

out revealing the keys. A key design principle of the cards is that they are tamper-resistant, which means that it is very difficult to disassemble them in order to modify their functionality, or to find out the inner workings. This task is especially difficult because the threat model includes the attacker having complete control of the device. The user has physical possession of the smart card, and can manipulate it in any way imaginable. Fortunately, a lot of research has gone into this area, and the current state of the art smart cards foil all but the most sophisticated attackers.

However, a more recent class of attacks has focused not on physical disassembly, but on detailed monitoring of the smart cards. They are based on the observation that during operation, the smart card leaks some information to the environment. These "side channel" attacks consist of close measurement of the timings[7] or the power usage[8] of the smart card, which can then be used to verify hypotheses about the internal state. The attacks based on timing can often reveal what operations are being performed; typical defenses against them are ensuring a constant execution path or padding the timings of all execution paths to be the same. The attacks based on power analysis are more sophisticated, and can often reveal not only what operations are being performed, but the values of the individual bits of the operands! Defenses to such attacks are not yet well understood, and they will be the subject of this paper.

We propose an architecture where we isolate a module which performs operations on secret data, and must be made power-analysis-resistant. The rest of the smart card can be designed without concern for power analysis. We then propose a design for this module using one of the previously suggested techniques for resisting power analysis. We show how to apply the technique to new arbitrary operations, discussing in particular those that are necessary for the implementation of a large number of ciphers. We also explore a number of the design and implementation issues associated with designing such a module. Because the topic of power-analysis-based attacks is still fairly new, our design is very conservative; it takes into account a number of potential new attacks.

Power analysis is described in more detail in the following section. Then, Section 3 describes the proposed architecture. Section 4 discusses the design of the module, and Section 5 discusses some implementation details that should be considered. Finally, Section 6 concludes the paper and discusses areas of further research.

2 Power Analysis

This section briefly describes power analysis and how it can be used to attack encryption algorithms. For a more detailed exposition, see [8] and [9]. Power analysis relies on measuring the instantaneous power utilization of a device like a smart card, and using such information to determine the structure of the operations, and information about the data they operate on. In its simplest form, the analysis consists of visually examining a trace, or a set of traces, and drawing observations from them. Often, some structure of the computation is

readily visible, and sometimes it is possible to tell which operations are being performed. When an attacker can find out, for example, which branch of an IF statement is taken, it becomes possible to use such information to draw conclusions about the secret key.

Differential Power Analysis is a statistical approach, where many traces are collected, and are examined for correlations. Typically, a characteristic function on the output is used to divide the traces into two sets. The function is dependent on a partial guess of a key, and it reflects whether the value of a particular bit in the computation is 0 or 1 (if the guess is correct). Then, the averages of the traces for the two sets are compared. If the guess is correct, then at the point corresponding to that bit of computation, the measurements from the first set will have a different bias than the ones from the second one. When the averages of the two sets are subtracted, there will be a spike in the difference, corresponding to that point. If the guess is incorrect, however, there will be no correlation, and values will be effectively random. In such manner, partial key guesses can be verified, and from them the entire key can be derived.

These techniques are really just a form of signal processing used to isolate a signal. With enough measurements, they are able to recover a very weak signal, even in the presence of a lot of noise. One impressive feature of the attack is that it is not necessary to know precise details of the internal structure of the implementation. The attack has been successfully applied to many existing implementations.

2.1 Countermeasures

Many countermeasures have been proposed. One approach is to remove correlations between the instantaneous power utilization of a chip and the internal bits of the computation; ie. eliminating or reducing the signal in the above discussion. Some advocate "balancing" each bit with its complement[5]. Due to design imperfections, timing differences, etc., this approach will not eliminate the signal. It is not yet known, however, whether it will reduce the signal enough to make DPA attacks impossible. Another approach is to use a circuit logic where the power usage is not dependent on the operations performed. The same concerns apply in this case; as well, it may be possible to detect transitions through power analysis, which may be sufficient to find out key material. Another approach is to apply a transformation on the computation, such that any bit of the internal state of the transformed computation is not correlated with any bits in the original. This is an approach that we have adopted in our design, and we explain it in more detail in Section 4. It may be still possible to detect correlations across several bits, but the complexity of the attack will be greatly increased.

