Configurable Memory Security in Embedded Systems

JÉRÉMIE CRENNE, ROMAIN VASLIN, GUY GOGNIAT, and JEAN-PHILIPPE DIGUET,
Université Européenne de Bretagne
RUSSELL TESSIER and DEEPAK UNNIKRISHNAN, University of Massachusetts, Amherst

System security is an increasingly important design criterion for many embedded systems. These systems are often portable and more easily attacked than traditional desktop and server computing systems. Key requirements for system security include defenses against physical attacks and lightweight support in terms of area and power consumption. Our new approach to embedded system security focuses on the protection of application loading and secure application execution. During secure application loading, an encrypted application is transferred from on-board flash memory to external double data rate synchronous dynamic random access memory (DDR-SDRAM) via a microprocessor. Following application loading, the core-based security technique provides both confidentiality and authentication for data stored in a microprocessor's system memory. The benefits of our low overhead memory protection approaches are demonstrated using four applications implemented in a field-programmable gate array (FPGA) in an embedded system prototyping platform. Each application requires a collection of tasks with varying memory security requirements. The configurable security core implemented on-chip inside the FPGA with the microprocessor allows for different memory security policies for different application tasks. An average memory saving of 63% is achieved for the four applications versus a uniform security approach. The lightweight circuitry included to support application loading from flash memory adds about 10% FPGA area overhead to the processor-based system and main memory security hardware.

Categories and Subject Descriptors: K.6.5 [**Management of Computing and Information Systems**]: Security and Protection—*Authentication*

General Terms: Security, Design

Additional Key Words and Phrases: AES-GCM, embedded system, FPGA

ACM Reference Format:
Crenne, J., Vaslin, R., Gogniat, G., Diguet, J.-P., Tessier, R., and Unnikrishnan, D. 2013. Configurable memory security in embedded systems. ACM Trans. Embedd. Comput. Syst. 12, 3, Article 71 (March 2013), 23 pages.
DOI: http://dx.doi.org/10.1145/2442116.2442121

1. INTRODUCTION

Cost-sensitive embedded systems are used to executing end-user applications in a wide range of computing environments which span a spectrum from handheld computing to automotive control. These systems often contain little more than a microprocessor, field-programmable logic, external memory, I/O (input/output) ports, and interfaces to sensors. In addition to standard concerns regarding system performance and power consumption, security has become a leading issue for embedded applications. The portable nature of many embedded systems makes them particularly vulnerable to

Authors' addresses: J. Crenne, R. Vaslin, G. Gogniat, and J.-P. Diguet, Université de Bretagne Sud-UEB, France; R. Tessier and D. Unnikrishnan, University of Massachusetts, Amherst, MA 01003.
Corresponding author's email: jeremie.crenne@lirmm.fr.

DOI: http://dx.doi.org/10.1145/2442116.2442121

a range of physical and software attacks. In many cases, attackers are uninterested in the details of the embedded system design, but rather desire information about the sensitive program code and data included in system memory storage. If left unprotected, off-chip memory transfers from microprocessors can easily be observed and may reveal important information. Some memory protection can be provided by simply encrypting data prior to external memory transfer. Although data encryption techniques are widely known and used, simply modifying the values of data and instructions is generally thought to be insufficient to provide full protection against information leakage [Anderson 2001].

The memory contents of embedded systems often require protection throughout the various phases of system operation from application loading to steady-state system operation. For microprocessors, application code is frequently kept in on-board flash to facilitate system loading. Following system initialization, application code and data are stored in external system main memory, which is usually interfaced to a microprocessor via a vulnerable external bus. The sensitivity of stored information varies from highly-sensitive to low importance, motivating configurable memory security which can be adjusted on a per-task and per-application basis. The real-time performance demands of most embedded systems indicate the need for a memory security approach which is implemented in on-chip hardware, adjacent to the microprocessor. This hardware implementation should just meet an application's security needs while minimizing area. Resource efficiency is typically a key for embedded computing in constrained environments.

Our work includes the development of a new lightweight memory security approach for embedded systems which contain microprocessors implemented within secure FPGAs. Our approach provides security to off-chip FPGA processor instruction and data accesses during both application loading and steady-state operation. Our new technique differs from previous embedded system memory security approaches [Elbaz et al. 2006; Lie et al. 2003; Suh et al. 2005] by limiting logic overhead and storing security tag information on-chip. After loading application code from flash memory, a security core based on the recently-introduced Advanced Encryption Standard Galois/Counter Mode (AES-GCM) block cipher [National Institute of Standards and Technology 2007] determines the appropriate data security level as memory accesses occur in conjunction with an embedded real-time operating system. Our approach allows for the optimization of security core size on a per-application basis based on memory footprint and security level requirements. To facilitate secure application loading during system bootup, a low overhead data decryption and authentication circuit has been implemented which can reuse some of the hardware used for main memory protection.

To demonstrate the effectiveness of our approach, we evaluate the hardware overhead required by the overall security system for four different multi-task applications, each requiring different mixes of security levels. The four applications have been quantified and tested using a Microblaze soft processor on a Xilinx Spartan-6 FPGA-based prototyping board. The Microblaze runs the MicroC/OS II operating system (OS) [LaBrosse 2002] to schedule tasks for each application. The improved security is achieved with about a 13% reduction in application performance.

Overall, our work provides the following specific contributions.

—Our approach provides a low-overhead implementation of authenticated encryption for FPGA-based embedded systems based on the AES-GCM policy approved by the National Institute of Standards and Technology (NIST). The approach minimizes logic and latency required for authentication and takes advantage of increased internal FPGA memory storage in contemporary FPGAs to store authentication information.

