Current Flattening in Software and Hardware for Security Applications

Radu Muresan
University of Guelph
Guelph, Ontario, Canada
N1G 2W1
1(519)824-4120x56730
rmuresan@uoguelph.ca

Catherine Gebotys
University of Waterloo
Waterloo, Ontario, Canada
N2L 3G1
1(519)888-4567x3539
c.gebotys@ece.uwaterloo.ca

ABSTRACT
This paper presents a new current flattening technique applicable in software and hardware. This technique is important in embedded cryptosystems since power analysis attacks (that make use of the current variation dependency on data and program) compromise the security of the system. The technique flattens the current internally by exploiting current consumption differences at the instruction level. Code transformations supporting current variation reductions due to program dependencies are presented. Also, a new real-time hardware architecture capable of reducing the current to data and program dependencies is proposed. Measured and simulated current waveforms of cryptographic software are presented in support of these techniques.

Categories and Subject Descriptors
E.3 [**Data**]: Data Encryption --- *Public key cryptosystems*; C.3 [**Computer System Organization**]: Special-Purpose and Application-Based Systems --- *Signal processing systems*; B.m [**Hardware**]: Miscellaneous.

General Terms: Security, Design, Measurement.

Keywords
Current flattening, power analysis attacks, hardware architecture.

1. INTRODUCTION
Due to recent increase in security usage, improved software and hardware countermeasures against power analysis attacks are needed. Current and power consumption dependency on program and data composition of a cryptographic hardware is exploited in various power analysis attacks and thus the security of an entire cryptosystem (hardware plus software) is compromised [1], [2], [3], [4], [5]. By conducting simple or statistical power analysis on power waveforms collected at a measurable point on the cryptosystem (also, called attacker's observation point) the secret key used for the secret communications can be revealed.

CODES+ISSS'04, September 8–10, 2004, Stockholm, Sweden.

The general idea of a power analysis attack is that the attacker does not have access to the secret information (secret key) of the user, but can access any non-secret information (algorithm implementation, public keys, portion of cipher text, hardware particularities of the system and so on) as well as a power consumption measurable point on the hardware [2], [3]. Various countermeasures against power analysis attacks are presented in literature [2], [3], [4], [5], [6], [8], [9].

So far, there are three main cryptosystems attacks based on power consumption analysis. These are: timing attacks, simple power analysis (SPA) attacks and differential power analysis (DPA) attacks. Timing attacks base their success on execution timing to data dependency [2]. Three basic countermeasures to the timing attacks are presented in [2]. These are based on equalizing the execution times for all operations of the protocol; randomizing the timing measures by introducing random delays within the processing time of a cipher text; and on using the blinding signature method. The SPA use power consumption dependency of program blocks on processed data [3]. Examples of program blocks considered are: multiplication, squaring, permutations, shifts [3], [4]. It was indicated in [4] that the SPA attack can be prevented by avoiding procedures that use secret sensitive information for conditional branching operations, by creating constant execution path algorithms, and by implementations of symmetric cryptographic algorithms. However, in microprocessors with large operand-dependent power consumption features, the SPA can pose a serious threat even for the constant execution path code [4]. The DPA uses statistical analysis of data to power trace correlations that exploit data path algorithm dependency on processed data [3]. In [3], Kocher et al. developed a DPA attack that targeted a DES implementation.

The power analysis attacks have been proven to work against any encryption technique. The elliptic scalar multiplication kP is a fundamental operation in the elliptic curve cryptosystem protocols and is the main target of the power analysis attacks [5], [8]. The scalar multiplication algorithm is central to the Diffie-Hellman and ElGamal key agreement elliptic curve (EC) protocols. In both of these protocol schemes the shared secret is obtained by performing scalar multiplication operations. In [8], Coron generalized the DPA attack to EC cryptosystems and described a DPA attack on EC Diffie-Hellman key exchange and EC El-Gamal type encryption. Considering the possibility of power analysis attacks against EC card implementations Coron suggested three countermeasures: randomization of the private exponent, blinding the point P, and randomized projective coordinates.

Current or power consumption flattening is a counterattack measure against power analysis attacks. The goal of a flattening technique is to reduce the variations of the current or power waveforms measured at an attacker point. This can be done by the use of external RLC filters or by power consumption compensators based on active components (FETs) [11]. However, these types of filters are not recommended as an efficient countermeasure against power analysis attacks [11].

