Green Secure Processors: Towards Power-Efficient Secure Processor Design

Article · January 2010

DOI: 10.1007/978-3-642-17499-5_13 · Source: DBLP

2 authors, including:

Yan Solihin
University of Central Florida
146 PUBLICATIONS **5,223** CITATIONS

Green Secure Processors: Towards Power-Efficient Secure Processor Design

Siddhartha Chhabra and Yan Solihin

Dept. of Electrical and Computer Engineering, North Carolina State University
Raleigh, USA
{schhabr, solihin}@ncsu.edu

Abstract. With the increasing wealth of digital information stored on computer systems today, security issues have become increasingly important. In addition to attacks targeting the software stack of a system, hardware attacks have become equally likely. Researchers have proposed *Secure Processor Architectures* which utilize hardware mechanisms for memory encryption and integrity verification to protect the confidentiality and integrity of data and computation, even from sophisticated hardware attacks. While there have been many works addressing performance and other system level issues in secure processor design, power issues have largely been ignored. In this paper, we first analyze the sources of power (energy) increase in different secure processor architectures. We then present a power analysis of various secure processor architectures in terms of their increase in power consumption over a base system with no protection and then provide recommendations for designs that offer the best balance between performance and power without compromising security. We extend our study to the embedded domain as well. We also outline the design of a novel hybrid cryptographic engine that can be used to minimize the power consumption for a secure processor. We believe that if secure processors are to be adopted in future systems (general purpose or embedded), it is critically important that power issues are considered in addition to performance and other system level issues. To the best of our knowledge, this is the first work to examine the power implications of providing hardware mechanisms for security.

Keywords: Power Analysis, Secure Processor Architectures, Memory Encryption, Memory Authentication, Embedded Systems Security

1 Introduction

Many applications handle security sensitive data like consumer credit card numbers, bank account numbers, personal information etc. With the increasing wealth of digital information stored on computer systems today, attackers have increased motivation to attack systems for financial gains. Traditionally, attackers have exploited vulnerabilities in application code and Operating System (OS) to mount software attacks resulting in the application leaking sensitive data.

However, with the computation becoming increasingly mobile and mobile devices being prone to theft or loss, attackers can get physical access to the system to launch *physical* or *hardware* attacks. Hardware attacks are made possible as most computer systems communicate data in its plaintext form between the processor chip and off-chip devices such as the main memory. This presents the attackers with a situation where they can place a bus analyzer that snoops data communicated between the processor chip and other chips [1]. In addition, data is also stored in its plaintext form in the main memory which allows an attacker having physical access to the system to dump the memory contents and scan it, possibly gaining a lot of valuable information such as passwords [2]. Although physical attacks may be more difficult to perform than software-based attacks, they are very powerful as they can bypass all software security solutions that might be deployed on the system. The recent proliferation of modchips in gaming systems has shown that given sufficient financial payoffs, hardware attacks are realistic threats.

Recognizing these threats, researchers have proposed secure processor architectures [3–17]. Secure processors assume that all off chip devices are vulnerable and the processor chip itself provides a natural security boundary. Secure processor architectures deploy hardware mechanisms to protect the *privacy* and *integrity* of application code and data. **Memory encryption** protects the privacy of data by encrypting data and code as it moves off the processor chip and decrypting it back once it is reloaded. Memory encryption provides protection against *passive* attacks, where an adversary tries to silently observe application data. **Memory authentication** protects the integrity of code and data by associating and verifying a Message Authentication Code (MAC) with each data block as it moves on and off the processor chip. Memory authentication provides protection against active attacks where the attacker tries to modify data in off-chip structures to change application behavior, potentially resulting in leaking sensitive information.

Secure processor research thus far has primarily focussed on reducing the performance and storage overheads of providing hardware mechanisms for security or on resolving system-level issues like lack of support for inter-process communication , virtual memory etc. Unfortunately power issues have largely been ignored for secure processor architectures. Figure 1 shows the power density increase for contemporary processors over the last 40 years.

The trend clearly indicates the need to consider power as one of the key design considerations. The hardware security mechanisms not only result in performance and storage overheads, but also increase the overall power consumption and based on the actual mechanisms used, this increase in power can be very significant(Section 2). There has been prior work in designing low power security cryptographic algorithms [18, 19], however, power issues have not been considered at the architecture level. If secure processors are to be adopted in future systems, it is critically important that power issues are considered in addition to performance and system-level issues. To the best of our knowledge, this is the first work to explore power implications of secure processor architectures.

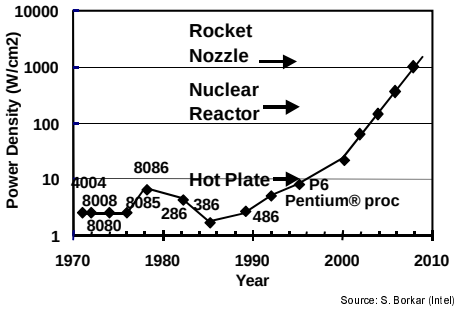

Fig. 1. Power Density Increase in Processors

Contributions: In this paper we analyze the power implications of using secure processor architectures. Overall, we make the following contributions:

- We analyze the sources of power consumption in various secure processor designs.
- We present a power analysis of various secure processor architectures in terms of their increase in power consumption over a base system with no protection and provide recommendations for designs that offer the best balance between performance and power without compromising security.
- We extend our study to the embedded domain and show that some design decisions offering the best power-performance balance in the general purpose domain do not necessarily apply to the embedded domain.
- We explore a novel *hybrid* cryptographic engine that combines multiple encryption mechanisms designed with the primary goal to minimize power overheads without compromising performance or security.

The rest of the paper is organized as follows. Section 2 presents the sources of power overhead in secure processor designs. Section 3 describes our experimental setup. Section 4 presents a power evaluation of currently proposed secure processor architectures. Section 5 presents a parallel evaluation for the embedded domain. Section 6 presents a discussion on our novel cryptographic engine and we conclude in Section 7.

2 Power Overheads in Secure Processors

Secure processors employ hardware mechanisms for memory encryption and authentication for protecting the privacy and integrity of data. Each mechanism contributes to increasing the overall power consumption of the processor. In this section, we first present the currently proposed mechanisms for memory encryption and authentication and then discuss the factors associated with each mechanism that contribute to power overheads in a secure processor architecture.

2.1 Memory Encryption

Memory encryption mechanisms are used to protect the privacy of data by encrypting and decrypting the data block as it moves on and off the processor chip. Direct encryption and counter-mode encryption form the most widely used forms of encryption for current proposals on secure processor architectures.

Direct Encryption: In direct encryption, as a cache block is evicted off the processor chip, an on-chip cryptographic engine encrypts it before storing it in the main memory [5]. When the block is reloaded from main memory, the cryptographic engine decrypts it before supplying it to the processor. Based on its operation, direct encryption results in the following power overheads:

Static power: Direct encryption in effect increases the latency of fetching a block from the main memory. The decryption of the block lies directly in the processor's critical path and this results in increased static power consumption for other processor structures that might be idle due to the decryption latency.

