# Implementing Virtual Secure Circuit Using A Custom-Instruction Approach

Zhimin Chen
Virginia Tech.
Blacksburg, VA 24060
chenzm@vt.edu

Ambuj Sinha
Virginia Tech.
Blacksburg, VA 24060
ambujs87@vt.edu

Patrick Schaumont
Virginia Tech.
Blacksburg, VA 24060
schaum@vt.edu

## ABSTRACT

Although cryptographic algorithms are designed to resist at least thousands of years of cryptoanalysis, implementing them with either software or hardware usually leaks additional information which may enable the attackers to break the cryptographic systems within days. A Side Channel Attack (SCA) is such a kind of attack that breaks a security system at a low cost within a short time. SCA uses side-channel leakage, such as the cryptographic implementations' execution time, power dissipation and magnetic radiation. This paper presents a countermeasure to protect software-based cryptography from SCA by emulating the behavior of the secure hardware circuits. The emulation is done by introducing two simple complementary instructions to the processor and applying a secure programming style. We call the resulting secure software program a Virtual Secure Circuit (VSC). VSC inherits the idea of a secure logic circuit, a hardware SCA countermeasure. It not only maintains the secure circuits' generality without limitation to a specific algorithm, but also increases its flexibility. Experiments on a prototype implementation demonstrated that the new countermeasure considerably increases the difficulty of the attacks by 20 times, which is in the same order as the improvement achieved by the dedicated secure hardware circuits. Therefore, we conclude that VSC is an efficient way to protect cryptographic software.

## Categories and Subject Descriptors

C.3 [**Computer Systems Organization**]: Special-Purpose and Application-Based Systems

## General Terms

Design, Security

## Keywords

Virtual Secure Circuit (VSC), balanced instructions, Side Channel Attacks (SCA)


*CASES'10,* October 24–29, 2010, Scottsdale, Arizona, USA.


## 1. INTRODUCTION

Cryptographic algorithms are designed to resist at least thousands of years of cryptoanalysis. That is, given that the attackers know the algorithm, the input data and the output data, any known method to extract the crypto-algorithm's secret key has an enormous computational complexity.

Unfortunately, the security features of an algorithm alone are not sufficient to guarantee that their implementations are also secure. One decade ago, Kocher, *et. al* introduced the Side Channel Attack (SCA) [10]. Besides the information used by cryptoanalysis, SCA also makes use of other information, such as the cryptographic implementations' execution time, power dissipation, or/and magnetic radiation. We call this additional information *side-channel leakage*. SCA is able to analyze the relationship between the side-channel leakage and the implementation's internal states and finally breaks a cryptographic system at a much lower computational cost. For example, if an attacker is able to check 1 billion AES keys per second, the Advanced Encryption Standard (AES) algorithm [14] can resist a brute force attack for $10^{13}$ years. In contrast, according to our experiments, SCA can break AES, which is implemented by embedded software, within 1 hour. This brings a great concern to the cryptography field, especially to the embedded security, since side channel leakage from embedded devices is often easy to obtain. This paper focuses on protecting cryptographic software implementations against SCA from a processor architecture perspective via custom instructions.

Countermeasures against SCA broadly fall into two categories, including *hiding* and *randomization*. Hiding offers protection by reducing the signal-to-noise ratio of the side channel leakage. Randomization techniques can randomize the processed values, called *masking* [2, 16, 8], or can randomize the time a sensitive value appears, e.g. random-code injection [3], or can randomize the voltage and frequency with Dynamic Voltage and Frequency Scaling (DVFS) [21]. Randomization techniques are not without problems; random-code injection were shown to be ineffective [7], and randomized voltage/frequency can easily observed and attacked. Moreover, randomization in the form of masking is algorithm-specific. This paper uses hiding rather than randomization, and it presents an algorithm-independent countermeasure.

In hardware, hiding can be implemented using Dual-Rail Pre-charge (DRP) circuits [6]. This paper builds the software equivalent of such DRP circuits. DRP circuits consist of two parts: a direct part and a complementary part. An ideal implementation of a DRP circuit has a constant static

and dynamic power dissipation (as will be described in Section 2). This way, the power dissipation signal of the DRP circuit contains no more side-channel information.

So far, the DRP technique has not been used to protect cryptographic software. This is because regular processors don't have support for DRP-like behavior. This paper proposes a solution to this problem and successfully ports the DRP technique to software. In general, the main concept is to add special custom instructions that enable part of the processor as a programmable DRP circuit. The new processor is called *balanced processor*. In addition, a special programming style further guarantees that the processors act exactly the same as the DRP circuits. As a result, the power dissipation of the processor should be ideally independent on the processed data and therefore resists SCA. We call the resulting software Virtual Secure Circuit (VSC).

The first advantage of the proposed solution is its flexibility. Once the balanced processor is built up, the side-channel resistance is achieved completely in software. The second advantage is that the proposed solution is not limited to a specific cryptographic algorithm. This makes our hiding-based technique more general than masking-based techniques currently used to protect crypto-software. The third advantage is the much lower hardware cost when compared with dedicated hardware DRP circuits. Our results show that hiding-based countermeasures can be applied using the existing ASIP technology. Hence, our technique can be directly applied by ASIP users with a need for side-channel resistance in their designs.

The contributions of this paper include 1) the concept that using a single balanced processor to support VSC to thwart SCA, 2) a low-cost solution to implement a balanced processor and the demonstration of its security, 3) a secure programming style to implement a VSC, and 4) the demonstration and quantification of the effectiveness of our solution with real side channel attacks.

The rest of this paper is organized as follows. Section 2 introduces the basic knowledge on SCA and the DRP technique. The proposed solution, VSC, is described and analyzed in Section 3. In addition, Section 4 presents a general way to program VSC. Section 5 demonstrates VSC's effectiveness with experiments and Section 6 makes a conclusion.

# 2. PRELIMINARIES

In this section, we introduce some preliminary knowledge on embedded security, including SCA, DRP technique, and the idea of quantifying security against SCA.

## 2.1 Side Channel Attacks

SCA can break a cryptographic device much faster than cryptoanalysis. The reason is that, by exploiting the side-channel leakage of a device, SCA can reveal the secret key bit by bit or piece by piece. This breaks the exponential complexity of a brute force key search. In this section, we describe the concept of SCA.

