

Making Register File Resistant to Power Analysis Attacks

Shuo Wang, Fan Zhang, Jianwei Dai, Lei Wang, and Zhijie Jerry Shi
University of Connecticut, Storrs, CT 06269, USA
Email:{shuo.wang, fan.zhang, jid06006, leiwang, zshi}@engr.uconn.edu

Abstract— Power analysis attacks are a type of side-channel attacks that exploits the power consumption of computing devices to retrieve secret information. They are very effective in breaking many cryptographic algorithms, especially those running in low-end processors in embedded systems, sensor nodes, and smart cards. Although many countermeasures to power analysis attacks have been proposed, most of them are software based and designed for a specific algorithm. Many of them are also found vulnerable to more advanced attacks. Looking for a low-cost, algorithm-independent solution that can be implemented in many processors and makes all cryptographic algorithms secure against power analysis attacks, we start with register file, where the operands and results of most instructions are stored. In this paper, we propose RFRF, a register file that stores data with a redundant flipped copy. With the redundant copy and a new precharge phase in write operations, RFRF provides data-independent power consumption on read and write for cryptographic algorithms. We validate our method with simulations. The results show that the power consumption of RFRF is independent of the values read out from or written to registers. Thus RFRF can help mitigate power analysis attacks.

I. INTRODUCTION

In recent years, security has become an important issue in the design of computer systems as more critical services are provided over the Internet and many networked devices communicate through wireless channels. Normally cryptographic algorithms are used to provide basic security functions such as confidentiality, authentication, and digital signature. Therefore their security is vital to any security mechanisms or protocols.

The security of cryptographic algorithms such as block ciphers and public-key algorithms relies on the secrecy of the key. Traditionally, when cryptanalysts examine the security of a cryptographic algorithm, they try to recover the secret key by observing the inputs and outputs of the algorithm. Assuming this type of attack models, cryptologists have made commonly-used cryptographic algorithms secure against such attacks.

However, a real computing device not only generates the outputs specified in algorithms but also inevitably produces some other information such as timing and power. These types of information, called side-channel information, can be exploited in *side-channel attacks* to retrieve secret keys. Side-channel attacks have successfully broken many algorithms that

are secure under traditional attack models [4], [7], [8]. Mobile devices and sensor nodes that work in the field, not protected by physical security mechanisms, are more vulnerable to side-channel attacks.

Among all side channel attacks, *power analysis*, which exploits the power consumption of a cryptographic system, can be carried out easily. It is very effective in breaking cryptographic algorithms [4], [7], [8]. In this type of attacks, adversaries learn what operations are performed and what data are processed by analyzing the power traces of computations. They can then figure out part or all of the bits in the secret key.

A lot of work has been done on the countermeasures against power analysis attacks. Many countermeasures studied software implementations, trying to make the power consumption of a crypto system either random or identical for different keys [4], [15]. The software countermeasures usually only work for specific algorithms and have large performance overhead. Very often, the countermeasures are found vulnerable to more advanced attacks. For example, randomized automata [15] for Elliptic Curve Cryptography operations are found not secure and can be broken by many new attacks [6], [13], [16], [19].

There are some hardware countermeasures [11], [12], [17]. The use of self-timed dual-rail logics is proposed in [11] to provide protection to power analysis attacks. Dynamic and differential logic is also employed in [17]. Both the two logic styles can make the power consumption of logic gates independent of the data values. One of the drawbacks of these methods is large area and power overhead. The method described in [12] compensates the power consumption of the system with voltage and frequency scaling techniques and an analog current injection circuit. However, besides large power overhead, frequency scaling affects the performance of software and may make the system vulnerable to timing attacks. It should be pointed out that memory security is not the primary focus of these above techniques.

Some other hardware countermeasures are at the architectural level [9], [10]. They randomize either the register renaming [9] or the issue of instructions in the instruction window [10] to make the power analysis attacks more difficult. However, these methods may not fit well with the low-end processors, which typically do not have register renaming mechanism or large instruction window to support out-of-order execution.