Counter Mode Encryption: Recently proposed memory encryption mechanisms have utilized *counter-mode encryption* [7, 9–15] due to its ability to hide cryptographic delays on the critical path of memory accesses. This is done by decoupling the cryptographic work from the actual data. In counter-mode encryption, a per-block *seed* is encrypted to generate a cryptographic *pad*, which is then XORed with the memory block to encrypt or decrypt it (Figure 2). However, the choice of seed is critical for both performance and security. The security of counter-mode encryption is contingent on the uniqueness of the pads which are XORed with blocks to encrypt/decrypt them. This essentially means that the seeds used to generate the pads must be unique. Prior works use the block address (virtual or physical) as a component of the seed to ensure *spatial* uniqueness when the blocks are stored in memory. In addition, a per-block counter incremented on every writeback of the block to main memory is also included as a component of the seed to ensure *temporal* uniqueness. From a performance point of view, the seed components must be known at cache miss time to overlap the pad generation (cryptographic) latency with the memory fetch latency. The block address is known at cache miss time and in order to have the block counter available too, an on-chip *counter cache* is used to cache the per-block counters. If the counter is found in the cache, the pad generation latency can be overlapped with the memory fetch latency. Based on its operation, counter-mode encryption results in the following power overheads:

Static power: While a counter cache hit will hide the cryptographic latency of generating the pad, a miss will result in a separate memory request issued to fetch the counter. Only when the counter is fetched can the cryptographic operation start. Thus a counter cache miss can increase the idle time of processor structures, thereby increasing their static power consumption.

Counter Cache: On each cache miss, the counter cache needs to be consulted to find the per-block counter. The addition of the cache contributes to both dynamic and static power consumption of the processor.

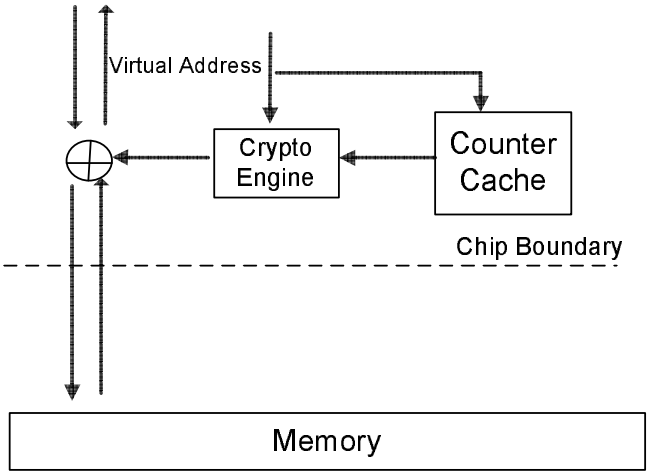

Fig. 2. Counter-mode Encryption

2.2 Memory Authentication

Memory authentication is used to protect the integrity of data. One of the early mechanisms for memory authentication [5] used a Message Authentication Code (MAC) associated with each block which is computed and verified as a block moves on and off the processor chip. However, per-block MAC based authentication is vulnerable to replay attacks, where an attacker can record an old data block with its MAC and replay it as the current value to the processor. Due to its security limitations, *Merkle tree* authentication was proposed and represents the family of memory authentication mechanisms used by current proposals.

Merkle tree Authentication: In Merkle tree memory integrity verification, a tree of MACs is computed over the memory. The root of this tree is stored securely in an on-chip register and never goes off the processor chip. On loading a block, its integrity is verified by checking its chain of MAC values up to the root MAC (Figure 3). Since the root of the authentication tree stores information about all the blocks in memory and never goes off the processor chip, an attacker cannot modify or replay any value without detection.

In the standard Merkle tree mechanism, a tree of MACs is built over the *entire* main memory. However, a recent work showed that if counter-mode encryption is used, it is not necessary to build the Merkle tree over the entire main memory. A tree of MACs built only over the per-block counters in main memory along with a per-block MAC can provide the same level of security as a standard Merkle tree, and at the same time result in a much smaller and shallower tree(Figure 4). This tree formation was called Bonsai Merkle Trees (BMTs) and it was shown that they can significantly reduce the overheads of memory authentication mechanism [20].

Based on its operation, Merkle tree authentication has the following power overheads:

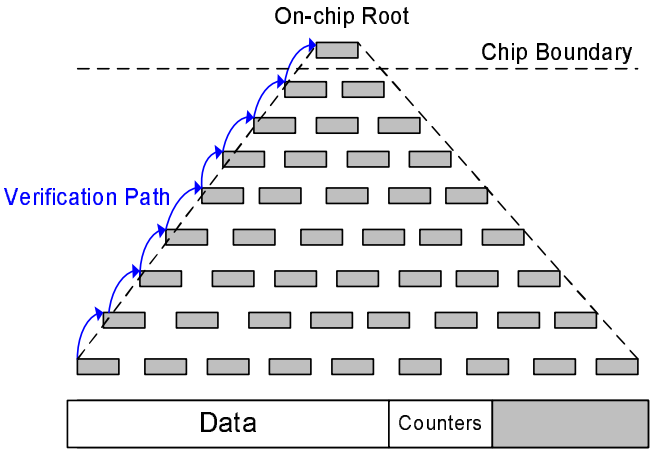

Fig. 3. Standard Merkle Tree Integrity Verification

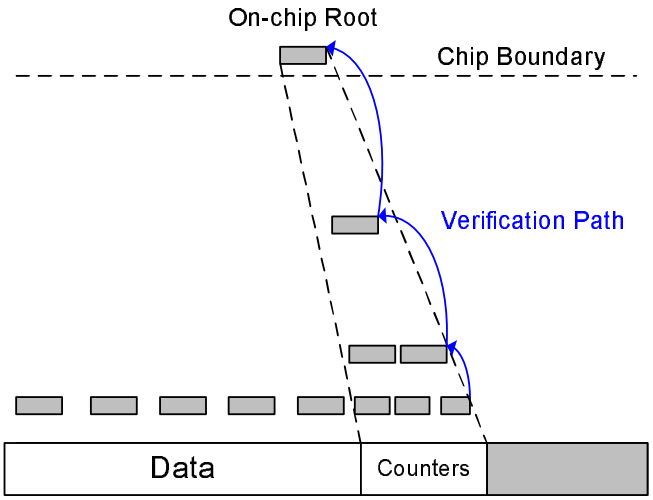

Fig. 4. Bonsai Merkle Tree Integrity Verification

Static Power: Authentication can be *imprecise*, where the processor is allowed to continue execution and retire the instruction loading the data. However, if *precise* authentication is used, the instruction cannot retire until the authentication is completed. Once again, this can result in a significant increase in the static power consumption depending on the number of MACs that need to be fetched and verified to establish the integrity of the block.
Dynamic Power: As an optimization, the MAC values can be cached and on verification of the first block found in the cache, the block can be considered to be verified for integrity as the MAC block found in the cache can be assumed to form the root of a small Merkle tree which is guaranteed to be secure as it is on-chip. These additional cache accesses result in an increase in the power consumption of caches and the processor as a whole.

2.3 Other Power Sources

There are other sources that contribute to the power consumption of a system equipped with hardware mechanisms for security. We describe the sources here, however, the results presented in the following sections do not account for these power sources and hence present a lower bound on the power overheads of secure processors.
Cryptographic engine power consumption: The on-chip cryptographic engine is responsible for all the cryptographic work required for memory encryption and authentication and forms another major source of additional power consumption in secure processor architectures.
Increased power consumption of other structures: Cryptographic metadata (counters and MACS) needs to be fetched on the processor chip for decrypting or verifying the block. This results in increased work for off-chip structures like the memory bus, memory controller, memory etc., thereby, contributing to a further increase in the power consumption of the system as a whole.

