A Security Approach for Off-chip Memory in Embedded Microprocessor Systems

Romain Vaslin*, Guy Gogniat*, Jean-Philippe Diguet*,
Eduardo Wanderley**, Russell Tessier***, Wayne Burleson***

Abstract

This paper describes a complete off-chip memory security solution for embedded systems. Our security core is based on a one-time pad (OTP) encryption circuit and a CRC-based integrity checking module. These modules safeguard external memory used by embedded processors against a series of well-known attacks, including replay attacks, spoofing attacks and relocation attacks. Our implementation limits on-chip memory space overhead to less than 33% versus memory used by a standard microprocessor and reduces memory latency achieved by previous approaches by at least half. The performance loss for software execution with our solution is only 10% compared with a non-protected implementation. An FPGA prototype of our security core has been completed to validate our findings.

Key words: Embedded systems, security, hardware architecture, FPGA

* European University of Brittany, CNRS, UMR 3192 - Lab-STICC, Centre de recherche - BP 92116, 56321 Lorient FRANCE
**CEFET-RN, Brazil
***University of Massachusetts, Dept of Electrical and Computer Engineering, Amherst, Mass. 01003, USA

Email addresses: `vaslin@univ-ubs.fr` (Romain Vaslin*, Guy Gogniat*, Jean-Philippe Diguet*,), `wanderley@cefetrn.br` (Eduardo Wanderley**,), `tessier@ecs.umass.edu` (Russell Tessier***, Wayne Burleson***).

1 Introduction

With the development of new wireless communication standards, the ubiquitous connectivity of embedded systems is becoming a reality. Since sensitive data, such as credit card numbers, passwords, etc., are often exchanged between these components, these transfers must be protected. As system use has become more diverse, security has become a substantial performance bottleneck, especially since embedded systems are often resource limited. More and more systems are facing hardware and software attacks (Dagon, 2004). As a consequence, various cryptographic solutions have emerged to improve system protection.

Often, existing system protection solutions require significant overhead to achieve adequate protection. It is essential that protection solutions for embedded systems meet tight constraints on memory size, performance and power consumption. In the following sections, we describe a solution to fully protect the confidentiality and integrity of embedded system external memory. The paper is organized as follows. Section 2 describes the threat model and state-of-the-art for existing memory protection solutions. Section 3 details one-time pad (OTP) protection and necessary extensions for integrity checking. In section 4, an example implementation of our solution, which uses an Altera NIOS II embedded processor (ALTERA, 2007), is described. Finally, section 5 offers conclusions and directions for future work.

2 State of the art

2.1 Threat model

The external memory of an embedded system can face a variety of attacks (Elbaz, 2006), including the probing of the bus between the processor core and memory. Since the bus is exposed, an adversary can easily examine data and address values with little effort. If the bus data is sensitive, it must be ciphered with an encryption algorithm, such as 3DES (3DES, 1995) or AES (AES, 2003). By using encryption, the confidentiality of the data is guaranteed. For several bus attacks, data ciphering alone does not provide a sufficient level of security. A spoofing attack (Figure 1(a)) occurs when an attacker provides a random data value on the bus, causing the system to malfunction. A relocation or splicing attack (Figure 1(b)) occurs when an instruction is copied from one memory location to a different location, overwriting an existing instruction. If the whole memory is encrypted with the same key, the swapped instruction will be executed instead of the original instruction. For example, a swapped

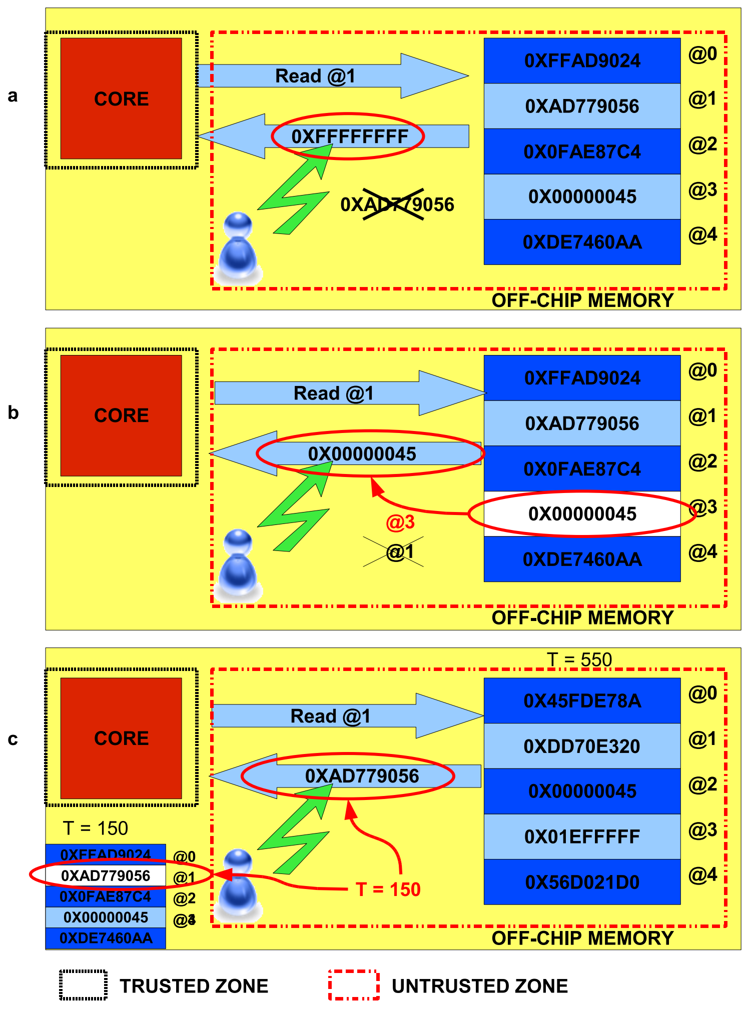

Fig. 1. (a) Spoofing attack (b) Relocation attack (c) Replay attack

instruction could make a program jump to malicious code stored in a non-ciphered part of memory. A replay attack (Figure 1(c)) is similar to a relocation attack since an existing instruction is overwritten with a new one. In this case, the new instruction is a copy of one which previously occupied the memory location, but was subsequently overwritten.

