Performance comparison on secure processors using a temporized page encryption technique

Osvaldo Espinosa Sosa, Luis Villa Vargas and Oscar Camacho Nieto

Center for Computing Research CIC-IPN, Av. Juan de Dios Bátiz, esq. con Miguel Othón de Mendizábal, México, D.F., 07738. México
espinosa@cic.ipn.mx , lvilla@cic.ipn.mx, oscarc@cic.ipn.mx
Phone: +(55)57296000 ext. 56519

Abstract. Secure processors have been proposed as the solution for problems in computer security such as hardware attacks, viruses and intruders as well as piracy of software. Researchers have proposed several techniques based on encryption of memory contents where the processor is the only capable entity of decrypt that information before use it. It is evident that encryption and decryption processes increase security levels on computer systems but there is a real penalty on processor performance due to the higher memory access latency. This paper shows the effect on performance of secure processors using temporized techniques for memory page encryption. The main objective is to show performance loses when secure processors are compared with baseline architectures (insecure processors). Performance loses are well justified with the major security level offered with the inclusion of encryption and decryption system. Two cases are evaluated: the first one using Direct Encryption mode and the second one using Counter Mode Encryption.

1 Introduction.

Nowadays, new attacks to computer systems constantly appears and they are produced by malicious software that take advantage of operating systems vulnerabilities and hardware weaknesses in computing systems. Duplication in an illegal way of software (piracy) [1] is also a very important problem; causing millionaire loses to software industry. To reduce these problems, several techniques have appeared at microprocessor level [3]. In these techniques the microprocessor is the only entity authorized to access to information, any other hardware component is considered vulnerable to the attacks due to the fact that anybody can be monitoring the information flowing through the buses [4]. Programs are then stored in memory in an encrypted form and only can be decrypted on-chip, taking into account that the encryption engine is usually placed between level two of cache and main memory.

This paper shows the effect in the performance of a superscalar processor due to the inclusion of an encryption and decryption system including the capacity to encrypt

Computer Engineering.
Research in Computing Science 30, 2007, pp. 97 - 105

pages of main memory at regular intervals of time, changing on every encrypted page the key to be utilized in order to raise the security level. It is clear that this encryption system will add more latency to main memory access and this will affect the overall processor performance. It is important to find a well balanced configuration between memory page size and the period of time used by page encryption in a manner that performance will not be affected severely and it can offer an adequate security level. This work contains a methodology used to evaluate the proposal (temporized page encryption) which uses an execution-driven simulator to obtain detailed statistics of realized experiments, results obtained are included also and finally we have conclusions and bibliographic references.

2 Encryption of memory in secure processors.

On previous work it has been proposed to encrypt data contained in main memory to offer a good security level against attacks [2]. The encryption and decryption system is usually inserted between level two of cache memory and main memory due to it is the place where processor performance is degraded in a minor quantity and because of the fact that levels one and two of cache memory normally reside on chip, it is used also as a bus interface with an insecure external world. When the processor performs a read operation to main memory, the data obtained must be decrypted to be used by the processor; likewise, when a write operation to main memory is performed the data must be encrypted before.

There are two approaches to do this: the first one is called *Direct Encryption Mode* where the encryption/decryption engine is placed serially between main memory and the second cache level. This mode encrypts and decrypts data moving between level two of cache and main memory. This encryption mode has the characteristic of exhibit the whole latency of encryption system and then an access to main memory increments its normal latency adding the encryption engine latency resulting in a higher total latency. The second approach is called *Counter Mode.* Unlike Direct Encryption system it does not need to wait until data arrives from memory (it does not work serially), instead of that, the system encrypts already well-known information at the moment of the access to memory such as memory address and/or the value of a counter, producing a new data called *data pad* which can be used for encryption or decryption. The data pad is calculated in parallel with memory access and the encryption/decryption latency is hidden with the memory access latency. When data read from cache (or ready to be written) and data generated for encryption engine (data pad) are available then both are XOR'ed to produce a new data decrypted to be read (or encrypted to be written). Latencies showed by Direct Encryption and Counter Mode Encryption are depicted on figure 1.