3 Experimental Setup

3.1 Machine Models

We use SESC [21], an open source execution driven simulator, to model the secure processor architectures evaluated in this paper. We use Wattch [22] power models for our power evaluations. For uniprocessor evaluations, we model a 3-issue, out-of-order processor with split L1 data and instruction caches. Both caches have 16KB size, 2-way set associativity, and 2-cycle hit latency. The L2 is a unified 1MB, 8-way set associative, cache with 10-cycle hit latency. All caches have 64-byte block size and use LRU replacement. We assume 2GB main memory with 490-cycle round-trip latency. The cryptographic engine used is a 16-stage pipelines, 128-bit AES [23] engine with 80-cycle latency [24]. The counter cache used for counter-mode encryption is a 32KB, 16-way set associative cache. The default MAC size is 128-bits. The process parameters for the simulated architecture are 5GHz clock and 70nm feature size. Note that we assume a very

optimistic latency of 80-cycles for the cryptographic engine to account for technological advances. Hence, the figures presented in this paper represent a lower bound and the energy overheads on a real system are likely to be even higher.

For the CMP evaluation, we model a two-core CMP where each core has private L1 data and instruction caches. The l2 cache and all lower levels of the memory hierarchy are shared by both cores. To better match current CMP configurations, we have increased the L2 cache size to 2MB. All other system parameters are the same as the uniprocessor case.

The simulated embedded processor is modeled after ARM's cortex A-8 processor [25] with the cryptographic parameters kept the same as the modeled general purpose processor.

3.2 Benchmarks

We use all C/C++ SPEC2K benchmarks [26] for our general purpose system evaluations. We use the reference input set for each benchmark and simulate it for 1 billion instructions after skipping 5 billion instructions. The figures show the individual results for benchmarks having an L2 miss rate of more than 20%, however, the average is calculated across all 21 benchmarks.

For our CMP evaluations, we have created 21 pairs of benchmarks using SPEC2K benchmarks. Each pair consists of two SPEC2K benchmarks which are spawned as two separate threads on each of the two cores of the modeled CMP system. To capture different memory behaviors, we classify the benchmarks into two categories: those that have an L2 miss rate of of more than 20%, when run alone, and those that have an L2 miss rate of less than 20%, when run alone. We select benchmarks from each group and combine them so all memory behaviors are represented. In the first group of benchmark pairs, the benchmarks in a pair are both taken from low miss rate category: *perlbmk_twolf* and *twolf_vpr*. In the second group of benchmark pairs, one benchmark is taken from the low miss rate category while the other is taken from the hiss miss rate category: *apsi_bzip2*, *gzip_applu*, *gzip_apsi*, *perlbmk_art*, *swim_gzip*, *swim_twolf*, *vpr_applu*, *vpr_art*, *applu_gzip*, and *swim_perlbmk*. The last group of benchmark pairs is the one where both benchmarks are taken from the high mate rate category: *apsi_art*, *art_mcf*, *art_swim*, *mcf_art*, *mcf_swim*, *swim_art*, *swim_mcf*, *equake_apsi*, and *mcf_apsi*. For each simulation, we use the reference input set and simulate for 1 billion instructions after skipping 5 billion instructions. The instructions are skipped only on the first benchmark in the benchmark pair and the simulation ends when the combined number of instructions simulated reaches 1 billion.

For our embedded domain evaluations, we use nine benchmarks from the MiBench [27] embedded benchmark suite (We excluded the benchmarks that had a compilation error in our simulation framework). The benchmarks have been picked from the four categories: Automotive and industrial control(*basicmath, bitcount, qsort, susan*), Network (*dijkstra, patricia*), Security (*rijndael, sha*), and telecommunications (*fft*). Each benchmark is simulated to completion.

4 Recommendations for Energy Efficient Secure Processor Design

In this section, we present a power evaluation of the currently proposed secure processor architectures and provide recommendations towards energy- efficient secure processor design. All figures plot the overall energy consumption of the discussed architecture, unless otherwise stated. We first present our study for general purpose processor architectures and then present a parallel discussion for embedded systems domain.

4.1 General Purpose Secure Processors

We first present the energy overheads for the most commonly used memory encryption and authentication mechanisms: Counter-mode encryption and standard Merkle trees. Figure 5 shows the energy overheads for SPEC2K benchmarks.

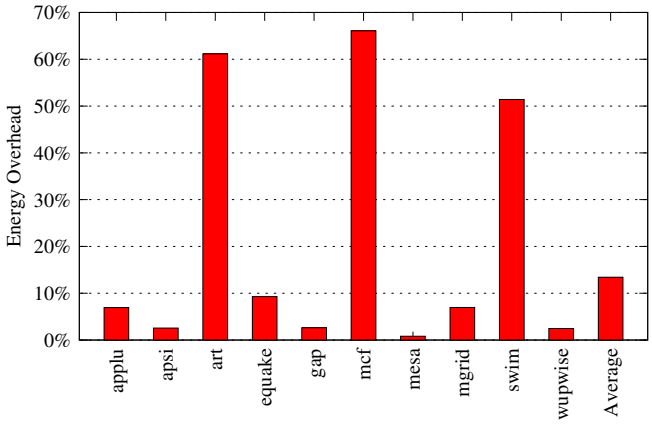

Fig. 5. Energy Overhead of Counter-mode Encryption with Standard Merkle Trees

It can be seen from the figure that secure processor mechanisms result in an average overhead of 13.42% across SPEC2K benchmarks. For memory intensive applications like art, mcf, and swim, the overheads are in excess of 50% with mcf resulting in as much as 67% overhead over a system with no protection. These overheads are extremely high considering the fact that the power increase from one processor generation to another, accompanied by a significant performance improvement due to increased clock speeds, is roughly around 10%. For example, Pentium III, running at 500MHz, consumes 7.8% additional power compared to a Pentium II, running at 233MHz.

Observation: Secure processor mechanisms add non-trivial power overheads, making power even more important in the context of secure processor design.

In order to better understand the overheads and possible avenues for power (energy) reduction techniques, we provide a breakdown of the overheads. Figure 6(a) shows the breakdown of power overheads into encryption and integrity verification components and Figure 6(b) shows the breakdown of the increase in energy consumption into the increase in pipeline energy vs increase in energy dissipated by caches (including the counter cache used for encryption).

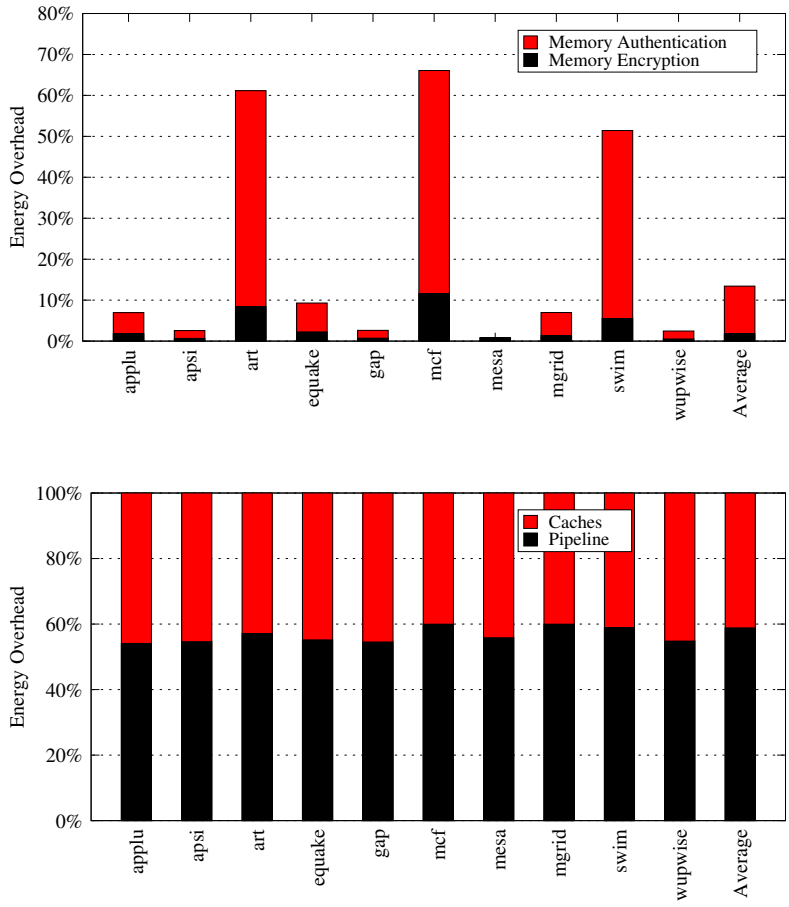

Fig. 6. Energy Overhead breakdown for Counter-mode Encryption with Standard Merkle Trees