2.2 Existing solutions

Three techniques have been developed to enhance the memory protection of processor systems. Two of these approaches, XOM (Lie, 2003) (Lie, 2000) and AEGIS (Suh, 2008) (Suh, 2005) (Suh, 2003a) (Suh, 2003b), also provide secure context switching and security level management, features which are not considered by our research. For each of the three techniques there are system concerns which impact the processor cores.

2.2.1 XOM

The *eXecute Only Memory* (XOM) (Lie, 2003) (Lie, 2000) memory protection approach is based on complex key management. Each memory partition is associated with a session key that is needed to decrypt its contents. Encrypted sessions keys are stored in main memory and can be decrypted using an asymmetric secure private key. Decrypted session keys are stored in the XOM key table. The private key required for asymmetric decryption is stored in the trusted zone of the architecture. The algorithm used for symmetric deciphering is AES256. When the core produces a cache miss, 256 bits of data must be read from memory and decrypted. For this case, the time required to perform AES decryption adds to the memory latency (case (b) in Figure 2). The integrity of each data value is ensured by a message authentication code (Krawczyk, 1997). This code consists of a hash of the data and its virtual address. The hash is ciphered with the data and stored in the external memory with the data value. Although effective, this solution does not protect the system against replay attacks. In (Yang, 2003), the authors replace standard AES ciphering with encryption based on one-time pad (OTP) operations. As described in section 3.1, OTP performs AES-based encryption of a data value using a combination of the value, its address, and a time stamp. Time stamp values guarantee protection against replay attacks.

2.2.2 AEGIS

AEGIS (Suh, 2008) (Suh, 2005) (Suh, 2003a) (Suh, 2003b) includes a memory security solution which uses one-time pad (OTP) operations to provide confidentiality. This encryption method has a small impact on memory latency at the cost of memory space overhead. A *cached hash tree* is used by AEGIS for integrity checking. This hashing approach is similar to a *Merkle tree* (Merkle, 1980) except that some hash tree nodes are stored in a cache to increase efficiency. For Merkle trees, only the root of the hash tree is securely stored. All hashes must traverse the tree until the root is reached. For cached hash trees, a hash is only performed until the desired node is found in the tree, increasing efficiency. Cached hash trees can only be considered secure if the hash cache memory is in a trusted zone of the system.

2.2.3 PE-ICE

PE-ICE (Elbaz, 2006) uses the spreading feature of block ciphering algorithms for AES to provide system confidentiality and integrity. Like XOM, a tag is added to a data value before ciphering. For read-only values, the tag includes the memory address to prevent relocation attacks. For read-write values, the address and a random value are included to prevent replay attacks. Due to

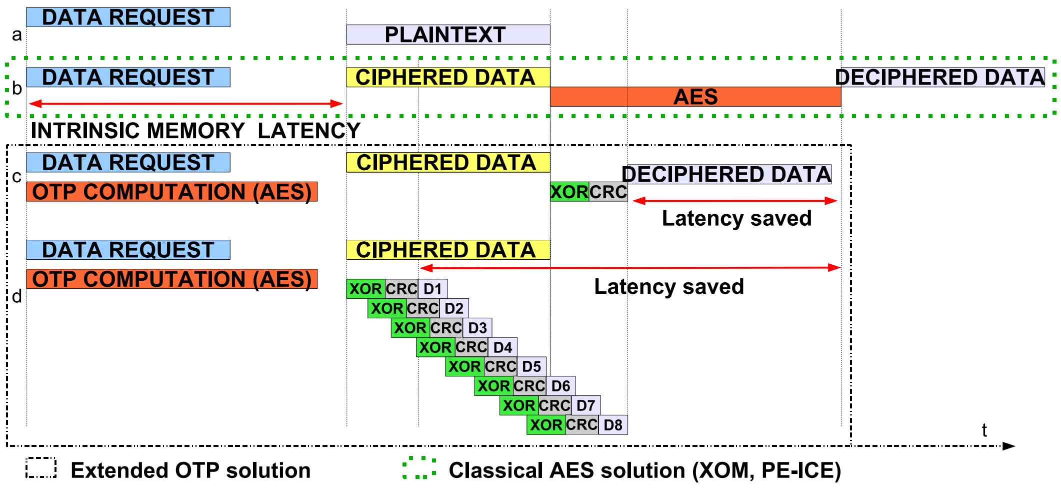

Fig. 2. Overview of the overhead of several security solutions for long latency memory reads: (a) No protection, (b) AES, (c) & (d) Extended OTP solution

the spreading feature of AES, the deciphered value of a data value will be greatly changed if even one memory bit is modified. During AES encryption, the plaintext and the tag are used as input. When the system performs a comparison between the deciphered tag and the original one concatenated with the data, it can detect if data integrity has been maintained. Like XOM, PE-ICE can have an impact on memory read latencies since decryption can only be performed after the read of a full cache line from external memory. Integrity checking is performed using a comparator for the address and the tag, so the amount of logic needed to guarantee integrity is not significant.

