A Parallelized Way to Provide Data Encryption and Integrity Checking on a Processor-Memory Bus

Reouven Elbaz[¶‡], Lionel Torres[¶], Gilles Sassatelli[¶]
Pierre Guillemin[‡], Michel Bardouillet[‡], Albert Martinez[‡]
[¶]LIRMM UMR University of Montpellier 2- CNRS C5506, Montpellier, FRANCE
[‡]STMicroelectronics, Advanced System Technology, Rousset, FRANCE
[¶]{surname}@lirmm.fr - [‡]{firstname.surname}@st.com

ABSTRACT

This paper describes a novel engine, called PE-ICE (Parallelized Encryption and Integrity Checking Engine), enabling to guarantee confidentiality and integrity of data exchanged between a SoC (System on Chip) and its external memory. The PE-ICE approach is based on an existing block-encryption algorithm to which the integrity checking capability is added. Simulation results show that the performance overhead of PE-ICE remains low (below 4%) compared to block-encryption-only systems (which provide data confidentiality only).

Categories and Subject Descriptors

E.3 [**Data**]: Data Encryption; C.5.3 [**Computer Systems Organization**]: Computer System Implementation - *Microcomputers, Portable devices*;

General Terms

Security, Performance, Design, Algorithms, Verification.

Keywords

Data Confidentiality and Integrity, Architectures, Bus Encryption.

1. INTRODUCTION

Today's embedded computing systems are considered as untrusted hosts because the owner, or anyone else who manages to get access, is a potential adversary. The bus between the System on Chip (SoC) and the external memory is one of the weakest points because external memories contain sensitive data (end users private data, software code…) which are usually exchanged in clear form over the bus. Therefore an adversary may probe this bus in order to read private data or retrieve software code (data privacy concern). Another possible attack relies on code injection (data integrity concern). A relevant example combining those two kinds of attacks is the one performs by Markus Kuhn [1], which consists of modifying the behavior of a software execution to obtain the plaintext version of the ciphered data stored in the off-chip memory.
Hardware protection must be provided to prevent such attacks and to guarantee a Private and authenticated Tamper Resistant (PTR) environment for application execution. This implies ensuring data confidentiality and integrity verification. In our context, the confidentiality security goal is to prevent the leakage of any useful information from off-chip, whereas the integrity security goal is to forbid execution of intentionally altered memory content.

In the specific processor/memory context, most of proposed engines in the literature [2] [3] [4] use one dedicated hardware-core (algorithm) per security property along with its own key. The direct consequence of such an approach is the serialization of both processes introducing an additional performance overhead.

This paper describes a Parallelized data Encryption and Integrity Checking Engine (PE-ICE) dedicated to processor-memory bus transaction which provide full parallelization of both processes on all kinds of read and write operations. The key aspect of this engine is the adding of the integrity checking capability to an existing block encryption algorithm.

This paper is organized as follows. Section 2 presents the considered threat model. Section 3 exposes prerequisites and gives an overview of techniques and domain-dedicated solutions proposed in the literature to provide data confidentiality and integrity. Section 4 focuses on the theoretical aspects allowing integrity checking within PE-ICE and presents an example of implementation. Simulation results are given in section 5. Finally, perspectives and improvements are proposed in section 6.

2. THREAT MODEL

The System on Chip (SoC) is considered as trusted. Only board level attacks with processor-memory bus probing are taken into account (Figure 1). We are particularly concerned by the "Man in the middle" attacks. The corresponding protocol consists of observing processor memory communication, intercepting the data on the bus (data confidentiality issue), and sending to the processor or storing in memory a chosen text (data integrity issue) – called in the following "fake" text. The choice of this "fake" by the attacker leads us to define three kinds of attack that the integrity checking engine must detect:

- *Spoofing attacks*: The adversary exchanges ciphertext transmitted on the bus with a random value. The attacker mainly alters program behavior but cannot foresee the results of his attack.
- *Splicing or relocation attacks*: the attacker swaps a memory block transmitted on the bus by another one previously recorded in the external memory. Such an attack may be viewed as a spatial permutation of memory blocks. When data are ciphered, the advantage of using a memory block copy as a fake is the knowledge of its behavior if this one was previously observed.
- *Replay attacks*: As with splicing attacks, an attacker records on the bus a ciphertext at a specified address. Then this recorded value may be reuse later on at the same address (current data value replaced by an older one). Such an attack may be viewed as a temporal permutation of memory blocks at a specific address location.