In this paper, a new current flattening technique applicable in software and hardware that attacks internally the emission states of a cryptosystem, is proposed. Code transformations, applicable to a processor core, capable of reducing the current to program dependency are introduced. Real current measurements for a polynomial multiplication implementation [7] show reduced current to program dependency when the proposed code transformations are used. Also, a real time power analysis attack resistant (PAAR) architecture that implements the current flattening technique in hardware is proposed. Unlike the software code transformations, the hardware architecture is capable of reducing current variations due to both program and data dependencies and thus providing a potential immunity to all power analysis attacks. Simulated results of an elliptic curve polynomial multiplication application running on an embedded processor core (SC140 [12]) are presented in support of the hardware architecture functionality.

The rest of the paper is organized as follows. Section 2 outlines previous research related to the current flattening technique presented in this paper. Section 3 presents the principles of the new current flattening technique. Section 4 introduces the new real time hardware architecture for current flattening that is a power analysis resistant architecture. Section 5 presents results of the current flattening technique applied in software and hardware. Section 6 presents the conclusions of our research.

2. PREVIOUS RESEARCH

In [14], a Non-Deterministic Instruction Stream Computer (NDISC) is proposed as a countermeasure to DPA. The NDISC applies a process called instruction descheduling in order to reduce the effectiveness of the DPA results. By instruction descheduling, at run time the processor will select, at random, an instruction to execute. In this way the instruction stream is randomized and the current traces will eventually be uncorrelated from one run to another. As a result, the architecture is capable of producing temporal misalignment of power traces which can help to defeat DPA.

In [10], a technique which allows the non-deterministic altering of the register to register or memory to memory transfers is proposed. This technique reengineers the circuitry that performs arithmetic operations or memory transfers and thus produces randomization and temporal misalignment of traces [10]. The approach employed in [10] is based on random register renaming that exploits the fact that the power consumption is related to the number of bits that are flipped when registers are overwritten.

In [11] the authors defined leakage immunity in cryptography and presented several leakage tests that confirm the probable existence of secret-correlated emanations. The results of the Difference of Mean Test (DoM-M) [11] performed on a smart-card chip, are used to evaluate the effectiveness of the RLC filters as countermeasures against power analysis attacks [11]. Due to the energy conservation principle and averaging characteristics of the RLC filters the use of these external filters on a smart card proved to worsen the fail rate of the DoM-M-leakage test. The observation was that the emission states used by a power analysis attack are not diminished. Instead, they are separated in two different stable levels that are easily detected by the test [11]. The effect of the RLC filters usage is that a flattened power trace is obtained. Based on the results of the leakage tests, the use of the RLC filters is not recommended by [11] as an effective countermeasure against power analysis attacks.

In [7] an internal current flattening technique (ICFT) in software and hardware and an energy analysis for secure implementations in embedded cryptosystems is proposed. The ICFT is capable of reducing the peak-to-peak current variation due to program dependencies and thus increase the security against simple power analysis attacks [7]. However, the current to data dependencies, and as a result the DPA attacks, are not efficiently supported by the software technique. Unlike the RLC filtering countermeasures the internal current flattening technique does not separate the emission states used by the power analysis attacks into two different stable levels. Instead, the emission states are driven to a common level.

3. THE PROPOSED INTERNAL CURRENT FLATTENING TECHNIQUE

In this section, the attacker's observation point [11] is considered to be the core's power supply pin. The electrical impact of a program at the observation point is modeled by an RC type circuit. Each program is assumed to be capable of delivering a charge to the capacitor at a rate determined by the instruction types, the composition of the execution sets, and the parallelism variation within the program [7]. The current consumption of a program is flattened by inserting non-functional instructions (NFI) within the program. A NFI is an instruction that neither alters the functionality of the program, nor consumes additional registers [7]. Examples of NFI for the SC140 processor are NOP; OR dn,dn; and ADD #0,dn. The purpose of introducing such instructions within the code is to extend the execution time of blocks that produce high currents, and increase the amount of discharge time used by these blocks. The NFIs could be also grouped into an execution set in order to increase the current consumption of a portion of the program when the corresponding current values are too low. Based on average or maximum current measurements at the instruction level, it is possible to divide the instruction set architecture of a target processor into classes. Within a class, the current consumption difference between the instructions should have a small measurable variation. The instruction set architecture of the SC140 target processor core was divided into seven current consumption classes. These classes are: ALU, composed of all ALU instructions; NOP, composed of the non-operational instructions; CONTROL, composed of all control instructions that perform a jump; LOOP, composed of the instructions used by the hardware loops; MOVE, composed of the move instructions from/to memory; VSL, composed of all the VSL instructions; DELAY, composed of the delay slot type instructions [7].