Figure 1 shows a cryptographic device. The device implements a cryptographic algorithm, represented by $f$. $f$ takes the plaintext ($PT$) and the key ($K$) as inputs, and generates the encryption result ($ER$) ($ER = f(PT, K)$). The internal secret key $K$ is not directly observable through the ports of the device. The objective of SCA is to reveal the value of $K$.

Each bit of $ER$ is related to every bit of $PT$ and every bit of $K$. Suppose $K$ has 128 bits, then a traditional *brute-force* attack needs to consider $2^{128}$ possible key values. It is this huge search space that ensures the security of the cryptographic algorithm. However, SCA does not try to break the entire $K$ at once. Instead, SCA divides $K$ into pieces, which are broken one by one. For example, a typical AES algorithm uses an 128-bit key. Using an SCA that reveals one key byte at a time, the search space for the entire key is reduced from $2^{128}$ to $16 * 2^8 = 2^{12}$.

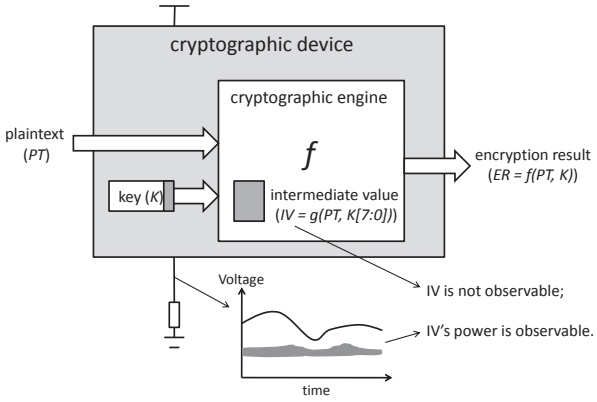


**Figure 1: An example of Side Channel Attack attacks a crypto-device and uncovers the key enclosed.**

Obviously, this is a drastic reduction, and SCA achieves this as follows. Although every bit of the output $ER$ is guaranteed to be related to every bit of $K$, this is not true for the intermediate values. We can always find intermediate values that are only related to a small part of $K$, e.g. one byte of $K$. For example, assume that we can find an intermediate value $IV$ which depends on a single key byte $K[7:0]$ and the plaintext $PT$. We can write $IV = g(PT, K[7:0])$. Then $K[7:0]$ can be discovered with only $2^8$ guesses.

To test which guess is correct, we therefore need to observe $IV$. This variable is inside of the implementation, but it is indirectly observable through its power dissipation. Indeed, the power dissipated by $IV$ is a part of the power dissipated by the entire device, which can be measured by the attackers. Hence, by measurement of the chip power dissipation, there is a way to test the guesses the attacker makes on a single key byte. Through proper correlation techniques, the chip overall power dissipation can be used in place of the power dissipation from $IV$; the power dissipated by unrelated components can be treated as noise.

In recent years, very powerful SCA techniques have emerged. Modern secure embedded systems therefore need to apply adequate countermeasures to prevent SCA. In the next section, we review a hardware countermeasure technique developed for this purpose.

## 2.2 DRP Technique

DRP technique is an effective countermeasure against SCA. Its basis is to reduce the side-channel information from the power dissipated by the intermediate variable $IV$. Hardware circuits implemented as DRP have a constant power dissipation. We use an example to explain the details.

Figure 2 explains the operation of the DRP technique using a DRP NAND gate. In this example, we approximate

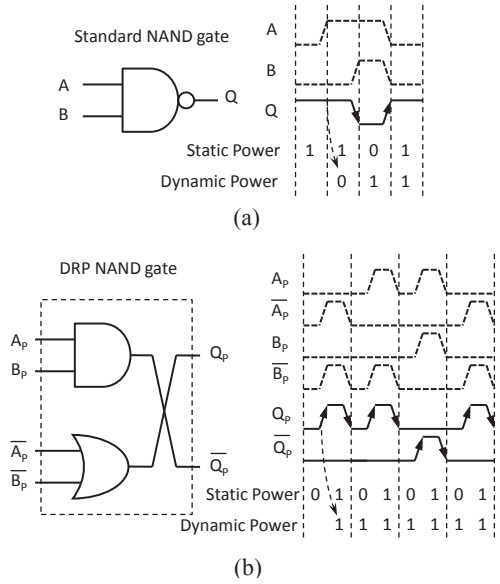


**Figure 2: (a) A CMOS standard NAND has data-dependent power dissipation; (b) A DRP NAND gate has a data-independent power dissipation.**

static and dynamic power dissipation of a logic gate through the Hamming Weight and Hamming Distance of its output respectively. In the case of a single NAND gate (Figure 2a), the static and dynamic power dissipation depend on the input values of the gate. For example, if the static power is 0, both inputs must be 1. This side-channel leakage is the basis for SCA.

Figure 2b shows the same test case on a DRP NAND gate. In this case, the circuit encodes each logic value with a complementary pair $(A_p, \overline{A_p})$. Furthermore, each pair is pre-charged to (0,0) in each clock cycle before evaluation. As a result, each clock cycle, every DRP signal pair shows exactly one transition from 0 to 1 and another one from 1 to 0. The resulting static and dynamic power dissipation are now independent of the input values of the DRP gate.

Despite the elegance of this concept, DRP circuits in hardware do have some disadvantages. First, DRP circuits are at least two times larger than equivalent standard CMOS circuits, and they have a much larger power dissipation. Second, the constant-power argument, based on Hamming Weight or Hamming Distance, does not hold when low-level electrical effects are taken into account. Small asymmetries between the direct and complementary paths of a signal pair still may lead to residual side-channel leakage. Nevertheless, careful design is able to reduce the imbalance to a very low level. This requires much more power measurements for a successful SCA [18].

So far, the DRP technique has been broadly used in hardware as secure circuits, for example in SABL [19], in WDDL [20], and in MDPL [17]. However, software DRP technique has not been developed. The major reason for this is that DRP technique requires the executions of the direct and complementary datapaths in parallel. In regular processors, this cannot be realized.

## 2.3 Quantifying Security

Researchers have realized that it is really difficult or even impossible to design one single SCA countermeasure that is absolutely secure. Usually, whether a countermeasure is secure or not depends not only on the countermeasure itself, but also on the secure system in which the countermeasure is integrated. Instead of evaluating a countermeasure as absolutely secure or not, a more practical way is to quantify the protection offered by a countermeasure.

A well accepted way to quantify the security of SCA countermeasures is to use the number of power measurements needed for a successful attack. When introducing a countermeasure (in hardware and software), the crypto-engineer makes a trade-off between the additional hardware/ software implementation cost, and the additional security offered by this countermeasure. In this paper, we use the same method to evaluate our solution. Results will show that the improvement obtained by the proposed software solution is in the same order as the one obtained by dedicated DRP hardware circuits.