As can be seen from Figure 6(a), the memory authentication mechanism is the primary contributor to the overall energy overheads (nearly 78% overhead comes from the authentication mechanism). Figure 6(b) shows that the overall increase in the energy consumption comes in a near equal proportion from both

the pipeline (58%) and on-chip caches (42%). The increase in pipeline energy is mainly the static component, as the processor pipeline does not carry out any of the cryptographic operations to decrypt or verify the integrity of a block, however, the pipeline structures can be idle for a longer duration due to the security mechanisms, thereby dissipating energy (leakage) and contributing to the overall energy increase. The increase in cache energy is mainly the dynamic component coming from increased switching due to data cache accesses to fetch MACs associated with a block to verify its integrity and the additional energy required for the counter cache operation.

Observation: It is important to design power-efficient memory encryption and authentication mechanisms, but a power-efficient authentication mechanism will result in greater savings than a power-efficient encryption mechanism. Mechanisms that reduce the overall delay associated with decryption and integrity verification will help reduce the increase in pipeline energy and mechanisms that reduce cache accesses for security will reduce the cache energy consumption.

As we discussed earlier, Bonsai Merkle trees were proposed to reduce the performance overheads associated with memory authentication mechanisms. BMT integrity verification affords the same security as a standard Merkle tree but requires the tree of MACs to be built only over the per-block counters used for encrypting/decrypting blocks along with a MAC associated with each data block. This results in a much shallower and smaller tree of MACs. The smaller and shallower the tree, the smaller will be the number of accesses to the cache to fetch MACs to verify the integrity of a block. Hence, intuitively, BMTs used with counter mode encryption should reduce the overall energy consumption of the processor. Figure 7 shows the energy overheads of using BMTs with counter-mode encryption.

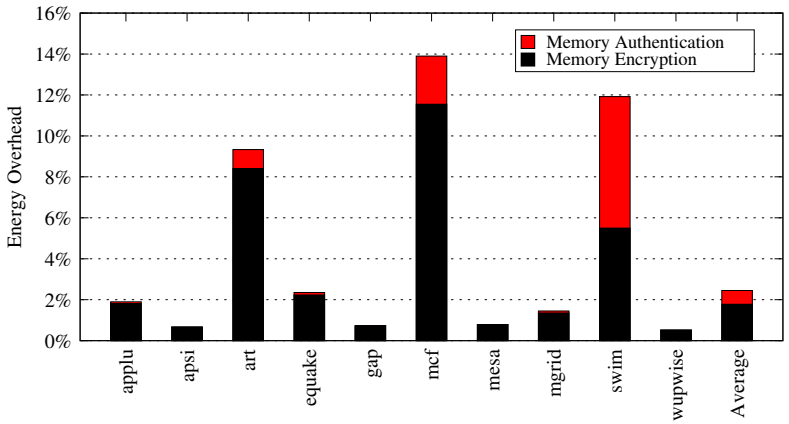

Fig. 7. Energy Overhead of Counter-mode Encryption with Bonsai Merkle Trees

Using BMTs as the integrity verification mechanism reduces the overall energy overheads from 13.42% on an average to 2.42%. In addition, the maximum overheads suffered by memory intensive benchmarks are reduced from 67% to 14%. Another interesting observation that we make is, of the 2.42% overheads, only 0.67% comes from the memory integrity verification mechanism. Hence, using BMTs significantly reduces the overall energy overheads of the secure processor. In addition to reducing the energy overheads, BMTs significantly also reduce the performance overheads of the memory integrity verification mechanism. Figure 8 compares the Energy-Delay product (EDP) for a secure processor using standard Merkle tree vs one using BMTs normalized to the EDP of a system with no protection. As can be seen from the figure BMT integrity verification mechanism has a normalized EDP of 1.04, while standard Merkle Tree integrity verification has a normalized EDP of 1.32. In essence, BMTs are very effective in reducing the overall performance as well as energy overheads of a secure processor with an EDP within 4% of a system with no protection.

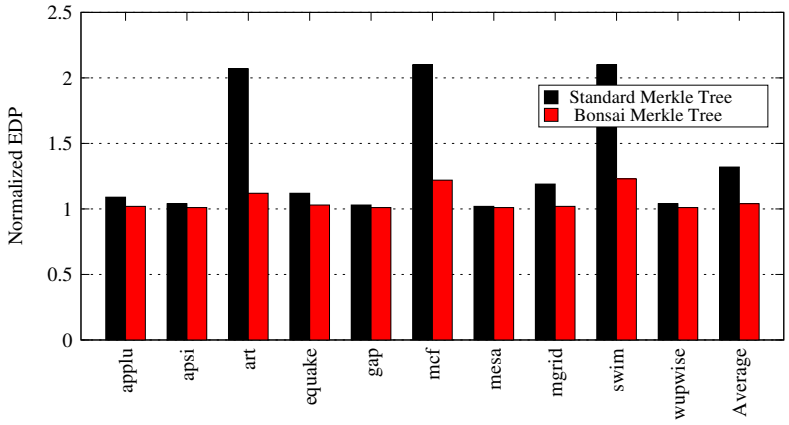

Fig. 8. Normalized Energy-Delay Product (EDP) for Standard vs Bonsai Merkle tree

We further strengthen this observation by showing the energy overheads of secure processor mechanisms on the simulated CMP system. Existing CMP designs [28–30] are typically organized with private L1 caches per core and some combination of shared and private lower-level caches, such as L2 and possibly L3 caches. All cores on the chip typically share a single, common memory bus and off-chip main memory. The memory integrity and verification mechanisms discussed above, can be applied to such CMP architectures in the same manner as a uniprocessor system. For our CMP evaluations, the number of instructions executed for each application in a benchmark pair can change based on the security mechanisms used. For example, using hardware mechanisms for security will make a memory intensive application stall more than a non-memory intensive application. This can be due to two primary reasons. One, for every

block fetched from main memory, a cryptographic pad needs to be generated to decrypt it. This can also result in extra misses (to fetch counters), thereby contenting with the demand fetches for memory bandwidth, further slowing the application down. Secondly, the authentication mechanisms fetch MACs for each block fetched from memory to verify the integrity of the block. These MACs are placed in the last level cache and contend for space with application data, which can further increase the miss rate of the application. As in our simulation infrastructure, the total number of instructions is kept constant, these increased stalls for memory intensive application results in the other application making more progress compared to the base case. Hence, instead of plotting energy numbers directly, we calculate the (*Energy Per Instruction (EPI)* for each individual benchmark and calculate the average EPI for the system as the average of the EPIs of the individual benchmarks. Figure 9 shows the average EPI overhead for using standard Merkle tree with counter-mode encryption.