Another direction is to hide the structure of the computation. Randomized timings, randomized order of execution, and other such techniques have been proposed. However, it is often possible to detect and reverse the effect of such randomizations. Finally, a suggestion which is more applicable to asymmetric ciphers is to translate the problem into a different domain and perform com-

putations there. It is not clear how generally applicable this technique is, as well as whether observations made in the different domain can be used to gain insight into key material. In general, defenses against DPA are still not very well understood.

3 Architecture

We propose an architecture which attempts to separate the parts of the computation which need to be protected from power analysis from those that do not. This is motivated by the observation that many of the countermeasures described in the previous section are complicated and costly to implement. Applying them requires very careful design and analysis, as well as special purpose operations, or even specialized circuit logic. By isolating the functionality which deals with sensitive data into a separate module, and making the module as simple as possible, we can reduce the cost and complexity of building a DPA-resistant device better. The probability of a simple mistake which could reduce the security of the system is also reduced.

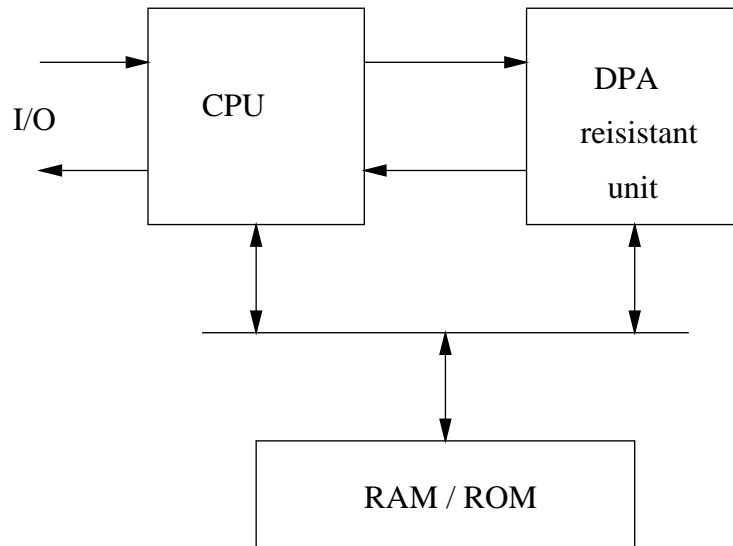


Figure 1: Architecture

Fortunately, this module, which we shall call the *DPA-resistant unit*, needs to perform only a small number of operations. This is because most ciphers are built from simple arithmetic and logic operations. The operations which we support in our design are XOR, NOT, ADD, bit-rotate, AND, and table lookups. This is already sufficient to implement many ciphers, including DES[10] and several of the finalists for the Advanced Encryption Standard such as Rijndael[4],

Serpent[1], Twofish[11]. It is possible to implement other operations, such as multiply, but we felt that supporting a wider variety of ciphers was not important enough to justify making the unit significantly more complex.

The unit itself has no knowledge of encryption algorithm. It is controlled by the CPU, which runs a program to perform the encryption. The unit also has access to memory to be able to pass operands back and forth, as well as access lookup tables (see Figure 1). The unit becomes a sort of ALU, which is designed with DPA-resistance in mind. This simplification is possible because any strong encryption algorithm does not rely on the secrecy of the algorithm to remain secure. Hence, the fact that power analysis may reveal the structure of the computation is acceptable, as long as the actual secret data remains unknown. This simplified design of the DPA-resistant unit has another advantage: it is possible to perform some empirical analysis to verify the resistance of the unit to power analysis. For example, for an 8-bit ADD, it is possible to run the unit with all of the 2^8 possible inputs, measure instantaneous power utilization, and look for any correlations in the resulting traces. Such an analysis is impossible for an entire cipher because the space of inputs is much too large. The unit is also flexible enough to be reused in various smart cards, even with different ciphers. This allows for the design cost to be shared among many applications.

4 Design of the Module

To protect data and operations from power analysis within our module, we use an encoding scheme first proposed in [3]. Each bit b of data internal to the module is split into k shares, b_1, \dots, b_k , such that $b = b_1 \oplus \dots \oplus b_k$. The shares should be assigned randomly, such that there exist no correlation in any $k - 1$ subset of the shares. Then, computation is performed on the shares separately, without recombining them until the output stage. This ensures that no single bit of the intermediate computation is correlated with the bits in the actual computation. Such measures have been shown to increase the number of samples required for differential power analysis exponentially.