No SCA countermeasure can provide perfect security by itself. However, in practice, security can be achieved by combining different countermeasures at different levels of abstraction. An example is to combine a protocol-level countermeasure that decreases the secret key's lifetime and the solution proposed in this paper that can increase the number of measurements. In this case, the secret key will be changed before the SCA can collect sufficient measurements.

# 3. VIRTUAL SECURE CIRCUIT ON A BALANCED PROCESSOR

This section proposes a new SCA countermeasure based on the DRP technique. Different from the previous DRP-circuit countermeasures, the new one is a software solution. It emulates the DRP circuits' behavior with software, and therefore, is called Virtual Secure Circuit (VSC).

## 3.1 Concept

The concept is illustrated in Figure 3. Similar to the DRP circuits, the processor has two parts: one performs direct operations and the other performs complementary operations. We call the processor a balanced processor. Every direct operation in the first part has a complementary counterpart in the second part. A direct operation takes input $in$ and generates output $out$. The complementary operation takes input $\overline{in}$ and generates output $\overline{out}$. $\overline{in} = NOT(in)$ and $\overline{out} = NOT(out)$. $NOT$ means the bit-wise inversion operation.

The balanced processor has a set of instructions, referred to as balanced instructions. Each balanced instruction chooses a pair of complementary operations from two parts of the processor and executes them at the same time. The cryptographic algorithm is programmed with this set of balanced instructions. Therefore, the cryptographic algorithm has both a direct execution path and a complementary path. To run the cryptographic algorithm, both direct and complementary plaintext (balanced input) are supplied to these two paths. Finally, the algorithm generates both the direct and the complementary encryption results (balanced outputs), shown in Figure 3a.

Besides the balanced instructions, the processor also has another set of instructions that perform the pre-charge operations in the same way as the DRP circuits. Before executing balanced instructions, a set of pre-charge instructions first

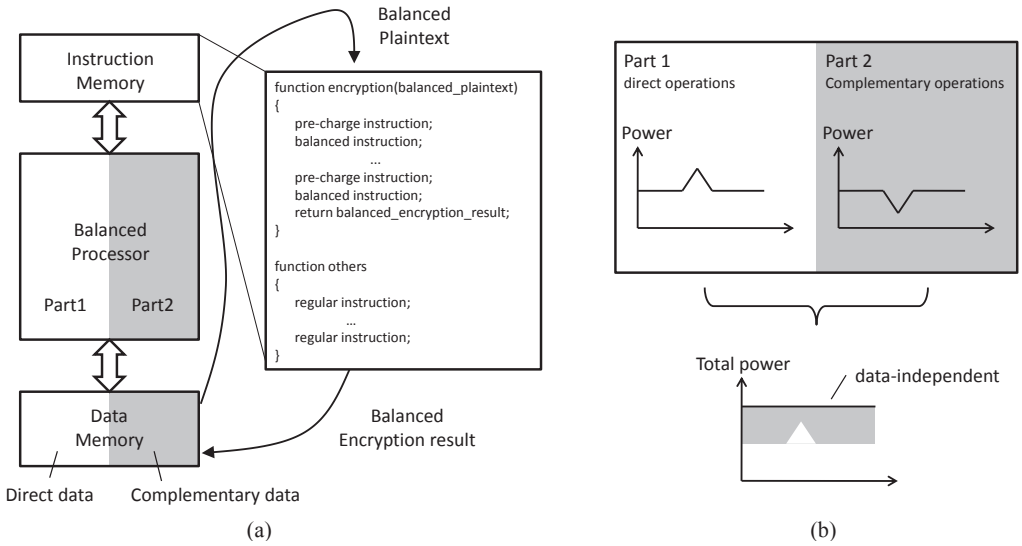


**Figure 3: (a) Concept of the proposed solution: balanced processor and VSC programming; (b) Power dissipation from the new balanced processor system does not reveal the processed data.**

clear the execution datapath. Following that, the balanced instruction finishes the calculation.

With the above concept, we obtain a software version of DRP circuit. For a balanced instruction, similar to the DRP gate shown in Figure 2, the power dissipation from the direct operation always has a complementary counterpart from the complementary operation. The sum of these two is a constant, shown in Figure 3b. Suppose the output of the balanced instructions contains the direct and complementary forms of the intermediate value $IV$ mentioned in Section 2, attackers are not able to go from the power of the balanced instructions to the value of $IV$. Therefore, $IV$ is protected. If every intermediate value in the cryptographic algorithm is processed by the balanced instructions, then the entire algorithm is secured. This kind of software program is the VSC.

## 3.2 Implementation

The previous section describes the concept of VSC. The next step is to map the concept to the processor implementation. This includes the implementation of the balanced instructions, the pre-charge instructions and the secure programming style that leads to DRP behavior.

### *3.2.1 Balanced Instructions*

The instruction set architecture of a processor includes different instructions. The logic function of the software is realized with logic instructions, such as AND, OR, and NOT. In theory, any algorithm can be realized with these three instructions. For higher efficiency, processors also implement the arithmetic instructions that are frequently used, such as ADD, SUBTRACT, and MULTIPLY. In addition to that, due to the processor architecture, shift instructions and data instructions, such as MOVE, LOAD, and STORE, also appear in the instruction set architecture. Finally, the processor also provides the control instructions, such as JUMP and conditional JUMP.

As mentioned above, each balanced instruction executes a pair of complementary operations. To make the balanced datapath compatible with the regular datapath, the width of either the direct or the complementary datapaths is designed to be half of the regular datapath's width. Thus, the entire datapath of a balanced instruction is as wide as the regular instructions.

After fixing the width of the balanced instructions, the next step is to fix the functions. The integration of balanced instructions follows the rule that balanced instructions take balanced inputs and generate balanced outputs. The proposed solution chooses balanced instructions according to the existing regular instruction set architecture. This enables the design flow of the VSC software to reuse the existing compiler for the regular instructions. The details are as follows.

Among the logic operations, NOT is the complementary instruction for itself. This means that if half of the input is direct value and the other half is complementary value, the output of NOT is still a pair of complementary values. Therefore, the regular NOT instruction can also be used as the balanced NOT instruction. The AND operation has OR operation as complementary counterpart and vice versa. Therefore, the balanced AND instruction (b_and) should be half AND for the direct input and half OR for the complementary input. Similarly, the balanced OR instruction (b_or) should be half OR and half AND, shown in Figure 4. With balanced AND, OR, and NOT, any complex logic function can be realized.