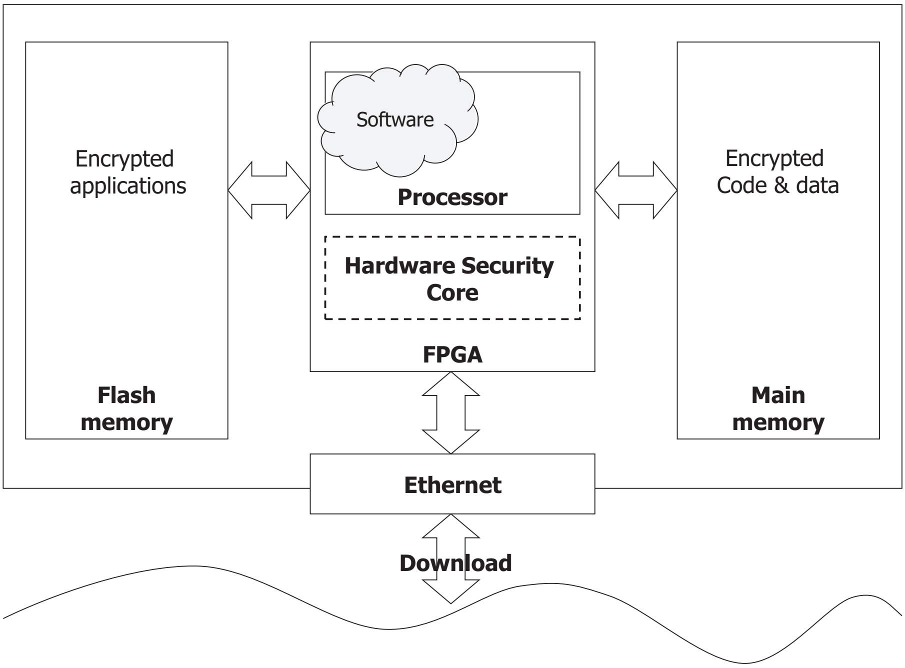

Fig. 1. High level model of a typical embedded system.

- Unlike previous encryption and authentication approaches for embedded systems [Vaslin et al. 2008], the new AES-GCM based authenticated encryption is synchronized and parallelized to enhance throughput.
- The overheads and performance of our authenticated encryption approach have been fully validated in the hardware of an FPGA-based embedded system for both main memory security and application loading from flash.
- The approach allows for a low overhead, flexible technique for providing selective confidentiality and authentication to different parts of an application without the need for processor instruction additions or significant operating system overhead.

Our approach can be used to protect any SRAM-based FPGA which supports bitstream encryption. This feature is available in all Altera Stratix II, III, IV, and V and Xilinx Virtex II, -4, -5, and -6 family devices.

The paper is organized as follows. Section 2 describes data security issues for embedded systems and previous approaches to address memory protection and secure application loading. Section 3 provides details of the developed AES-GCM based security core. Section 4 focuses on our mechanism for the low-overhead, secure application loading. In Section 5, the integration of the new memory security core with a microprocessor is described along with its use with four embedded applications. A description and analysis of experimental results regarding steady-state memory protection are provided in Section 6. Section 7 discusses results from secure application loading. Section 8 concludes the paper and offers directions for future work.

2. BACKGROUND

2.1. System Model

The system model that is addressed by this work is shown in Figure 1. The embedded system is connected to the external world via a communication port (e.g., an Ethernet connection, 802.11 wireless, etc). An on-board microprocessor executes one or more applications which are stored in off-chip flash memory. Following power-up or system reset, protected application code is loaded from the flash to the main memory (in this case DDR-SDRAM) via the microprocessor implemented in the FPGA. This model has been used in a series of previous embedded system studies [Lee and Orailoglu 2008; Pasotti et al. 2003]. Generally, code execution is not performed from flash memory due to high read latencies. The time needed to load the microprocessor-based application from flash is dependent on the code size and the data fetch latency. During steady state system operation, both program data and instructions are stored in DDR-SDRAM. These off-chip accesses consist of both read and write operations.

The memory security approach described in this paper addresses two specific memory protection scenarios based on the system model shown in Figure 1.

—*System loading of a microprocessor application.* To provide application code security, flash memory contents must be protected against attack. Since application code is stored in the flash, the same sequence is performed every time following system reset.
—*System main memory protection.* Following application loading, both code and data stored in DDR-SDRAM must be protected.

These memory security mechanisms must consume minimal area and power and provide effective performance for embedded systems. Although Figure 1 shows a potential system capability to securely download new applications from an external source to flash or DDR-SDRAM, this issue is not addressed by the work described in this manuscript.

2.2. Embedded System Memory Threats

The system memory of an embedded system can face a variety of attacks [Elbaz et al. 2006] resulting from either the probing of the interface between a processor and the memory or physical attacks on the memory itself (fault injection). Bus probing results in the collection of address and data values which can be used to uncover processor behavior. The encryption of data values using algorithms such as the Advanced Encryption Standard (AES) or Triple Data Encryption Standard (3DES) prior to their external transfer guarantees data confidentiality. Data encrypted with these algorithms cannot be retrieved without the associated key. However, even encrypted data and their associated addresses leave memory values vulnerable to attack. Well-known attacks [Elbaz et al. 2006] include spoofing, relocation, and replay attacks. A spoofing attack occurs when an attacker places a random data value in memory, causing the system to malfunction. A relocation attack occurs when a valid data value is copied to one or more additional memory locations. A replay attack occurs when a data value, which was previously stored in a memory location, is substituted for a new data value which overwrote the old location. Processor instructions are particularly vulnerable to relocation attacks since specific instruction sequences can be repeated in an effort to force a system to a specific state. Specific approaches that maintain the authentication of data from these types of attacks are needed to secure embedded systems. Authentication in this context indicates that the retrieved data from a memory location is the same as the data which was most recently written.