3 OTP encryption with extensions for integrity checking

3.1 Standard OTP encryption

OTP encryption was initially proposed by Gilbert Vernam during World War I (Anderson, 2001), but was only recently adapted for digital memory protection (Suh, 2003b). This protection approach uses the delay created by memory reads to compute a random OTP key. After key generation, it is XORed with the ciphered data to obtain the retrieved plaintext. Each OTP is created before a memory write and is used for encryption. The same OTP is also used for subsequent decryption.

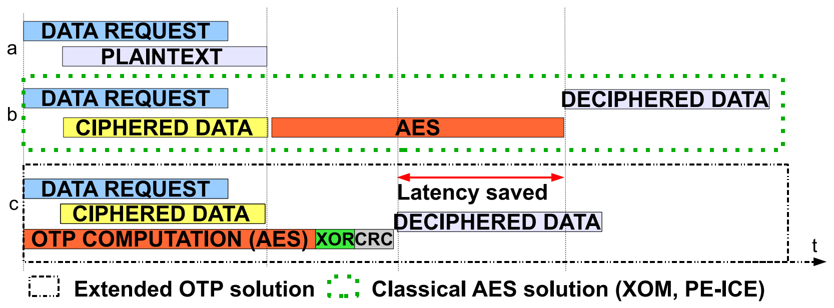

Fig. 3. Overview of the overhead of several security solutions for short latency memory reads: (a) No protection, (b) AES, (c) Extended OTP solution

AES encryption is a vital operation in OTP protection. The memory address of a data value is used as an AES core input for OTP generation. To prevent replay attacks, time stamps (TS) are used. As shown in Algorithm 1, the TS value associated with each data address is incremented by 1 after each OTP generation. For each new cache line memory write request, the system will compute a different OTP since the value of TS is incremented. The TS values are stored in a memory for later use during memory read operations. During a read, the original TS value is retrieved (Algorithm 2) and provided to AES during the read request. The result of AES will give the same OTP as the one produced for the write request and the encrypted data will become plaintext after being XORed (Algorithm 2).

Read-only data does not require protection against replay attacks because these data are never modified. No TS values are needed for these data so the amount of TS memory space can be reduced. Read-only data may be the target of relocation attacks but the address used to compute the OTP guarantees protection against these attacks. If the size of the address and the TS are not long enough to completely fill the AES encryption block input, a padding value (PV) is used. The value used for padding has no impact on the security of the OTP computation. Even if an attacker knows the TS, address and padding values, he will not be able to obtain the generated OTP key since the secret key used by the AES core is unknown.

In most systems, memory accesses require a long latency. As a result, the cache line read latency may be long enough to perform OTP computation with AES. As shown for cases (c) and (d) in Figure 2, the latency added by encryption is reduced compared to case (a) which represents previous solutions (XOM, PE-ICE). These previous solutions use the stored data as the input for AES. In case (c) in Figure 2, the latency added by OTP encryption is the latency of a logical XOR operation. In general, the time needed to retrieve the data from the memory is longer than the time needed to compute the OTP with AES.

Figure 3 illustrates the situation when data fetch time is shorter than OTP computation time. Even in this case, the latency needed to obtain the deciphered data is shorter when OTP is used instead of AES. From a security standpoint, it is essential that the OTP key is used only one time. The OTP key is obtained with AES, so the AES inputs also need to be used just one time. If an OTP key is used several times, information leakage may occur. The attacker may be able to determine if data ciphered with a same OTP have the same values.

Algorithm 1 - ***Cache memory write request:***

```
$1 - CRC\,(@) = CRC\{plaintext\}$
$2 - Time\ stamp\ incrementation:\ TS\,(@) = TS\,(@) + 1$
$3 - OTP\ computation:\ OTP = AES\{TS\,(@)\,, @, PV\}$
$4 - Ciphered\ data = plaintext \oplus OTP$
$5 - Ciphered\ data \Rightarrow memory$
$6 - TS\,(@) \Rightarrow TS\ memory$
$7 - CRC\,(@) \Rightarrow CRC\ memory$
```

Algorithm 2 - ***Cache memory read request:***

```
$1 - Get\ TS\,(@) \Leftarrow TS\ memory$
$2 - Get\ CRC\,(@) \Leftarrow CRC\ memory$
$3 - OTP\ computation:\ OTP = AES\{TS\,(@)\,, @, PV\}$
$4 - Get\ ciphered\ data \Leftarrow memory$
$5 - Plaintext = Ciphered\ data \oplus OTP$
$6 - CRC\,(@) \equiv CRC\{plaintext\}$
$7 - Plaintext \Rightarrow cache\ memory$
```

Highlighted operations are only available for the extended OTP solution proposed here with integrity checking

As mentioned earlier in this section, the use of time stamps and data addresses for OTP protects a system against replay and relocation attacks. If a data value is replayed, the TS used for ciphering will differ from the one used for deciphering. If a data value is relocated, its address will differ from the one used to generate the OTP. In both cases, the deciphered data will be invalid. To use this information, the secure memory access system must be able to detect that the deciphered data is incorrect. Thus, we present an extension to OTP encryption in section 3.2. Our OTP implementation is efficient because it performs OTP computation (operation 3 in Algorithm 2) in parallel with memory data requests (operation 4 in Algorithm 2). Figures 2 and 3 provide a view of the nature of the benefit.