Acknowledgments: This research was supported by the NSF grants CCF-0621947 and CT-0644188, and the University of Connecticut Faculty Research Grants 446751 and 447388.

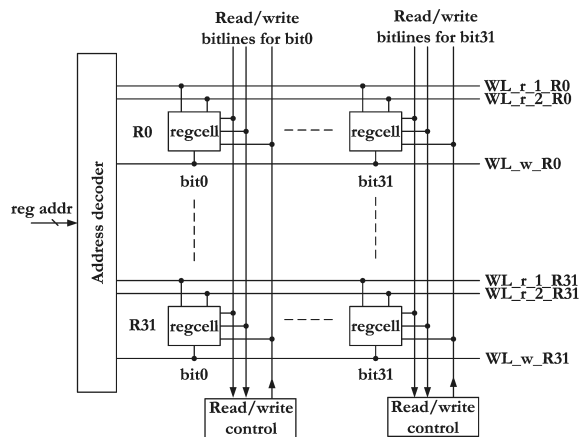


Fig. 1. An example of a 32×32 register file.

Despite the previous work, it is still desirable to study an effective, low-cost, algorithm-independent countermeasure for general-purpose processors, especially for low-end processors used in mobile devices or sensor nodes. Considering the fact that most of current attacks exploit power traces on memory and register accesses [1], [2], [14], in this paper we study how to make register file resistant to power analysis attacks. Admittedly, the register file is only one of many modules that produce power information that can be exploited. Improving the register file alone is not enough to make a device secure against power analysis attacks. However, the work on register file will help identify the effective mechanisms that mitigate power analysis attacks and hence can be extended to other components in the system as well.

The organization of this paper is as follows. Section II describes a typical register file in a processor core. Section III presents our methods. Section IV validates our methods with simulations. We conclude the paper in Section V.

II. VULNERABILITY OF REGISTER FILES TO POWER ANALYSIS ATTACK

On-chip memories are a major source of power information leakage that can be exploited in power analysis attacks. In particular, the register file, most frequently accessed, contributes a significant part to the power traces of computation.

Figure 1 shows the generic architecture of a conventional register file. For illustration purpose we consider the register file to comprise of 32 registers with 32 bits in each register. The address decoder selects the target register cells according to the register address and asserts the corresponding wordlines. Each register cell is an SRAM cell with multiple read/write ports [5] (2r/1w in this example). During a read operation, the wordline $WL_{r,1}$ or $WL_{r,2}$ is asserted, and the data stored in the selected register are evaluated by the read bitlines. Upon a write operation, data can be written from the write bitlines into the selected register. Typically, read logic is implemented in dynamic CMOS circuits that are precharged in the first half of the clock cycle and evaluated and sensed in the second half of the cycle, whereas write logic is in static circuits that are

activated in the first half of the cycle. Thus, this register file supports back-to-back read and write operations with single-cycle latency.

While the average power consumption of a register file is usually small, its frequent activities can generate substantial transient responses that contribute to data-dependent power traces. To illustrate, we consider the power consumption on register bitlines, a dominant component in the power profile of register file as bitlines typically have much larger parasitic capacitance than bit cells. We will also consider capacitive power consumption due to the high switching activities and the small size of register files where the leakage power is relatively small. During a register read, all read bitlines are precharged to the supply voltage V_{dd} in the first half of a clock cycle. During the next half of the cycle, some read bitlines will be discharged if 0's (or 1's depending on the read logic) are stored in corresponding bit cells. As a result, the charging/discharging of the involved bitlines will draw/release currents from the power supply. The power consumption is data-dependent, as the number of 0's in a register determines the amount of the charging/discharging current. For example, consider that registers $R0$ and $R31$ in Fig. 1 are read at different clock cycles, and the data (32 bits) stored in $R0$ and $R31$ are 0000_0000 and $FFFF_FFFF$, respectively (For convenience, we use eight hexadecimal digits to denote 32 bits). All the 32 bitlines are discharged when $R0$ is read, while there is no discharging current when $R31$ is read. Such different activities can be observed from the power traces and hence be exploited in power analysis attacks. Similarly, during register write, $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions are expected on the write bitlines and in bit cells, which also contribute to the data-dependent power traces. It is worth to notice that read and write operations induce different data-dependent patterns on power traces. Since write logic is typically implemented in static circuits, both $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions can be observed on the write bitlines during write access. On the other hand, as read logic is usually designed with dynamic circuits that are precharged to V_{dd} , only $1 \rightarrow 0$ transitions can be observed on the read bitlines during the evaluation phase (the $0 \rightarrow 1$ transitions can be observed during the precharge phase though).