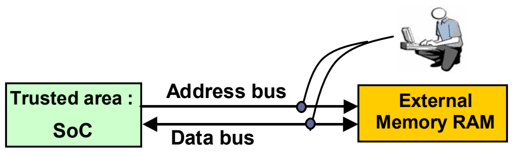


Figure 1. Board level attack with bus probing

3. STATE OF THE ART

3.1 Prerequisites

Security mechanisms localization: Hardware mechanisms guaranteeing data confidentiality and integrity are usually inserted between the cache memory and the memory controller to i) prevent any CPU (Central Processing Unit) modification which could lead to performance degradations ii) remain on the trusted area (SoC). For the same security reason, secret values like encryption keys, needed for those mechanisms are stored on-chip.

Confidentiality is usually ensured by using symmetric encryption algorithms, which are divided in two families: stream cipher algorithms – encryption is done bit per bit - and block cipher algorithms - the plaintext is first split into blocks, and then each block is ciphered separately. For more details on encryption techniques refer to [7].

Data integrity is guaranteed by checking that read data has not been tampered with during external storage or transmission over the bus. In the following, the term "chunk" is used to define the size of the atomic memory block loaded from the off-chip memory, decrypted and verified by the integrity checking engine on read operations. As Load/Store processor architectures fetch memory blocks of a cache line size on a cache miss, a chunk is usually of this size.

To fulfill the integrity checking objective, a value is appended to each chunk stored in the external memory. This value called tag is computed on-chip with MAC (Message Authentication Code) algorithms [7] on write operations. Such algorithms based on hash functions or on symmetric primitives [7] accept as inputs the chunk data and a key. Theoretically they create a unique Chunk/Tag pair. Only the SoC is capable to compute the tag as the secret key is stored on-chip. On read operations, the integrity of the loaded chunk is checked: the tag is recomputed on the loaded data and compared to the one retrieved from off-chip memory along with the data. If the tag matching process fails, an integrity checking flag informs the CPU which in turn adopts an adequate behavior (for instance a HALT instruction to stop processor execution).

3.2 Data Confidentiality and Integrity

The conventional way to satisfy the confidentiality-and-integrity property is to make two passes on data. A first pass is dedicated to encryption and a second one is done to compute a tag with a MAC algorithm. Three different schemes defined in [6] are depicted in Figure 2. Encrypt-then-MAC (Figure 2a) encrypts the plaintext to get a ciphertext C, and then appends to C a tag computed with a MAC algorithm over C. MAC-then-Encrypt (Figure 2b) calculates a tag over the plaintext P, appends the resulting tag to P and then encrypts them together. The third scheme, Encrypt-and-MAC (Figure 2c) encrypts the plaintext P to get a ciphertext C and append to C a tag computed over P. The main drawback of such techniques is the non-parallelizable property of the two security mechanisms (encryption and MAC computation) on read or on write operations or on both. On write operations, for the Encrypt-then-MAC scheme, the tag computation starts only at the end of the encryption process while for the MAC-then-Encrypt, encryption starts after the completion of the tag calculation. Concerning read operations, for Encrypt-and-MAC and MAC-then-Encrypt schemes, the tag matching process begins only when the decryption process is completed.

Generic composition: In the specific processor/memory context, proposed engines [2][3][4] are based on one of the conventional way schemes (two passes on the data) to satisfy both integrity and confidentiality properties. Such engines belong to the "generic composition" [6] family as they use a different key for each algorithm. They generally use Encrypt-then-MAC approach. The cost to achieve data confidentiality-and-integrity is then the cost to encrypt plus the cost of MAC computation on write operations.

In order to thwart replay attacks hash or Merckle trees [8] are proposed in [2][3][9]. Such a technique implies a huge performance loss due to multiple hash computation and is unaffordable in a computing system.

Authenticated encryption scheme is a shared-key transform whose goal is to provide both privacy and authenticity [6]. Several AE modes have been proposed to the NIST (National Institute of Standard and Technology) standardization process [5]. However, AE does not mean one pass on data; all AE modes are based on one of the conventional way schemes and still require to chain two computations on read or on write operations. The most relevant AE modes for our application domain seem to be OCB (Offset CodeBook) and GCM (Galois Counter Mode) [5] in term of computation performance. Unfortunately, no implementation or evaluation exists of such modes for our application domain.

4. PE-ICE: Parallelized Encryption and Integrity Checking Engine

PE-ICE is a dedicated solution providing strong encryption and integrity checking to data transferred on the processor-memory bus of an embedded computing system. It has been designed in order to optimize latencies introduced by the hardware security mechanisms on read and write operations. In order to achieve this goal, PE-ICE uses a single block-encryption algorithm. Data confidentiality is thus guaranteed by encryption while integrity checking relies on the spreading feature of block-encryption algorithms and on resources available on-chip.