2.3. System Threat Model

The scope of our main memory security work is limited by the same threat model assumed by earlier approaches [Elbaz et al. 2006; Suh et al. 2003; Lie et al. 2003]. The following specific assumptions are made regarding the threat model.

Table I. Security Against Brute Force Attack for Memory Protection

	PE-ICE [Elbaz et al. 2006]	XOM [Lie et al. 2003]	AEGIS [Suh et al. 2003]	Yan-GCM [Yan et al. 2006]
Security	$\frac{1}{2^{32}}$	$\frac{1}{2^{128}}$ or $\frac{1}{2^{160}}$	$\frac{1}{2^{160}}$	$\frac{1}{2^{128}}$

—The FPGA and its contents (e.g., a microprocessor) are secure and keys and other configuration and user information in the FPGA cannot be accessed by either physical or logical attacks. These attacks include differential power attacks (DPA), side channel attacks, and probing. The FPGA is the *trusted area*.

—The FPGA configuration bitstream is encrypted and stored external to the FPGA (e.g., in on-board flash). The bitstream is successfully decrypted inside the FPGA using bitstream decryption techniques available from commercial FPGA companies. A number of effective FPGA bitstream encryption [Xilinx Corporation 2005] and secure bitstream download [Badrignans et al. 2008] techniques have been developed and tested for commercial devices. An assessment of the ability of these encryption techniques to protect FPGA contents has previously been performed [Altera Corporation 2008].

—Any on-board component outside the FPGA is insecure. These resources include the DDR-SDRAM, the flash memory which holds application code, and flash memory which holds the bitstream information. These components may be subject to physical attacks which attempt to read and/or modify data in the components.

—The interconnections between the components and the FPGA are also vulnerable to attack. Data on the interconnect can be observed or modified by an attacker.

Interconnect and components outside the FPGA are located in the *untrusted area*. Our approach provides protection against replay, relocation, and spoofing attacks caused by threats.

2.4. Related Work

A number of techniques have been developed that provide data confidentiality and authentication to main memory in processor-based systems. For these systems [Elbaz et al. 2006; Lie et al. 2003; Suh et al. 2005; Suh et al. 2003; Yan et al. 2006], confidentiality is provided via data encryption using AES or 3DES. Data is encrypted prior to off-chip transfer and decrypted following data retrieval from memory. Data authentication is typically maintained by hashing data values in a hierarchical fashion. The *brute force* security level of these schemes, summarized in Table I, measures the likelihood that an attacker could break the provided authentication using a changed data value that could pass the authentication check (AC). This type of attack would likely consist of a large number of attempts using varied data values. Even though these previous solutions have been shown to be effective, the cost of security can be high in terms of the memory space needed to store hash [Gassend et al. 2003], compressed hash [Suh et al. 2003], and AC tags for each data item and increased read latency due to the AC. Our new approach limits authentication time, although it does require the on-chip storage of security information. This overhead is quantified in Section 6. The use of AES-GCM for accelerated authentication for external memory accesses was first proposed by Yan et al. [2006]. This work involved the simulation of a full microprocessor-based system, including a multi-cache level memory hierarchy. Our work focuses on quantifying both the performance and area costs for AES-GCM based security for low-end computing typically used in embedded systems.

A distinguishing feature of our new low-overhead approach is its ability to offer configurable data security levels for different tasks in the same application. The confidentiality and authentication approach in AEGIS [Suh et al. 2005] most closely matches our approach. AEGIS also allows for the selection of different security levels for different portions of memory. However, new instructions are added to the processor for OS use. These instructions take advantage of hardware security primitives to enter and access operating system (OS) kernel primitives. Overall, this approach adds complexity to both the processor architecture and the operating system. Although it appears that other data confidentiality and authentication approaches [Elbaz et al. 2006; Lie et al. 2003 Yan et al. 2006] could be easily extended to multiple tasks, selective security based on memory addresses has not been reported for them.

2.4.1. Secure Application Loading. The secure application loading from flash memory for processor-based systems has been extensively examined in the context of both desktop [Arbaugh et al. 1997] and mobile devices [Dietrich and Winter 2008]. Like our approach, the techniques primarily include code stored in flash memory which is external to the processor [Heath and Klimov 2006]. This code is encrypted and protected by additional authentication values stored in flash memory. Early secure loading approaches [Arbaugh et al. 1997] integrated authentication with a processor's basic input/output system (BIOS) to ensure proper loading. This process is generally thought to be overly complex for embedded systems [Dietrich and Winter 2008]. More recent approaches, such as TrustZone [Alves and Felton 2004], overcome the need for read-only memory (ROM) authentication by integrating the ROM onto the same chip as the processor. Although effective, not all embedded systems use FPGAs which contain embedded ROM. A recent reevaluation of secure application loading [Dietrich and Winter 2008] for mobile platforms notes that software and hardware flexibility is allowed in the implementation of secure loading implementations in embedded systems. The Trusted Computing Group (TCG) mobile working group defines a hierarchy of loading activities including the initial retrieval of processor code. Our implementation focuses at this lowest level of this hierarchy, the fetching of code for the processor from an external, unsecured location. Unlike earlier techniques, our approach specifically targets minimization of required logic overhead in the initial stages of the secure loading process and the use of synchronized authenticated encryption using AES-GCM.