Shift and data instructions are similar. They both move the programs' intermediate values from one storage location to another. Since 'move' operations do not change the values of the operands, their complementary counterparts are themselves. Similar to NOT instruction, shift and data instructions are shared by both balanced and regular instructions, shown in Figure 4.

Currently, we have not found easy and efficient ways to design balanced arithmetic and control instructions. But fortunately, a number of cryptographic algorithms can avoid using these instructions. For example, the AES algorithm has a fixed control flow. So arguments of the control in-

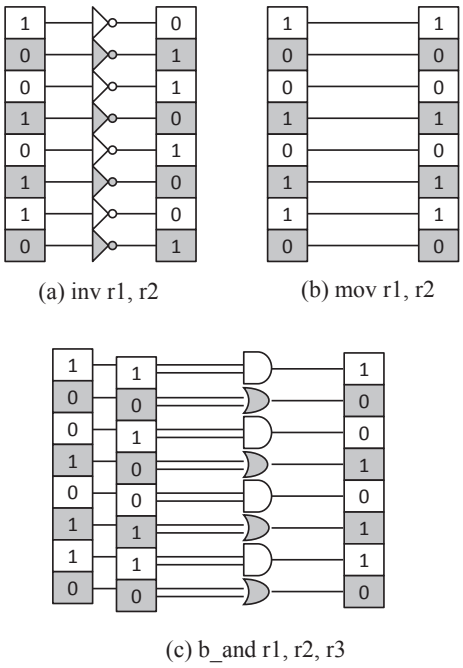


**Figure 4: (a) Regular INV can be used as a balanced INV; (b) Regular MOV can be used as a balanced MOV; (c) Balanced AND instruction uses half AND operators and half OR operators.**

structions are not functions of the key or the plain text input. It does not influence the security even if the attackers know these intermediate values. Another possible question is that, without balanced arithmetic instructions, whether it is practical to realize the software only with balanced logic instructions. Normally, using logic instructions to replace arithmetic instructions is very inefficient. However, for a number of cryptographic algorithms, only logic instructions are needed. Therefore, even though we do not have balanced arithmetic instructions, these cryptography can still be implemented without decreasing the efficiency too much.

*In summary, the proposed solution only needs to add two balanced instructions (balanced AND and balanced OR).* NOT, shift, and data instructions can be shared between regular and balanced instructions. This makes the modification to the processor very tiny.

### 3.2.2 Pre-charge Instructions

As mentioned in Section 2, besides complementary datapath, DRP technique also requires pre-charge operations. The proposed solution also faces the problem on how to implement pre-charge instructions for the same purpose. A pre-charge operation needs to pre-charge not only the result storage but also the computational datapath. For a balanced processor, a pre-charge instruction needs to pre-charge the processor's datapath and the destination registers or memory locations.

In the DRP circuits, pre-charge is usually done by pre-charging the inputs. The proposed solution uses a similar way to realize the pre-charge operations. It turns out that by setting the input operands of different balanced instructions to 0, the corresponding outputs are either all 0 or all 1 (for

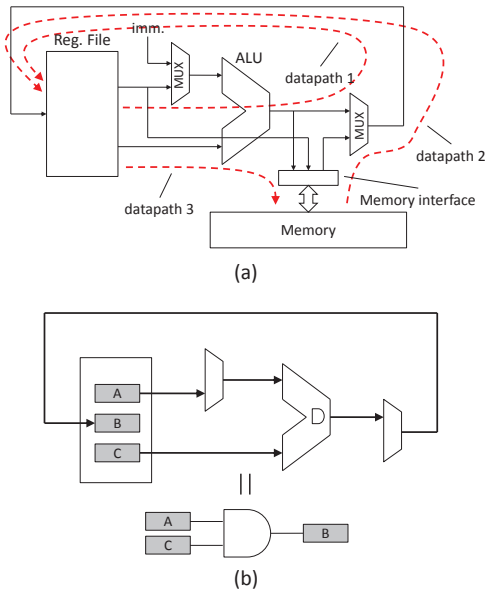


**Figure 5: Processor architecture: (a) Three datapaths in a processor [15]; (b) Datapath for AND operation.**

NOT). Since all 0 and all 1 are both acceptable pre-charge results, the pre-charge operation of a balanced instruction can be done by just executing the same balanced instruction with all 0 inputs.

*In summary, there is no need to add additional dedicated pre-charge instructions.* Every balanced instruction can finish the pre-charge operation for itself by taking all 0 as the inputs.

## 3.3 Security Analysis

While a VSC is software, its ultimate objective is to reduce the side-channel leakage originating from *hardware*, and more specifically from the processor that executes the VSC. This section explains why VSC has the same security feature as the DRP circuits.

we first analyze what parts of the processor are potential sources of side-channel leakage. For this purpose, we analyze the flow of information within a processor, as shown in Figure 5a. We can distinguish three different datapaths [15]. The first datapath is the computational datapath. It starts from the register file, goes through the Arithmetic Logic Unit (ALU), and returns to the register file. For memory-operations, there are two additional datapaths. The memory-load datapath is used to transfer information from memory to the register file. The memory-store datapath is used to transfer information from the register file to the memory.

Each of the above datapaths is a potential source of side-channel leakage. For example, Figure 5b shows the execution of an AND operation, which configures the computational datapath as an array of AND gates. Clearly, this operation leads to data-dependent power dissipation which must be avoided. *Therefore, if we can protect each of these three datapaths, the microprocessor instructions that only make use of these datapaths will also be secure.*