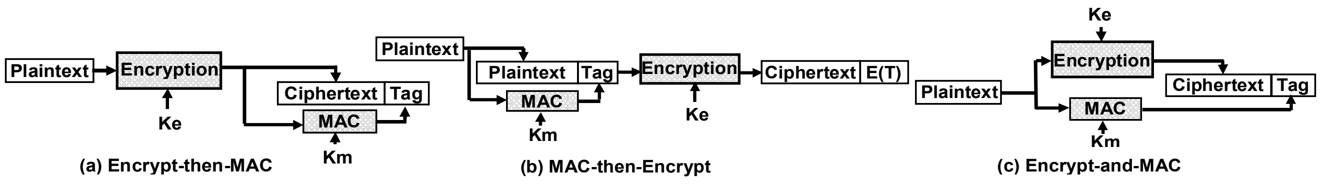


Figure 2. The conventional way to provide the confidentiality-and-integrity property to data

4.1 Confidentiality

Data privacy is provided by AES [10] (Advanced Encryption Standard) encryption. AES is a block cipher algorithm processing 128-bit blocks with a 128-bit key.

4.2 How to Add the Integrity Checking Capability to Block Encryption

The spreading feature of block encryption algorithms: Once a block encryption is performed, the position and the value of all bits in a ciphertext block C are influenced by each bit of the corresponding plaintext block P. Consequently if P is composed of two distinct data (P_L and T), after ciphering, it is impossible to distinguish the P_L ciphered part from the T one in C. Moreover if one bit is modified in C, after decryption all bits of the corresponding plaintext block will be impacted

PE-ICE integrity checking process: The previous property is used to add the integrity checking capability to block-encryption. On a write operation (Figure 3a), a "tag" value is inserted in each plaintext block P before encryption. As a result, P is composed of a payload P_L (data to protect) and a tag (T). Such a tag must be a nonce, a Number used ONCE, for a given encryption key and does not need to be calculated over the data with a specific algorithm; it may be for example generated by a counter. After encryption, an indistinguishable and unique pair P_L/T is created and the resulting ciphered block C is written in the external memory. On a read operation (Figure 3b) C is loaded and decrypted. The tag T issued from the resulting plaintext block is compared to an on-chip regenerated tag called the tag reference T'. If T does not match T', it means that at least one bit of C has been modified (spoofing attack), PE-ICE raises an integrity checking flag to prevent further processing.

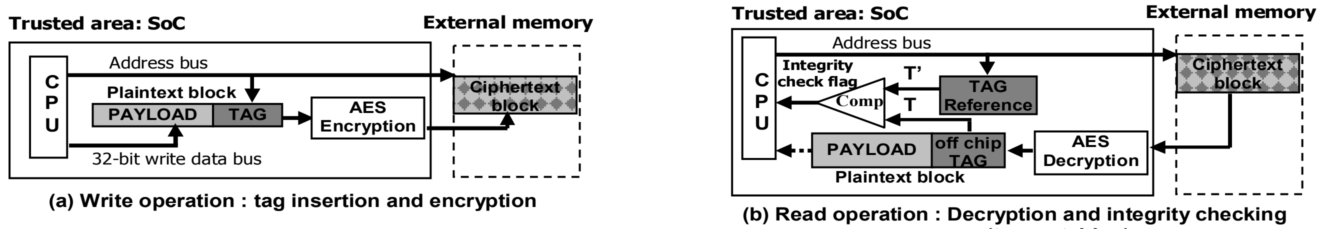
The tag generation: In the processor-memory communication context, the SoC is alone achieving both the encryption and the decryption process. Therefore, the SoC has to hold the tag value T of each ciphered block between the encryption and the decryption or must be able to regenerate it on read operations to perform the integrity checking process. The challenge is to reach this objective by storing as less as possible tag information on the SoC to optimize on-chip memory usage. Moreover T may be public because an adversary needs the secret encryption key to create an accepted P_L/T pair.