4.1 Random Source

To split a bit into shares, it is necessary to generate some random data. In a hardware implementation, it can sometimes be tempting to draw data from environmental sources, but given the context of smart cards, one has to proceed on the assumption that the adversary can control the environmental conditions of the device. This would enable the adversary to predict the way in which the input bits are split, and hence it may be possible to find key-dependent correlations between intermediate steps in the computation and the output or input data. This would lead to a DPA attack.

Therefore, it is important that the attacker cannot predict how the shares are computed. A practical way to achieve this is to use a cryptographically strong pseudo-random number generator. The complexity of such generators can be

non-trivial; frequently, they are built using an existing cipher in a special mode. Furthermore, the PRNG must be keyed, and the key must be kept secret from the attacker, the same as the key of the main encryption algorithm. However, the PRNG does not need the same kind of protection as the main encryption algorithm, because it is significantly harder to mount a DPA on it. DPA relies on finding correlations between the known plaintext or ciphertext and the internal state of the computation. However, in the case of the PRNG, the output of the generator only affects the internal state of the computation, and does not produce a visible difference in the output.

Hence, it should be possible for the PRNG algorithm to run on the CPU, and feed the random numbers to the DPA-resistant unit. The operation of the PRNG must still be designed with care. For example, there exists a class of power analysis attacks which does not require any known plaintext or ciphertext [2], based on the hamming weight of some of the internal state. Furthermore, it may be possible to gain great insight into the output of the PRNG if the plaintext is known, as it is possible to look for correlations between the plaintext bits before the sharing transformation and the resulting shares, when the plaintext is being loaded into the DPA-resistant unit.

A possible way to implement this PRNG is to use a cipher which is resistant to hamming weight analysis of the key schedule in output feedback mode. This leaves the attacker with no information about the input or the output of the PRNG cipher. To reduce the value of the correlation attacks, in the cases where the plaintext for the main cipher is known, we can change the key used in the PRNG cipher by passing it through a one-way hash function, such as SHA[12], after some number of uses of the PRNG. This was suggested in [8] as a general mechanism to prevent DPA, as re-keying in such manner should effectively destroy any built information accumulated through correlation sampling. It is harder to apply this principle to the main cipher, since changing keys often requires complex modification of the protocols and/or the application which uses the cipher.

The above design of the random number generator may be over-engineered. We have chosen a conservative approach because we are aiming to defend from attacks which are not yet very well understood. The current very little about DPA attacks which involve no known ciphertext or plaintext, and some of the higher order correlation attacks exist only as informal conjectures in the cryptography research community. A further exploration of these topics should make it more apparent what level of security the PRNG needs to achieve, and may lead to a faster and cheaper implementation of the PRNG. As it is, we are hopeful that our design ensures that the PRNG is not the weakest point of the system.

4.2 Basic operations

When a byte is loaded into the DPA-resistant unit, it needs to be split into k shares before it is stored in a register. (This implies that the registers need to be kn bits wide, where n is the size of the input data). Since the random data is

generated on the CPU, a simplification is to have the CPU compute the shares of the input data itself, and load those into the DPA-resistant unit. A simple way to generate a shared bit is to generate $k - 1$ random bits b_1, \dots, b_{k-1} , and let $b_k = b_1 \oplus \dots \oplus b_{k-1} \oplus b$, where b is the value of the data bit. This operation can be easily translated to a byte or word operation to split an entire byte or word of the input simultaneously. Key material will be stored in ROM or non-volatile RAM, and it should be stored in an already-split state. The split can be calculated at keying time, which means that any faults in the on-card random number generator will not affect how uniform the key splitting action is done.

Operations such as XOR, NOT, and bit rotates can be easily performed on shares. This is because they are linear operations in $GF(2)$, which means that the operation can be performed separately on each share, and the results will be the correct shares for the result of the whole computation. For example, to XOR two 8-bit values, A and B , each of which has been split into two shares, A_0, A_1 , and B_0, B_1 respectively, we compute the two new shares $C_0 = A_0 \oplus B_0$ and $C_1 = A_1 \oplus B_1$. It is best if operations on both shares are performed simultaneously, since it will make it harder to isolate each each half of the computation; this is easily done on a custom-built DPA-resistant unit.