2.4.2. Relationship to the Authors' Previous Work. This manuscript extends our previous work in main memory security [Vaslin et al. 2008] by integrating main memory protection with secure application loading. These two security components complement each other by allowing for resource sharing. The authentication of main memory security is now performed using AES-GCM, a block cipher mode of operation which has previously been proven to be secure [National Institute of Standards and Technology 2007]. Our previous work used a less secure authentication approach performed by an abbreviated AES operation.

3. MAIN MEMORY SECURITY ARCHITECTURE

3.1. Security Policy

Before discussing our approach for secure application loading from flash memory, security for system main memory is described. Our main memory security approach, shown in Figure 2, relies on a hardware security core (HSC) fashioned from logic and embedded memory which is able to manage different security levels depending on the memory address received from the processor.

A *memory segment* corresponding to a user-defined software task can be determined from compiled code or associated data. Depending on the security policy required by the software designer, numerous memory segments can be built. Each memory segment is defined by 4 parameters.

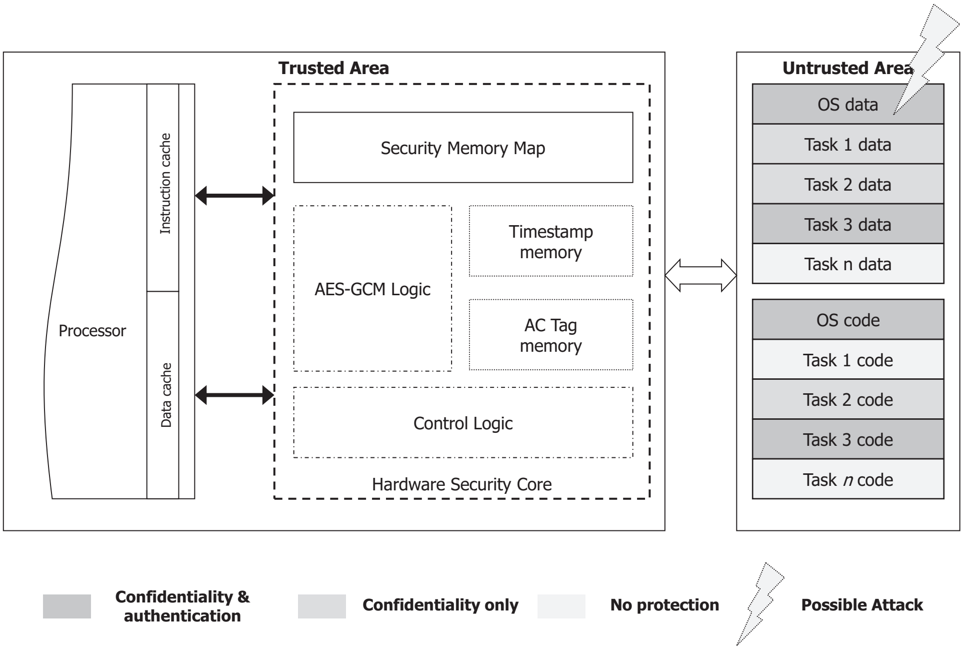

Fig. 2. Main memory security system overview. Main blocks in the trusted area (FPGA) include the Security Memory Map (SMM), AES in Galois Counter Mode of operation block (AES-GCM), timestamp (TS) memory, and authentication check (AC) tag memory storage.

—The segment base address.
—The segment size.
—The segment security level.
—The kind of segment: code (read-only) or data (read-write).

A small lookup table security memory map (SMM) is included in the hardware security core to store the security level of memory segments associated with tasks. We consider three security levels for each memory segment: confidentiality-only, confidentiality and authentication, or no security. The implementation of the security policy in the SMM is independent of the processor and associated operating system. The isolation of the SMM from the processor makes it secure from software modification at the expense of software-level flexibility. Although authentication-only is another possible security level, we do not claim support for this level since we were not able to evaluate it experimentally with our application set. Since the HSC directly works with the memory segment address, no specific compiler or compilation steps are necessary.

3.2. Security Level Management

The use of an operating system with most embedded system processors provides a natural partitioning of application code and data. In Figure 2, the application instructions and data of Task 1 have different security levels and require different memory segments. The availability of configurable security levels provides a benefit over requiring all memory to perform at the highest security level of confidentiality and authentication checking. The amount of on-chip memory required to store tags for authentication checking can be reduced if only sensitive external memory must be protected. Additionally, the latency and dynamic power of unprotected memory accesses are minimized since unneeded security processing is avoided.

