation phase and dashed characters such as p' for probabilities during the execution phase.

The multiplier is initialised by computing $c{\times}d$ and then executes $a{\times}b$ during which gate switching is counted. The probability of a bit of a being 1 is $\frac{1}{k}\text{HW}(a)$, and the probabilities for b, c and d can be similarly expressed in terms of their Hamming weights. Denote these probabilities for a, b, c and d by p', q', p and q, respectively.

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, when averaged over all initial conditions ($p = q = \frac{1}{2}$), the power variation is constant. So we need not concern ourselves further with this.

On average, the AND gates at the top of the multiplier will be initialised with their outputs set to 1 with probability pq. Then, when $a{\times}b$ is computed the AND gates output a 1 bit 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.

This is a very reliable component of the total variation in gate counts, unsullied by independence assumptions of unproven merit: without making any use of such additional assumptions, exactly $\mathrm{HW}(a)\mathrm{HW}(b)$ gates produce outputs of 1, and each gate has exactly probability $k^{-2}\mathrm{HW}(c)\mathrm{HW}(d)$ of being set to 1 during initialisation. Hence the average count for these gates is precise. It is a component which can be usefully separated out from calculations when trying to understand the behaviour and effects of the 3-bit counters.

This reveals a substantial part of the Hamming weight dependency that we expected. In fact, there are just $k(k{-}2)$ 3-bit counters, at most $\mathrm{HW}(a)\mathrm{HW}(b)$ of them can reduce their bits totals, and, moreover, each reduction causes a data dependent increase in the order of 1.2 gates[1] . This yields about $1.2\mathrm{HW}(a)\mathrm{HW}(b)$ gate switches from the counters for an average initialisation where $p = q = \frac{1}{2}$, compared with $\frac{1}{4}k^2{+}\frac{1}{2}\mathrm{HW}(a)\mathrm{HW}(b)$ from the AND gates. So, without the benefit of the forthcoming analysis, we expect that the variation for AND gate switches will normally account for about 0.5/(1.2+0.5), i.e. 30%, of the total data dependent variation, which is indeed what we find.

7.2 Inclusion of the Rows of 3-bit Counters

We now combine the results from previous sections. Theorem 1 provides the probabilities of each input being 1 at a given depth in the adder tree both after initialisation by $c{\times}d$ and during the computation of $a{\times}b$. If we denote by p_i the initialisation probability of a 1 being fed into a counter at depth i, then $p_0 = pq$ and $p_{i+1} = {p_i}^3{-}\frac{3}{2}{p_i}^2{+}\frac{3}{2}p_i$ by Lemma 1. Similarly, if p'_i denotes the probability

[1] from Table 8 and Theorem 2(i), if $pq = p'q' = \frac{1}{4}$ then the average extra gate switching per counter when the bit count is reduced is about $\frac{1}{2} + (1.5417{-}\frac{1}{2}){\times}\frac{1}{3}\sum_{i=0}^{\infty}(\frac{2}{3})^i(\frac{3}{4})^i$ $= \frac{1}{2}{+}\frac{2}{3}{\times}1.0417 \approx 1.2$ gates. This assumes no bit reduction causes only negligible data dependent gate changes, as is the case.

of a 1 in the next data being fed into a counter at depth i, then $p'_0 = p'q'$ and $p'_{i+1} = {p'_i}^3 - \frac{3}{2}{p'_i}^2 + \frac{3}{2}p'_i$. Substituting p'_i in place of p in Theorem 1(i) gives the probabilities $p'_{i\delta}$ for a counter at depth i reducing its number of bits set to 1 by δ, namely $p'_{i0} = 1 - 3{p'_i}^2 + 2{p'_i}^3$ and $p'_{i1} = 3{p'_i}^2 - 2{p'_i}^3$. The substitutions of p_i and p'_i for p and p' in part (iii) yields the average number $g_{i\delta}$ of gates switched when the counter makes a reduction of δ bits. If n_i is the number of 3-bit counters in row i, then under the multiplier model of Section 6, the ith row $(i = 0, 1, 2...)$ of the counter tree has approximately $n_i = \frac{2^i}{3^{i+1}}k^2$ counters and the total number of gates switched in that row is $n_i(p'_{i0}g_{i0} + p'_{i1}g_{i1})$. Throwing in the earlier contribution of $k^2(pq + p'q' - 2pqp'q')$ from the AND gates, and ignoring the negligible contribution from the small number of 2-bit counters, the total number of gates switched is approximately