4.3 AND

AND is not a common operation in of ciphers; however, we shall use it as a building block for all non-linear operations. AND has the property that it is impossible to perform a computation on each share independently and obtain a correct result. However, recombining the shares to perform an AND is unacceptable. We propose a design which avoids recombining the shares. Let $A = A_0 \oplus A_1$ and $B = B_0 \oplus B_1$, and let us try to compute shares of AB . We start by observing the distributive property of AND and XOR:

$$\begin{aligned} AB &= (A_0 \oplus A_1)(B_0 \oplus B_1) \\ &= A_0B_0 \oplus A_0B_1 \oplus A_1B_0 \oplus A_1B_1 \end{aligned}$$

We can compute shares for AB as, for example $(AB)_0 = A_0B_0 \oplus A_1B_1$ and $(AB)_1 = A_0B_1 \oplus A_1B_0$. Observe that we did not have to recombine the inputs at any point. However, these shares are non-uniformly distributed: for a random setting of A_0, A_1, B_0, B_1 , $(AB)_0$ agrees with AB with probability $5/8$. While it's unclear whether this property can lead to an attack directly, we would like to do better.

Observe that for any function $f(A_0, A_1, B_0, B_1) \rightarrow \{0, 1\}$, we can let

$$\begin{aligned} g(A_0, A_1, B_0, B_1) &= f(A_0, A_1, B_0, B_1) \oplus \\ &\quad (A_0 \oplus A_1)(B_0 \oplus B_1) \end{aligned}$$

Then we can set $(AB)_0 = f$ and $(AB)_1 = g$, which will be correct shares of AB . However, we want the choice of the shares to be uncorrelated. To be more precise, for a random choice of A_0, A_1, B_0, B_1 , we want each share to agree with

A_0	A_1	B_0	B_1	f	g
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	1	1
0	0	1	1	1	1
0	1	0	0	0	0
0	1	0	1	0	1
0	1	1	0	1	0
0	1	1	1	1	1
1	0	0	0	1	1
1	0	0	1	1	0
1	0	1	0	0	1
1	0	1	1	0	0
1	1	0	0	1	1
1	1	0	1	1	1
1	1	1	0	0	0
1	1	1	1	0	0

Figure 2: $f = A_0 \oplus B_0$

AB with probability $1/2$. We also want to ensure that each share agrees with A with probability $1/2$, and with B with probability $1/2$. This way, it will be impossible to find correlations between the shares and the actual values in the computation.

An examination of the 2^{16} possible values of f shows that there are 1296 choices which satisfy these constraints. One such f is shown in Figure 2. We can compute f and g directly (for example, $f = A_0 \oplus B_0$ in this case), or simply use an 4-to-2 bit lookup table to implement this operation. In either case, we obtain shares which are uniformly distributed. We favor the lookup table approach, since some of the intermediates computed in the direct computation can be slightly correlated with one of the inputs or the outputs. A lookup table should, in theory, offer fewer possibilities of finding a signal correlated with any of the above, but more tests are needed to verify this in practice. Another useful property of the lookup table is that any of the 1296 choices may be used equivalently. The exact choice of lookup table can be easily kept secret, and it can even be changed at runtime. Such a design also maps easily onto an FPGA-based implementation, since one or more CLBs can be used to implement a function such as f directly.

It is an easy to see that functions with these constraints exist for $k > 2$ as well. For larger values of k , it may become impractical to even consider the direct implementation of the function. The lookup table size grows with k ; however, observe that we only need one lookup table to support all AND operations.