3.2 An integrity checking extension for OTP

The memory security circuitry must be able to detect and report an OTP mismatch error following a data read. Our integrity checking approach uses CRC operations to minimize resource overhead and operation latency. Prior to OTP generation, the CRC of the cache line to be encrypted is generated (operation 1 in Algorithm 1) and later stored in a cache (operation 7 in Algorithm 1). Later, when the processor core requests a read, the CRC result of the final XOR operation is compared with the CRC value stored in the memory (operation 6 in Algorithm 2). If data is changed following storage, the CRC of the retrieved value will differ from the stored value, so the attack is detected. As shown in Figure 2, the latency added to the original OTP solution by our extension is the latency of CRC computation and checking. This CRC computation can be completed in one clock cycle. With the extended OTP, the minimum latency added to a memory access is the time to obtain the result of the XOR and the CRC check (case (b) in Figure 2).

If the CRC is performed on a full data cache line (CRC32), the operation can only be performed when all the data values have been read from the memory and XORed (case (c) in Figure 2). A way to decrease the data retrieval latency is to perform the XOR and CRC on a 32 bit word (CRC8) and not on a full cache line (256 bits, for example). As a result, the system does not have to wait for a full cache line fetch from external memory. As soon as the first 32 bits have been read, the 32 bit word is deciphered and checked for integrity. Decryption and integrity checks for the remaining 32 bit words of the cache line can then be pipelined (case (d) in Figure 2). This approach reduces the memory latency caused by the extra security steps but doubles memory consumption, since the 32 bit CRC of each 32 bit word must be stored. To address this issue, we use an 8 bit CRC to reduce the required storage for partial cache lines.

3.3 Strengths and weaknesses of this architecture

An integrity checking approach based on CRC is generally weaker than approaches based on MD5 (MD5, 1992) and SHA-1 (SHA-1, 2001) algorithms, although our implementation mitigates this weakness. Our approach assumes that an attacker does not have physical access to the plaintext. The CRC computation is performed on a clear cache line and only ciphered data is stored in memory. Due to the OTP-based ciphering, there is not a direct link between the result of the CRC computation and the data in memory. In this case, the the CRC result cannot be attacked because only the ciphered data is visible.

For a 32 bit CRC implementation, an attacker has one chance out of 2^{32} of randomly creating a data value which has the same CRC result as the original data value. Moreover, even if two data values have the same CRC result, it may not be possible for an attacker to extract information about the system. A 64 bit CRC could be used to increase the security level of the architecture, effectively doubling the required memory overhead.

For an 8 bit CRC implementation, an attacker has one chance out of 2^{8} of randomly creating a data value which has the same CRC result as the original data value. For many systems, this level of integrity protection may be inadequate, although it may be sufficient for embedded systems. More powerful approaches, such as MD5 and SHA-1, require a minimum input data size of 512 bits, which may be prohibitive for some processor-based embedded architectures. Many of these architectures do not support 512 bit cache lines. In addition, the output size of the tag values generated by these algorithms is 128 or 160 bits, so the memory overhead required to store information for these approaches is substantial (around 30%). Finally, the latency required to obtain a result for MD5 and SHA-1 is substantially longer than our CRC-based approach. Unlike CRC generation, which requires 1 clock cycle, MD5 requires 64 clock cycles and SHA-1 requires 80 clock cycles. Assuming an SDRAM latency of 10 cycles, the cache miss latency associated with critical data would be at least 74 or 90 cycles.

It is possible to trade off memory overhead, security level and system performance. Table 1 shows a summary of the tradeoffs discussed in section 2.2. For PE-ICE, the security level is $1/2^{32}$, since a tag of 32 bits is added to the data prior to ciphering with AES. AEGIS has a security level of $1/2^{160}$ since SHA-1, which has a signature of 160 bits, is used for integrity checking. This increased security comes at the cost of increased memory usage and decreased system performance. For XOM, the level of integrity is determined by the hash algorithm that is deployed (MD5 or SHA-1).

	OTP + CRC32	OTP + CRC8	PE-ICE Elbaz (2006)	XOM (Lie, 2003)	AEGIS (Suh, 2003b)
Performance loss	low	very low	average	very high	high
Memory cost	very low	low	high	high	high
Security level	$1/2^{32}$	$1/2^{8}$	$1/2^{32}$	$1/2^{128}$ or $1/2^{160}$	$1/2^{160}$
Target	Embedded system	Embedded system	Embedded system	Workstation	Workstation