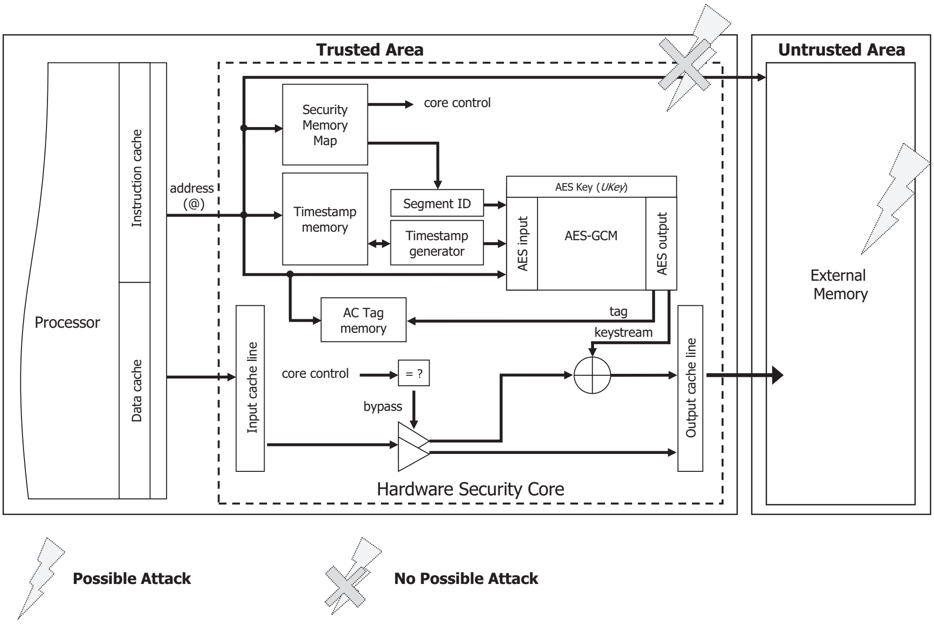

Fig. 3. Hardware security core architecture for a write request.

3.3. Memory Security Core Architecture

Our core for management of memory security levels is an extension of a preliminary version [Vaslin et al. 2008] which provides uniform security for all tasks and memory segments and uses one-time pad (OTP) operations [Suh et al. 2003] for confidentiality and an abbreviated AES sequence for authentication checking. Confidentiality in our new system employs AES-GCM [McGrew and Viega 2004; National Institute of Standards and Technology 2007] which has previously been proven to be secure. Unlike other cipher modes of operation, such as cipher block chaining (CBC), electronic code book (ECB), and counter mode (CTR), AES-GCM synchronizes and ensures confidentiality and authenticity (authenticated encryption) and can be both pipelined and parallelized.

Rather than encrypting write data directly, our approach generates a keystream using AES, which operates using a secret key stored inside the FPGA. In our implementation, a timestamp (TS) value, the data address, and the memory segment ID of the data are used as inputs to an AES-GCM encryption circuit to generate the keystream. This keystream is then XORed with the data to generate ciphertext which can be transferred outside the FPGA containing the microprocessor. The timestamp is incremented during each cacheline write. The same segment ID is used for all cachelines belonging to a particular application segment. Like previous OTP implementations [Suh et al. 2003], a benefit of this $Exec_{GCM}$ policy based on AES-GCM versus direct data encryption of the write data can be seen during data reads. The keystream generation can start immediately after the read address is known for read accesses. After the data is retrieved, a simple, fast XOR operation is needed to recover the plaintext. If direct data encryption was used, the decryption process would require many clock cycles after the encrypted data arrives at the processor. One limitation of this approach is a need to store the timestamp values for each data value (usually a cacheline) in on-chip storage so it can be used later to verify data reads against replay attacks. A high-level view of the placement of security blocks is seen in Figures 3 and 4.

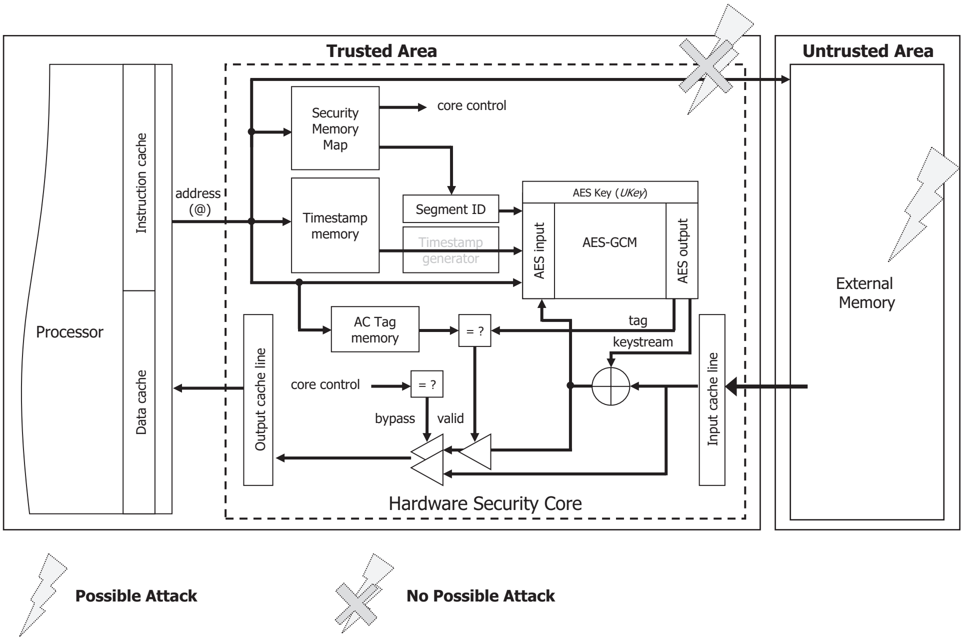

Fig. 4. Hardware security core architecture for a read request.

Figure 5 shows the AES-GCM operations necessary to cipher 256 bits of a plaintext cacheline (a similar scheme is applied for deciphering). The figure shows two 128-bit AES operations E_{Ukey} each using a 128-bit secret key, $UKey$. A 128-bit AES input includes the 32-bit timestamp (TS), the 32-bit data address (@), and the 64-bit memory segment ID (SegID). The 0 || $Len(C)$ value is a 128-bit word resulting from padding the length of ciphertext C with zero bits. Two 128-bit ciphertexts and a 128-bit authentication tag are generated from the two 128-bit plaintext input values. The tag is not ciphered since it is stored in secure on-chip embedded memory.