Keeping in mind that there is a noticeable difference (visually as well as measurably) between the current classes, the goal of a

cryptographic software developer should be to flatten the current variation at the observation point as much as possible in order to diminish the current to program dependencies. This current flattening could be done automatically to a certain extent by a compiler that has its current classes (for the target processor) well defined. The major problem faced by such compiler would be determining the places and the number of NFIs needed in a program in order to achieve a desired flattening effect. The following subsection presents an empirical methodology that can solve some of these problems.

3.1 Empirical Method for Generating Code Transformation Applicable to ICFT

The method developed in this subsection is based on the current equation of the discharging cycle in a source-free RC circuit $i(t) = I_{d0} \cdot e^{-t/\tau_d}$ [7]; where I_{d0} is the initial value of the current before the discharging cycle starts and $\tau_d = RC$ is the discharge time constant of the RC circuit. The program execution is considered to be composed of two types of cycles [7]: the charging cycles (that are due to the execution of instructions that are part of the unaltered functional code); and the discharging cycles (that are due to execution of NFI blocks that are introduced to achieve current flattening). Due to processor's complex current dependency on program and data, direct I_0, R, and C values for the current model of a program are practically impossible to determine. As a result, a discharging estimation time approximation is used. Given an initial code for a program and an initial code transformation called FLATTEN0 two current measurements are performed.

Measurement 1: This measurement determines the peak current value $I_{\max 1}$ and the program execution time T_1 of the original unaltered program.

Measurement 2: This measurement determines the peak current value $I_{\max 2}$ and the execution time T_2 for an altered version of the program. The altered version of the program preserved the original functionality but used NFIs throughout the program such that the average number of ALU instructions per execution sets was made uniform to a fixed value u_V. This transformation process is referred to here as an uniformization process.

These two measurements are made to provide initial values (the discharging time τ_d of the processor executing NFI blocks) for calculating the best code transformation for ICFT. Experiments were performed on small programs (less than 1000 clock cycles) executing on the SC140 processor core. The code transformations used in these experiments (See Table 1) were based on simple current class consumption approximations above a fixed current level called NOP level [7], [13]. The NOP level was the current consumption of the processor executing an infinite string of NOP instructions. All operational instructions have greater current consumption than the NOP instruction and are responsible for the current variation observed at the power supply pin. The current variation of the target processor builds up above the NOP current level as the processor executes operational instructions [7], [13]. The ALU class current consumption was considered to be the reference measurement unit for the flattening transformations.

Figure 1 presents two current waveforms collected at the processor pin while the processor executed an assembly language application. The application was composed of two code blocks. The first block was a highly parallel block with an average instruction count per execution set of five [7]. The second block had an average instruction count (per execution set) of two. The FLATTEN0 code transformation presented in Table 1 was used to perform the process of ALU uniformization for the unaltered version of the program used in Measurement 1. After the program uniformization process, for any two consecutive execution sets, the average number of ALUs was equal to a fixed uniformization value u_V. Considering any execution set ES that had an ALU equivalence of k ALU units, the number n of NFIs that were introduced after this execution set was calculated based on the following relation: $n = u_V(k+1)$. In Figure 1, the blue (lighter) waveform represents the unaltered version (Case 1 Version) and the red (darker) waveform (Case 2 Version) represents the altered version. Using the current discharge equation and the experimental values $I_{\max 1}$, T_1, $I_{\max 2}$, and T_2, the value of the discharging time τ_d was approximated by the following relation [7]: $\tau_d = (T_2 - T_1)/\ln(I_{\max 1}/I_{\max 2})$.

