Bitslice Ciphers and Power Analysis Attacks

Joan Daemen, Michael Peeters, and Gilles Van Assche

Proton World Intl.
Rue Du Planeur 10, B-1130 Brussel, Belgium
Email: {daemen.j, peeters.m, vanassche.g}@protonworld.com
http://www.protonworld.com/research

Abstract. In this paper, we present techniques to protect *bitslice* block ciphers against power analysis attacks. We analyze and extend a technique proposed in [12]. We apply the technique to BaseKing, a variant of 3-Way[9] that was published in [7]. We introduce an alternative method to protect against power analysis specific for BaseKing. Finally, we discuss the applicability of the methods to the other known bitslice ciphers 3-Way and Serpent [1].

1 Introduction

The inherent security offered by a block cipher is best evaluated by investigating the cipher's resistance against the set of known cryptanalytic attacks. It is generally agreed that a cipher for which there are attacks that are more efficient than exhaustive key search have an inherent weakness. In our opinion, the absence of this kind of attacks is rightfully the primary criterion for comparison of ciphers.

Although interesting, in many cases the relevance of these cryptanalytic attacks is mostly academic: in practical applications of the cipher they turn out to be irrelevant for several reasons. The attacks might require an unrealistic amount of plaintext/ciphertext pairs, such as differential cryptanalysis and linear cryptanalysis of DES [2,15]. In other cases, the theoretical weakness only manifests itself in very rare cases. This is the case for weak keys such as in IDEA [10]. In still other cases there is an easy way to protect against the weaknesses. Weak keys of DES and related-key attacks [11] can be avoided by generating keys independently or by deriving them using one-way functions.

In the model for theoretical cryptanalysis, the key is considered to be unknown and an attacker has access only to plaintext and ciphertext, and can possibly manipulate the key in certain ways. He has no access to intermediate computation results. In practical implementations the secrecy of the key, that is considered to be a given in cryptanalysis, must be accomplished by effective physical and logical protection. This is invariably the most expensive and problematic aspect of any serious application using cryptography. Particularly challenging are distributed applications, where smart cards used by consumers, and terminals used by merchants and service providers, use cryptography to secure fund transfers or the conditional access to services (e.g., GSM). As has been shown recently in several publications [14,5], implementing cryptographic algorithms on these platforms is not trivial. One of the reasons for these problems is that a combination of hardware characteristics and algorithm coding or the presence of (induced) errors might give away information on intermediate computation results. By using statistical techniques, this information is then used to find the key.

In the long term, this problem can best be dealt with by incorporating on the smartcards and terminal security modules dedicated cryptographic hardware that performs integrity checks and minimises the leakage of information. Such components are not available yet and use must be made from existing components. In this paper, we discuss techniques to protect implementations of bitslice ciphers on state-of-the-art smartcards against power analysis attacks.

2 Implementation Attacks

2.1 Timing Attacks and Simple Power Analysis

In timing attacks, the dependence of the execution time of the cipher on plaintext or key bits is exploited to derive key or plaintext information. An effective protection against timing attacks is writing the code in such a way that the number of cycles taken by an execution is independent of the value of key or plaintext bits (preferably a constant).

In so-called *simple power analysis* attacks, the attacker makes use of some quantity, measurable at the outside of a cryptographic device, to detect instructions being executed inside it. Typically, this measurable quantity is the power consumption or radiation. This may leak information on key or plaintext bits if the instructions that are executed depend on the values of data that are being processed. An effective way to protect against this type of information leak is to program the cipher as a fixed sequence of instructions. This also implies a constant execution time, effectively sealing up the timing leak.

2.2 Differential Power Analysis

In more advanced power analysis attacks, e.g., *differential power analysis* (DPA) [14] the correlation between the power consumption (or radiation, ...) and the values of the operands of the instructions is exploited. Usually, this correlation is weak with respect to the noise on the power consumption. Even if no special measures are taken, several (tens to thousands, depending on the type of instruction and quality of the processor) cipher executions are required to exploit these correlations in an attack.

The basic principle of differential power analysis is that the probability distribution of the power consumption, given that a certain event occurs, can be distinguished from the average probability distribution. The attack is mounted as follows:

- The first step of the attack is to instruct the cryptographic device to perform a number of cipher computations. For each of these computations the power consumption pattern P_i is measured and stored together with the known parameters of the computation, that is, a_i the plaintext or the ciphertext. This is the data acquisition phase which produces a data set $D = \{(a_i, P_i) \mid i = 1 \dots z\}$.
- Then, an event is specified whose occurrence depends on the value of a number of plaintext (or ciphertext) bits and key bits we wish to determine. Such an event can be, for instance, that the result of an intermediate cipher calculation gives a certain result (which must be at some time present in a CPU register, ALU, bus, or memory cell). We call the *target subkey* the key bits the specified event depends on.
- For each of the possible values of the target subkey, the following check is performed. In the hypothesis that the target subkey s^* is correct, the set of power consumption patterns are divided into two groups: those for which the event occurs $D_1 = \{(a_i, P_i) \mid f(s^*, a_i) = 1\}$ and the complementary set $D_0 = \{(a_i, P_i) \mid f(s^*, a_i) = 0\}$. (As suggested by these formulas, f indicates whether the event occurs given the known and hypothesis values.)