$$\begin{aligned} n(pq, p'q') \;=\;& k^2(pq + p'q' - 2pqp'q') \\ +\; k^2 & \left\{ \textstyle\sum_{i=0}^{\infty} \frac{2^i}{3^{i+1}} \left(1 - 3{p'_i}^2 + 2{p'_i}^3\right) \left(p_i(5 - 2p_i) + \frac{p'_i}{1+2p'_i}(1 - 2p_i)^2(5 - 6p_i)\right) \right\} \\ +\; k^2 & \left\{ \textstyle\sum_{i=0}^{\infty} \frac{2^i}{3^{i+1}} \left(3{p'_i}^2 - 2{p'_i}^3\right) \left((3 + 2p_i + 4{p_i}^2)(1 - p_i) + \frac{1-p'_i}{3-2p'_i}(4p_i - 1)^2(2p_i - 1)\right) \right\} \end{aligned}$$

The average over all initial states is then given by

$$n(p'q') \;=\; \frac{1}{k^2} \sum_{p=0}^{k} \sum_{q=0}^{k} \binom{k}{p} \binom{k}{q} \; n\!\left(\frac{pq}{k^2}, \frac{p'q'}{k^2}\right)$$

7.3 Evaluation of the Model

The value of $n(p'q')$ was computed in the case of $k = 8$ and compared with the results presented in Table 4. Because $n(0) = 126.44$ and $n(1) = 169.46$ are substantially different from the values given in the table, it is clear that such a model Hamming weight multiplier is inadequate. Although the contribution from the k 2-bit adders has been omitted, that could only lead to at most $2k$ more gates switching. It would not quite be sufficient to enable $n(1)$ to reach the true value of 193.14 and it would take $n(1)$ further away from its true value of 77.30.

Several improvements might be made to this model. As remarked, including the 2-bit adders does not help at least until other changes are adopted. However, the two main simplifications used in this multiplier were identified at the start of this section: a *Non-Integrality Simplification* and an *Indpendence Assumption*. The first of these can be removed entirely by following a specific construction for the multiplier, such as that used earlier for the digit-by-digit simulations. Using this construction also enables a weaker assumption to be made about the independence of inputs to the various counters. This led to a second model.

8 A Successful Hamming Weight Multiplier

The hope of a generic Hamming weight multiplier which would predict gate switching accurately seems doomed from the attempt described in the previous section. A much more accurate model certainly has to be built. Specifically, the construction for a real multiplier must be used and a weaker independence assumption. The key simplification needed for the new model below is the following:

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

8.1 The Multiplier

In the revised model, we took the original digit-by-digit multiplier used for the results in Section 3. Every Boolean output was replaced by a real number probability. For each pair of Hamming weights for the multiplier inputs the corresponding probabilities were assigned to each input bit. Then the independence assumption made it possible to obtain the probability of a 1 appearing at the outputs from each AND gate, each full adder and each half adder. Moreover, these probabilities also enable the average number of gate switchings to be computed in each counter, and hence in the whole multiplier. After averaging over all initial states, the gate counts in Figure 10 were obtained for $k = 8$.

HW(a)×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

Fig. 10. Gate Switching Activity for the 8-bit Hamming Weight Multiplier.

8.2 Evaluation

There is now a much better match with the correct values than with the previous Hamming weight multiplier. The maximum relative error between Tables 4 and 10 is in the region of 7.5%. Moreover, for all small k the forms of the discrepancies between the new values and the true values are similar. For example, model values for $(0, i)$ are consistently marginally smaller than true values by about 5%. It therefore seems reasonable to assume that extrapolating to large cases will yield fairly accurate results and indeed, a standard correction could be applied to remove most of the error.

Back in Section 4 it was noted that there is a rather nice connection between the Hamming weights of the inputs to a product and the Hamming weight of the result. So a useful health check to perform on such a model is to construct a table of output weights and compare it with the expected values, as illustrated in Figure 7. Each output digit is now a vector of probabilities associated with its member bits. The sum of these probabilities is the average Hamming weight predicted by the model for the output. The table of these output weights was created for various k up to $k = 32$ and the same overall features were found between the surface it generates and that for the true values. Indeed, the correspondence between the two was much closer than expected: the same steep sides were found when one Hamming weight is small, the same large plateau region occurs for the majority of values, and there is the usual extra rise just before the value (k, k). The only main feature that was missing was the slight depression near (k, k). However, a very much better approximation was obtained this way than by using the theoretical means which led to Figure 6.