III. REGISTER FILE WITH REDUNDANCY

As explained in section II, conventional register files are vulnerable to power analysis attacks as the power consumption is highly data-dependent. In this section, we propose a method to make register files resistant to this kind of attacks.

A. Modified Register File

As power consumption is determined by the patterns of $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions, one approach to protect register file from power analysis is to make the number of transitions independent of the data values accessed during read and write operations. The proposed method exploits this idea by introducing redundancy in the register file so that each register access will always incur the same number of transitions on the

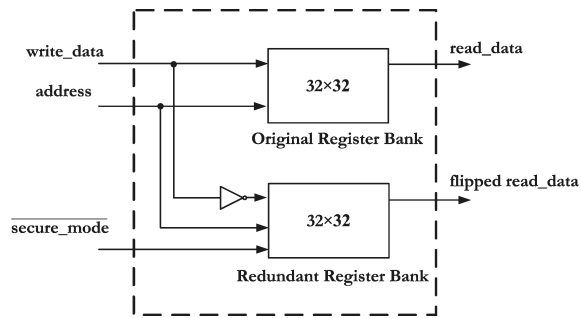


Fig. 2. Modified Register File.

read and write bitlines. Specifically, we introduce an additional register bank that is activated only when security programs are running. Data are stored at two locations in the register file. One location has the normal bit representation of the value and the other has the complemented bits. Writing to a register updates the bits at both locations. Reading from the register file also retrieves values from both locations. The new register file will be referred to as *RFRF* (register file with redundant, flipped copies).

Figure 2 shows the design of RFRF. In comparison with the conventional register file, RFRF comprises two identical register banks. For example of a typical embedded processor, each bank has 32 registers and each register has 32 bits. One bank is used to store the original data values, referred to as the original bank. The other bank, referred to as the redundant bank, stores the complemented bits. For convenience, we use R_i and R_i' to denote the register i in the original and redundant banks, respectively.

One may think that the proposed approach will introduce large hardware overhead due to the additional register bank. However, we expect this overhead to be well paid-off for most embedded systems where protection from power analysis is becoming a top priority. To reduce the power overhead due to redundant reads and writes, we define two operation modes for RFRF, normal mode and secure mode. The redundant operations are enabled only when the system is in the secure mode. In the normal mode, RFRF works in the same way as a conventional register file. Thus, it only consumes extra power in the secure mode for security programs.

As shown in Fig. 2, in the normal mode the data $write_data$ goes directly to the original bank, whereas the redundant bank is shutdown by the signal $secure_mode$ ($secure_mode = 1$). Thus, only the original bank is accessed and the register file works as a conventional one.

In the secure mode ($secure_mode = 0$), both the original and redundant banks are enabled. The redundant bank performs write and read operations on the flipped bits of $write_data$. On a write, $write_data$ and its flipped bits are simultaneously stored in the original and redundant banks, respectively. On a read, both banks are accessed. The original and flipped $write_data$ are read out from the register file, where the corresponding read bitlines are evaluated by these data. Both the original data and the flipped copy are read out

to the pipeline registers, which are placed between pipeline stages, however only the original data will be actually used in the functional units for further computations. Thus, the capacitive load of the two read ports can match each other in order to avoid power information leakage. Again, this capacitive load of the flipped read-out does not incur power overhead in the normal mode where the redundant register bank is shut down.