  It is assumed that the two subsets D_1 and D_0 can be statistically distinguished for the correct hypothesis. We therefore define some distance between the two distributions. The subkey value for which this distance is maximized, is taken as the correct value. In general, a wrong target subkey value will divide the power consumption patterns in two sets in which the event occurs an average number of times. (However, if the round function has certain algebraic properties, several subkey values, among which the correct one, may be suggested.)

2.3 Higher-Order DPA

Depending on the usage of the power consumption pattern we can distinguish differential power attacks of different orders. Generally speaking, N-th order DPA makes use of N different intermediate values calculated at N different times during the execution of the cipher algorithm.

- In first-order DPA as described by Kocher [14], the event mentioned above is typically the fact that a particular bit (or set of bits) in a CPU register, bus, or memory has the value 1 (or 0). It is usually sufficient to distinguish the two data sets D_0 and D_1 by their average.
- In second-order DPA, the event is typically the fact that two bits of operands occurring at different times during the computation, are equal (or different). This situation occurs when one can group a set of samples according to the value of the exor of two operand bits rather than an absolute value of an operand bit.

Protection against DPA can take different forms. Here are two examples:

– The alignment of the power consumption patterns can be made harder by building in random time differences in the cipher execution. This reduces the effectiveness of the attack described above by requiring more power consumption patterns.
– Theoretically, first-order DPA can be made impossible by programming the cipher in such a way that the operands bear no correlation with intermediate block cipher states or key bits. The techniques proposed in this paper attempt to do exactly that, through the introduction of random biases and state/key splitting. This can be generalized to resistance against N-th order DPA, where it is required that no set of N operands has a correlation with state or key. These techniques were already proposed in [4].

In practice, a second-order attack is more difficult to mount than a first-order one. The reasons for this are:

– **A more complex layout.** In first-order DPA, the probability distributions are one-dimensional, e.g., they represent the power consumption at a given stage in the cipher computation. Usually, for any given stage a subkey hypothesis can be tested by taking the average for the two subsets and use the difference between them as distance. In the second order DPA, the equality (or inequality) of two bits is of interest. Here, a subkey hypothesis has to be tested by determining whether it divides the power samples in two groups with the following properties: in one group, the two power consumption samples have a tendency to increase or decrease together, in the other group they fluctuate in opposite directions. Computing the distance between the two distributions takes more computations than just taking the difference between the averages. More complex processing is thus required.
– **Increased memory and processing requirements.** Especially when a cipher implementation uses random delays, the exact location of the cycles where a certain operand is processed in the power consumption samples is unknown a priori. Hypothesis testing has to be performed for all possible locations. If n samples are of interest for a first order attack, one gets n^2 pairs of samples for a second order attack, thus greatly increasing the demand of data storage and processing.
– **Increased number of power consumption patterns.** To distinguish two distributions from each other, one needs enough samples before a statistically significant result appears. For the same amount of noise, bi-dimensional distributions are harder to distinguish than their equivalent 1D distributions. This is detailed in appendix A. Typically, if z power consumption patterns are needed in the 1D case, about $2z^2$ patterns are necessary in the 2D case. Furthermore, the use of random delay spreads the effect of the event in a single dimension, decreasing the signal-to-noise ratio linearly. In second-order DPA, this effect is spread in two dimensions, decreasing the signal-to-noise ratio quadratically.

Before discussing the different protection methods, we discuss correlation and decorrelation. For the sake of brievety, no proofs are given in this version of the paper.

2.4 On Correlation and Decorrelation

The correlation between two binary variables (bits) f and g is given by $C(f, g) = 2\Pr(f = g) - 1$. If f and g can be expressed in terms of a word a of n bits $(a_1, a_2, \ldots, a_n)$, i.e., $f(a), g(a)$, this can be computed as [7,8]:

$$C(x, y) = 2^{-n} \sum_a (-1)^{f(a) \oplus g(a)}.$$

Consider a bit f that can be expressed as a function of two words a and b, i.e., $f(a, b)$. This bit is said to be decorrelated from a if for any linear combination of bits of a, denoted by $u^{\mathrm{t}}a$ (with u a *selection vector* [8]), we have $C(f, u^{\mathrm{t}}a) = 0$.

A word d that is a function of two words a and b is said to be decorrelated from a if for all linear combinations of bits of d, denoted by $w^{\mathrm{t}}d$, and all linear combinations of bits of a, denoted by $u^{\mathrm{t}}a$, we have $C(w^{\mathrm{t}}d, u^{\mathrm{t}}a) = 0$. Clearly, this implies that all bits of d are also decorrelated from a.