We can notice the minor total latency of Counter Encryption Mode. For encryption/decryption is usually utilized the AES algorithm (Advanced Encryption Standard) which is an algorithm of fixed parameters. The AES requires as input a data

block of 16 bytes. In order to encrypt a cache memory line of 64 bytes, four AES blocks are required as shown in figure 2.

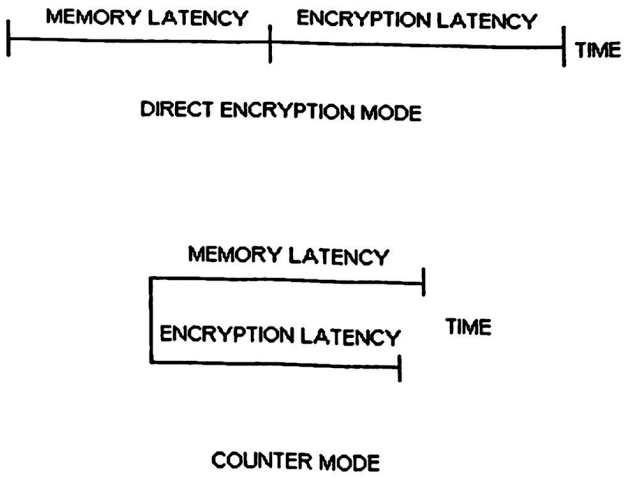

Fig. 1 Two approaches for encryption/decryption

The encryption or decryption process can use always the same key, which represents vulnerability due to the possibility that the algorithm can be broken by an expert intruder.

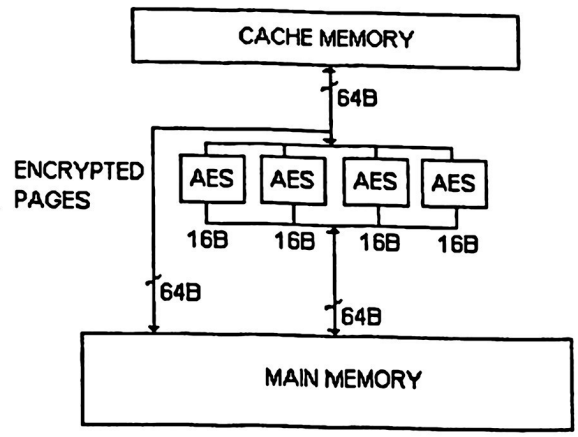

Fig. 2 Encryption and decryption circuit detail.

Other option is to change the key after certain number of encryptions and decryptions, however when the key is replaced the main memory must be re-encrypted causing that system could be stopped a large interval of time (in the order of seconds) in a system working at frequencies in the order of Ghz. To improve security in a computer system without important performance degradation we propose a system where periodically keys are replaced using a pull of keys, even more, we can use different keys for each memory page, as we are going to explain in the next section.

3 Proposed architecture.

Our encryption system is shown in figure 3. The main memory is divided in pages of fixed size (e.g. 4KB). There is a *timer* that activates the mechanism for key replacing periodically at regular intervals of time. Keys are generated in a random form. Every time this mechanism is activated, a memory page is decrypted using the old key and re-encrypted using the new key. Afterwards, the encrypted information is sent back to main memory.

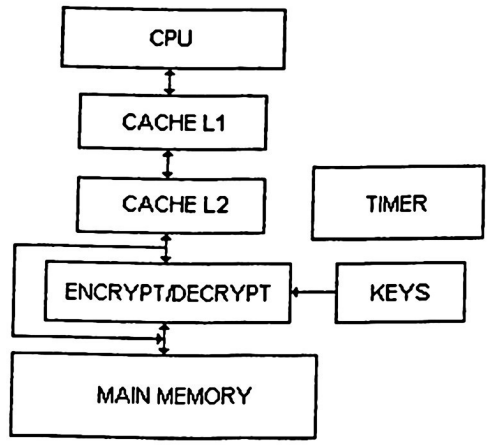

Fig. 3 Proposed architecture.