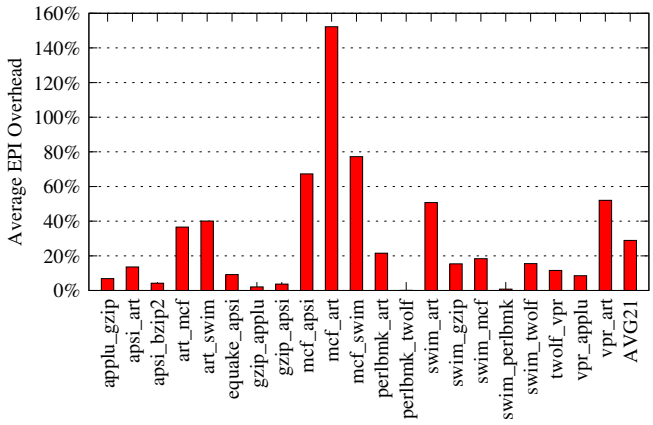

Fig. 9. Average EPI Overhead of Counter-mode Encryption with Standard Merkle Trees

As can be seen from the figure, majority of the pairs suffer from a high average EPI overhead, with the simulated benchmark pairs suffering an average EPI overhead of 29.8% on an average. For memory intensive benchmark pairs, the overheads can be as high as 152% (*mcf_art*). Hence, the energy overheads of using standard Merkle tree with counter-mode encryption are significantly higher for CMPs that for the uniprocessor case. CMPs are likely to be used in server platforms where Energy (Power) consumption is even more important. A recent article [31] pointed out that power could cost more than the servers themselves. Hence, if hardware mechanisms for security are to be adopted for server platforms, it is even more important that their overheads in terms of power are brought down significantly.

Figure 10 shows the average EPI overhead for using BMTs with counter-mode encryption.

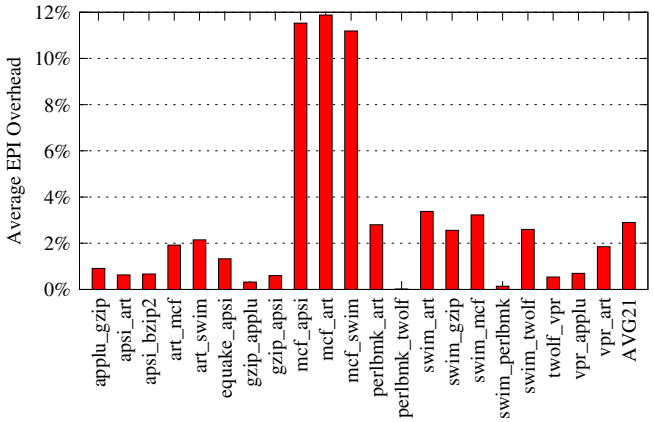

Fig. 10. Average EPI Overhead of Counter-mode Encryption with Bonsai Merkle Trees

As can be seen from the figure, BMTs are even more effective, compared to the uniprocessor case, in reducing energy overheads of CMP systems with the simulated benchmark pair suffering an average EPI overhead of 2.9% on an average with the worst case average EPI overhead declining steeply to 11.8% (compared to 152% with Standard Merkle trees). Hence, if hardware mechanisms for security are to be adopted for server systems utilizing CMPs, it is imperative that they use BMTs as their integrity verification mechanism.

Recommendation: In order to minimize the energy consumption of a secure processor, Bonsai Merkle trees must be used as the memory integrity verification mechanism. BMTs are not only important to reduce the performance overheads, they achieve the best power-performance balance without compromising security.

Now that we have established BMTs as the preferred memory integrity verification mechanism for general purpose processors, we next look at the two most prominently used memory encryption mechanisms, direct and counter mode encryption, and analyze them in greater detail to see if we can achieve further power savings. Figure 11 shows the power overheads of using counter-mode and direct encryption alone, without using a memory authentication mechanism. We observe that counter-mode encryption, even with better latency hiding capabilities suffers 2× higher energy overheads than direct mode encryption. In particular, for memory intensive benchmarks counter-mode encryption results in even higher overheads (*mcf: 12% vs 4%, art: 8% vs 2% and swim: 5.5% vs 2.2%*).

In addition to counter-mode encryption having 2× higher energy overheads compared to direct encryption, counter-mode encryption requires the mem-

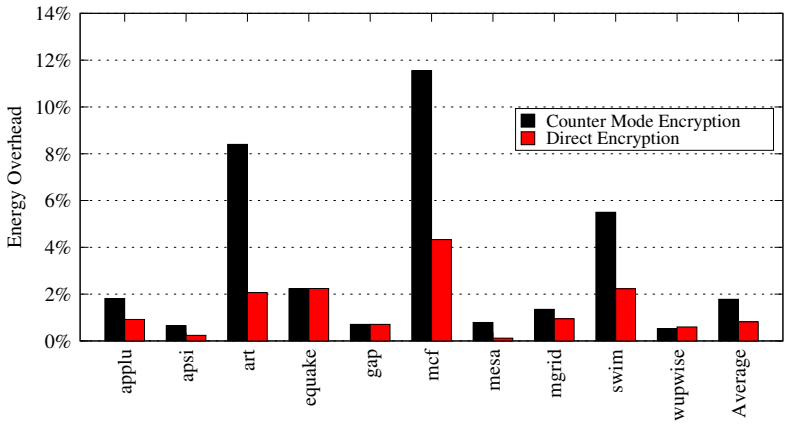

Fig. 11. Energy overhead for Direct vs Counter-mode Encryption

ory authentication mechanism for its security. More specifically, the security of counter mode encryption is contingent on the counters being fresh (unmodified and not replayed). This necessitates an authentication mechanism to be in place for the security of the encryption mechanism. Direct encryption, on the other hand, has no such limitation, it can be used independently of the authentication mechanism. There can be environments which do not have authentication requirements (i.e. active attacks are not possible) but only privacy needs to be guaranteed. For such environments using counter-mode encryption will result in unnecessary overheads, both from the encryption mechanism and the authentication mechanism needed to ensure its security.

The above discussion would suggest using direct encryption in energy constrained environments instead of counter-mode encryption. However, direct encryption as discussed previously lies directly in the critical path of memory fetches and results in non-trivial performance overheads. Figure 12 shows the EDP for counter-mode encryption vs direct encryption. As shown in the figure, despite having lower energy overheads, direct encryption has a higher normalize EDP of 1.12 vs 1.02 compared to counter-mode encryption. Hence, neither counter-mode nor direct encryption offers the best balance in power and performance. This observation is the basis of our design for hybrid cryptographic engine discussed in Section 6.

Recommendation: Use direct encryption for power-constrained environments, where power forms the first design constraint ahead of performance. Direct encryption not only has lower power overheads compared to counter-mode encryption, but also does not need the authentication mechanism for its security, thereby, further reducing the overall power overheads. For environments, where performance and power both are of equal importance, a hybrid cryptographic engine (described in Section 6) should be used.

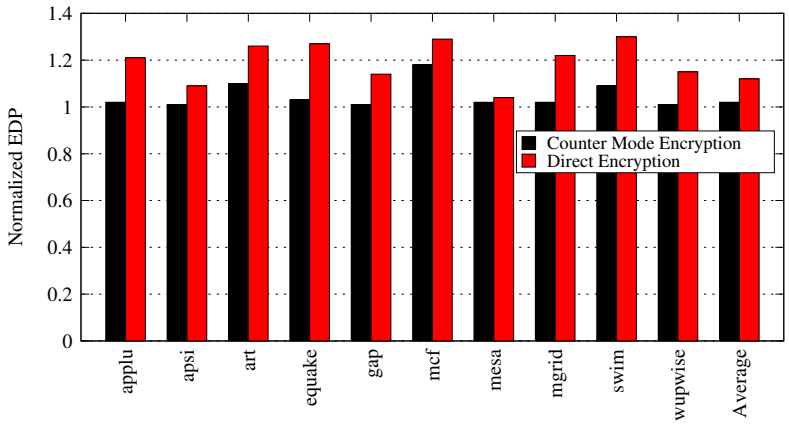

Fig. 12. Normalized Energy-Delay Product (EDP) for Counter-mode vs Direct Encryption

5 Embedded Secure Processors

Embedded devices like mobile phones, PDAs etc. represent a category of devices increasingly used for computation and storage, but they also represent the category of devices that can easily be stolen or lost. This gives the attackers an increased opportunity to get physical access to the system and conduct hardware attacks to extract sensitive information off the system. Hence, hardware mechanisms for security assume even more importance for embedded devices. However, a majority of embedded systems are battery powered and naturally power constrained. In addition, the recent years have seen a significant growth in the computation requirements of embedded devices but the battery capacity has not scaled with the computation needs. Figure 13 shows this trend.