In words, knowledge of a bit f (or word d) gives no information whatsoever on the word a if the word b is unknown.

Consider an operand $a'(a, \delta)$ that is defined by $a' = a$ if $\delta = 0$ and $a' = \bar{a}$ otherwise. Using techniques introduced in [8] it can easily be verified that all bits of a' are decorrelated from a. The complete word a' is however not decorrelated from a. δ is called the *masking bit*.

For an operand $a'(a, a'')$ that is defined by $a' = a \oplus a''$ it can easily be shown that it is decorrelated from a. Obviously, thanks to symmetry, the operand $a'' = a \oplus a'$ is also decorrelated from a. a'' is called the *masking word*.

It can be shown that a word b that is the result of the computation of two operands $a'_1(a_1, a''_1)$ and $a'_2(a_2, a''_2)$ is decorrelated from both a_1 and a_2 if a''_1 and a''_2 are mutually decorrelated (or equivalently, independent).

Moreover, all bits of a word b that is the result of a bitwise logical computation of two operands $a'_1 = a_1 \oplus a''_1$ and $a'_2 = a_2 \oplus a''_2$ are decorrelated from both a_1 and a_2 if the bits at corresponding positions of masking variables a''_1 and a''_2 are mutually decorrelated (or equivalently, independent).

Finally, for a state $a = a' \oplus a''$, any computation involving only terms of a' (or only a''), will have only operands that are decorrelated from a.

3 The Duplication Method

Louis Goubin and Jacques Patarin [12] propose the *Duplication Method*: a method for increasing DPA-resistance based on secret sharing. It aims to remove all correlation between operands and intermediate state or key, thus making first-order DPA impossible. In this section, we discuss the limitations of the methods as proposed in [12].

The basic principle of the Duplication Method is to split any variable V of a given cipher computation into a set of k other variables $V_1, V_2, \dots V_k$ such that the variable can be reconstructed through the use of a function f: $V = f(V_1, \dots V_k)$. The cipher computations are performed on the variables V_i such that the relation $V = f(V_1, \dots V_k)$ holds at all times and without having to calculate V explicitly. Furthermore, the knowledge of $k-1$ of the variables V_i does not give any information on V itself.

In the practical methods they propose, the Duplication Method is applied to DES and the variables V are split in two parts V_1 and V_2 with $V = V_1 \oplus V_2$. For the linear operations in DES (expansion, bit permutation, exor of output of f-function), the computation can be done on V_1 and V_2 separately and independently (for a linear function L, we have $L(V_1) \oplus L(V_2) = L(V_1 \oplus V_2) = L(V)$). For the key addition it is sufficient to add it to one of the two variables.

For the S-box evaluation however, the computations involving V_1 and V_2 cannot be kept separated due to the nonlinearity of the S-boxes. Goubin and Patarin [12] propose the use of 12-bit to 8-bit lookup tables $T_i(v_1, v_2)$ satisfying

$$(v_1', v_2') = T(v_1, v_2) = (A(v_1, v_2), S(v_1 \oplus v_2) \oplus A(v_1, v_2)) \tag{1}$$

where A is a *randomly-chosen* secret transformation and S is the original 6-bit or 4-bit S-box. Clearly, $v_1' \oplus v_2' = S(v_1 \oplus v_2)$, so that the table-lookup preserves the condition $v = v_1 \oplus v_2$. The computation of v_1', v_2' is performed as a lookup in a table of 4096 bytes. Unfortunately, the size of these tables that replace the S-boxes makes this method prohibitively expensive for current smartcards, where memory is a scarce resource.

For this reason, they propose a variant of their method that uses more compact lookup tables. The S-box computation is performed in the following two steps:

$$v_0 = \varphi(v_1 \oplus v_2), \tag{2}$$

and

$$(v_1', v_2') = S'(v_1, v_2) = (A(v_0), S(\varphi^{-1}(v_0)) \oplus A(v_0)) \tag{3}$$

with φ a secret bijective function.

Although φ can be chosen such that v_0 can be calculated by combining individual computations (e.g., if φ is linear, it reads $v_0 = \varphi(v_1) \oplus \varphi(v_2)$), the intermediate state value v is fully determined by a single operand: v_0. The inability to exploit v_0 for hypothesis testing is only based on on the secrecy of φ. Apart from disclosure by a manufacturer, an attacker can learn more about φ if he or she has access to a sample card where the cipher can be run with a known key. The problem can thus be factored in two sub-problems, namely, learning more about φ and mounting a first-order DPA attack focusing on the v_0 values rather than v. Moreover, the linearity of φ (or the fact that it is quadratic) may really help a lot in determining it.

From an academic point of view, both variants of the method face the problem that they do not guarantee decorrelation between operands and intermediate state values.