B. Resistance to Power Analysis

We now discuss how RFRF enables data-independent power consumption during read and write register accesses.

1) *Read Operation*: As discussed in section II, for a read operation, the numbers of $1 \rightarrow 0$ transitions (during the evaluate phase) and $0 \rightarrow 1$ transitions (during the precharge phase) on read bitlines are determined by the number of 0 bits in the data being read out. It can be shown that in the secure mode, the total number of 0's in the original and the flipped $write_data$ is always equal to W , where W is the register width. Consider a register data with n bits of 0's, where ($0 \leq n \leq W$). In the secure mode, the register in the original bank stores these 0's and the corresponding register in the redundant bank stores the flipped copy that has $(W - n)$ bits of 0's. Thus, the total number of 0's is exactly W . One can consider that the modified register file has a width of $2W$ bits and half of these bits are always 0.

Consider a specific example where $W = 32$, $R_0 = 0000_0001$, and $R_0' = FFFF_FFFE$. Among all the bits in R_0 and R_0' are there 32 0's. During the read operation, these 0-value bits will discharge the read bitlines. Note that no matter what value R_0 has, the number of discharged read bitlines is always 32 when the flipped copy R_0' is also included. Then, when the read bitlines are precharged, these 32 bitlines will be charged back to V_{dd} . Therefore, a read access in the secure mode will always introduce an equal number of $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions that are data-independent.

2) *Write operation*: For a write operation, things become a little more complicated because the number of bit transitions depends not only on the value being written to the register but also on the value currently stored in the register. Simply writing a redundant copy does not defeat power analysis attacks. Even worse, it enlarges the power variance among different data values and makes it easier for adversaries to launch power analysis attacks.

Consider an example where R_0 is 0000_0000 and will be updated with 0000_0001 . In the normal mode, there is one $0 \rightarrow 1$ transition in the original bank. In the secure mode, in addition to this transition, there is one more $1 \rightarrow 0$ transition in the redundant bank. In total, there are two bit transitions. If the original value in R_0 is $FFFF_FFFF$ instead, there will be a total of 62 ($= 2 \times 31$) transitions. The redundant bank actually doubles the number of transitions, making it easier for adversaries to observe the power differences.

Therefore, we need to modify the write operation so that its power consumption cannot be exploited. Inspired by the precharge in read operations, which makes the discharging

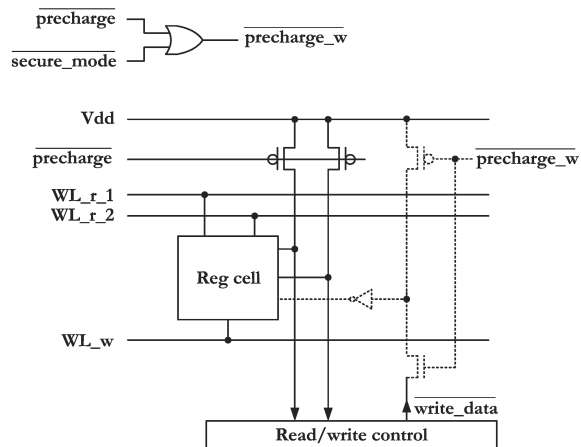


Fig. 3. Modified write circuitry. (Dotted lines represent the modified write bitlines. $\overline{secure_mode} = 0$ indicates the secure mode.)

power dependent only on the data value in the current cycle, we add a precharge phase in write operations.

Figure 3 shows the modified write circuitry. In the figure, a signal $\overline{precharge_w}$, generated from $\overline{secure_mode}$ and $\overline{precharge}$ (the precharge control in read operations), controls the precharge in write operations. In addition, an inverter is added to prevent the write bitline from being pulled down by a 0 in the cell. If the register file is in the normal mode ($\overline{secure_mode} = 1$), the NMOS in dotted line is open while the PMOS in dotted line is closed. The register file works in the same way as a conventional implementation. If the register file is in the secure mode ($\overline{secure_mode} = 0$), however, the write bitline will be precharged in the first half of the clock cycle by the PMOS. 0 will be written into register that are selected by the write wordline, replacing the old value. In the second half of the cycle, the PMOS is closed and the NMOS is open. New data will be written into the selected register.