At the end of the period, a new page of main memory will be selected for re-encryption following a Round Robin scheme. The new encryption process will use a new key (and a new one for every page) to offer a higher level of security in comparison with proposals which use only one key for whole memory.

Using this new page encryption model the security level increases without important performance degradation. In order to know what memory pages are active on memory, the processor has a special group of page registers working together with operating system which is the responsible of resource management (memory in this case). The processor includes a set of special registers to store the old key for decryption (it comes from main memory) and other one to store the new key that will

be used for page re-encryption before are sent back to main memory. It is important to notice the following fact: when a page is re-encrypted, information pass through the decryption circuit and data is sent back directly to main memory without affecting the cache contents.

4 Methodology.

We have evaluated our models using the simplescalar 3.0 simulator tool set which performs a very detailed simulation of a superescalar processor with out of order execution [5]. The simulator was configured as an Alpha 21264 because this architecture has been considered the best superscalar processor at its time of appearance. This processor contains two level 1 cache memories (Instructions and data) of 64 KB 2-way associative with 64 byte blocks. The second level of cache is unified with size of 1 MB being 8-way associative with 64 byte block. The SPEC CPU 2000 was used as a benchmark set, which is composed of twelve applications fixed point and fourteen floating point programs. In this work the performance is monitored whenever we change the page size or the period of time for page re-encryptions. Five hundred millions instructions were simulated for each program skipping the first 1×10^8 instructions with the aim of eliminate initialization effects on statistics. Results are shown in terms of IPC average for the 26 SPEC CPU programs.

5 Evaluation.

As we can see on figure 4, our reference is the bar labelled as *baseline*, which corresponds to an insecure processor and corresponds to the maximum performance attainable. We compare the *baseline* with Direct Encryption Mode (XOM) using pages of 4 KB, encrypting every 1,000,000 cycles and 100,000 cycles (XOM+1e6 and XOM+1e5 respectively). We can notice that performance is diminished significantly with the inclusion of encryption engine. Direct Encryption reduces performance to 90,84% in average respect to the baseline but including page encryption every 1,000,000 cycles performance degradation is minimal (descends to 90.23%) . If page encryption is now realized every 100,000 cycles the performance degradation is more important (84.73%). It is clear that increasing security (reducing re-encryption's period) performance will be reduced in a major form. Figure 5 shows how the page size impacts on performance. We consider pages of 4 KB, 8 KB and 16 KB (labelled XOM+4K, XOM+8K and XOM+16K respectively) and there is a comparison with baseline and Direct Encryption Mode (XOM). It is evident that increasing page size, performance decreases. In this case, page re-encryptions take place every 100,000 cycles. With pages of 4 KB the performance is 90.23% respect to the baseline and 85.62% for pages of 8 KB. The performance is even worse with 16 KB pages, where it is reduced until 69.84%. Figure 6 shows results with a period of 1,000,000 cycles between page re-encryptions. We can see that with a higher period the latency of encryption system is better hidden. As it is depicted, page size could be bigger and the system is less sensitive to the latency inserted by the encryption engine, having

performances of 90.65% on average respect to the baseline with 8 KB pages and 88.04% with 16 KB pages.

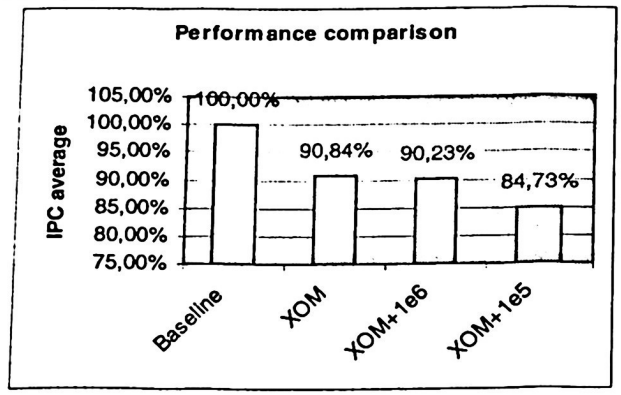

Fig. 4 Direct Encryption Mode with 4 KB pages and different periods.