For the first variant of the Duplication Method, the lookup-table output (v_1', v_2') can be expressed in terms of (v_1, v) using an equivalent table T' by including the linear transformation $(v_1, v_2) = (v_1, v \oplus v_1)$. To have decorrelation of the bits of v_1', v_2' from v, it is a requirement that $C(b, u^{\mathrm{t}}v) = 0$ for all 8 bits b of v_1' or v_2' and for all 64 possible selections u. If A is chosen randomly it is very unlikely that this is the case.

For the second variant of the Duplication Method, bits of operand v_0 are correlated to linear combinations of bits of v. As a matter of fact, we have:

$$\sum_u C^2(b, u^{\mathrm{t}}v) = 1$$

for any of the output bits. In the case that φ is linear (as proposed in [12]), every bit of v_0 is correlated to a linear combination of bits of v with correlation 1.

4 Bitslice Ciphers

Bitslice ciphers can be implemented using only bitwise logical instructions and (cyclic) shifts. The term bitslice cipher was introduced by Eli Biham referring to the AES candidate Serpent [1] designed by Eli Biham, Ross Anderson and Lars Knudsen. Older examples of bitslice ciphers are 3-Way[9] published in 1993 and BaseKing. BaseKing is a variant of 3-Way that was described in [7] but never presented at a conference.

4.1 BaseKing

BaseKing has a block and key length of 192 bits (24 bytes). It is an iterated block cipher with a round transformation composed of a number of steps, each with its own function. These steps treat the intermediate encryption result, called the *state*, in a *uniform* way. The state, denoted by a consists of 12 16-bit words denoted by a_0 to a_{11}. The round transformation has 5 steps:

– **key addition:** the cipher key and a round constant is added to the state.

$$a \leftarrow a \oplus k \oplus Cr_j$$

– **diffusion:** the words are transformed with a linear transformation with high diffusion (branch number 8):

$$a_i \leftarrow a_i \oplus a_{i+2} \oplus a_{i+6} \oplus a_{i+7} \oplus a_{i+9} \oplus a_{i+10} \oplus a_{i+11}$$

– **early shift:** the words are cyclically shifted over 12 different offsets:

$$a_i \leftarrow a_i \ll r_i$$

– **S-box:** the words are transformed with a nonlinear transformation operating in parallel on sets of 3 bits:

$$a_i \leftarrow a_i \oplus (a_{i+4} \vee \overline{a_{i+8}})$$

– **late shift:** the words are cyclically shifted over 12 different offsets:

$$a_i \leftarrow a_i \gg r_{11-i}$$

The vector of rotation constants used in the shift operations is

$$r = (0, 8, 1, 15, 5, 10, 7, 6, 13, 14, 2, 3).$$

The round constants are given by:

$$Cr_j = (0, 0, q_j, q_j, 0, 0, 0, 0, q_j, q_j, 0, 0)$$

with q_j given by the following pseudo-c program:

```
q[0] = 0x000B;
if ((q[j+1] = q[j]<<1) & 0x0100) q[j+1]^= 0x0111;
```

BaseKing has 11 rounds and a final output transformation. The final output transformation consists of a key addition and a diffusion step (as described above) followed by a transformation that inverts the order of the words:

$$a_i \leftarrow a_{11-i}$$

Thanks to the arrangement of the steps and the algebraic properties of the operations, the inverse cipher is exactly the same as the cipher itself, with the exception of the round constants. For a detailed treatment of these aspects, we refer to [7].

4.2 Cryptanalysis

The design of BaseKing aims at providing strong resistance against differential and linear cryptanalysis and the absence of symmetry properties. We refer to [7] for a development of this point.

For 3-Way, it has been shown that the lack of a real key schedule allows mounting of a related-key attack [13]. This attack is also applicable to the cipher BaseKing. However, in applications requiring only encryption, MACing and key derivation, related-key attacks can be easily prevented by the application of sound key management principles, i.e., by avoiding key variants.

5 Protecting Bitslice Ciphers against DPA

This section describes our methods of protecting bitslice ciphers against first order DPA attacks.

Before executing the cipher, the initial value of the state a, i.e., the plaintext, is split into two state shares a' and a'' with $a = a' \oplus a''$. All computations will be performed on the state shares in such a way that the relation $a = a' \oplus a''$ holds at all times. The linear steps (early shift, diffusion, late shift) can be applied to the state shares a' and a'' independently and therefore provide decorrelation from state words.

5.1 Key Addition

The key addition can be applied by adding the round key to one of the two split states: $a' \leftarrow a' \oplus k$. In the assumption that the attacker has no information on a', a first-order DPA cannot be used to gain information on the key k.

However, in [3], Eli Biham and Adi Shamir describe an attack that uses Hamming weight information on round key words to retrieve the key. The Hamming weight information is obtained by taking the average consumption over multiple cipher computations with the same key. The ability to use this Hamming weight information to derive the key strongly depends on the key schedule. For DES (or Triple-DES) the complete key can be found using "standard techniques from error correcting codes".