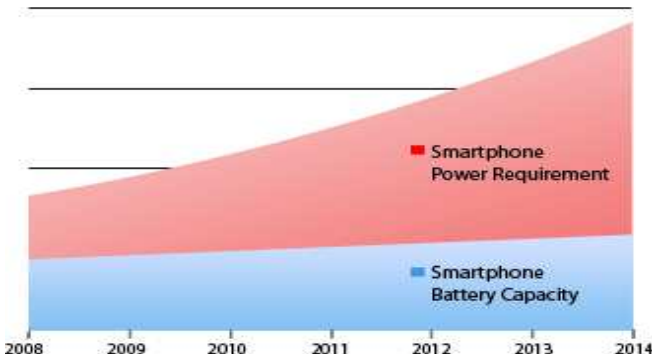

Fig. 13. Power Requirement vs Battery Capacity (Source: Quicklogic [32])

Hence, the energy overheads of providing hardware mechanisms for security assume a greater importance for such systems. In this section, we present our parallel evaluation for embedded environments, however, due to space limitations, we do not present figures for all the results but discuss our main findings.
Memory Authentication: As with general purpose systems, if counter-mode encryption is used with standard merkle tree, it results in much higher overheads (4.57×) compared to counter-mode encryption with BMTs. Figure 14 presents these results. Since, applications running on embedded systems are typically much smaller and much less memory intensive, the absolute energy overheads for provide hardware security mechanisms are much lower than for a general purpose processor. However, at the same time, embedded systems present environments where any energy overheads can pose showstopper issues causing, for example, degradation of battery life etc. Hence, even with small absolute values, it is critically important to minimize these overheads for embedded environments.
Recommendation: Even for embedded systems, BMTs must be used for memory integrity verification, similar to general purpose systems.
Memory Encryption: In terms of encryption mechanism, direct mode encryption in addition to saving energy (similar to general purpose processors), also does not result in any significant performance overheads for applications running on embedded processors. Figure 15 show the EDP for all the benchmarks simulated, normalized to a base system with no protection. As can be seen from the figure, direct encryption closely follows counter-mode encryption, with a worst case degradation of 2% in normalized EDP.

This is primarily due to the fact that embedded benchmarks are not memory intensive and hence suffer very few cache misses. Since encryption overheads are exposed only when blocks are loaded from memory, the low cache miss rate of embedded benchmarks ensures that direct encryption does not result in any significant performance overheads. We observe EDP for direct mode encryption and counter-mode encryption to be within 0.05% for all the embedded benchmarks we simulated.
Recommendation: For embedded systems, the application characteristics make direct encryption as the chosen mode for encryption, offering the best power-performance balance.

6 Power-Efficient Hybrid Cryptographic Engine

In this section, we describe the outline for a power-efficient cryptographic engine design. The design outlined here is based on the observation that direct encryption consumes less power when compared to counter-mode encryption. Also, in addition, direct encryption can avoid the overheads of a memory authentication mechanism for environments which do not need authentication. Counter-mode encryption on the other hand necessitates a memory authentication mechanism for its own security. Without an authentication mechanism that defends against replay attacks, the block counters used by counter-mode encryption to generate seeds (pads) can be replayed, resulting in pad reuse, thereby, breaking the se-

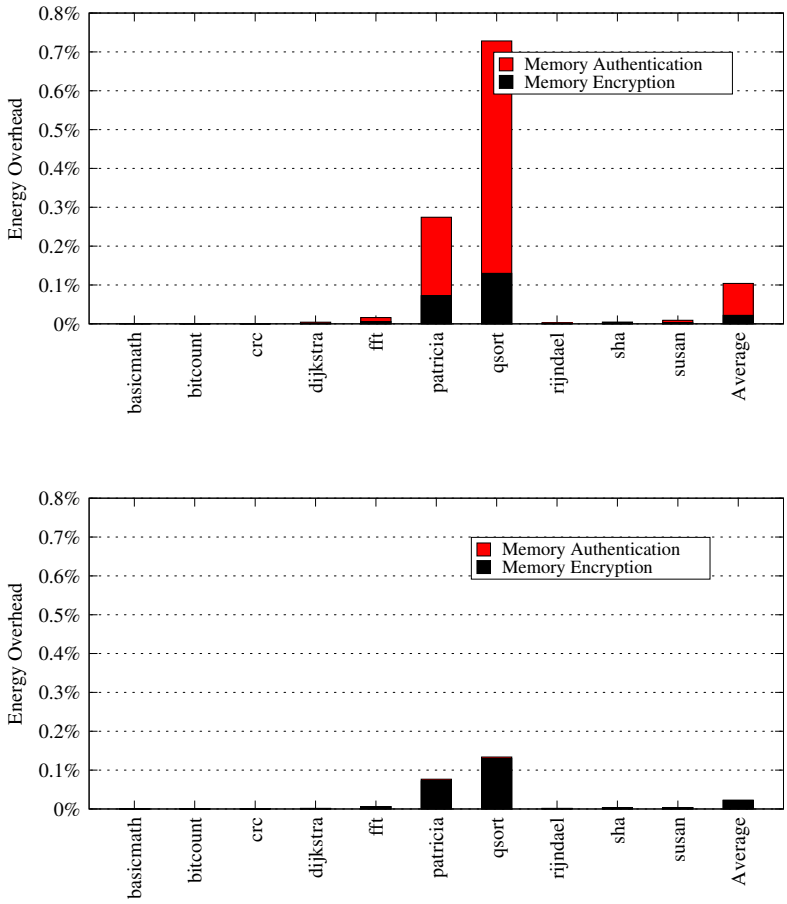

Fig. 14. Energy Overhead breakdown for Counter-mode Encryption with Standard Merkle Trees(a), and Bonsai Merkle Trees (b)

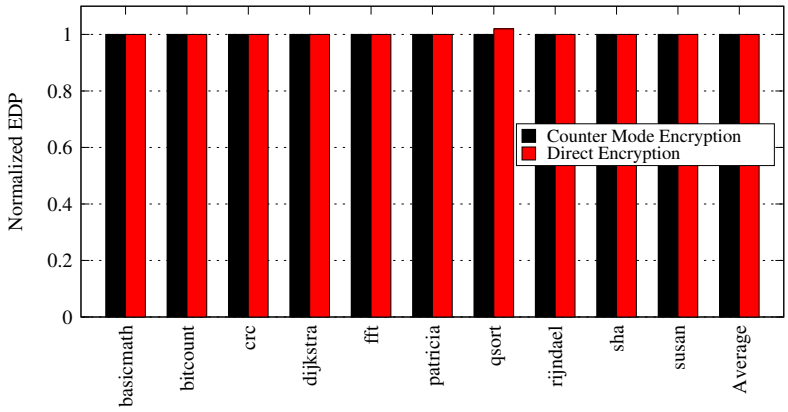

Fig. 15. Normalized Energy-Delay Product (EDP) for Counter-mode vs Direct Encryption

curity of counter-mode encryption. However, despite its power benefits, direct encryption can result in significant performance overheads, particularly for memory intensive applications. However, for applications that are compute intensive and do not miss much in the cache, direct encryption can result in potential power savings. Achieving the best power-performance balance necessitates alternating between the two modes of encryption. For memory-intensive applications (or phases), counter-mode encryption along with the authentication mechanism should be used. For compute-intensive applications (or phases), direct encryption can be used in isolation, if the environment does not need authentication.