4.4 ADD

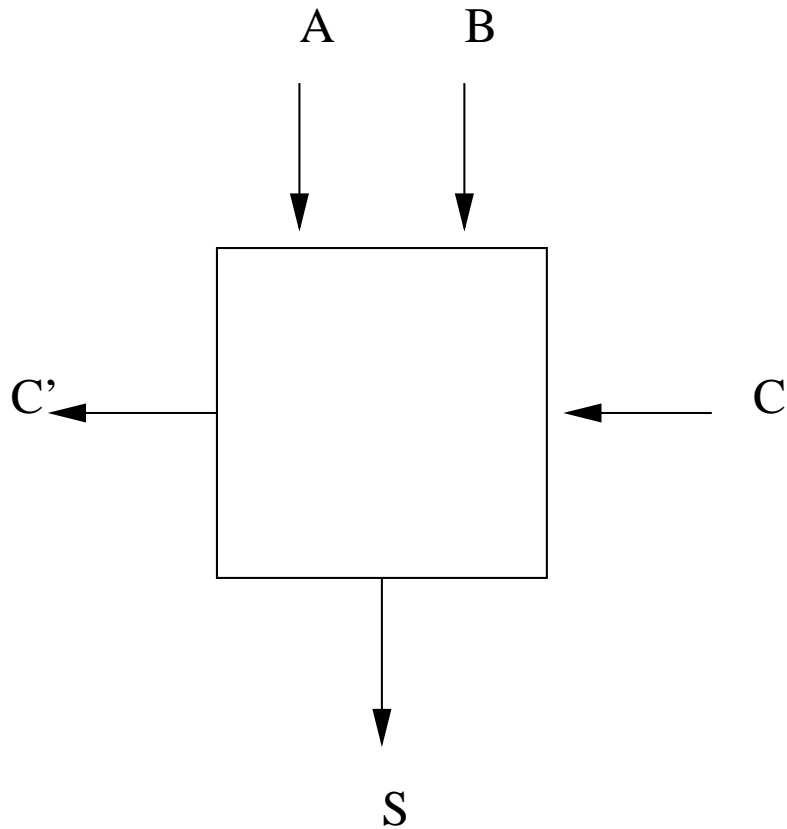


Figure 3: A 1-bit adder

ADD is a non-linear operation more commonly found in ciphers. We demonstrate how to build a 1-bit adder, as shown in Figure 3. C is the carry in and C' is the carry out. Observe that:

$$S = A \oplus B \oplus C$$

$$C' = AB \oplus AC \oplus BC$$

We can therefore build the adder using AND and XOR operations (this is in fact true for any boolean operation). However, we encounter a problem. Suppose the f from Figure 2 is used to implement each AND operation. Then

$$\begin{aligned} C'_0 &= (AB)_0 \oplus (AC)_0 \oplus (BC)_0 \\ &= A_0 \oplus B_0 \oplus A_0 \oplus C_0 \oplus B_0 \oplus C_0 = 0 \end{aligned}$$

So we have $C'_0 = 0$, and hence $C'_1 = C'$. This is because, while each split share is uncorrelated with the inputs, there is correlation in how the shares are split in each AND operation. In general, combining results of related AND operations can lead to correlated results. One way to avoid this is to "refresh the split" at the end of each AND operation, by XORing each share with a random bit. This effectively destroys any correlation.

However, for the ADD example, we would require 3-random bit for each 1-bit adder; or in general $3n$ bits to add two n -bit numbers. To avoid putting such strain on the PRNG, we can use an explicit lookup table for the adder instead. We can construct f and $g = f \oplus C'$ as above, and ensure that there are no correlations. Assigning f to be $A_0 \oplus B_0 \oplus C_0$ satisfies the constraint,

If there are related ADDs used in the cipher, we still need to mix in random bits, to avoid correlations as with ANDs before. The DPA-resistant unit should have an instruction which resplits an internal bit (or byte, or word) based on input from the PRNG. To be clear about the terminology, the 1-bit adders in a ripple adder are not related, as no bit is used as an input to more than one adder. Many ciphers include a large amount of key-dependent operations between each ADD, so it may be safe to forego the mixing step, as the additional key material should destroy any possible correlations between the shares.

4.5 S-Boxes

Many ciphers also employ a general (in some cases key-dependent) non-linear function called an S-box. The S-box is simply a lookup table which takes n input bits to m output bits. Use of an S-box is very common, and it is probably the most difficult part of building a power-analysis-resistant cipher implementation.

We can extend the ideas in the previous sections to support the use of S-boxes in our DPA-resistant module. The most natural approach to implement a lookup table based on split inputs is to build a bigger table. Instead of an m -to- n lookup table, we can build a km -to- kn lookup table. Once again, we have to ensure that there are no correlations present, but we have an incredibly large number of possible assignments, so it should be easy to find one satisfying this constraint. However, the memory requirements for such lookup tables can be excessive. They grow at an exponential rate as k increases, and seeing as memory is already a scarce commodity on smart cards, this approach is not viable for $k > 2$ (and sometimes even for $k = 2$). a viable approach.