A step-by-step description of $Exec_{GCM}$ protected data writes and reads based on the AES-GCM block in Figure 5 are shown in Algorithms 1 and 2. From a security standpoint, it is essential that the keystream applied to encrypt data is used only one time. Since the keystream is obtained with AES, the AES inputs also need to be used just one time. If the same keystream is used several times, information leakage may occur since an attacker may be able to determine if data encrypted with the same keystream have the same values. The use of the data memory address in the generation of the keystream (Figures 3 and 4) protects the data value from relocation attacks. To prevent replay attacks, a simple 32-bit counter or linear feedback shift register (LFSR) counter is used for timestamp generation.

ALGORITHM 1: Cache Memory Write Request

```
1 − $Timestamp\ incrementation: TS = TS + 2$
2 − $\{Keystream, Tag\} = AES - GCM\{SegID, @, TS\}$
3 − $Ciphertext = Plaintext \oplus Keystream$
4 − $Ciphertext \Rightarrow external\ memory$
5 − $Timestamp\ storage: TS \Rightarrow TS\ memory(@)$
6 − $Authentication\ Tag\ storage: Tag \Rightarrow Tag\ memory(@)$
```

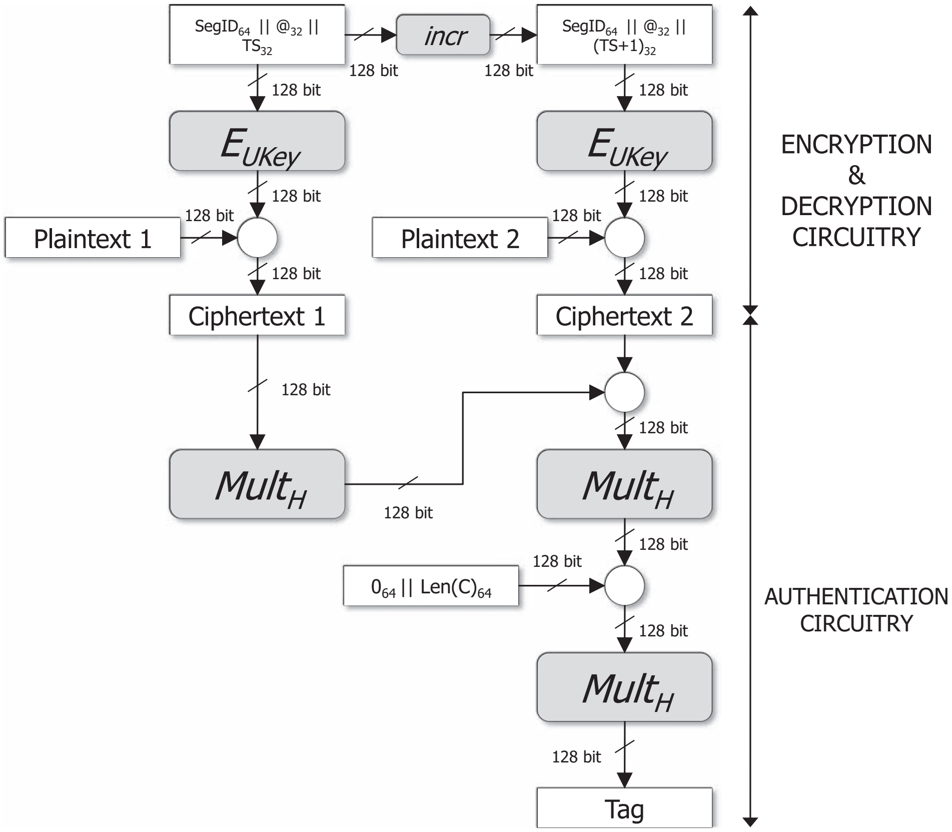

Fig. 5. The AES-GCM architecture for an authenticated encryption operation of a 256-bit plaintext cacheline. $Mult_H$ denotes Galois field multiplications in $GF(2^{128})$ by the hash key H, and *incr* denotes an increment function. The symbol || denotes concatenations between words. This figure was adapted from National Institutes of Standards and Technology [2007].

As shown in Algorithm 1, the 32-bit TS value is incremented by 2 after each write to the memory since two separate TS values are used by the circuit in Figure 5. It is stored in the Timestamp memory based on the data memory address. For each new cacheline memory write request, the system will compute a different keystream since the value of TS is updated. During a read, the original TS value is used for comparative purposes (Algorithm 2). The retrieved TS value is provided to AES during the read request. This value is fetched from the Timestamp memory using the data memory address. The AES result will give the same keystream as the one produced for the write request and the encrypted data will become plaintext after being XORed (step 5 in Algorithm 2).

The use of an AES-GCM core allows for provably secure data authentication without a costly latency penalty. If two 128-bit AES cores are used to cipher or decipher 256-bit data values (Figure 5), the authenticated encryption and authenticated decryption process can be done in 13 cycles: 10 cycles for ciphering or deciphering and 3 cycles for authentication. Each multiplication in $GF(2^{128})$, when implemented using a fully parallel datapath with XOR and AND operations, takes 1 cycle [National Institute of Standards and Technology 2007]. XOR operations shown at $Mult_H$ inputs in Figure 5 are implemented directly in the $Mult_H$ core, so no additional cycles are required. This cycle count does not include the bus-based memory read/write latency.