Table 1. Example of current equivalence chart for SC140

Class	Rule #	FLATTEN0 [ALU Units]	FLATTENi [ALU Units]
ALU	1	1	1
NOP	2	0	0
CONTROL	3	4	3
LOOP	4	No Action	Insert 1 Nop after loopstart
MOVE	5	4	3
VSL	6	5	4
DELAY	7	No Action	Non-Delay Slot Instructions

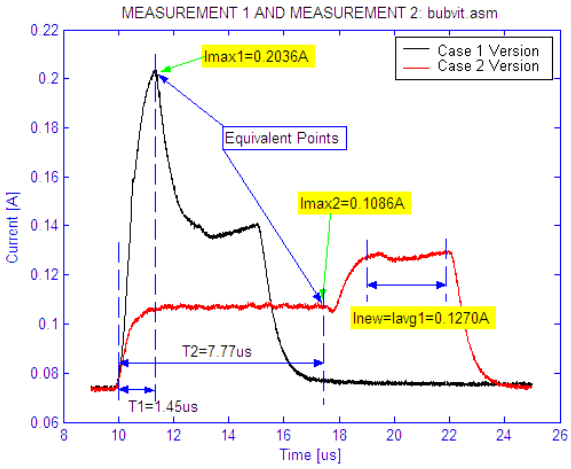

Figure 1. Example of Measurement 1 and Measurement 2 Currents for the SC140 processor.

If the current value I_{max2} obtained by applying the FLATTEN0 transformations is unsatisfactory, further current adjustments are made by using the calculated discharging time value. Given a new target current value $I_{new} < I_{max1}$, the extra amount of time needed to extend the program execution (in order to introduce a free discharge cycle of the processor's capacitor to this new current value) is calculated by the following equation: $t_{extra} = T_{new} - T_1 = \tau_d (\ln(I_{\max 1} / I_{new}))$ [7].

Based on the results of this and other similar experiments, an improved code transformation called FLATTENi was derived for the SC140 processor (See Table 1). The FLATTENi current equivalence rules together with the uniformization process can be applied by a compiler in order to eliminate current to program dependency and achieve an internal current flattening through software. In this paper the efficiency of the FLATTENi code transformations was estimated by direct measurements presented in Section 5. The deficiency of the software flattening technique as applied to counterattack power analysis attacks is that it does not support current to data dependencies. However, the principles of this technique show that by manipulating the instructions that a processor core executes, internal current regulation and filtering is possible (See Section 5). Transformations that take place at compilation time cannot address current changes that are only known at execution time. As a result, a real-time hardware architecture that implements the current flattening technique into hardware is presented in the next section.

4. REAL TIME HARDWARE ARCHITECTURE FOR CURRENT FLATTENING

The real time power analysis resistant architecture, called PAAR architecutre, is composed of 2 modules: feedback current flattening module called FCFM and the pipeline current flattening module called PCFM. The FCFM is responsible for measuring the instantaneous current consumption at the processor supply pin and generating two feedback signals to the PCFM. The PCFM is responsible for inserting non-functional instructions into the pipeline in order to bring the current consumption of the processor to a value that is within two programmable current variation limits (L1 to L2).

Figure 2 presents the block diagram of the FCFM and the necessary signal interfacing to a general processor core architecture. The following describes the main characteristics of this block diagram. The FCFM has software programmable capabilities and can be activated only when needed. After reset, the block's signals are inactive until the current level registers are programmed. The FCFM has 6 control lines and one 8-bits output data bus. A current resistor Rp is connected between the power supply line Vdd and the power supply entry to the processor core. Two voltage signals, V1 and V2, are used to measure the voltage drop across the current resistor Rp. The voltage drop across the current resistor is proportional with the processor's current at all times. A differential amplifier DA1 is used to bring the small signal value of the voltage drop to a comparable signal level CLp that is proportional with the instantaneous current consumption of the processor. The architecture assumes that the two current level registers are loaded with the binary values L1 and L2. These values can be loaded every time a secure communication is executing. The binary values L1 and L2 are converted by digital to analog (D/A) converters into the current levels CL1 and CL2, respectively. The current levels CL1 and CL2 represent the minimum and the maximum current level allowed, respectively. The current levels CL1 and CL2 are compared with the instantaneous current level CLp by two comparator blocks C1, and C2, respectively. The comparator blocks C1 and C2 produce the feedback digital output signals of the FCFM to the PCFM. The feedback signals are 'Cut Current Signal' (CCS) and 'Increase Current Signal' (ICS). The CCS signals that the processor's instantaneous current is above the allowed maximum value and the ICS signals that the processor's instantaneous current is below the allowed minimum value.