Finally, we demonstrate that a VSC is functionally equal

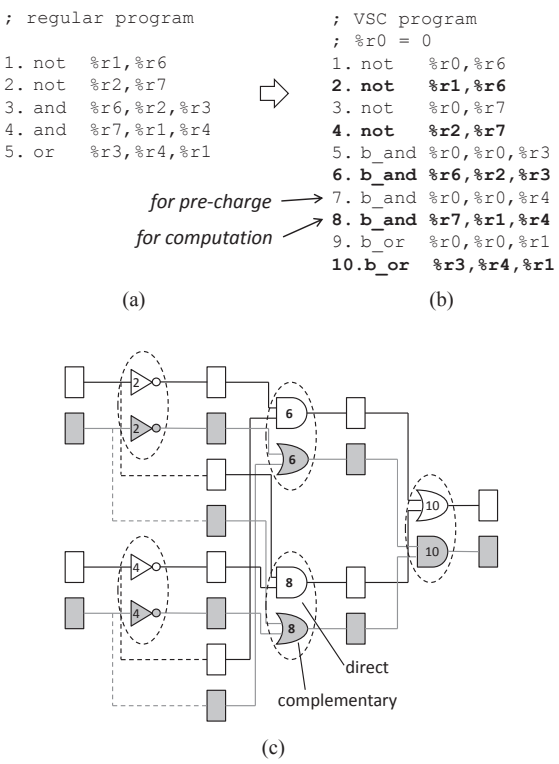


**Figure 6: (a) Regular program of Leon3 [1] assembly code; (b) VSC program of Leon3 assembly code; (c) Equivalent DRP circuit.**

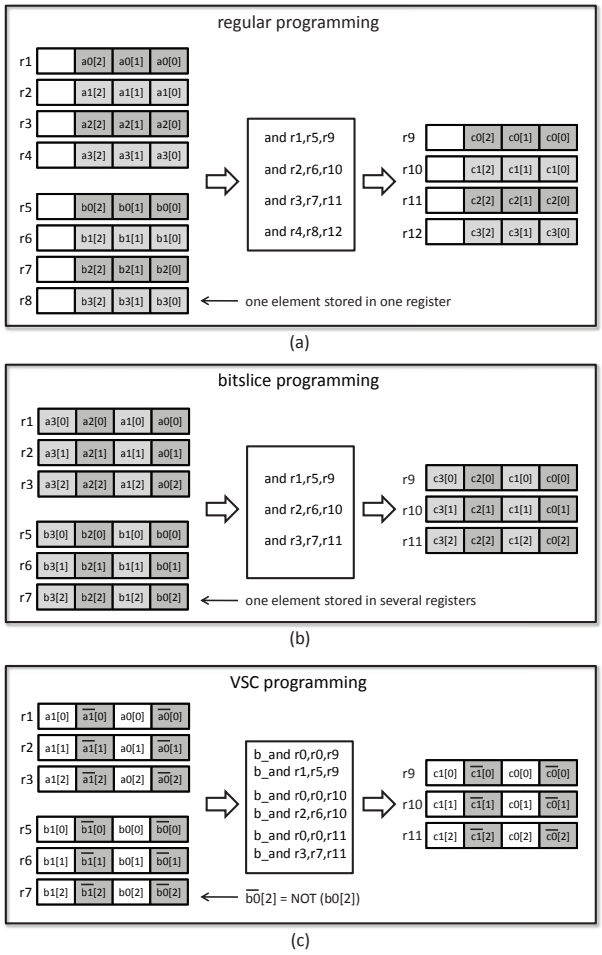


**Figure 7: Processing arrays A:** $(a0, a1, a3, a4)$ **and B:** $(b0, b1, b2, b3)$ **with AND operations to generate C:** $(c0, c1, c2, c3)$**. Each array element is 3-bit long while the processor's datapath is 4-bit long. (a) regular programming; (b) bitslice programming; (c) VSC programming.**

to a DRP circuit. Figure 6 shows an example of a regular program (Figure 6a), a VSC program (Figure 6b) and a DRP circuit (Figure 6c). All of them accomplish the same function. In the VSC program, bold instructions act as balanced instructions. Other instructions before each of them are used for the pre-charge purpose. 'b_and' and 'b_or' are balanced AND and OR instructions.

Two successive pre-charge and balanced instructions correspond to the pre-charge and evaluation activities of a logic gate (vector) in the circuit in Figure 6c. The numbers annotated within the logic gates correspond to line numbers of the balanced instructions in the VSC program. In between each logic gate, a register is inserted. They are equivalent to the registers in the processor's register file. The pre-charge instruction will reset that register before loading it with sensitive intermediate values. In addition, the direct and complementary operations are activated by the same balanced instruction at the same time. Therefore, when executing a pre-charge instruction followed by a balanced instruction, the circuits inside the processor behave exactly the same as a part of DRP circuit in Figure 6c. We conclude that a VSC is a sequentialized version of a DRP circuit. Moreover, the VSC may inherit the properties of a DRP circuit.

# 4. VSC PROGRAMMING

This section presents a method for VSC programming on the balanced processors so that SCA can be thwarted. Compared with the regular programming, VSC programming has two basic constraints. First, only the balanced instructions can be used. Second, the direct or the complementary datapath can only take half of the datapath of the processor. There could be different methods to deal with these two constraints. Here, we show a general solution based on bitslice programming.

## 4.1 Bitslice Programming

Bitslice programming is a special programming method. The idea is to break the software applications into only logic operations at the bit level, such as AND, OR, NOT, and XOR. The software's logic function is realized bit by bit. This is similar to the hardware implementation, with every operation equivalent to one logic gate. Porting this idea to a $n$-bit processor results in $n$ application instances running in parallel. Figure 7 shows a simple example on bitslice programming.

Suppose the application is AND operation and runs on a 4-bit processor. We use this application to process two

arrays A: $(a0, a1, a2, a3)$ and B: $(b0, b1, b2, b3)$ and generate the result C: $(c0, c1, c2, c3)$. $ci = ai$ AND $bi$. Every array element is 3-bit long, for example $a0$ can be represented as $(a0[2], a0[1], a0[0])$. Figure 7a shows the process based on regular programming. Every element is stored into a processor register. AND operations are used to process the elements, with each operation generating one element of C. In contrast, bitslice programming views the problem in a different way. Each array element is distributed over different registers. The operation on each array element is broken down to the bit level. Therefore, instead of 1 AND instruction, 3 AND instructions are needed to generate one element of C array. However, since the processor has a 4-bit wide datapath, another 3 exactly the same application instances can be done in parallel. Hence, the operations in Figure 7a can also be finished in Figure 7b.

Bitslice programming guarantees that the processor's datapath is under full usage, which may make it more efficient. However, it also has limitations. First, bitslice programming only uses logic instructions for computation. So for software applications that use a lot of arithmetic operations, bitslice programming loses efficiency. Second, running $n$ identical application instances in parallel generates $n$ times of intermediate results. These results need to be moved out to the memory in case the processor's registers are not enough. This will also reduce the efficiency.

Even though it has the above limitations, bitslice programming for symmetric-key cryptography is still popular. By now, we have seen bitslice programming under intensive research on two most important symmetric-key algorithms, including DES [4] and AES [9, 11, 13]. The reason is that symmetric-key cryptography seldom uses arithmetic operations, which makes the first constraint not a problem.