But recall our earlier statement that any boolean function can be expressed in terms of AND and XOR. This means that if we can decompose the S-box function into a combination of ANDs and XORs, we can compute it directly without using any lookup tables (other than the one used to implement the AND). Of course, the code size necessary to implement the sequence of ANDs and XORs is quite large, but it should be comparable to the size of the original S-Box. Most importantly, the amount of storage necessary grows in direct proportion to k , and not exponentially.

Depending on the size of the table, the performance impact of computing the S-box this manner can be significant. We are in the process of investigating

several ways to ameliorate this, through either the use of time-memory tradeoffs, or through approaches analogous to Karnaugh maps which would let us find the minimal set of operations necessary. We are also considering allowing a richer set of operations. Fortunately, space is a larger concern for most smart cards than performance, since they typically operate infrequently and on small amounts of data.

This approach also imposes a significant load on the PRNG, since the computation is bound to involve many related ANDs, and "refreshing" of the shares is required. Overall, we feel that S-box lookup is the weakest point of our design, which is reflective of the state of the art in DPA countermeasures. Our hope is that future research will allow for much better S-box implementations.

5 Implementation Concerns

5.1 Choice of k

It is important to choose a reasonable value of k . A larger value of k will result in more samples being necessary to mount a DPA attack on the system, but it also imposes significant overhead. The direct impact is that for every bit in the original computation there are k bits in the modified one. The size of the lookup tables for an operation like AND or ADD increase exponentially with k , which can be a limiting factor. For example, the lookup table for ADD has the size of $2k \cdot 2^{k^3}$ bits, which requires 96 megabytes of storage for $k = 3$, clearly unrealistic for a smart card. The lookup tables for S-boxes also increase exponentially; in some cases, even the case of $k = 2$ can be not viable.

Implementing these operations as a combination of ANDs and XORs can reduce the exponential blow up. The lookup table for AND may be feasible for $k = 4$ on larger smart cards, since its size is only $k \cdot 2^{k^2}$, and it can be reused for all operations. However, this approach has a penalty on performance, which is especially heavy in the case of S-box lookup. It also puts a heavy strain on the random number generator, as resplitting of shares must be done after successive AND operations. We recommend using the value of $k = 2$, or, if the implementation constraints allow it, $k = 3$. Most current DPA attacks require hundreds or thousands of samples; with even a quadratic blow-up, the number of samples can easily be driven into the impossible range.

5.2 Resplitting of Shares

Resplitting of shares for a single bit consists of XORing $k - 1$ of the shares with random bits, and XORing the last bit with their parity. This results in a new share assignment for the bits. This operation has the property that if the shares were already uncorrelated, they will remain uncorrelated, regardless of the randomness characteristic of the new bits from the PRNG; in other words they do not introduce new correlations. On the other hand, even if correlation existed in the shares before the resplitting, if the PRNG bits are truly random,

the correlation will disappear.

Hence, resplitting of shares of a bit does not affect the correctness of the algorithm, and should not have negative security implications, either. As such, resplitting could potentially be done at every intermediate stage of the computation. But we feel that doing this is unwarranted; in most cases, resplitting seems to be added complexity with no visible gain. We recommend resplitting bits after a non-linear function application (such as AND, ADD, S-box lookup, etc.), to destroy correlations which may have been introduced there. We also recommend resplitting the key material every encryption, or every few encryptions. Otherwise, it may be possible after many traces to determine how the key material is split², and then use those traces to mount a DPA attack.

5.3 Non-Volatile RAM

Such resplitting, as well as operation of the PRNG, assumes the availability of non-volatile RAM in the smart card. However, non-volatile storage is expensive, the cost of updating it can be large, and the number of duty cycles can be smaller than is desirable (sometime in the 10,000 range). To partially address these issues, possible to use a comparatively small amount of non-volatile storage which is updated with every use of the device, to keep a usage counter. The counter can be used to ensure that the PRNG produces different values on each run, and it can also be used to resplit the key material every n iterations, for some n .