Ironically, its lack of a key schedule gives BaseKing an excellent protection against this attack. The cipher key is applied at the end of every round by just exoring it with the state. The subkey words are the same for every round. In the case of 8-bit words, this attack gives on the average 2.54 bits of information per byte, leaving still $24 * (8 - 2.54) = 131$ bits to guess. In the case of 32-bit words, this becomes 3.55 bits per word, leaving still $6 * (32 - 3.55) = 171$ bits to guess.

Anyway, the knowledge of key information might be exploited to further attack the cipher. A simple way to protect against the key schedule attack is to apply secret sharing on the key. The key k is split into two parts k' and k''. The exor with k is then executed by a word-by-word exor of the state with k' followed by a word-by-word exor of the state with k''. The addition of two subkeys per round can even be done with some linear steps in between, if one of the two subkeys undergoes a linear transformation: $L(a + k) = L(a) + k_1$ with $k_1 = L(k)$.

In combination with the state secret sharing method, the key secret sharing method can make key addition really symmetric: $a' \leftarrow a' \oplus k'$ and $a'' \leftarrow a'' \oplus k''$.

5.2 Full State Splitting

Similar to the duplication method of Goubin and Patarin, the state a is split in a' and a'' with a' generated randomly before the computation and only recombined at the end of the cipher computation.

The BaseKing S-box operates on sets of three words of the state (e.g., a_0, a_4 and a_8), and transforms them by

$$a_i \leftarrow a_i \oplus (a_{i+4} \oplus 1)a_{i+8} \oplus 1, \qquad (4)$$

with index additions modulo 12. Applying secret sharing gives rise to $a_i = a'_i \oplus a''_i$, $i = 0, 1, \ldots 11$. We must determine functions f' and f'':

$$\begin{aligned} a'_i &\leftarrow f'(a'_i, a'_{i+4}, a'_{i+8}, a''_i, a''_{i+4}, a''_{i+8}) \quad \text{and} && (5)\\ a''_i &\leftarrow f''(a'_i, a'_{i+4}, a'_{i+8}, a''_i, a''_{i+4}, a''_{i+8}) && (6)\end{aligned}$$

that preserve the relation $a = a' \oplus a''$:

$$f' \oplus f'' = a'_i \oplus a''_i \oplus (a'_{i+4} \oplus a''_{i+4} \oplus 1)(a'_{i+8} \oplus a''_{i+8}) \oplus 1, \quad i = 0, 1, \ldots 11. \quad (7)$$

The restriction on f' and f'' is that during their computation there are no operands that bear correlation with a. Using the distribution rule of $\mathbf{F}_2$, we get:

$$\begin{aligned} f' \oplus f'' = a'_i \oplus a''_i \oplus a'_{i+8} \oplus a''_{i+8} \\ \oplus a'_{i+4}a'_{i+8} \oplus a'_{i+4}a''_{i+8} \oplus a''_{i+4}a'_{i+8} \oplus a''_{i+4}a''_{i+8} \oplus 1. \quad (8)\end{aligned}$$

A computation involving only components of a' or a'' cannot involve operands that have a correlation with a. However, due to the presence of the mixed terms, i.e., with components of a' and a'', the computation of f' will necessarily involve terms of a'' or vice versa. For instance, one gets:

$$\begin{aligned} f' &= a'_i \oplus a'_{i+8} \oplus a'_{i+4}a'_{i+8} \oplus a'_{i+4}a''_{i+8}\\ f'' &= a''_i \oplus a''_{i+8} \oplus a''_{i+4}a'_{i+8} \oplus a''_{i+4}a''_{i+8} \oplus 1.\end{aligned}$$

To guarantee decorrelation of all operands, the order in which these functions are computed is important. Consider the expression for f' given above. If it is evaluated from right to left, after the addition of the two rightmost terms, the following operand occurs: $a'_{i+4}a''_{i+8} \oplus a'_{i+4}a'_{i+8} = a'_{i+4}a_{i+8}$. Clearly each bit of this operand has a correlation of 1/2 with the corresponding bit of state word a_{i+8}. Since a' is random and independent of a, we have:

$$a'_{i+4}a_{i+8} = \begin{cases} a_{i+8} & \text{when } a'_{i+4} = 1,\\ 0 & \text{when } a'_{i+4} = 0, \text{ and thus}\end{cases} \quad (9)$$

$$C\left(a'_{i+4}a_{i+8}, a_{i+8}\right) = \frac{1}{2}. \quad (10)$$

If the expression for f' is evaluated from left to right, it can be shown that no operands occur that have a correlation with the state. The computations of all terms except the last one involve only words of a', hence here decorrelation from a is automatic. For the addition of the last term to the intermediate result of the computation, the presence of a'_i in the first term implies that the masking words of two terms are decorrelated.