With the precharge phase, RFRF has $W\ 1 \rightarrow 0$ transitions during the write precharge, and also exactly $W\ 0 \rightarrow 1$ transitions when the new data is actually written into the bit cells.

Let us look at the previous example again, in which $W = 32$, $R0 = 0000_0000$, and $R0' = FFFF_FFFF$. Now consider a write operation that updates $R0$ to 0000_0001 (and $R0'$ to $FFFF_FFFE$). In the precharge phase, both $R0$ and $R0'$ are set to 0. Since the total number of 1's in $R0$ and $R0'$ is 32, there are 32 $1 \rightarrow 0$ transitions in total. In the second half of the cycle, there is one $0 \rightarrow 1$ transition in $R0$ and 31 $0 \rightarrow 1$ transitions in $R0'$. Thus, there is a total of 32 $0 \rightarrow 1$ transitions in the second half of the cycle. No matter what value $R0$ has, the number of bit transitions is the same.

Note that the additional overwriting operation introduces energy overhead, but the overhead is confined in the secure mode. In the normal mode, no precharge is performed and RFRF behaves as a conventional register file.

Many processors require back-to-back write and read accesses on the same register in the same cycle. However, the write operations in RFRF will finish in the second half of

the clock cycle due to the introduced precharge phase. (Note that finishing the writes in the first half or the second half of the cycle does not have a necessary impact on the processor performance.) To support back-to-back write and read, a small circuit is needed to check whether the register being written to is also being read at the same time and bypass the written data to the read bitlines when necessary.

C. Support for the Modified Register File

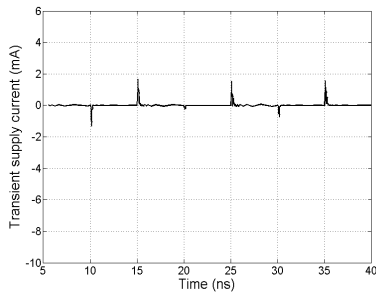
As mentioned at the beginning of this section, we maintain two operation modes, the normal mode and the secure mode, to reduce the energy overhead. Accessing the redundant bank and precharging write bitlines are only enabled in the secure mode. Thus, we need to add a processor status bit $\overline{secure_mode}$ to indicate the operation mode of the register file. In addition, new instructions need to be added to allow the security applications to enter and leave the secure mode and to allow the operating system to save and restore the status bit.

When an application needs to perform a cryptographic algorithm, it first enters the secure mode with the new instruction that sets $\overline{secure_mode}$ to 1. When the cryptographic operation is done, the application leaves the secure mode by setting $\overline{secure_mode}$ to 0.

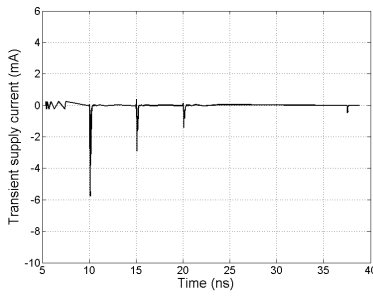
IV. SIMULATIONS

In this section, we use simulation to validate our design and estimate the energy overhead. In simulations, we assume a register file that has 32 registers of 32-bit each. The register file has 4 read bit lines and 3 write bit lines. In order to obtain supply current traces as accurately as possible, we implemented the physical design of the register file in TSMC 0.18- μm process technology [18] and ran simulations at the transistor level, considering parasitics with Spectre (a SPICE simulation tool from Cadence [3]). The simulations ran with a clock of 100MHz.