If no NVRAM is available at all (at least not for update on every use), then it is difficult for the PRNG to generate different splits for multiple runs using the same input data, since the attacker can duplicate the external conditions in each run with great precision. It is possible, however, to produce different random values based on something like the hash of the input. This may be sufficient, since DPA relies on collecting many runs on different values of inputs, as opposed to the same one. It is important that these hashes be mixed with some secret material, otherwise an attacker who knows the structure of the device will be able to predict the splits.

5.4 Artificial Delays

Introducing artificial delays into the computation further reduces the feasibility of a DPA attack. By increasing the computational latency, we effectively decrease the number of samples that the attacker can collect within a certain period of time. However, care must be applied to ensure that such delays are not easily circumventable. For example, in an authentication token, it is tempting to introduce the delays *after* the authentication takes place. The advantage of doing so is that there is no visible effect for a normal user. However, a clever attacker can cut the power to the card as soon as the authentication data is

²There are no current attacks which do this. However, we conjecture that there may exist sophisticated future attacks which are able to achieve this, and we would like to defend against them.

received from the device. Therefore, such delays must be introduced before output is generated. Note that the same concern applies to actions which update the NVRAM after an encryption.

6 Conclusion

We have presented an architecture for building DPA-resistant smart card, where the resistance to power analysis is isolated to a small module. We also presented a design for this module, using an encoding technique. In doing so, we have discovered a new way to implement arbitrary functions and make them resistant to power analysis attacks. Because of the scope of our research project, we did not implement a prototype; however, the detailed discussion of the design and implementation issues that we offer should be helpful in doing so in the future.

Our proposed design can be used as a basis for theoretical and empirical analysis of the design principles that it contains. The area of power analysis and defenses against it is still not very well understood; the current research in the area feels like an arms race of clever defenses and more clever attacks (this is somewhat true in the field of cryptography in general). Our work furthers the exploration of this area, and is a step towards a better understanding of power analysis. Through more of these steps, we can hope to reach some fundamental results in this area, which will let us make strong statements about this class of attack.

References

- [1] E. Biham, R. Anderson, and L. R. Knudsen. Serpent: A new block cipher proposal. *Lecture Notes in Computer Science*, 1372:222–??, 1998.
- [2] E. Biham and A. Shamir. Power analysis of the key scheduling of the AES candidates. In *Second AES Candidate Conference*. National Institute of Standards and Technology, May 1999.
- [3] S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi. Towards sound approaches to counteract power-analysis attacks. *Lecture Notes in Computer Science*, 1666:398–412, 1999.
- [4] J. Daemen and V. Rijmen. Rijndael. NIST AES Candidate Algorithm, Description available at <http://www.esat.kuleuven.ac.be/~rijmen/rijndael>.
- [5] Joan Daemen and Vincent Rijmen. Resistance against implementation attacks: A comparative study of the AES proposals. In *Second AES Candidate Conference*. National Institute of Standards and Technology, May 1999.
- [6] Ian Goldberg and David Wagner. GSM cloning. <http://www.isaac.cs.berkeley.edu/isaac/gsm.html>.

- [7] P. Kocher. Cryptanalysis of Diffie-Hellman, RSA, DSS, and other cryptosystems using timing attacks. In Don Coppersmith, editor, *Advances in cryptology, CRYPTO '95: 15th Annual International Cryptology Conference, Santa Barbara, California, USA, August 27–31, 1995: proceedings*, volume 963 of *Lecture Notes in Computer Science*, pages 171–183, Berlin, Germany / Heidelberg, Germany / London, UK / etc., 1995. Springer-Verlag.
- [8] P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In Michael Wiener, editor, *Advances in cryptology — CRYPTO '99: 19th annual international cryptology conference, Santa Barbara, California, USA, August 15–19, 1999 proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397, Berlin, Germany / Heidelberg, Germany / London, UK / etc., 1999. Springer-Verlag.
- [9] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Introduction to differential power analysis and related attacks. <http://www.cryptography.com/dpa/technical/>.
- [10] National Bureau of Standards. *Data Encryption Standard*. U. S. Department of Commerce, Washington, DC, USA, January 1977.
- [11] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson. Twofish: A 128-bit block cipher. Technical report, Counterpane Systems, 101 East Minnehaha Parkway, Minneapolis, MN 55419, June 1998.
- [12] William Stallings. SHA: The Secure Hash Algorithm. *Dr. Dobb's Journal of Software Tools*, 19(4):32, 34, April 1994.