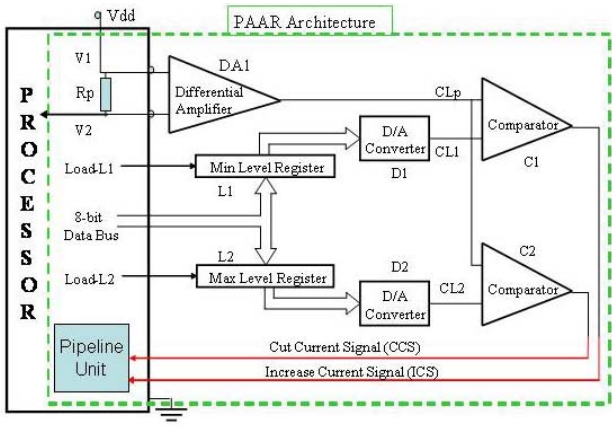

Figure 2. Feedback current module-FCM.

Figure 3 presents the block diagram of the PCFM. The following explores the main functionality details of this module for the case of a 2-instructions issue pipeline block. The pipeline module is composed of the following pipeline stages: instruction fetch and prefetch (F); dispatch (D); execution (E); and write back (W).

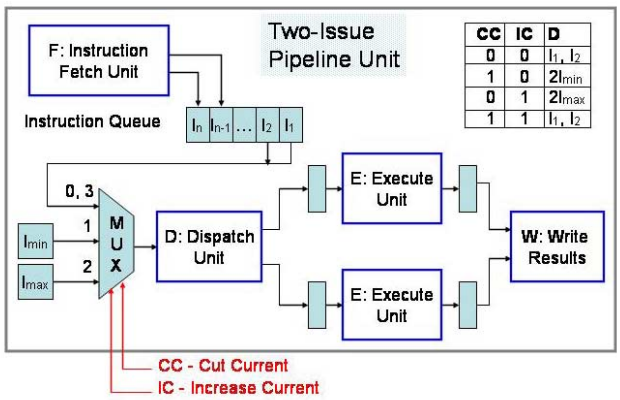

Figure 3. Pipeline current flattening module – PCFM.

The PCFM assumes that the instruction set architecture of the processor has 2 special non-functional instructions called I_{min} and I_{max}. The I_{min} instruction is equivalent to a NOP instruction. Its characteristic is that the instruction does not perform any operations and as a result, its current consumption is minimal. On the other hand the I_{max} instruction activates a special high current

consumption block such that the current consumption of this instruction is adjustable. Based on the CCS and ICS values the MUX allows one or two instructions from the instruction queue to be passed to the dispatch unit, or one or two I_{min} or I_{max} instructions (See the MUX control table of Figure 3). When the current consumption at the processor pin is measured by the FCFM to be above/under the maximum/minimum programmed current limit, the PCFM introduces I_{min}/I_{max} instructions in the pipeline's dispatch unit until the current is regulated to its prescribed limits. The final goal of the PAAR architecture is to regulate the current consumption to a fixed current value that falls under the minimum current value of an application that is sensitive to power analysis attacks. In this way there will be no current variation that can be possibly linked to the secret key of the implementation. The number of I_{min} or I_{max} instructions introduced in the program execution are expected to vary from one cycle to another depending on data manipulated by the application. As a result, real time instruction issue randomization will also be obtained.

5. RESULTS

This section presents results of the internal current flattening technique. First, real current measurement results obtained by applying the code transformation rules of Table 1 are presented. Secondly, simulated results for a possible add-in PAAR module attached to the SC140 processor core are presented. The experiments used the polymulNIST.asm assembly language code of an elliptic curve scalar multiplication implementation kP, with P a fixed point on a known elliptic curve and k a secret key [7]. The coordinates of the elliptic curve points were represented in polynomial basis and the basic polynomial field operations (addition, reduction, multiplication, inversion) were implemented using SC140 assembly language optimized routines [1]. The executable code used a NIST elliptic curve over the $GF(2^{163})$ field with a fixed NIST base point and the following prime polynomial [1]: $x^{163}+x^7+x^6+x^3+1$. Using an instantaneous current collection setup [7], [13], entire current traces for smaller and full scale bits (k=163) were collected. By applying the FLATTENi code transformations presented in Section 3 (Table 1) a new application was obtained (polymulNISTsmooth.asm).