Table 1
Security and performance levels for memory protection approaches

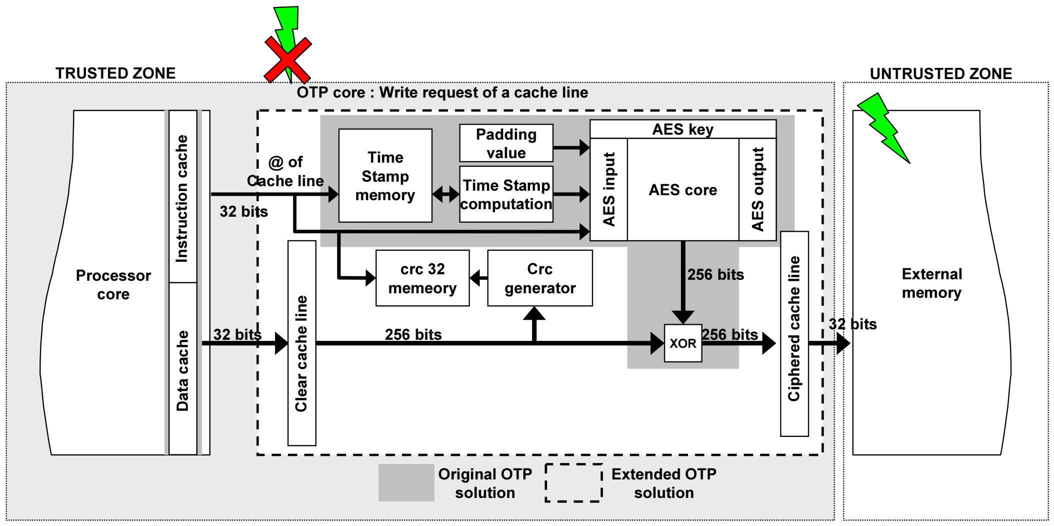

Fig. 4. Write request including OTP operations

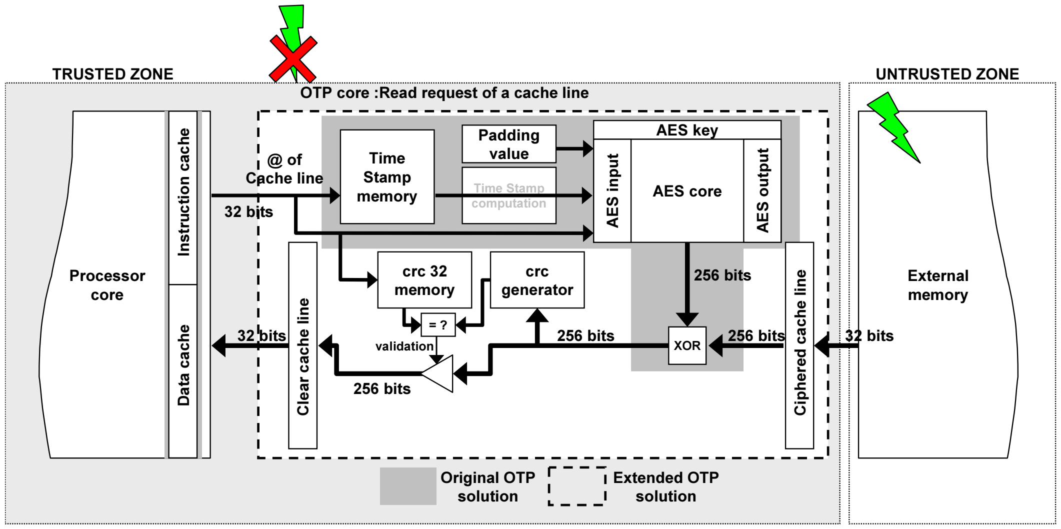

Fig. 5. Read request including OTP operations

4 Example implementation with an embedded processor

4.1 Architectural features

An embedded platform based on an Altera NIOS II microprocessor was used to validate our new memory protection approach. The NIOS II configuration includes instruction and data caches, each with 512 total bytes, and cache lines of 256 bits. As seen in Figures 4 and 5, NIOS caches are interconnected to the OTP design via 32 bit connections. A 32 bit wide bus is also used to connect the NIOS II to 4 Mbits of off-chip SDRAM.

For this work, it is assumed that the OTP core and TS and CRC storage cannot be attacked using techniques such as fault injection or side channel attacks because they are located in the trusted zone. The memory space required to store the time stamps and CRC values is summarized in Equation 1. Consider a system configuration that has a total memory size of 512 KB and a 32 bit CRC. Half of the memory values are read-only processor instructions and half are read-write (RW) data. From Equation 1, $OTP_{\text{STORAGE}} = 96$ KB (32 KB for TS_{STORAGE} and 64 KB for $CRC32_{\text{STORAGE}}$ with a $TS\ SIZE$ and a $CRC32\ SIZE$ of 32 bits). Time stamps are unnecessary for read-only data.

Equation 1 - ***OTP memory consumption***

$$OTP_{\text{STORAGE}} = TS_{\text{STORAGE}} + CRC_{\text{STORAGE}}$$

$$TS_{\text{STORAGE}} = \left(\frac{RW\ DATA\ MEMORY\ SIZE}{CACHE\ LINE\ WIDTH}\right) * TS\ SIZE$$

$$CRC32_{\text{STORAGE}} = \left(\frac{TOTAL\ MEMORY\ SIZE}{CACHE\ LINE\ WIDTH}\right) * CRC\ SIZE$$

For this work, a 128 bit AES core is used to minimize the hardware impact of OTP on the overall design. Since the AES core generates 128 bit OTP values and each cache line has 256 bits per line, each OTP value must be used twice to encrypt a full cache line. The CRC32 module has an input width of 256 bits (a full cache line). This module produces a 32 bit output which is stored in the CRC32 memory (Figure 4) or compared to a stored value from the cache (Figure 5). For the pipelined version of the design, a 32 bit AES core is used. This CRC8 core has a 32 bit input width and an 8 bit output width. Based on Equation 1, the memory needs for CRC storage are 160 KB if 256 KB of instruction memory and 256 KB of data storage are used. For the pipelined version, a 256 bit cache line will generate a 64 bit CRC result (8 bit CRC for each 32 bit word). Thus, the CRC storage required for the pipelined version is twice as large as the unpipelined version (64 versus 32 bits per 256 bit word).

4.2 The cost of security

The cost of adding security to the NIOS II based system is significant. Table 2 shows that the number of required look-up tables (ALUTs) is nearly tripled (281% increase) and memory usage is increased by 18.8% versus a basic NIOS system. These overheads were determined using Equation 1. The pipelined version of the security circuitry requires a greater memory overhead (31.3%).