The state splitting method can be generalized to provide protection against second and higher order DPA attacks. In principle, this enables the smartcard designer to adjust the level of security by making DPA attacks arbitrarily difficult. For instance, protecting against second-order DPA requires the state a to be split into three parts, namely $a = a' \oplus a'' \oplus a'''$. The evaluation of the S-box requires care when deciding in which order the operations must be performed.

5.3 The Bias Vector Method

In the bias vector method one of the two split states is in a particular sub-class that can be kept invariant under the cipher computations. For BaseKing this is the class of states where each 16-bit word is either all-0 (0) or all-1 ($\bar{0}$). These particular split states are called *bias states*. A bias state can be represented by a 12-bit vector, called a *bias vector*.

A bias state is invariant under the shift operations. The diffusion operation maps every bias state to another bias state, since it operates in parallel on the bits of the words. Thanks to the linearity of the diffusion operation, computing the bias vector at the output of a diffusion step from the bias vector at its input can be done with some table-lookups and exors. For example: 3 table lookups in tables with 16 (2^4) entries and two exors. Thanks to their compactness, input-output bias vector pairs can be computed beforehand and stored in memory for later usage.

The cipher computation operates on a biased state A equal to $a \oplus d$. The bias vector corresponding with d is denoted by δ.

For the linear steps including the key addition, the computation is performed on A. The evolution of the bias vector in the linear steps is computed using the table-lookups described above. For each round a new random 12-bit bias vector is introduced.

We explain the computation of the first word of the output of the nonlinear transformation corresponding with:

$$a'_0 = a_0 \oplus (a_4 \vee \overline{a_8})$$

All other words are computed in the same way. The computation of A'_0 is done as follows:

1. **Computation of required values:** Compute $A_4 \oplus A_8$ and store it in a register. Store 0 in a register (A_4 and A_8 are already assumed to be in registers);
2. **Nonlinear computation:** We compute $A_4 \vee \overline{A_8}$ and store it as G.
3. **Computation of correction term**: depending on the bias vector bits δ_4, δ_8, one of the four registers containing 0, A_4, A_8 or $A_4 \oplus A_8$ is selected and its complement is stored. The selected register or the one containing its complement (depending on γ) is exored to G:
   - $\delta_4 = 0, \delta_8 = 0$: complement 0 and store as H. If $\gamma = 0$ add 0 to G, else add H to G;
   - $\delta_4 = 0, \delta_8 = 1$: complement A_4 and store as H. If $\gamma = 0$ add H to G, else add A_4 to G;
   - $\delta_4 = 1, \delta_8 = 0$: complement A_8 and store as H. If $\gamma = 0$ add H to G, else add A_8 to G;
   - $\delta_4 = 1, \delta_8 = 1$: complement $A_4 \oplus A_8$ and store as H. If $\gamma = 0$ add $A_4 \oplus A_8$ to G, else add H to G;

   This is programmed such that the sequence of instructions is independent of the branch that is taken, the only difference are the source/target registers.

4. **Computation of A'_0 and δ'_0:** $A'_0 = A_0 \oplus G$ and $\delta'_0 = \delta_0 \oplus \gamma$.

This computation can be repeated for all state words. The bits of all operands in the computation are decorrelated from state words:

- Bits of A_4 and A_8 are decorrelated independently (via d_4 and d_8) from state words, hence the results of $A_4 \oplus A_8$ and $A_4 \vee \overline{A_8}$ are also decorrelated.
- Depending on the values of δ_4 and δ_8 the operand is $0, A_4, A_8$ or $A_4 \oplus A_8$. The bits of individual operands are all decorrelated, except 0 that is obviously constant.
- In the subsequent computations of G and A'_0 the bits of all operands are decorrelated.

The advantage of the bias vector method is that second order DPA will be more difficult to accomplish than in the case of full state splitting thanks to the compact (and possible delocalized) processing of the bias vector. Unfortunately, the bias vector method has the disadvantage that decorrelation is only reached at bit-level and not at word level (e.g., if in a word a_i of the state the two LSB bits are equal, the two LSB bits of A_i will be equal). In cyclic shift operations, this might give away information via second-order effects.

5.4 Coding Results

To evaluate the implementability and cost of the methods described above, several versions of BaseKing were programmed on the ARM7 RISC processor (see `www.arm.com` for technical and other information). This processor is well know for its very high computation power vs. consumption ratio, and hence is ideal for smartcards. Special care has been taken to guarantee immunity against timing (for instance, handling of pipeline clearing) and SPA attacks. The following table summarizes the code size and execution time for the different versions.

version	cycles	code size
timing and SPA resistant	1949	776
full state splitting	4593	1148
bias vector	4505	2804
bias vector, guarded instr.	3845	1844

Clearly, the application of the anti-DPA methods has a considerable cost in execution time and code size. In the case of the bias vector method, most of the overhead comes from the computation of the S-box (2119 cycles, 1216 bytes code size).