8.3 Extrapolated Results

It is now apparent that the Hamming weight model for the multiplier describes an overall level of power consumption which is essentially the same for multipliers of any size. Total gate switching for $(0, 0)$ and (k, k) is listed in Table 11. 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 we are able to predict what the true values are 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. Thus it would seem that above $k = 8$ the model will provide an upper bound on the actual values for (k, k).

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

Fig. 11. Maximum and Minimum Gate Switching Activity, namely that for Hamming Weights $(0, 0)$ and (k, k). (*Predicted Values in italics*).

For Hamming weights, switching for $(0, 0)$ always gives the minimal activity, and that for (k, k) the maximal activity. In all cases of small k this coincided with the minimum and maximum for digit-by-digit products (*see* the Appendix). We assume therefore that the table also provides accurate predictions for the maximum and minimum gate switching activity experienced during any multiplication performed by a multiplier built according to the specification given in Section 6 *when averaged over all possible initialisations.* Of course, without such averaging over the initial state the extremes are greater: for example, performing $a{\times}b$ immediately after $a{\times}b$ should result in no gates 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 11. In particular, using these and a few more values, 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 simplified Hamming weight model; whereas 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 simplified Hamming weight model. These functions fit within the bounds imposed by the hardware. Since they also provided a very reasonable fit to all the ten (respectively 15) points used rather than just three (a maximum of about 3% relative error in the worst case), they were used to suggest the approximations for the true values which were too costly to compute in Table 11. Thus, on average between about a quarter and just over a half of the gates are switched during each multiplication.

9 Conclusion

It has been shown that the variation in power used by a multiplier is closely related to the Hamming weights of the two arguments when averaged over all possible initial states. This is a potential weak point which attackers of embedded cryptosystems may try to exploit. A simplified model of a multiplier was constructed which accurately determines power usage for all Hamming weights in polynomial time with respect to the number of bits in the inputs. This contrasts markedly with the exponential time required to consider all input bit patterns in a full simulation of the multiplier. The model relies on a key simplification, namely that inputs to full adders can be assumed to be totally independent of each other, although this cannot be the case in practice. However, this simplication was verified for all small cases, and, by avoiding combinatorial explosion, it makes it possible to deduce properties of much larger multipliers than would otherwise be the case.

Appendix

Here are the digits ordered by increasing average gate counts for the multiplier a and multiplicand b in a small k-bit multiplier for $a \times b$ constructed in the manner described in Section 6.

$k = 5$:
Multiplicand Order:

0	1	2	4	8	16	17	3	5	9	18	6	10	20	12	24
25	21	7	19	11	13	26	28	22	14	29	27	23	15	30	31

Multiplier Order:

0	1	2	4	8	16	17	3	9	5	18	6	20	10	12	24
25	19	21	7	11	13	28	22	26	14	29	27	23	15	30	31

$k = 6$:
Multiplicand Order:

0	1	2	8	4	16	32	33	9	17	3	34	10	5	18	6
12	36	40	24	20	48	49	35	41	11	25	7	13	19	50	37
42	14	21	26	38	56	52	22	44	28	51	57	15	43	39	53
27	58	23	45	29	60	46	54	30	59	61	55	47	31	62	63

Multiplier Order:

0	1	4	2	8	16	32	33	17	3	9	5	34	18	36	6
20	12	10	24	48	40	49	35	7	37	19	25	41	21	13	11
50	52	38	42	14	22	44	26	28	56	51	57	39	53	45	15
23	29	43	27	58	60	54	46	30	59	61	55	47	31	62	63

$k = 7$:
Multiplicand Order:

0	2	1	4	16	8	32	64	65	17	18	66	33	34	3	9
10	20	6	5	68	12	36	24	72	80	96	40	48	67	97	19
98	35	81	25	82	21	73	49	69	7	22	70	50	26	13	11
14	37	74	41	100	38	84	42	28	76	52	44	104	112	88	56
99	113	83	23	51	114	71	15	27	105	85	89	101	102	29	39
75	77	57	106	53	86	30	78	90	43	45	116	58	46	54	108
92	120	60	115	121	117	31	103	87	107	91	93	79	59	109	55
122	118	47	61	110	94	124	62	123	119	125	111	95	63	126	127

Multiplier Order:

0	4	1	2	32	16	8	64	33	65	17	34	36	66	9	68
5	3	18	20	6	12	10	48	40	24	72	96	80	97	35	67
49	37	69	81	73	41	19	98	7	25	100	21	11	50	13	70
38	52	82	84	42	26	14	22	28	44	74	76	112	104	56	88
113	99	101	51	71	105	39	83	85	89	57	102	53	43	15	114
75	27	77	116	45	29	23	106	108	60	78	58	92	54	86	46
90	30	120	115	103	117	121	107	109	93	79	91	59	87	47	31
55	61	118	124	122	110	94	62	119	125	123	111	95	63	126	127

References

1. R. Anderson & M. Kuhn, *Tamper Resistance - a Cautionary Note*, Proc 2nd USENIX Workshop on Electronic Commerce, Oakland, California, November 18-21, 1996, pp 1-11.
2. R. M. Best, *Microprocessor for Executing Enciphered Programs*, US Patent 4,168,396, 8th Sept, 1979.
3. W. Diffie and M. Hellman, *New directions in cryptography*, IEEE Trans. Info. Theory, **IT-22**, 1976, pp 644-654.
4. W. van Eck, *Electromagnetic Radiation from Video Display Units: An Eavesdropping Risk*, Computers and Security, vol. **4**, 1985, pp 269–286.
5. T. El-Gamal, *A public key crypto system and a signature scheme based on discrete logarithms*, IEEE Trans. Info. Theory, **IT-31**, 1985, pp 469–472.
6. K. Gandolfi, C. Mourtel & F. Olivier, *Electromagnetic Analysis: Concrete Results*, Cryptographic Hardware and Embedded Systems – CHES 2001, Ç. Koç, D. Naccache & C. Paar (editors), Lecture Notes in Computer Science, **2162**, Springer-Verlag, 2001, pp 251–261.
7. *Index of National Security Telecommunications Information Systems Security Issuances*, NSTISSC Secretariat, US National Security Agency, 9 January 1998.
8. P. Kocher, *Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems*, Advances in Cryptology, Proc Crypto 96, N. Koblitz editor, Lecture Notes in Computer Science, **1109**, Springer-Verlag, 1996, pp 104–113.
9. P. Kocher, J. Jaffe & B. Jun, *Differential Power Analysis*, Advances in Cryptology – Crypto '99, M. Wiener (editor), Lecture Notes in Computer Science, **1666**, Springer-Verlag, 1999, pp 388–397.
10. R. Mayer-Sommer, *Smartly Analyzing the Simplicity and the Power of Simple Power Analysis on Smartcards*, Cryptographic Hardware and Embedded Systems (Proc CHES 2000), C. Paar & Ç. Koç editors, Lecture Notes in Computer Science, **1965**, Springer-Verlag, 2000, pp 78–92.
11. T. S. Messerges, E. A. Dabbish, R. H. Sloan, *Power Analysis Attacks of Modular Exponentiation in Smartcards*, Cryptographic Hardware and Embedded Systems (Proc CHES 99), C. Paar & Ç. Koç editors, Lecture Notes in Computer Science, **1717**, Springer-Verlag, 1999, pp 144–157.
12. P. L. Montgomery, *Modular Multiplication without Trial Division*, Math. Computation, **44**, 1985, pp 519–521.
13. R. L. Rivest, A. Shamir & L. Adleman, *A Method for obtaining Digital Signatures and Public-Key Cryptosystems*, Comm. ACM, **21**, 1978, pp 120–126.
14. W. Schindler, *A Timing Attack against RSA with Chinese Remainder Theorem*, Cryptographic Hardware and Embedded Systems (Proc CHES 2000), C. Paar & Ç. Koç editors, Lecture Notes in Computer Science, **1965**, Springer-Verlag, 2000, pp 109–124.
15. C. D. Walter, *Sliding Windows succumbs to Big Mac Attack*, Cryptographic Hardware and Embedded Systems – CHES 2001, Ç. Koç, David Naccache & Christof Paar (editors), Lecture Notes in Computer Science, **2162**, Springer-Verlag, 2001, pp 286–299.
16. C. D. Walter & S. Thompson, *Distinguishing Exponent Digits by Observing Modular Subtractions*, Topics in Cryptology – CT-RSA 2001, D. Naccache (editor), Lecture Notes in Computer Science, **2020**, Springer-Verlag, 2001, pp 192–207.