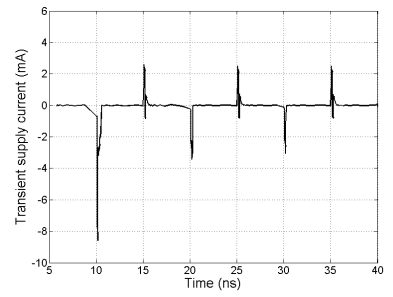
To validate our proposed technique, we first compare RFRF with the conventional approach. Figure 4 shows the transient supply current for a conventional register file. Figure 4(a) shows the results for three read operations, each reading a different register. The data in the three cycles are 0000_0000 , 0000_FFFF , and 0000_5555 . The three reads are performed through the same bit lines, which has the value $FFFF_FFFF$ before the first read. The last downward spike shown in Fig. 4(a) is the precharge for the next read. Figure 4(b) shows the results for three write operations, writing 0000_0000 , 0000_FFFF , and 0000_5555 to the same register that holds $FFFF_FFFF$ originally. The same write bit lines were employed for all three write operations. Figure 4(c) shows the results of performing two read operations and one write operation per cycle for three cycles. The operations are on different registers, but the data that are read out or written to registers are same in each cycle. The data in cycles 1, 2, and 3 are 0000_0000 , 0000_FFFF , and 0000_5555 , respectively. From the figures, we can see that the power consumption of the conventional register file is highly data-dependent.



(a) One read per cycle

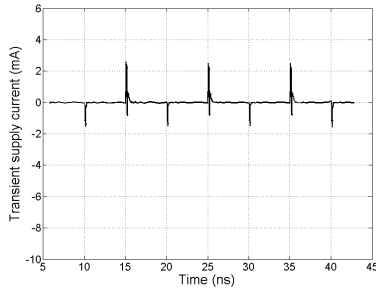


(b) One write per cycle

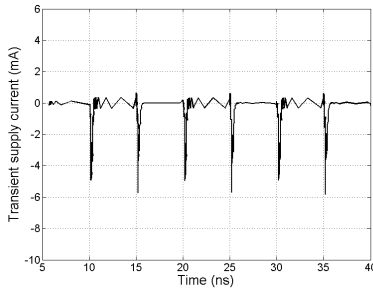


(c) Two reads and one write per cycle

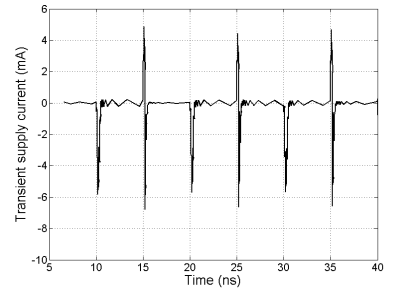
Fig. 4. Supply current for conventional register file.



(a) One read per cycle

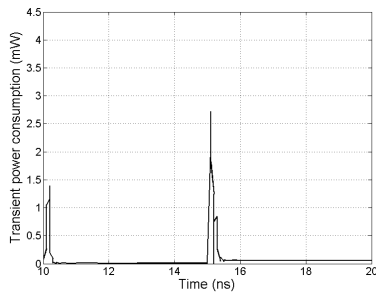


(b) One write per cycle

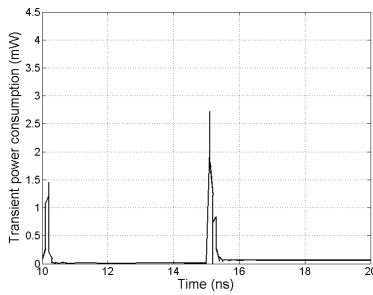


(c) Two reads and one write per cycle

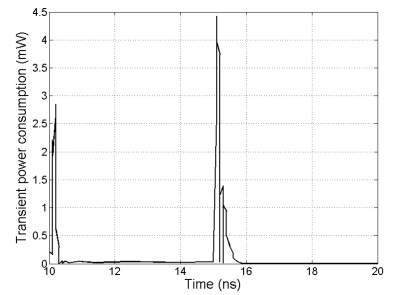
Fig. 5. Supply current for RFRF.