CPUs process two kinds of data, Read Only (RO) data and Read/Write data. The composition of the tag is different for each kind of data and depends of their respective properties. RO data are only written once in external memory and are not modified during program execution. In addition, the secret encryption key is changed for each application downloaded in the external memory. Therefore, the tag can be fixed for each plaintext block of RO data. PE-ICE uses the most significant bits of the ciphered block address as tag (Figure 4a). If an attacker performs a splicing attack, the address used by the processor to fetch a block and by PE-ICE to generate the tag reference (T') will not match the one loaded as tag (T). RW data are modified during software execution and are consequently sensitive to replay attacks. Using only the address as tag is not enough to prevent such attacks because the processor cannot verify that the data stored at a given address is the most recent one (temporal permutation). For that reason the tag is composed of the most significant bits of the address of each ciphered block, concatenated with a random value (Figure 4b). The random value of each plaintext block is changed on each write operation and is stored on-chip to be able to re-generate the tag reference on read operations.

4.3 Example of Implementation

The targeted processor for PE-ICE implementation is a Load / Store RISC (Reduced Instruction Set Computer) architecture, meaning that on a cache miss, the CPU loads a payload of a cache line size. PE-ICE has been designed to encrypt and to check integrity of such a payload. In the following PE-ICE configuration is given considering 256-bit cache lines.

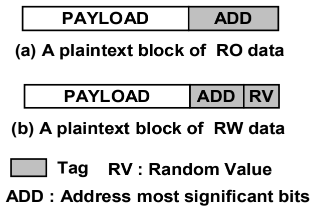
The memory block M (payload + tags) loaded by the processor on a cache miss must be a multiple of the ciphered-block length. As a consequence, the minimum size of all tags needed for a payload of a whole 256-bit cache line is one AES block (128-bit). The resulting memory block M to load is three AES blocks long (384-bit) and therefore requires three tags. The tags distribution among plaintext blocks composing M is depicted in Figure 5. Tags storage take up to 33% of the external memory.



(a) Write operation : tag insertion and encryption

(b) Read operation : Decryption and integrity checking process (tag matching)

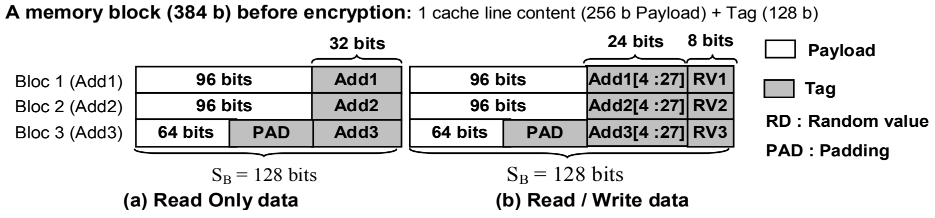
Figure 3. PE-ICE principle



(a) A plaintext block of RO data

(b) A plaintext block of RW data

Figure 4. Plaintext blocks and tags composition



(a) Read Only data

(b) Read / Write data

Figure 5. Tag distribution over a memory block (for a cache line of 256-bit)

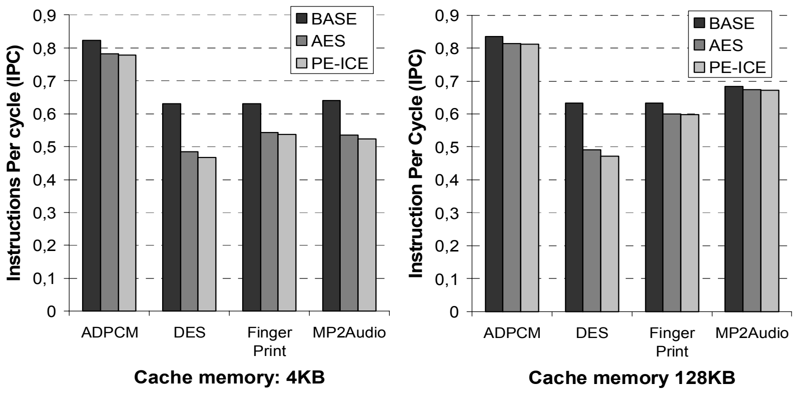


Figure 6. Performance overhead of AES Encryption and PE-ICE for two different cache sizes with cache lines of 256-bit

Table 1. Memory access penalties of PE-ICE and AES encryption depending on the kind of operation performed

Operation	Latencies introduced on external memory access (cycles)	
	AES	PE-ICE
8 to 32-bit Write	36	37
8 to 32-bit Read	15	15
4-word Write	15	38
4-word Read	15	17
8-word Write	15	18
8-word Read	15	17

Concerning RO data, the 32-bit address of each ciphered block address is used as tag (Figure 5a). Such a configuration protects up to 4GB of external memory space dedicated to RO data against relocation (splicing) attacks. For RW data, the tag is composed of (at minimum) the 24 most significant bits of the ciphered block address plus an 8-bit random value (Figure 5b). Such tag protects up to 256 MB (minimum 24 addr. bit addressing 128-bit blocks ⇔ 2^{28}) of external memory space dedicated to RW data against relocation attacks. Moreover an adversary has a $1/2^8$ chance to succeed with a replay attack. This implies to have 96 KB of on-chip memory for storing the random value. In addition, for both kinds of data the tag is minimum 32-bit long, hence an adversary has a $1/2^{32}$ chance to succeed with a spoofing attack.