The added circuitry has an effect on the latency of data retrieval. In our case,

	Base NIOS	NIOS + OTP128 + CRC32		NIOS + OTP128 + CRC8		NIOS + OTP256 + CRC32	
	cost	cost	overhead	cost	overhead	cost	overhead
Logic (ALUTs)	2198	6193	281%	6095	277%	6767	317%
Memory (KB)	512	600	18.8%	662	31.3%	603	19.7%
Read latency (cycles)	0	11	11	3	3	11	11
Write latency (cycles)	0	12	12	12	12	12	12

Table 2
Cost of security for NIOS II

which is similar to example in Figure 2(c), the intrinsic memory latency is long enough to overlap the OTP generation, which requires 12 cycles.

A total of 8 cycles are required to request the eight 32 bit cache line values from the SDRAM. A four delay is then incurred by the SDRAM to process the request. These 12 cycles of latency are always present in the NIOS architecture, even for the base system which does not include security. Our OTP128 approach must wait 8 additional cycles to fetch a full cache line from SDRAM before CRC computation can be performed. As soon as the cache line is fetched from the memory, 3 additional cycles are needed to perform XOR and CRC operations. Thus, a total of 11 cycles (8+3) are added to the read transaction delay of the base NIOS II architecture (Table 2). This overhead is significant, but as shown in Figure 2 and in Table 5, the overhead is less for our new approach versus non-OTP approaches, which require the retrieval of the entire cache line before the use of AES. For CRC8, the latency is reduced to the 3 clock cycles needed to perform the XOR and CRC8 operations. Partial cache lines of 32 bits can be immediately decoded following retrieval, reducing the latency overhead. The latency overhead beyond the base case for a write request is 12 cycles. The 12 cycles are due to the time required to perform OTP management (AES computation and XOR operation). Table 2 illustrates the trade off between memory and latency overheads. This tradeoff can be tuned based on the application using the configurability of the FPGA.

The number of AES cores used in the architecture impacts the design size, leading to additional tradeoffs. As described in section 3.1, each OTP value should not be used more than once. Since an AES core generates a 128 bit result and each cache line has 256 bits per line, each 128 bit OTP must be used twice to encrypt a full cache line (Equation 2-1). In this case, information leakage can occur. An adversary will be able to determine that the first 128 bits of the OTP are the same as the last 128 bits, but the unciphered values cannot be determined (NIST, 2001). If a more secure implementation is required or if the attacker can access some of the unciphered data, a 256 bit AES implementation is mandatory (Equation 2-2). In this case, the architecture will need to use two AES cores to create a 256 bit OTP. The overhead in logic and memory due to this architectural modification is shown in Tables 3 and 4. The impact of the OTP implementation on memory size is insignificant. The small memory difference between the 128 and 256 bit AES implementations is a result of SBOX implementation with memory blocks. The two AES cores use the same time stamp, so TS storage is not increased for the larger OTP implementation. The addresses used as AES input depend on the addresses of the cache line subblocks (@1 & @2) (Equation 2-2). The latency of OTP generation is unaffected since the two 128 bit AES cores run in parallel. For the OTP128 core, the 128 bit AES block represents 36% of the total design size. For the 256 bit version, the two 128 bit cores represent 42% of the total design size. CRC size is roughly constant for both cases.

Equation 2 - ***Two security levels***

With one AES core:

$1 - OTP_{128} = AES_{128}\left(TS\left(@_{\text{BASE}}\right), @_{\text{BASE}}, padding\right)$

$1 - Ciphered\ data_{256} = plaintext_{256} \oplus \{OTP_{128}, OTP_{128}\}$

With two AES cores:

$2 - OTP1_{128} = AES_{128}\left(TS\left(@_{\text{BASE}}\right), @_1, padding\right)$

$2 - OTP2_{128} = AES_{128}\left(TS\left(@_{\text{BASE}}\right), @_2, padding\right)$

$2 - Ciphered\ data_{256} = plaintext_{256} \oplus \{OTP1_{128}, OTP2_{128}\}$

	OTP128 core + CRC	OTP control	AES128	CRC32
Logic (ALUTs)	4059	1918	1479	662
Memory (KB)	98.6	32	2.6	64

Table 3
Resource breakdown for 128 bit OTP implementation

	OTP256 core + CRC	OTP control	AES256	CRC32
Logic (ALUTs)	4633	1999	1972	655
Memory (KB)	101.1	32	5.1	64

Table 4
Resource breakdown for 256 bit OTP implementation

4.3 Evaluation

The cost of our new security approach compares favorably with the existing solutions described in section 2.2. A comparison of relative logic sizes is difficult due to a lack of available data from previous approaches. In general, each approach requires at least one AES core, although the number of cores and method of integrity checking varies. For PE-ICE, no overhead is required for integrity checking. For AEGIS, a time-consuming software implementation of the SHA-1 algorithm on a cached hash tree is used for integrity checking. The developers of AEGIS do note (Suh, 2003a) the possibility of a hardware SHA-1 implementation.

Table 5 summarizes a number of additional relevant cost values. All of these approaches support some level of confidentiality and integrity for off-chip memory. In terms of memory, our solution consumes less space than other solutions (Figure 6) even though an overhead of 32% is required for the CRC8 version versus the base NIOS implementation. AEGIS also guarantees confidentiality using OTP so it also requires space for time stamps. However, the use of a cached hash tree for integrity checking causes a memory overhead of 33%. For XOM, no memory overhead figures have been published. However, since the XOM integrity check uses a MAC solution, some storage space will be needed to store hash signatures. Memory overhead for PE-ICE results from tags (address and random values) added to the data and from on-chip storage needed to securely store random values.