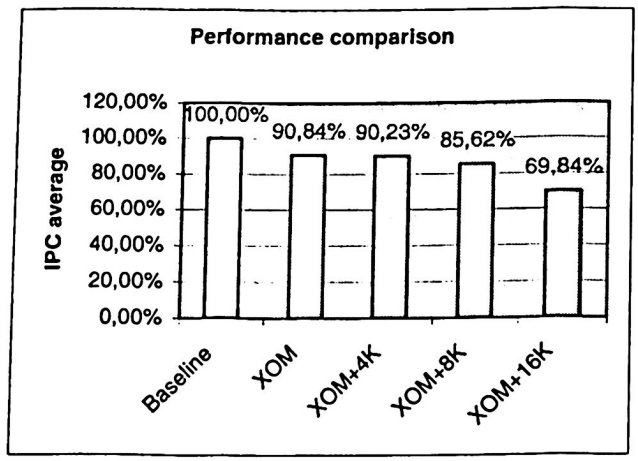

Fig. 5 Direct Encryption Mode with page encryption every 100,000 cycles.

On the other hand, figure 7 shows results for the Counter Mode Encryption system, the insecure processor is labelled baseline and represents the maximum performance attainable by the system. The Counter Mode only encryption system corresponds to the bar CM, and when we include page encryption every 1,000,000 and 100,000 cycles (labelled CM+1e6 and CM+1e5 respectively). Is evident that performance loses are lower than Direct Encryption Mode, in fact performance reductions for CM+1e5 is less than 5%. Figure 8 depicts the case of page re-encryption every 100,000 cycles, showing that we can select page sizes of 8 KB having the performance loss to approximately 10%, better than Direct Encryption case.

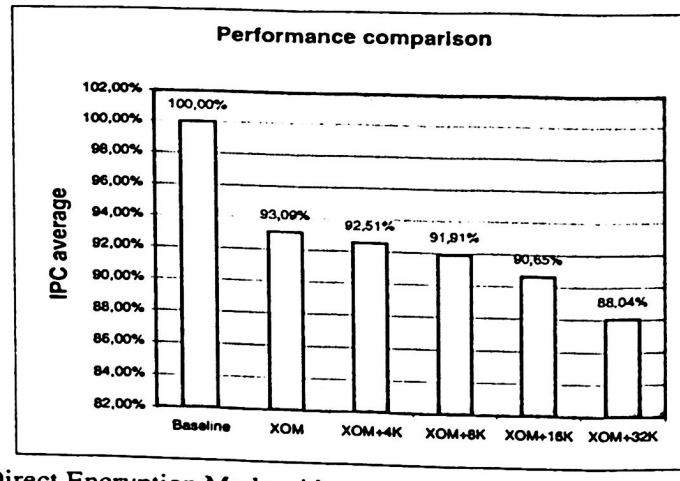

Fig. 6 Direct Encryption Mode with page encryption every 1,000,000 cycles.

Figure 9 corresponds to the case of page encryption every 1,000,000 cycles. Reducing the period of time between re-encryptions the system is less sensitive to an encryption process. We can notice that page size of 32 KB represents a performance loss of less than 5%.

Comparison of performance loses give us an idea that Counter mode is better in terms of performance instead of Direct Encryption which have better levels of security (by the fact that Counter Mode exhibits part of information through the buses for example).