To deal with the second constraint, improved bitslice schemes have been proposed. The main idea is to reduce the number of encryption instances running in parallel, while maintaining a full usage of the processor's datapath. In this case, instead of taking only 1 bit of the register, one encryption instance takes multiple bits. Similar intermediate bits in one encryption instance are organized in one register. The operations on them are usually the same, even though not always identical as in the basic idea of bitslice programming. Once the operations for different bits in the same register are different, shift operations are employed to separate them for different operations and join them together after that. This sacrifices some computational efficiency but reduces data movements between the registers and the memory. The bitslice programming for AES that we will use for demonstration follows this method.

## 4.2 VSC Programming Based on Bitslice

Bit-slicing programming is a very suitable starting point for VSC programming. First, bitslice programming only uses bit-level logic instructions for computation. These instructions can easily find complementary counterparts. Second, bitslice programming can easily meet the requirement that direct operation only takes half of the processor's datapath. This can be done by simply changing the number of instances. Therefore, it is very easy to convert the bitslice programming to VSC programming.

Figure 7c shows how to convert the example depicted in Figure 7b to a VSC. Instead of running 4 direct application instances together, we maintain 2 instances as direct and change another 2 to the complementary instances. This easily fits into the balanced processor. In detail, doing VSC programming based on bitslice programming can simply go with the following two steps.

- Do bitslice programming for the cryptographic application. Implement even number of application instances in parallel. Make sure that each application instance either falls into the direct part or the complementary part of the balanced processor. This guarantees that, after converted a VSC, the application instance either remains as a direct instance or is completely converted to a complementary instance.
- Convert the regular instructions to balanced instructions. Add the correlated pre-charge operation before each balanced instruction.

To demonstrate the effectiveness of VSC, we chose the AES algorithm as the attack target. The bitslice AES was designed according to the method presented in [11]. Following the above two steps, we obtained a VSC AES on a 32-bit processor. Every AES instance takes 16 bits of the processor's datapath. So in VSC AES, one direct AES and one complementary AES run in parallel. One detail about VSC AES is with the XOR instruction. Since the complementary counterpart of XOR, XNOR, cannot be pre-charged by simply applying 0 on the inputs, we expand XOR to AND, OR, and NOT instructions. Actually, the logic operation shown in Figure 6 is a XOR operation that has already been expanded.

From the existing efficient designs of bitslice symmetric-key cryptography, we can see that the mainstream symmetric-key cryptography algorithms can all be easily converted to VSC. So although the current version of VSC may not support the software with arithmetic operations well, it can still be applied to a great amount of security systems.

# 5. EXPERIMENTAL RESULTS

To demonstrate that VSC on the complementary processor improves the resistance against SCA, this section presents the experimental results based on real attacks.

## 5.1 Experimental Setup

We used Leon3 embedded processor [1] to build a balanced processor prototype. Both the regular bitsliced AES and the VSC bitsliced AES discussed in Section 4 were chosen as the attack targets. To convert Leon3 to a balanced processor, we replaced the operations of two existing instructions (ANDN and ORN) with complementary AND and OR operations. Since ANDN and ORN instructions are not used by the bitsliced AES, this does not influence the correctness of the software and helps to quickly build up the prototype.

The prototype was realized on a FPGA board with a Xilinx Spartan 3E-1600 FPGA. The balanced Leon3 processor was implemented with FPGA resources, clocked at 50 MHz, while the software applications under test were stored in the on-board DDR2 SRAM memory. An AES encryption key was stored in the DDR2 SRAM and could not be read out of the board.

The experimental setup contained a FPGA board, an oscilloscope (Agilent DSO5032A) and a PC, shown in Figure 8. A RS232 cable connected the FPGA board and the PC to

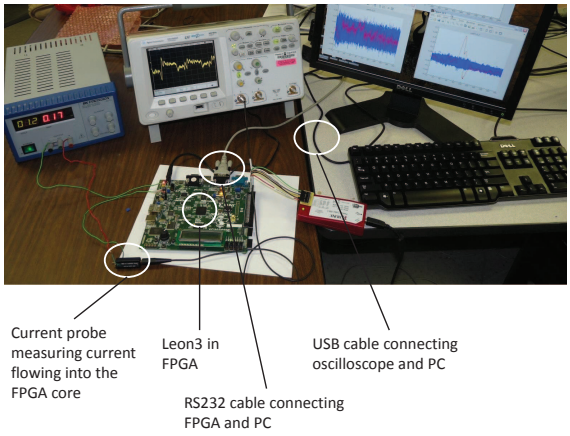


**Figure 8: Experimental setup.**

**Algorithm 1** Program for Leon3 on FPGA

```
exp_key ⇐ KeyExpansion(key)
loop
  if UART is not empty then
    wait until 16 bytes received
    in[15 : 0] ⇐ UART_FIFO
  end if
  out[15 : 0] ⇐ cipher(in, exp_key)
end loop
```

enable the communication between these two. The oscilloscope used a current probe (Tektronix CT-2) to monitor the current flowing into the FPGA core. Here, current is used to represent the power of the FPGA. Between the oscilloscope and the PC, a USB cable was used for communication. PC was able to send command to the oscilloscope, for example to require the oscilloscope to sample current traces with the current probe. After one sampling finished, PC was able to obtain the current sample trace also through the USB cable. Power measurement for SCA required the collaboration of these three parts. For every measurement, the following steps of operations are performed.

- The PC sends a random plaintext message (16 bytes) ($PT$) to the FPGA board through the RS232 cable. The $PT$ is also recorded in the PC.
- The balanced Leon3 processor in the FPGA receives $PT$ from the RS232 cable, calculates the corresponding complementary message $\overline{PT}$ and starts the cryptographic program with $PT$ and $\overline{PT}$. The program running on the Leon3 is shown in Algorithm 1. *cipher* is the cryptographic application under attack.
- After sending out $PT$, the PC sends command to the oscilloscope to sample the current trace when Leon3 in the FPGA is performing encryption.
- When sampling is done, the oscilloscope sends the current trace $TR$[999:0] (1000 sampling points) back to PC for further analysis.