Challenges: Designing such a *hybrid cryptographic engine* poses interesting challenges. One, in order to switch between the two encryption modes, on-chip circuitry is needed to establish whether an application (or phase) is memory intensive or not. To this end, one could envisage designing a miss rate monitoring system which keeps track of the number of cache accesses and cache misses, to calculate a running miss rate for the system. However, we observed that the miss rate does not directly correlate to energy overheads, for example, the correlation coefficient for miss rate and the energy overheads of counter-mode encryption is rather low at 0.43, which implies that miss rate cannot directly be used as an indicator to switch the encryption mode for the hybrid cryptographic engine. We observe that Misses-per-Instruction (MPI) correlates extremely well with the energy overheads, with the correlation coefficient between MPI and counter-mode encryption energy overheads being 0.984. Hence, MPI can serve as a good input to the hybrid cryptographic engine. To this end, we introduce a low cost and reliable MPI Monitoring System (MMS). Our MMS requires two counters to keep track of the number of retired instructions, and cache misses [1]. Each counter

[1] Note that these counters might already be available as hardware performance counters, thereby eliminating the need for maintaining additional counters

is 4 bytes in size. Therefore, our MMS requires a total storage overhead of 8 bytes. The hybrid cryptographic engine uses direct encryption by default due to its lower energy overheads. However, to keep the performance overheads low as well, MMS keeps track of the MPI and when the MPI exceeds a certain threshold, X, it switches the cryptographic engine to use counter-mode encryption. The value of X, can be decided based on the characterization of the workloads that are expected to run on the system. The value of X also depends on the primary goal (minimum power overheads or minimum performance overheads) of the system. Figure 16 plots the MPI and the energy overheads of counter-mode encryption and direct encryption for the SPEC2000 benchmark suit. If for example, the *security energy budget* for the system is fixed at 4%, then fixing the MPI at 0.07 would ensure that the energy overheads are always lower than 4% as the engine would always use direct encryption which suffers less than 4% overheads for all the benchmarks. A similar (but more involved) analysis can be done to fix the value of X, if performance is to be factored in as well in the overall goals of the system.

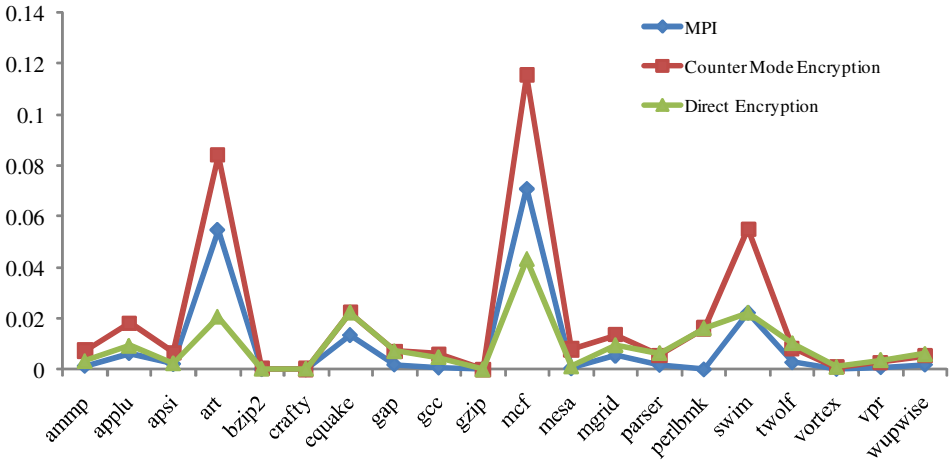

Fig. 16. Determining the value of X

As another challenge, the main memory will now store blocks that have been encrypted using either counter-mode or direct encryption. When a block is loaded from the main memory, it is necessary to identify the mechanism used for its encryption, so it can be decrypted correctly. To this end, we propose tagging the main memory blocks with a single bit, the *counter-mode encrypted bit*. This bit is set when a block is evicted off the processor chip and the cryptographic engine used counter-mode encryption to encrypt it. Our hybrid cryptographic design is show in Figure 17.

Qualitative Analysis of Hybrid Cryptographic Engine: The hybrid cryptographic engine outlined will add complexity in terms of extra on-chip cir-

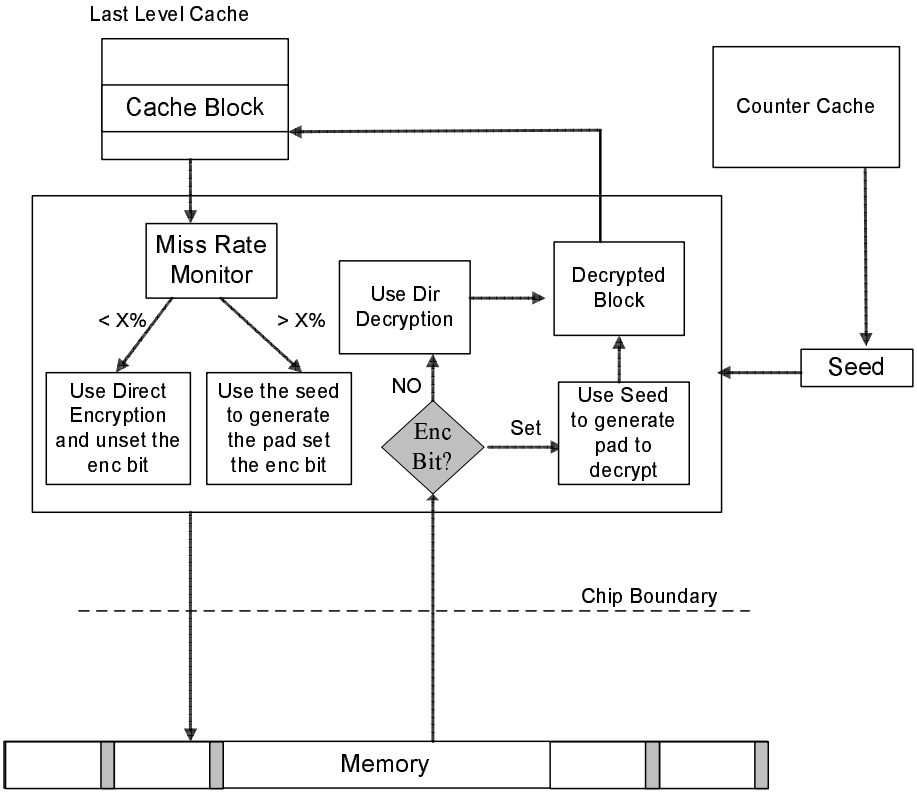

Fig. 17. Power-Efficient Hybrid Cryptographic Engine

cuitry to switch between the two cryptographic modes of encryption. However, the additional circuitry needed to switch between the two modes can be as simple as division circuitry to calculate the current MPI of the system (MMS), using the two 4-byte counters that we introduced, which is then fed to a comparator to determine the mode of encryption to be used. The core of the cryptographic engine, however, needs no modifications. Lets assume that AES is used as the encryption algorithm. For a secure processor substrate using direct encryption, the on-chip cryptographic hardware will consist of an AES engine to which the data blocks are fed for encryption/decryption as they move off and on the processor chip. On the other hand, for a secure processor substrate using counter-mode encryption, the on-chip cryptographic hardware will still consist of an AES engine. However, in this case, instead of feeding the data block directly, the seed associated with the data block is fed to the AES engine to generate a cryptographic pad which is then XORed with the data block to encrypt/decrypt it. Hence, the proposed hybrid cryptographic engine, using both direct and counter-mode encryption, does not add any additional cryptographic hardware. The simple MMS circuit introduced will determine whether to feed the seed associated with the block to generate a cryptographic pad (for counter-mode encryption) or to feed the data block directly (for direct encryption).