We did not conduct actual power measurements and therefore were unable to verify whether the use of guarded instructions endangers the immunity against DPA attacks, and hence the last line of the table is only included for information.

More detailed information on the coding and updates will be made available at `http://www.protonworld.com/research`.

5.5 Applicability to 3-Way and Serpent

The bias vector method makes use of particular symmetry properties of the cipher BaseKing and cannot be extended to 3-Way or Serpent. The full state splitting method however can be extended to any bitslice cipher. For 3-Way, both the linear steps and the S-box evaluation can be done in exactly the same way as described for BaseKing.

In Serpent, the linear steps of the round function pose no problem. However, we do not expect that implementing the method for the Serpent S-boxes will be trivial. The BaseKing S-box mapping is very simple, containing only a single nonlinear term with only two factors per output word computation. Moreover, the expression is the same for all output words, only the input words differ. In Serpent, there are 8 different S-boxes, and the expressions of the output bits contain more terms with degrees up to 3. This is likely to give more mixed terms in the expressions of Serpent's equivalents of the f' and f'' functions and a relatively more important reduction in performance. Special care must be taken in the order of evaluation of these functions to guarantee correlation immunity (if possible). Moreover, due to the lack of symmetry in the Serpent S-boxes, this may give rise to an important overhead in code size.

Acknowledgments. We would like to thank Mr. Philip Theunissen for proofreading the final version of this paper.

6 Conclusions

We have applied and extended techniques to protect block ciphers against power analysis attacks to the bitslice block cipher BaseKing. These techniques have been validated by actual coding in assembly language on an ARM processor.

One of the techniques described generalises readily to the block cipher 3-Way. We have shown that this technique can be applied to Serpent, but more analysis and research is required to see what is the performance penalty in this case.

Finally, in the appendix we show the difference in power between first-order and second-order DPA with information-theoretical arguments.

References

1. E. Biham, R. Anderson, and L. Knudsen. Aes proposal serpent. *AES CD-1: documentation*, 1998.
2. E. Biham and A. Shamir. Differential cryptanalysis of des-like cryptosystems. *Journal of Cryptology*, 4(1):3–72, 1991.
3. E. Biham and A. Shamir. Power analysis of the key scheduling of the aes candidates. In *2nd AES Candidates Conference*, March 1999.
4. S. Chari, C. Jutla, J. Rao, and P. Rohatgi. A cautionary note regarding evaluation of aes candidates on smart-cards. In *Proceedings of the 2nd AES Candidates Conference*, March 1999.

5. S. Chari, C. Jutla, J. Rao, and P. Rohatgi. Towards sound approaches to counteract power-analysis attacks. In *Advances in Cryptology - CRYPTO'99*, pages 398–412. Springer-Verlag, 1999.
6. T. M. Cover and J. A. Thomas. *Elements of Information Theory.* John Wiley & Sons, 1991.
7. J. Daemen. *Cipher and Hash Function Design.* PhD thesis, Katholieke Universiteit Leuven, March 1995.
8. J. Daemen, R. Govaerts, and J. Vandewalle. Correlation matrices. In R. Anderson, editor, *Fast Software Encryption*, pages 275–285. Springer-Verlag, 1994.
9. J. Daemen, R. Govaerts, and J. Vandewalle. A new approach towards block cipher design. In R. Anderson, editor, *Fast Software Encryption*, pages 18–33. Springer-Verlag, 1994.
10. J. Daemen, R. Govaerts, and J. Vandewalle. Weak keys of idea. In *Advances in Cryptology - CRYPTO'93*, pages 224–231. Springer-Verlag, 1994.
11. D.W. Davies. Some regular properties of the des. In *Advances in Cryptology - CRYPTO'82*, pages 89–96. Plenum Press, 1983.
12. L. Goubin and J. Patarin. Des and differential power analysis. In *CHES'99*, volume 1717, pages 158–172. Springer-Verlag, 1999.
13. J. Kelsey, B. Schneier, and D. Wagner. Key-schedule cryptanalysis of idea, g-des, gost, safer and triple-des. In *Advances in Cryptology - CRYPTO '96*, page 237. Springer-Verlag, 1996.
14. P. Kocher, J. Jaffe, and B. Jun. Introduction to differential power analysis and related attacks. *The article can be found at* `http://www.cryptography.com/dpa/technical/index.html`, 1998.
15. M. Matsui. Linear cryptanalysis method for des cipher. In *Advances in Cryptology - EUROCRYPT'93*, page 386. Springer-Verlag, 1993.

A First Order vs. Second Order DPA

Using a simple statistical model we compare the distinguishing power of second order DPA and first order DPA, i.e., we compute the required number of samples for both methods under identical noise levels.