ALGORITHM 2: Cache Memory Read Request

```
1 – $TS\,loading : TS \Leftarrow TS\,memory(@)$
2 – $Tag\,loading : Tag \Leftarrow Tag\,memory(@)$
3 – $\{Keystream,\ Tag\} = AES - GCM\{SegID,\ @,\ TS\}$
4 – $Ciphertext\,loading : Ciphertext \Leftarrow external\ memory$
5 – $Plaintext = Ciphertext \oplus keystream$
6 – $Authentication\,checking : Tag \equiv Tag$
7 – $Plaintext \Rightarrow cache\ memory$
```

Read-only data, such as processor instructions, do not require protection from replay attacks because these data are never modified. No TS values are needed for these data so the amount of on-chip TS memory space can be reduced accordingly. Read-only data may be the target of relocation attacks but the address used to compute the $Exec_{GCM}$ policy guarantees protection against these attacks. The use of TS and data addresses for $Exec_{GCM}$ policy protects read/write data 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 keystream. In both cases, data authentication will fail and the deciphered data will be considered invalid.

The need for unique TS creates a problem if the TS generation counter rolls over and starts reusing previously-issued TS. A typical solution to this issue involves the reencryption of stored data with new TS [Yan et al. 2006]. A solution which uses multiple TS generator counters [Yan et al. 2006] was proposed to address this issue. If a TS counter reaches its maximum value, only about half the data must be reencrypted. With our security approach, the same idea can be applied based on segment IDs. If a TS associated with a segment rolls over, the segment ID value is incremented. All the data included in the segment are reencrypted with the new segment ID value. The use of the segment ID in keystream generation helps avoid the issue of matching TS values in this case. If reencryption due to counter rollover is needed, only a portion of the external memory is affected. Although not considered in the current implementation, reencryption could commence in the background prior to counter overflow or larger TS values could be used for certain applications, limiting or eliminating down time. In our prototype, a 32-bit TS is used, indicating a need to reencrypt only after 2^{32} cacheline writes for a given memory segment. This event is expected to occur infrequently.

The tag of the cacheline to be encrypted (step 2 in Algorithm 1) is stored in the authentication check (AC) tag memory (step 6 in Algorithm 1). Later, when the processor core requests a read, the tag result of the final XOR operation is compared with the AC tag value stored in the memory (step 6 in Algorithm 2). If data is replayed, relocated, or modified, the tag of the retrieved value will differ from the stored value, so the attack is detected.

The storage required for AC tag values impacts the security level provided. To limit this storage without compromising the security, between 64 and 128 of the most significant bits (MSB) of the tag are kept for a 256-bit cacheline. For an n-bit tag, an attacker has a 1 out of 2^n probability of successfully modifying the deciphered value and achieving the original tag value. NIST ensures the security of AES-GCM authenticated encryption for tags of these sizes [National Institute of Standards and Technology 2007].

The sizes of the on-chip TS and AC tag memories represent an important overhead of our approach which can vary widely on a per-application basis. These overheads are calculated and analyzed for four applications in Section 6.3.

4. SECURE APPLICATION LOADING

As described in Section 2.1, at system power up or reset, application code must be loaded from flash memory. As part of the application loading process, the contents of flash memory are copied to main memory by the system microprocessor as soon as the processor's registers are configured. To maintain both the confidentiality and authentication of application code, instructions stored in flash must be appropriately protected via ciphering and authentication checking. In our secure system, two distinct scenarios are considered.

—*Application code loading.* In this scenario, the SMM is already loaded in hardware so that only application instruction loading to main memory is needed. This scenario may occur if the SMM is included in an FPGA bitstream.

—*Application code and SMM loading.* The SMM information must be loaded into a memory-based table adjacent to the microprocessor and application code must be loaded to main memory. SMM loading takes place first, followed by application code loading.

The details of these two scenarios are now described.

4.1. Secure Application Code Loading from Flash Memory

Our secure application approach for loading code from flash memory (Figure 6) uses an AES-GCM core in a similar fashion to the $Exec_{GCM}$ technique for main memory described in the previous section. In general, secure loading is less constrained than main memory protection leading to AES-GCM optimization. Since data writes and replay attacks are not an issue for embedded system flash memory, the segment-based timestamp and per-cacheline AC approach used for main memory exhibits unnecessary overhead for flash-based code. The need for address and segment data as AES-GCM inputs is eliminated. Thus, our new $Load_{GCM}$ policy only uses a single initial 32-bit timestamp and a 96-bit initialization vector (IV) which is unique for each application. These values replace the input to E_{Ukey} and *incr* in the upper left of Figure 5. Except for AES secret keys and the *IV/TS* inputs, the same AES-GCM circuitry used for $Exec_{GCM}$ main memory security is reused for $Load_{GCM}$ application instruction loading.

ALGORITHM 3: Application Loading

```
1 – The IV is copied to the $AES_{GCM}$ running the $Load_{GCM}$ policy
2 – The TS is copied to the $AES_{GCM}$ running the $Load_{GCM}$ policy
Pipelined loop (for all application code)
3 – The encrypted application code is copied to the $AES_{GCM}$ running the $Load_{GCM}$
policy
4 – The encrypted application code is decrypted with the $Load_{GCM}$ policy
5 – The decrypted data is encrypted with the $AES_{GCM}$ running the $Exec_{GCM}$ policy
6 – The encrypted data is copied in main memory
End Loop
7 – The application tag is compared with the one generated by the $AES_{GCM}$ running the
$Load_{GCM}$ policy. If the two tags match, the application is securely loaded and can be safely
used for secure execution
```

A secure hardware architecture for application code loading for an embedded system (Figure 6) uses both $Load_{GCM}$ and $Exec_{GCM}$ policies to securely load instructions from the flash to the main memory. This process can be done in a pipelined fashion with multiple or shared 128-bit AES cores used for the $Load_{GCM}$ and $Exec_{GCM}$ policies. As