(a) Conventional register file

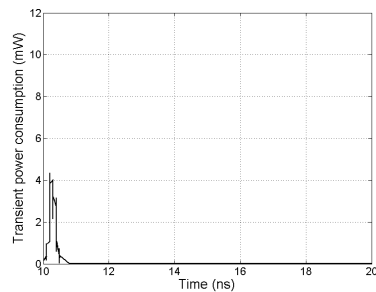


(b) RFRF in the normal mode

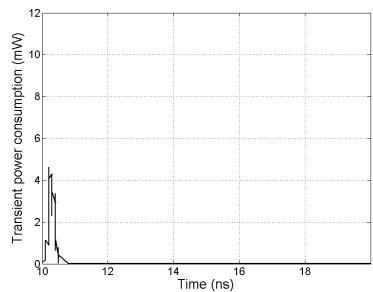


(c) RFRF in the secure mode

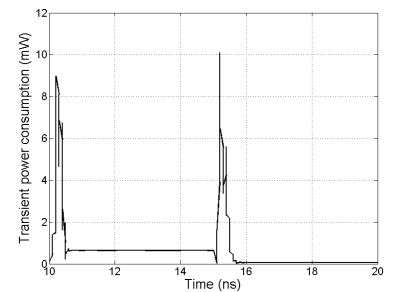
Fig. 6. Transient power consumption for a read operation.



(a) Conventional register file



(b) RFRF in the normal mode



(c) RFRF in the secure mode

Fig. 7. Transient power consumption for a write operation.

In Fig. 5, we present the simulation results for RFRF performing the same operations. Figures 5(a), (b), and (c) show the results for reading one register, writing to one register, and reading two register and writing to one register in the same cycle, respectively. Figure 5(b) clearly shows the precharges for write operations. It can be seen that the power consumption of each type of operation is similar in all three settings. Since the power consumption is independent of data value, the proposed RFRF can mitigate power analysis attacks. The access latencies of RFRF read and write operations are only slightly increased from those of the conventional register file, which are acceptable for low-end processors vulnerable to power analysis attacks. Note that although there is an additional precharge phase for the write operations, performance would not be degraded as long as the writes can finish in the second half of the cycle and the back-to-back write and read accesses are supported.

As mentioned in Section III, the improvement in security comes at the cost of energy overhead. We compare the average power consumption for the following three settings: conventional register file, RFRF in the normal mode, and RFRF in the secure mode. Figures 6 and 7 compares the transient power consumption for read and write operations, respectively. In the simulations, we assume that half of the bits read from a register are 0 and that half of the bit cells are switched during write operations. From the figures, we can see that in the normal mode, RFRF consumes almost the same amount of power as the conventional design, for both read and write. In the secure mode, however, the power consumption of RFRF is doubled for read operations. Because of the newly added precharge phases, write operations consumes almost 3.6x power than the conventional design. The overhead is large. Fortunately, the overhead is only for the secure mode. The overall power overhead is determined by how often and how long the specific workloads need the system to stay in the secure mode.

V. CONCLUSIONS

In this paper, we propose RFRF, a register file that is resistant to power analysis attacks. By storing a flipped copy of data and adding a precharge phase for write operations, RFRF has the same number of transitions on bitlines for all read or write operations. We also validate our method with simulations. The results show that the power trace of RFRF is independent of data values. When combined with similar solutions for other processor components, it is expected that the proposed method can strongly protect the system from power analysis attacks.

The overhead of RFRF mainly comes from the redundant register bank and additional circuitry for the precharge phase in writes and for the data bypassing during back-to-back write and read. However, the overhead is considered worthwhile as chip security is becoming a top priority in most embedded systems, although efforts should be spent on improve the cost-efficiency of the solution. To reduce the power overhead, the RFRF has two working modes where only security applications incur energy overhead for the security enhancement.