## 5.2 Correlation Power Attack

With the above measurement results, a SCA was mounted called Correlation Power Attack (CPA) [5]. In our experiments, CPA focused on the outputs of the *SubBytes* step (16 bytes) in the first round of the AES algorithm. Every output was only related to one byte of the key. CPA is a searching process. To attack one key byte, it goes through every possible value of that key byte ($2^8 = 256$ possible values). For different key guesses, we have different outputs of the SubBytes operation. Accordingly, different power dissipations are expected, called *power hypothesis*. A decision function calculates the correlation coefficient of the power hypothesis and the actual current measurement traces, and identifies the correct key value that is used by the encryption: the correct key guess results in larger correlation coefficient than any other incorrect key guess.

Every sample trace has 1000 sample points. These points cover a certain range of time (500 us in our experiments) which includes the calculation of the first round of the AES. Although the attackers do not know which points correspond to the instruction execution related to the *SubBytes*'s outputs, they can still try every point. This increases the analysis complexity but the analysis time is still acceptable.

## 5.3 Results

This section assumes that the reader is familiar with the SCA result analysis [12]. The key point is Table 1, which shows the protected AES is much harder to attack.

Experiments first compared the unprotected AES (`AES`) and the protected AES (`VSC-AES`) implementations. Figure 9 shows an example of the attack results on one of `AES`'s key bytes with 5120 measurements. The x-axis represents the time that each measurement covers. The first round execution of `AES` occurs within this time range. The y-axis represents correlation coefficient between the power hypothesis and the actual current measurement. At each sampling point (1000 in total), we mount the CPA analysis, which results in 256 correlation coefficients for 256 possible key values. After finishing SCA analysis at 1000 different sampling points, Figure 9 is obtained. The black trace corre-

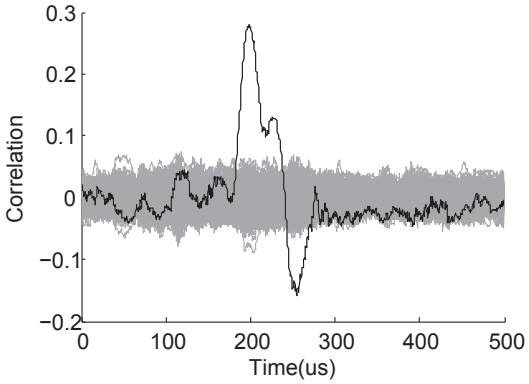


**Figure 9: Attack result on unprotected AES: Correlation between the sampled current and the power expectations with 5120 measurements. *Correct key's trace is plotted in black, while all other key's traces are in gray. The emerged black trace means successful attack.***

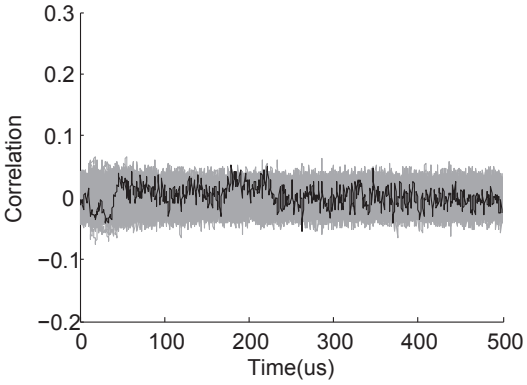


**Figure 10: Attack result on protected VSC-AES: Correlation between the sampled current and the power estimations with 5120 measurements.** ***Correct key's trace is plotted in black, while all other key's traces are in gray. The buried black trace means unsuccessful attack.***

sponds to the correlation coefficients at 1000 different time points when the correct key byte value is used to calculate the power hypothesis. The gray traces corresponds to other incorrect key byte values. It turns out that the black trace differentiates itself from all the other gray traces (around 200$\mu$s). This means that the correct key byte value can be identified and the attack is successful.

The experiment also mounted the same CPA attack on `VSC-AES` implementation. The example of the attack on the same key byte as in Figure 9 is shown in Figure 10. The black trace is totally buried into the gray traces. So the correct key byte value is not identifiable. Therefore the attack is not successful.

Figure 9 and 10 give two visualized attack examples when the sampling trace number is 5120. To quantify the security improvement, researchers usually use *Measurement to Disclosure* (MTD). MTD means the number of measurements required for a successful attack. Larger MTD indicates better security. Here, we define the successful attack to be uncovering all 16 bytes of the AES key. Table 1 presents more detailed attack results. For `AES`, 1280 measurements are sufficient to uncover all 16 key bytes. For `VSC-AES`, the number of measurements for full attack increases to 25600. *The improvement is as high as 20.* These results were obtained from the power expectation which is the Hamming weight of the entire byte of the *SubBytes*'s output. We name this as *byte-based* Hamming weight model. We also tested the power expectation which is the Hamming weight of each bit of the *SubBytes*'s output, named *bit-based* Hamming weight model. In our experiments, byte-based model is more efficient.

To further verify the correctness of VSC, we mounted attacks on variants of `VSC-AES`: 1) only the direct path of `VSC-AES` without complementary path (`VSC-AES` nocomp), and 2) `VSC-AES` without pre-charge operations (`VSC-AES` noprch). All these designs could be broken with a similar effort to break `AES`, as shown in Table 1.

Table 1 also shows that `VSC-AES` pays 6.5 times decrease of throughput and 3.3 times increase of footprint for higher capability of resisting CPA. We estimate that the energy cost increases 6.5 times since the power remains more or less the same. Since our AES implementations unrolled the major loop of AES: 10 rounds of operations. So the footprints for both `AES` and `VSC-AES` are relatively large. However, it is reasonable to expect both footprints to decrease by 10 times if the unrolling is not done.

## 5.4 Analysis Based on Results

The experimental results shown above demonstrate that VSC on the complementary processors presents an effective way to improve cryptographic software's resistance against SCA. The side-channel resistance improvement of `VSC-AES` is in the same order as the improvement offered by the WDDL prototype IC chip [18], a real DRP circuit countermeasure. Compared with the unprotected design, `VSC-AES` provides 20 times improvement while the WDDL prototype IC offers 100 times improvement.

Compared with the WDDL prototype IC, VSC did not spend much effort to the structural symmetry between direct and complementary datapaths, although they are located in adjacent bits (Figure 4). Due to this reason, it is reasonable to see a smaller improvement than the WDDL prototype IC. Since intermediate values in software usually have longer life time than their counterparts in the hardware, software is usually more vulnerable than hardware. As a result, `VSC-AES` can be attacked with fewer measurements than the WDDL prototype IC. However, VSC possesses a potential for improvements. This is because VSC has a high flexibility. Other countermeasures, for example masking, can be easily integrated to VSC. In this case, the baseline design is not a unprotected software but a protected software with masking solutions. This combination of hiding and masking countermeasures has already been demonstrated to be effective by the MDPL circuits [17]. Due to the similarity between VSC and DRP circuits, it is reasonable to expect the same improvement for VSC based masking software countermeasures.