Figure 4 presents the waveforms plot of the instantaneous current measurements performed at the power supply pin of the processor core while executing altered and unaltered versions of the polymulNIST.asm program [7]. In this plot, M1 represents the current waveform of the unaltered program and M7 represents the current waveform of the program that applied all of the FLATTENi transformation code rules. M2 and M4 are current waveforms that applied only Rules 1 to 6 of the FLATTENi transformation code rules. The most effective transformation was version M7 that used all of the transformation rules and NOP as the NFI instruction. All of the transformed versions used an uniformization factor u_v = 2. Table 2 presents a summary of the measurements performed on these waveforms. As seen from this table the FLATTENi code transformation (waveform M7) showed a Pk-Pk current reduction of approximately 78% with an increase of approximately 74% in energy consumption due to longer execution time [7]. Note that the energy increase and the performance depreciation apply only to the routines that are vulnerable to the power analysis attacks.

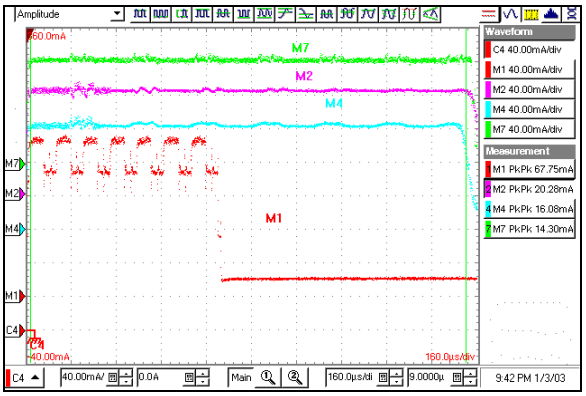

Figure 4. ICFT measurements based on Table 1 code transformations applied to polymulNIST.asm.

Table 2. Data analysis for ICFT applied to polymulNIST.asm

Version	Start Time [µs]	End Time [µs]	Energy Program [µJ]	Mean [mA]	S [mA]	Pk-Pk [mA]
M1	25	681	209.9	0.168	18.85	67.7
M2	25	1569	363.2	0.124	1.60	20.3
M4	25	1545	359.9	0.125	1.91	16.8
M7	25	1569	365.5	0.125	1.98	14.3

Figure 5 and Table 3 present simulated data analysis for the PAAR architecture. The functionality of the hardware implementation of the current flattening technique was simulated by the use of a MATLAB instantaneous current simulator [7], [13]. One of the security sensitive routines that is targeted by the power analysis attacks is the binary multiplication routine polymul.asm that is part of the polymulNIST.asm program [7]. The simulator used assumed that the PAAR module was added to the SC140 processor core. The simulator used instantaneous current measurements collected for the instruction architecture of the processor [7], [13]. Figure 5 presents four current waveforms for the polymul.asm subroutine. The current projection assumed that the PAAR block had a constant current consumption that was not significant to this simulation. Figure 5 plots with blue the measured current and with red the simulated current when the PAAR block was inactive. The figure also shows with magenta and black, the simulated currents when the PAAR block was active with the CL2 level at 70 mA, and at 60 mA, respectively. In these cases the CL1 level was zero. Table 3 presents the execution times, the mean, max, Pk-Pk, energy and standard deviation (s) values for the measured current and the simulated current. The standard deviation was calculated over the interval T=[2500 ns, 21300 ns] for all of the current traces presented in Figure 5. As shown by the Pk-Pk and s values the current variation decreases when the PAAR block is activated. However, longer execution times and energy consumption were recorded for the simulated currents when the PAAR block was activated. As a result, the hardware and the software flattening techniques showed an increase in the energy consumption and execution time per program. Due to possible low current variation regulation (See the black waveform of Figure 5) the PAAR architecture could assure

theoretically maximum security against power analysis attacks by trading energy consumption and execution time for security.

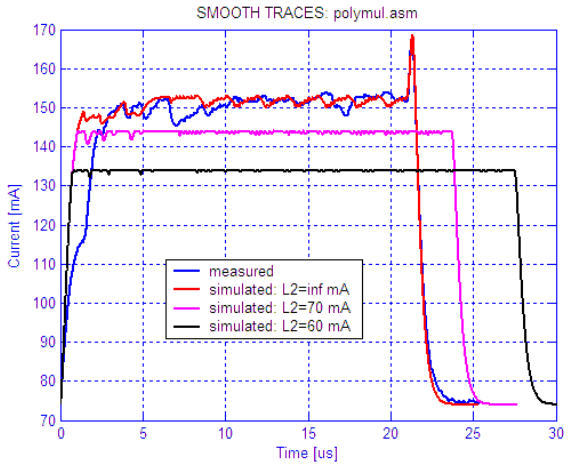

Figure 5. Current simulation for the SC140 executing the polymul.asm application when PAAR is activated.