It should also be pointed out that due to variations in process, voltages, and temperature, as well as the difference in routing for the two register banks, the power consumption may not always be absolutely independent on data values. However, this effect is considered difficult to exploit for the power analysis attacks. In the future, we will study more efficient implementation of RFRF and incorporate the countermeasures into on-chip cache and memory. Besides, RFRF will also be studied for other purposes such as thwarting fault analysis attacks and improving the reliability of devices.

REFERENCES

- [1] T. Akishita and T. Takagi. Zero-value point attacks on elliptic curve cryptosystem. In *ISC2003, Lecture Notes in Computer Science*, 2851, pages 218–233, December 2003.
- [2] D. J. Bernstein. Cache-timing attacks on AES. In *Preliminary report. Available at cr.yp.to/antiforgery/cachetiming-20050414.pdf*, 2005.
- [3] Cadence: <http://www.cadence.com>.
- [4] J. S. Coron. Resistance against differential power analysis for elliptic curve cryptosystems. In *Proceedings of CHES'99*, pages 292–302, August 1999.
- [5] E. Fetzter, L. Wang, and J. Jones. The multi-threaded, parity protected, 128 word register files on a dual-core Itanium[®] family processor. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 382–383, February 2005.
- [6] D. Han, N. S. Chang, S. W. Jung, Y. H. Park, C. H. Kim, and H. Ryu. Cryptanalysis of the full version randomized addition-subtraction chains. In *Proceedings of 8th Australasian Conference on Information Security and Privacy, ACISP 2003*, pages 67–78, July 2003.
- [7] J. Jaffe. A first-order DPA attack against AES in counter mode with unknown initial counter. In *Cryptographic Hardware and Embedded Systems - CHES 2007, LNCS, vol. 4727*, pages 1–13, September 2007.
- [8] P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *Proceedings of CRYPTO'99, LNCS 1666*, pages 388–397, August 1999.
- [9] D. May, H. L. Muller, and N. Smart. Random register renaming to foil DPA. In *Cryptographic hardware and embedded systems-CHES 2001*, pages 28–38, May 2001.
- [10] D. May, H. L. Muller, and N. P. Smart. Non-deterministic processors. In *Proceedings of ACISP 2001, LNCS 2119*, pages 115–129, July 2001.
- [11] S. Moore, R. Anderson, P. Cunningham, R. Mullins, and G. Taylor. Improving smart card security using self-timed circuits. In *Proceedings of ASYNC 2002*, pages 211–218, 2002.
- [12] R. Muresan, H. Vahedi, Y. Zhanrong, and S. Gregori. Power-smart system-on-chip architecture for embedded cryptosystems. In *Proceedings of CODES+ISSS'05*, pages 184–189, October 2005.
- [13] K. Okeya and K. Sakurai. On insecurity of the side channel attack countermeasure using addition-subtraction chains under distinguishability between addition and doubling. In *Proceedings of 7th Australasian Conference on Information Security and Privacy, ACISP 2002*, pages 420–435, July 2002.
- [14] D. A. Osvik, A. Shamir, , and E. Tromer. Cache attacks and countermeasures: the case of AES. In *CT-RSA 2006: The Cryptographers' Track at the RSA Conference, LNCS, vol. 3860*, pages 1–20, November 2006.
- [15] E. Oswald and M. Aigner. Randomized addition-subtraction chains as a countermeasure against power attacks. In *Proceedings of CHES 2001*, pages 39–50, May 2001.
- [16] Z. J. Shi and F. Zhang. New attacks on randomized ECC algorithms. In *Proceedings of EITC 2006*, pages 22–25, August 2006.
- [17] K. Tiri and I. Verbauwhede. Securing encryption algorithms against DPA at the logic level: Next generation smart card technology. In *Proceedings of CHES 2003, LNCS 2779*, pages 127–137, September 2003.
- [18] TSMC: <http://www.tsmc.com>.
- [19] F. Zhang and Z. J. Shi. Power analysis attacks on ECC randomized automata. In *Proceedings of ITNG 2007*, pages 900–901, April 2007.