In (Yang, 2003), the authors describe an OTP implementation which stores TS values off-chip. A small cache of TS values is kept on-chip to improve processor performance. This same approach could be used by our system to store TS and CRC values off-chip. Of course, these values would require the same data protection as other instructions and code. The area and performance cost of such a solution would likely be high since our current OTP approach overlaps OTP generation with data fetches.

The system latency of our new approach also compares favorably to previous AES-based approaches (XOM and PE-ICE). For PE-ICE, the time required to verify a tag read from memory against one stored in on-chip memory extends memory read latency. For OTP-based AEGIS, software-based integrity checking adds significant delay, impacting the performance of the system. It has previously been reported (Suh, 2003a) that 4715 cycles are need to execute the SHA-1 algorithm in software. A hardware implementation of one hash cycle would require 80 clock cycles. Since AEGIS integrity checking uses Merkle trees (section 2.2.2) multiple hash cycles are required, extending the latency. For an AEGIS write request, the hash computation is not in the critical path since it can be performed after the data has been written to the memory. In

	base AES (no integrity)	OTP + CRC32		OTP + CRC8		XOM AES + HMAC		PE-ICE AES		AEGIS OTP + hash trees	
	cost	cost	overhead	cost	overhead	cost	overhead	cost	overhead	cost	overhead
Memory (KB)	512	600	+18.8%	662	31.3%	unclear	unclear	776	+50.7%	768	+50%
Read latency (cycles)	22 = (14+8)	11 = (8+3)	-10	3 = (0+3)	-19	86=(22+64) 102=(22+80)	+64 +80	25 = (17+8)	+3	≈(SHA-1)	>80
Write latency (cycles)	22 = (14+8)	12	-10	12	-10	86=(22+64) 102=(22+80)	+64 +80	26 = (18+8)	+4	12	-10

Table 5
Overhead comparison of security approaches solutions with basic AES data encryption. Latencies include data fetch time and encryption time.

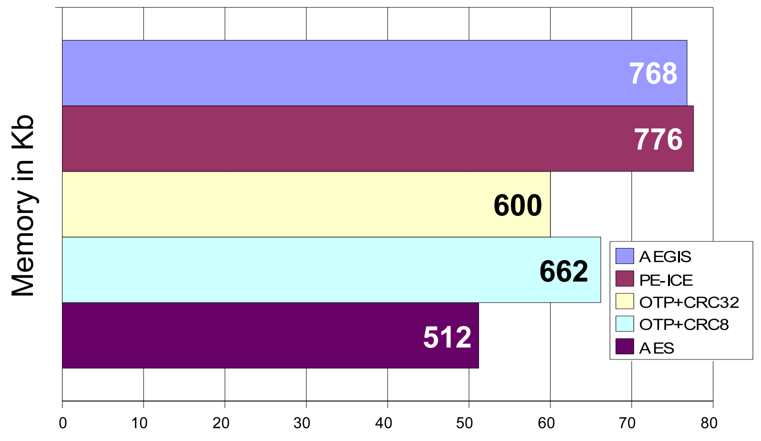

Fig. 6. Memory footprint comparison

constrast, the XOM approach (section 2.2.1) encrypts data with the hash signature requiring a significant delay during both data reads and writes. These actions include either a 64 cycle (MD5) or 80 cycle (SHA-1) memory access delay. Table 5 shows that our approach reduces latency compared to other approaches. Only 3 or 11 cycles are needed for memory reads instead of the 22 cycles required by previous AES-based solutions.

Figure 7 gives an overview of the performance of our security approach and other security approaches versus an unprotected memory implementation. Our approach suffers about a 10% slowdown versus the unprotected case, depending on the application cache miss rate, but the approach achieves a 36% performance improvement versus basic AES data ciphering with no integrity checking. The main cost of our approach is increased on-chip memory usage. For the pipelined version of our approach used with a NIOS II processor, a 32% increase in memory usage is reported. This result presents a choice: a secured system with a moderate memory footprint and an average latency or a low security system with a small memory overhead and low latency.

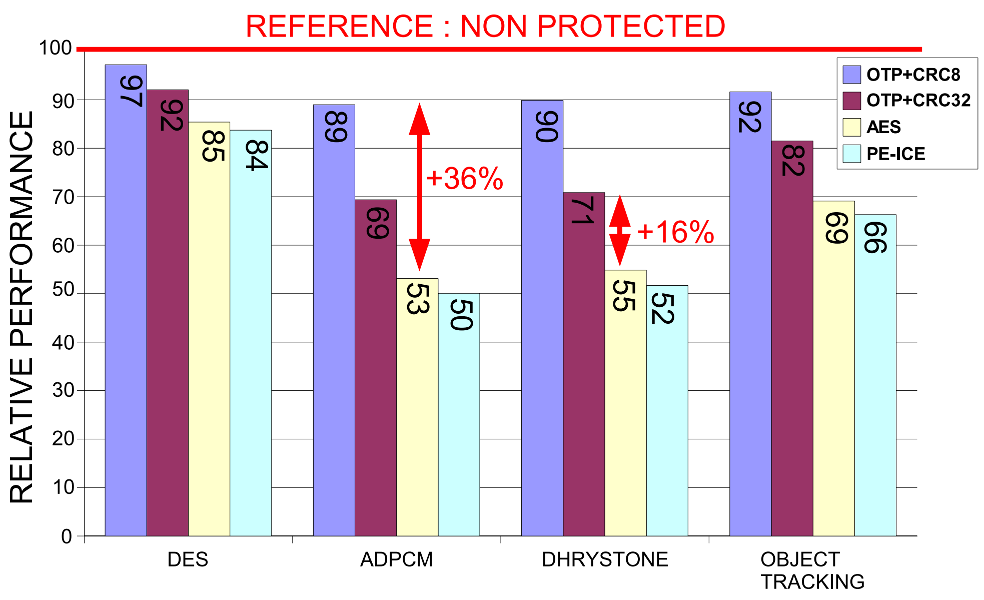

Fig. 7. Performance comparison for memory security approaches versus an unprotected case.