Table 3. Measurements for the waveforms of in Figure 5

Wave-form	Run Time [µs]	Energy [µJ]	Max [mA]	Mean [mA]	S [mA]	Pk-Pk [mA]
blue	21.3	6.04	167.7	137.5	2.47	22.4
red	21.3	6.14	168.5	139.5	1.87	19.2
magenta	23.7	6.47	144.0	133.8	0.38	1.2
black	27.5	7.01	134.0	126.6	0.13	0.6

6. CONCLUSIONS

This paper presented the mechanisms of a novel internal current flattening technique as a countermeasure against power analysis attacks in embedded cryptosystems. The software code transformations that follow this technique are capable of reducing the current variations of an application that are caused by current to program dependencies. Current variation reductions of 78% were obtained for the SC140 processor core. A real time hardware architecture called PAAR (power analysis attack resistant) architecture that implements the software principles of the current flattening technique in hardware was developed. Theoretically the PAAR architecture could overcome all power analysis attacks without employing any of the software based countermeasure techniques. The simulated results showed that both the data and program current dependencies are controlled. However, the processor used in these experiments had no cache capabilities. Unlike the randomized methods of hardware architecture that have the final effect of changing the power consumption pattern [10], the PAAR architecture introduced in this paper targets a dynamically controlled current level with a very low detectable current variation. A random sequence of instructions is controlled by the data, program and the control logic of the PAAR architecture.

This research opens new directions into the development of secure embedded cryptosystems against power analysis attacks. An important future work of this research is to investigate a real hardware implementation of the PAAR architecture and find schemes that have improved power and performance factors.

7. ACKNOWLEDGMENTS

Our thanks to Motorola, NSERC and CITO for supporting this research and to CODES-ISSS for publishing this paper.

8. REFERENCES

[1] Gebotys, C., Design of secure encryption in dsp embedded processors. In *Proceedings of DATE, 2002*.

[2] Kocher, P., Timing attacks on implementations of Diffie Hellman, RSA, DSS, and other systems. In *Proceedings of CRYPTO*, LNCS 1109, Springer, pp. 104-113, 1996.

[3] Kocher, P., Jaffe, J., and Jun, B., Introduction to differential power analysis and related attacks. Technical Report. http://www.cryptography.com/dap/technical, 1998.

[4] Kocher, P., Jaffe, J., and Jun, B., Differential power analysis. In *Proceedings of Advances in Cryptography (CRYPTO '99),* pp. 388-397, 1999.

[5] Hasan, M. A., Power analysis attacks and algorithmic approaches to their countermeasures for koblitz curve cryptosystems. *IEEE Transactions on Computers,* 50, 10 (Oct. 2001), 1071-1082.

[6] Chari, S., Rao, J. R., Jutla, C. S., and Rohadgi, P., Towards sound approaches to counteract power-analysis attacks. In *Proceedings Advances in Cryptology (CRYPTO'99),* pp. 398-412, 1999.

[7] Muresan, R., *Modeling and Applications of Current Dynamics in a Complex Processor Core*. PhD Thesis. University of Waterloo, Canada, 2003.

[8] Coron, J-S., Resistance against differential power analysis for elliptic curve cryptosystems. In *CHES'99,* pp. 292-302.

[9] Kommerling, O., Kuhn, M. G., Design principles for tamper-resistant smartcard processors. In *Proceedings of the USENIX Workshop on Smartcard Technology (Smartcard '99)*, Chicago, May 1999, pp. 9-20.

[10] May, D., Muller, H. L., and Smart, N. P., Random register renaming to foil dpa. In *Proceedings of the 3rd International Workshop on CHES*, Paris, May 2001, pp. 28-38.

[11] Coron, J-S., Kocher, P., Naccache, D., Statistics and secret leakage. LNCS, Berlin: Springer-Verlag, Vol. 1962 pp. 157-173, 2001.

[12] Motorola Inc., *StarCore 140 Software Development Platform*. Hardware Reference Manual, Motorola, April 2000.

[13] Muresan, R., Gebotys, C., Instantaneous current modeling in a complex vliw processor core. ACM Transactions in Embedded Computing Systems (TECS), To Appear, 2004.

[14] May, D., Muller H., and Smart, N. P., Non-deterministic processors. In *ACISP 2001*, Springer Verlag, LNCS, July 2001.