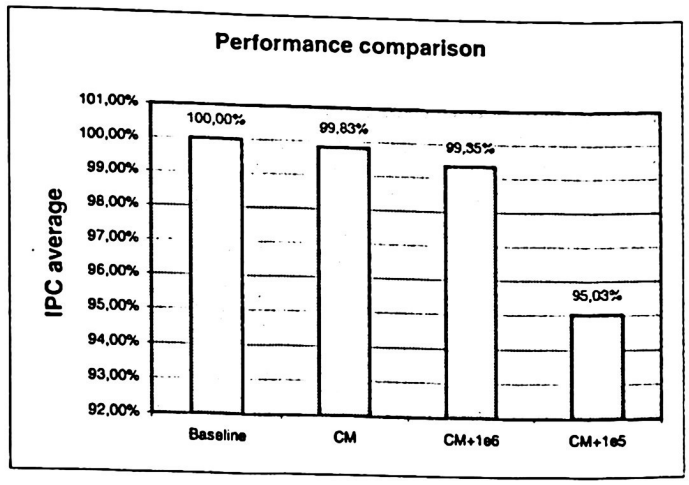

Fig. 7 Counter Mode Encryption with 4 KB pages and different periods.

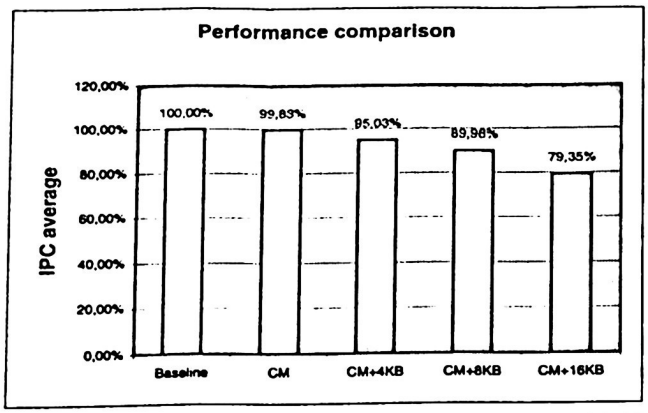

Fig. 8 Counter Mode Encryption with page encryption every 100,000 cycles.

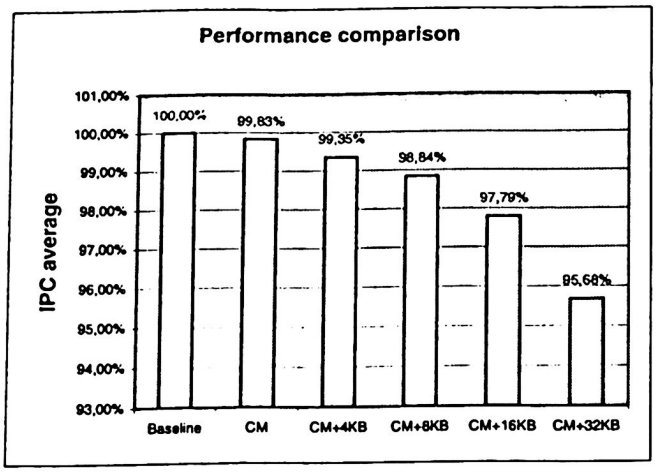

Fig. 9 Counter Mode Encryption with page encryption every 1,000,000 cycles.

6 Conclusions.

As it is studied in this work, to include an encryption system reduces processor performance in an important manner, if additionally we insert the temporized page encryption technique we are adding additional loss. These performance loses are well justified in terms of increased security. We can eliminate 10% of performance in order to gain a higher level of security. Evidently, choosing an adequate period for page re-encryption and the page size could increase even more the security of the system. Direct Encryption is better in terms of security but Counter mode is better for system performance.

7 References.

1. Yang,Zhang,Gao. Fast secure processor for inhibiting software piracy and tampering. Proceedings of the 36th International Symposium on Microarchitecture MICRO 36-2003.

2. Ruby B. Lee, Peter C. S. Kwan. Architecture for protecting critical secrets in microprocessors. Proceedings of the 32nd Annual International Symposium on Computer Architecture 2005.

3. T.Kgil, L.Falk and T. Mudge. ChipLock: support for secure microarchitecures. Workshop on architectural support for security and anti-virus, 2004.

4. Chenyu Yan, Brian Rogers et. Al. Improving cost, performance and security of memory encryption and authentication. International Symposium on computer architecture ISCA 2006.

5. Burger,Dough. The simplescalar toolset, version 3.0 *Computer Architecture News*, 25 (3), pp. 13-25, June, 1997.