Another advantage of VSC is the low hardware cost. The only modification made to the hardware is two simple balanced instructions. In contrast, DRP circuits is usually from 3.4 times [20] to 12 times [22] larger than the regular circuits. This is a critical issue that prevents their broad acceptance by the industry. What VSC pays is not the hardware cost, but the performance and the memory storage. Considering the memory storage is becoming cheaper and cheaper and is reusable, if the reduced performance is still acceptable to the applications, VSC sees a promising future.

# 6. CONCLUSION

This paper presents a new software countermeasure against Side Channel Attacks. The new countermeasure emulates the secure hardware circuit. Due to this, the processor system has similar power dissipation as the secure circuits and therefore can resist Side Channel Attacks as the secure circuits do. The corresponding software is called Virtual Secure Circuits. To support this emulation, this paper also presents the concept and the implementation of the balanced processors, the balanced instructions, and the method to program Virtual Secure Circuits. Real attacks showed that Virtual Secure Circuits improve the security by 20 times. Besides the security improvement already obtained, Virtual Secure Circuits still have a large improvement space to cooperate with other countermeasures for

Table 1: Attack results summary.

| Parameter | AES | VSC-AES | | VSC-AES | VSC-AES |
|---|---|---|---|---|---|
| | | byte-based | bit-based | nocomp | noprch |
| throughput (kb/s) | 207*2=414 | 64 | | – | – |
| footprint (kB) | 45.7 | 150.2 | | – | – |
| key bytes **NOT** found | | | | | |
| @ 512[1] | 3 | 16 | 16 | 4 | 7 |
| @ 1280[1] | 0 | 15 | 15 | 0 | 0 |
| @ 5120[1] | 0 | 9 | 10 | 0 | 0 |
| @ 12800[1] | 0 | 4 | 5 | 0 | 0 |
| @ 25600[1] | 0 | 0 | 2 | 0 | 0 |

[1] Number of measurements.

even better security. Moreover, Virtual Secure Circuits also see a promising future because of its high flexibility and low requirement for additional hardware cost.

## 7. ACKNOWLEDGMENTS

This research has been supported in part by National Science Foundation (NSF) Grant No. 0644070.

## 8. REFERENCES

[1] Aeroflex Gaisler. LEON3 Multiprocessing CPU Core. *available at* `http://www.gaisler.com/doc/leon3_product_sheet.pdf`.

[2] M.-L. Akkar and C. Giraud. An Implementation of DES and AES, Secure against Some Attacks. *CHES 2001*, LNCS 2162:pp. 309–318, 2001.

[3] J. A. Ambrose, R. G. Ragel, and S. Parameswaran. "rijid: Random code injection to mask power analysis based side channel attacks". *DAC 2007*, pages pp. 489–492, 2007.

[4] E. Biham. A Fast New DES Implementation in Software. *FSE 2005*, LNCS 1267:pp. 260–272, 1997.

[5] E. Brier, C. Clavier, and F. Olivier. Correlation Power Analysis with a Leakage Model. *CHES 2004*, LNCS 3156:pp. 16–29, 2004.

[6] S. Chari, C. Jutla, J. Rao, and P. Rohatgi. Towards Sound Approaches to Counteract Power-Analysis Attacks. *CRYPTO 1999*, LNCS 1666:pp. 398–412, 1999.

[7] C. Clavier, J.-S. Coron, and N. Dabbous. Differential Power Analysis in the Presence of Hardware Countermeasures. *CHES 2000*, LNCS 1965:pp. 252–263, 2000.

[8] C. H. Gebotys. A Split-Mask Countermeasure for Low-Energy Secure Embedded Systems. *ACM Transactions on Embedded Computing Systems*, Vol. 5(No. 3):pp. 577–612, 2006.

[9] E. Käsper and P. Schwabe. Faster and Timing-Attack Resistant AES-GCM. *CHES 2009*, LNCS 5747:1–17, 2009.

[10] P. C. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. *CRYPTO 1999*, LNCS 1666:pp. 388–397, 1999.

[11] R. Könighofer. A Fast and Cache-timing Resistant Implementation of The AES. *CT-RSA 2008*, LNCS 4964:pp. 187–202, 2008.

[12] S. Mangard, E. Oswald, and T. Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer, 2007.

[13] M. Matsui and J. Nakajima. On the Power of Bitslice Implementation on Intel Core2 Processor. *CHES 2007*, LNCS 4727:pp. 121–134, 2007.

[14] NIST. Annoucing the ADVANCED ENCRYPTION STANDARD (AES). *available at* `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`, 2001.

[15] J. Nurmi. *Processor Design*. 10.1007/978-1-4020-5530-0. Springer Netherlands, 2007.

[16] E. Oswald and K. Schramm. An Efficient Masking Scheme for AES software Implementations. *WISA 2005*, LNCS 3786:pp. 292–305, 2005.

[17] T. Popp and S. Mangard. Masked Dual-Rail Pre-charge Logic: DPA-Resistance Without Routing Constraints. *CHES 2005*, LNCS 3659:pp. 172–186, 2005.

[18] K. Tiri, D. Hwang, A. Hodjat, B. Lai, S. Yang, P. Schaumont, and I. Verbauwhede. Prototype IC with WDDL and Differential Routing - DPA Resistant Assessment. *CHES 2005*, LNCS 3659:pp. 354–365, 2005.

[19] K. Tiri and I. Verbauwhede. Securing Encryption Algorithms against DPA at the Logic Level: Next Generation Smart Card. *CHES 2003*, LNCS 2779:pp. 125–136, 2003.

[20] K. Tiri and I. Verbauwhede. A Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation. *Proceeding of DATE 2004*, vol. 1:pp. 246–251, 2004.

[21] S. Yang, W. Wolf, N. Vijaykrishhnan, D. N. Serpanos, and Y. Xie. Power Attack Resistant Cryptosystem Design: A Dynamic Voltage and Frequency Switching Approach. *DATE 2005*, pages pp. 1530–1591, 2005.

[22] P. Yu and P. Schaumont. Secure FPGA Circuits Using Controlled Placement and Routing. *CODES+ISSS 2007*, pages pp. 45–50, 2007.