5 Conclusion and Perspectives

In this paper, we have evaluated the impact of a new off-chip memory security approach for an embedded processor architecture and provided an estimation of the memory access latency costs. A possible next step would be to assess the impact of the approach on a range of cache line and CRC sizes. For example, it would be interesting to evaluate implementations using 64 and 128 bit CRCs.

Since many embedded systems require battery-based operation, power consumption is an important issue. A complete analysis of the power costs of our approach is needed to evaluate the overhead of our approach on power consumption. From a security standpoint, additional work is needed to protect on-chip memory used to store TS and CRC values. This memory could be targeted by fault injection attacks leading to incorrect system operation.

The work presented in this paper uses a reconfigurable target (FPGA). The features of reconfigurable architectures provide interesting perspectives for security. It may be possible to adapt the security level of the architecture in response to different threat levels. In (Gogniat, 2006), the authors propose reconfigurable mechanisms to provide for a fault tolerant AES. Another security adaptation opportunity might involve real-time operating systems (RTOS). An RTOS may have specific services to enable hardware security primitives. The isolation of non-sensitive data would reduce the amount of memory needed to store TS and CRC tags.

References

[Dagon, 2004] David Dagon, Tom Martin and Thad Starner, Mobile Phones as Computing Devices: The Viruses are Coming!, IEEE Pervasive Computing, Vol. 3, No. 4, pp 11-15

[ALTERA, 2007] http://www.altera.com/

[Elbaz, 2006] Reouven Elbaz, Lionel Torres, Gilles Sassatelli, Pierre Guillemin, and Michel Bardouillet and Albert Martinez, A Parallelized Way to Provide Data Encryption and Integrity Checking on a Processor-Memory Bus, DAC '06: Proceedings of the 43rd Annual Conference on Design Automation, pp 506-509

[AES, 2003]AES RFC 3565, ftp://ftp.rfc-editor.org/in-notes/rfc3565.txt

[3DES, 1995] 3DES RFC 1851, ftp://ftp.rfc-editor.org/in-notes/rfc1851.txt

[Lie, 2003] David Lie, Chandramohan A. Thekkath, and Mark Horowitz, Implementing an Untrusted Operating System on Trusted Hardware, SOSP '03: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, pp 178-192

[Lie, 2000] David Lie, Chandramohan Thekkath, Mark Mitchell, Patrick Lincoln, Dan Boneh, John Mitchell and Mark Horowitz, Architectural Support for Copy and Tamper Resistant Software, ASPLOS-IX: Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems, pp 168-177

[Suh, 2008] G. Edward Suh, Charles W. O'Donnell, and Srinivas Devadas, Aegis: A Single-Chip Secure Processor, IEEE Design & Test of Computers, Vol. 24, No. 6, pp 570-580

[Suh, 2003] G. Edward Suh, Dwaine Clarke, Blaise Gassend, Marten van Dijk and Srinivas Devadas, AEGIS: Architecture for Tamper-Evident and Tamper-Resistant Processing, ICS '03: Proceedings of the 17th International Conference on Supercomputing, pp 160-171

[Suh, 2005] G. Edward Suh, Charles W. O'Donnell, Ishan Sachdev and Srinivas Devadas, Design and Implementation of the AEGIS Single-Chip Secure Processor Using Physical Random Functions, ISCA '05: Proceedings of the 32nd Annual International Symposium on Computer Architecture, pp 25-36

[Suh, 2003] G. Edward Suh, Dwaine Clarke, Blaise Gassend, Marten van Dijk, and Srinivas Devadas, Efficient Memory Integrity Verification and

Encryption for Secure Processors, MICRO 36: Proceedings of the 36th IEEE/ACM Annual International Symposium on Microarchitecture, pp 339-350

[Merkle, 1980] R.C. Merkle, Protocols for Public Key Cryptosystems, IEEE Symposium on Security and Privacy, pp 122

[Krawczyk, 1997] H. Krawczyk, M. Bellare, and R. Canetti, HMAC: Keyed-hashing for Message Authentication, RFC 2104, http://www.faqs.org/rfcs/rfc2104.html

[Anderson, 2001] Ross J. Anderson, Security Engineering: A Guide to Building Dependable Distributed Systems, John Wiley, New York, NY

[Gogniat, 2006] Wayne Burleson, Guy Gogniat, and Tilman Wolf, Reconfigurable Security Support for Embedded Systems, HICSS '06: Proceedings of the 39th Annual Hawaii International Conference on System Sciences, pp 250

[MD5, 1992] RFC 1321, ftp://ftp.rfc-editor.org/in-notes/rfc1321.txt

[SHA-1, 2001] RFC 3174, ftp://ftp.rfc-editor.org/in-notes/rfc3174.txt

[Yang, 2003] Jun Yang, Youtao Zhang, Lan Gao, Fast Secure Processor for Inhibiting Software Piracy and Tampering, MICRO 36: Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture, pp 351-360

[NIST, 2001] Recommendation for Block Cipher Modes of Operation, Methods and Techniques, NIST special publication 800-38A, 2001