We believe that our hybrid cryptographic engine will reap the power benefits of using direct encryption and at the same time ensure that applications do not suffer any major performance penalties by switching to counter-mode encryption, when need be. A quantitative evaluation of the proposed cryptographic engine is left as future work.

7 Conclusion

Secure processor architectures have been proposed to defend against hardware attacks. While previous works have concentrated on resolving the performance, storage and system-level issues of secure processor architectures, power issues have largely been ignored. In this paper, we evaluated the sources of power in currently proposed secure processor mechanisms. We analyzed the power overheads of various hardware security mechanisms for general purpose as well as embedded systems. Finally, we outlined the design of a hybrid cryptographic engine that has been designed with the primary goal of minimizing power overheads, but at the same time ensuring an insignificant loss in performance.

References

1. Huang, A.: Hacking the Xbox: An Introduction to Reverse Engineering. No Starch Press, San Francisco, CA (2003)
2. Kumar, A.: Discovering Passwords in Memory. http://www.infosec-writers.com/text_resources/ (2004)
3. Gassend, B., Suh, G., Clarke, D., Dijk, M., Devadas, S.: Caches and Hash Trees for Efficient Memory Integrity Verification. In: Proc of the 9th International Symposium on High Performance Computer Architecture (HPCA-9). (2003)

4. Gilmont, T., Legat, J.D., Quisquater, J.J.: Enhancing the Security in the Memory Management Unit. In: Proc. of the 25th EuroMicro Conference. (1999)
5. Lie, D., Mitchell, J., Thekkath, C., Horowitz, M.: Specifying and Verifying Hardware for Tamper-Resistant Software. In: IEEE Symposium on Security and Privacy. (2003)
6. Lie, D., Thekkath, C., Mitchell, M., Lincoln, P., Boneh, D., MItchell, J., Horowitz, M.: Architectural Support for Copy and Tamper Resistant Software. In: Proc. of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems. (2000)
7. Rogers, B., Solihin, Y., Prvulovic, M.: Efficient data protection for distributed shared memory multiprocessors. In: International Conference on Parallel Architectures and Compilation Techniques. (2006)
8. Shi, W., Lee, H.H.: Authentication Control Point and Its Implications for Secure Processor Design. In: Proc. of the 39th Annual International Symposium on Microarchitecture. (2006)
9. Shi, W., Lee, H.H., Ghosh, M., Lu, C.: Architectural Support for High Speed Protection of Memory Integrity and Confidentiality in Multiprocessor Systems. In: Proceedings of the International Conference on Parallel Architectures and Compilation Techniques. (September 2004) 123–134
10. Shi, W., Lee, H.H., Ghosh, M., Lu, C., Boldyreva, A.: High Efficiency Counter Mode Security Architecture via Prediction and Precomputation. In: Proceedings of the 32nd International Symposium on Computer Architecture. (June 2005)
11. Shi, W., Lee, H.H., Lu, C., Ghosh, M.: Towards the Issues in Architectural Support for Protection of Software Execution. In: Proceedings of the Workshop on Architectureal Support for Security and Anti-virus. (October 2004) 1–10
12. Suh, G., Clarke, D., Gassend, B., van Dijk, M., Devadas, S.: Efficient Memory Integrity Verification and Encryption for Secure Processor. In: Proc. of the 36th Annual International Symposium on Microarchitecture. (2003)
13. Yan, C., Rogers, B., Englender, D., Solihin, Y., Prvulovic, M.: Improving cost, performance, and security of memory encryption and authentication. In: Proc. of the International Symposium on Computer Architecture. (2006)
14. Yang, J., Zhang, Y., Gao, L.: Fast Secure Processor for Inhibiting Software Piracy and Tampering. In: Proc. of the 36th Annual International Symposium on Microarchitecture. (2003)
15. Zhang, Y., Gao, L., Yang, J., Zhang, X., Gupta, R.: SENSS: Security Enhancement to Symmetric Shared Memory Multiprocessors. In: International Symposium on High-Performance Computer Architecture. (February 2005)
16. Chhabra, S., Rogers, B., Solihin, Y., Prvulovic, M.: Making secure processors os- and performance-friendly. ACM Transactions on Architecture and Code Optimization **5**(4) (2009) 1–35
17. Chhabra, S., Rogers, B., Solihin, Y.: SHIELDSTRAP: Making Secure Processors Truly Secure. In: ICCD. (2009)
18. NIST: Cryptographic hash algorithm competition. http://csrc.nist.gov/groups/ST/hash/sha-3/index.html (2008)
19. Stajano, F., Anderson, R.: The resurrecting duckling: Security issues for ubiquitous computing (supplement to computer magazine). Computer **35** (2002) 22–26
20. Rogers, B., Chhabra, S., Solihin, Y., Prvulovic, M.: Using Address Independent Seed Encryption and Bonsai Merkle Trees to Make Secure Processors OS- and Performance-Friendly. In: Proc. of the 36th Annual International Symposium on Microarchitecture. (2007)

21. J. Renau et al: SESC. http://sesc.sourceforge.net (2004)
22. Brooks, D., Tiwari, V., Martonosi, M.: Wattch: a framework for architectural-level power analysis and optimizations. In: ISCA '00: Proceedings of the 27th annual international symposium on Computer architecture, New York, NY, USA, ACM (2000) 83–94
23. FIPS Publication 197: Specification for the Advanced Encryption Standard (AES). National Institute of Standards and Technology, Federal Information Processing Standards (2001)
24. Kgil, T., Falk, L., Mudge, T.: ChipLock: Support for Secure Microarchitectures. In: Proceedings of the Workshop on Architectural Support for Security and Anti-Virus (WASSA). (October 2004)
25. http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0344c/index.html
26. Standard Performance Evaluation Corporation: http://www.spec.org (2004)
27. http://www.eecs.umich.edu/mibench/: (2001)
28. Barroso, L.A., Gharachorloo, K., McNamara, R., Nowatzyk, A., Qadeer, S., Sano, B., Smith, S., Stets, R., Verghese, B.: Piranha: A scalable architecture based on single-chip multiprocessing. In: Proc. of the 27th International Symposium on Computer Architecture, New York, NY, ACM (2000) 282–293
29. Mowry, T.C., Lam, M.S., Gupta, A.: Design and Evaluation of a Compiler Algorithm for Prefetching. In: 5th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems. (1992)
30. Sinharoy, B., Kalla, R.N., Tendler, J.M., Eickemeyer, R.J., Joyner, J.B.: Power5 system microarchitecture. IBM Journal of Research and Development **49**(4/5) (2005) 505–521
31. CNET: Power could cost more than servers, Google warns. *http* : *//news.cnet.com/Power − could − cost − more − than − servers, −Google − warns/*2100 − 1010_3 − 5988090.*html*
32. Quicklogic: Display Power Optimizer. *http* : *//www.quicklogic.com/display − power − optimizer − dpo − overview/* (2008)