Assume an emitter has chosen one of the two random process (with x a continuous variable, e.g., the power consumption at a given stage):

- Process f, probability density $f(x)$;
- Process g, probability density $g(x)$.

The observer receives a sequence of x_i and wants to determine whether it comes from f or from g. From the observer's point of view, the probability that values in $[x_i, x_i + dx_i]$ come from f is:

$$\begin{aligned} P(f \mid x_1 x_2 \ldots x_z)\, dx_1 \ldots dx_z &= \frac{P(x_1 x_2 \ldots x_z \mid f) P(f)}{P(x_1 x_2 \ldots x_z)}\, dx_1 \ldots dx_z \\ &= P(f) \prod_i \frac{P(x_i \mid f)}{P(x_i)}\, dx_i, \end{aligned}$$

and similarly for g,

$$P(g \mid x_1 x_2 \dots x_z)\, dx_1 \dots dx_z = P(g) \prod_i \frac{P(x_i \mid g)}{P(x_i)}\, dx_i.$$

For simplicity, we now assume that $P(f) = P(g)$, and thus one gets

$$\frac{P(f \mid x_1 x_2 \dots x_z)}{P(g \mid x_1 x_2 \dots x_z)} = \prod_i \frac{P(x_i \mid f)}{P(x_i \mid g)} = \prod_i \frac{f(x_i)}{g(x_i)}$$

In order to detect f over g, we must reach the situation where

$$P(f \mid x_1 x_2 \dots x_z) \geq \lambda P(g \mid x_1 x_2 \dots x_z).$$

For simplicity, let $\lambda = e$. Taking the logarithm on both sides, it reads:

$$\sum_i (\log(f(x_i)) - \log(g(x_i)) \geq 1 \tag{11}$$

The main question we address is how many samples (parameter z) are required to reach this condition. Assuming that the emitter chose the random process f, i.e., distribution $f(x)$ applies, each new x_i will on average contribute to the sum in (11) as much as:

$$D(f \| g) = \int f(x)(\log(f(x)) - \log(g(x))dx,$$

where $D(f\|g)$ happens to be the relative entropy of f and g (see [6]).

Therefore, the average number of samples required to clearly distinguish f from g is $z \sim 1/D(f\|g)$.

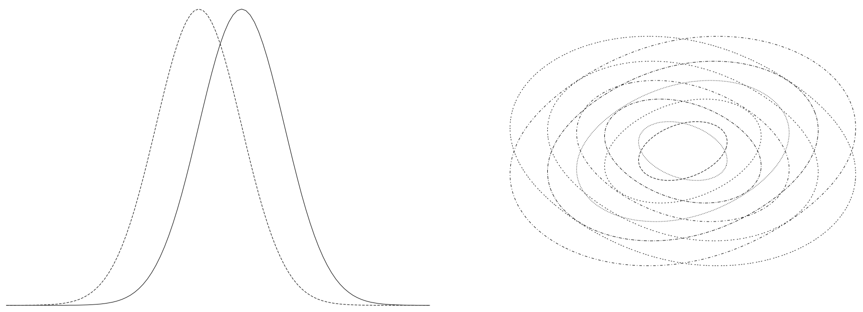

Fig. 1. First order vs. second order DPA distributions. The 2D distributions clearly have more overlap than 1D distributions.

The above result does not depend on a particular form of $f(x)$ or $g(x)$. We will now illustrate this with normal distributions:

- First order DPA: One has to distinguish between two noisy sets with slightly different averages. We thus create two classes, one for logical zero $f \sim N(0,\sigma)$ and one for logical one $g \sim N(1,\sigma)$. See Figure 1 (left).
- Second order DPA: The same individual distributions are used. However, one is interested in the exor of two independent bits. Therefore, we create two 2D distributions. See Figure 1 (right).
  - Since $0 = 0 \oplus 0 = 1 \oplus 1$, f consists of a balanced mix of $N(0,\sigma) \times N(0,\sigma)$ and $N(1,\sigma) \times N(1,\sigma)$.
  - Since $1 = 0 \oplus 1 = 1 \oplus 0$, g consists of a balanced mix of $N(0,\sigma) \times N(1,\sigma)$ and $N(1,\sigma) \times N(0,\sigma)$.

Notice that symmetry implies $D(f\|g) = D(g\|f)$ in this case. We numerically evaluate $D(f\|g)$ with varying σ for both first order and second order DPA. The results are listed in the table below. It appears that the number of samples necessary for second order DPA is much higher than that of first order DPA with the approximate relationship $z_{2^{nd}\mathrm{ODPA}} \approx 2z^2_{1^{st}\mathrm{ODPA}}$.

σ	$z_{1^{st}\mathrm{ODPA}}$	$z_{2^{nd}\mathrm{ODPA}}$
4	8	143
5.7	16	543
8	32	2111
11.3	64	8319
16	128	33023
22.6	256	131586
32	512	525326