5. EVALUATION

The simulation framework is based on the SoC Designer (MaxSim technology) toolset [11]. The processor core modeled is the ARM926EJ-S.

PE-ICE as AES encryption introduces a performance loss only on external memory accesses i.e. on read and write cache misses and on operations on non-cacheable data. Penalties of such operations on an ARM AMBA-AHB bus for PE-ICE and for AES were obtained by RTL VHDL simulations and are given in Table1. They are used to define all external memory latencies depending on the kind of access.

Four benchmarks used in embedded systems serve to perform a first evaluation of PE-ICE: ADPCM (Adaptive Differential Pulse Code Modulation), DES (Data Encryption Standard), FingerPrint (Fingerprint skeletization) and MP2Audio (MPEG2 audio encoder). Figure 6 depicts simulation results for two different sizes of cache memory (4KB and 128KB). Increase the cache memory size could decrease performance overhead, however if external memory accesses are mainly due to operations on non-cacheable data, performance overhead remain almost unchanged (e.g. DES). Figure 6 shows that PE-ICE introduces a performance degradation of 4% in the worst case (DES) when compared to AES encryption alone (without integrity checking).

6. CONCLUSION AND PERSPECTIVES

PE-ICE provides a parallelized way to encrypt and check integrity of data stored in off-chip memories of embedded computing systems. Data encryption and integrity checking are mandatory to obtain a private and authenticated tamper resistant (PTR) environment for software execution.

For commercial devices using block cipher algorithms for encryption, the PE-ICE implementation seems to be more realistic than Merckle trees to provide integrity checking in addition to confidentiality. While the global performance overhead of CHTree [3] (Cached Hash Tree), is between 20 and 50%, first simulation results show that the PE-ICE integrity checking mechanisms introduce less than 4% of performance degradation.

On-going work includes the implementation of PE-ICE and its validation on the LEON2 processor [12]. One improvement objective is to decrease the internal memory usage due to random value storage.

7. REFERENCES

[1] M. G. Kuhn: Cipher Instruction Search Attack on the Bus-Encryption Security Microcontroller DS5002FP, IEEE Trans. Comput., vol. 47, pp. 1153–1157, Oct. 1998.

[2] G. E. Suh, D. Clarke, B. Gassend, M. van Dijk, and S. Devadas, AEGIS: Architecture for Tamper-Evident and Tamper-Resistant Processing, in Proc. Intl Conf. Supercomputing (ICS ’03), pp. 160–171, June 2003.

[3] G. E. Suh and al, Efficient Memory Integrity Verification and Encryption for Secure Processors, 36th IEEE/ACM Intl. Symposium on Microarchitecture, p.339, 2003.

[4] D. Lie, C. Thekkath, M. Mitchell, P. Lincoln, D. Boneh, J. Mitchell, and M. Horowitz. Architectural Support for Copy and Tamper Resistant Software. In Proceedings of the 9th Intl Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-IX), pages 169-177, Nov 2000.

[5] http://csrc.nist.gov/CryptoToolkit/modes/proposedmodes/

[6] M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic construction paradigm. In T. Okamoto, editor, Asiacrypt 2000, volume 1976 of LNCS, p. 531-545. Springer-Verlag, Berlin Germany, Dec. 2000.

[7] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. Handbook of Applied Cryptography. CRC Press, 1996.

[8] R. C. Merkle. Protocols for public key cryptography. In IEEE Symp. on Security and Privacy, pages 122–134, 1980.

[9] B. Gassend, G. E. Suh, D. Clarke, M. v. Dijk, and S. Devadas, Caches and Hash Trees for Efficient Memory Integrity Verification, in Proc. of the 9th Intl Symposium on High Performance Computer Architecture (HPCA9), February 2003.

[10] N. I. of Science and Technology. FIPS PUB 197: Advanced Encryption Standard (AES), November 2001.

[11] http://www.arm.com/products/DevTools/MaxSim.html

[12] http://www.gaisler.com/