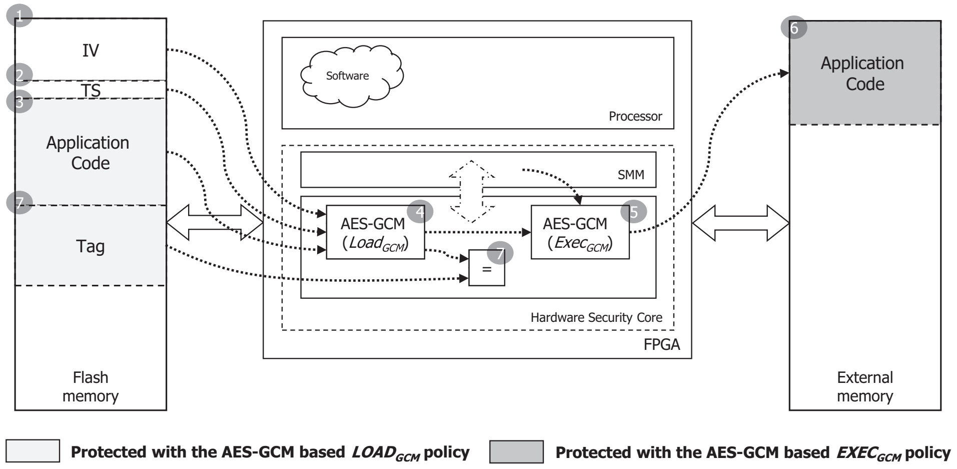

Fig. 6. Secure hardware architecture for application code loading. The SMM configuration is a part of the FPGA bitstream.

shown in Figure 6, while $Load_{GCM}$ circuitry deciphers data from the flash, $Exec_{GCM}$ policy write operations (outlined in Algorithm 1) are applied to the instructions before they are stored in main memory. Steps 3, 4, 5, and 6 in Algorithm 3 are performed repetitively instruction-by-instruction in a pipelined fashion until the application is loaded. When loading is complete, the 64-bit ciphered tag located in flash is checked against the tag generated by the AES-GCM block running the $Load_{GCM}$ policy to ensure application loading authentication (Step 7). Unlike the $Exec_{GCM}$ policy, which needs multiple tags to be stored in a secure on-chip memory (one per protected cache line), the single 64-bit tag stored in flash is used to authenticate application loading.

4.2. Application Code and SMM Loading from Flash Memory

Most microprocessor platforms require the capability to load and execute different applications at different times. This issue requires not only the initialization of an application, but also the SMM on a per-application basis. As mentioned in Section 4.1 for FPGA-based processors, the flexibility could be provided by loading a different bitstream which has an alternate SMM configuration for each application. An alternative strategy is to load both the SMM entries and the application code from flash memory.

The SMM configuration for an application is stored in flash memory in an application header (Figure 7). Like application code, headers provide important security information and must be protected. Figure 7 exhibits the layout of a memory block in flash which includes a protected application header and application code. The header contains information which is necessary to perform application loading including the SMM configuration, the application size, and the initial main memory load address. The header size will vary depending on the size of the SMM configuration. Specific components for the tested implementation include the following.

—A 96-bit initialization vector (IV).
—A 32-bit timestamp (TS).
—A 64-bit authentication check tag for SMM configuration.
—A SMM configuration containing the following.
  —A 32-bit application address (@).
  —A 32-bit application size.
  —64-bit values for each segment. Each value indicates the segment security level.

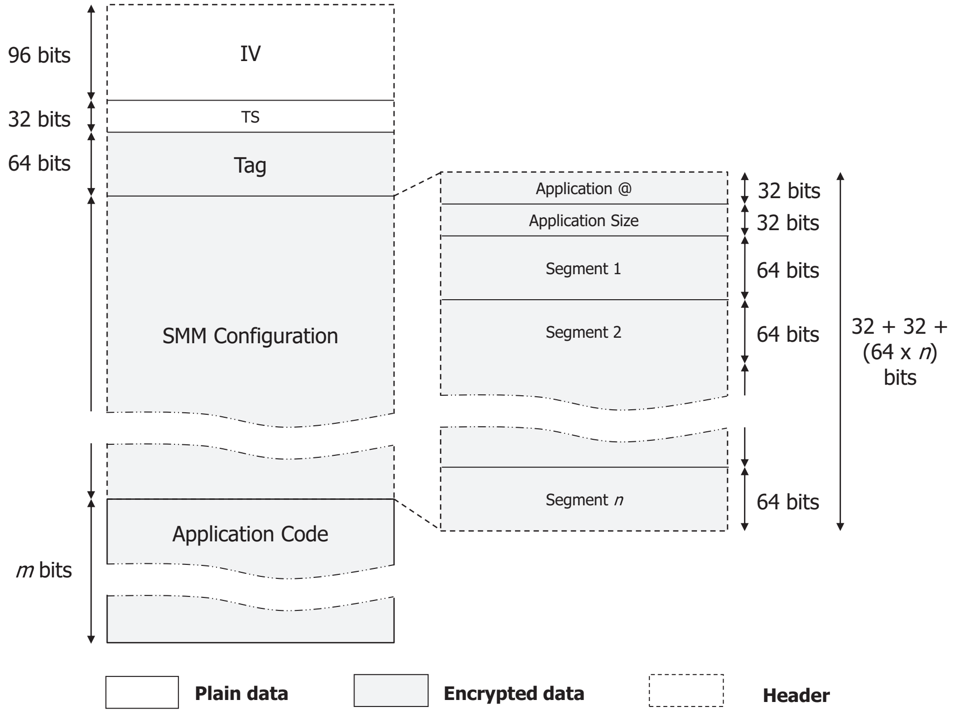

Fig. 7. Detailed flash application header, necessary to protect an application with the $Load_{GCM}$ policy.

The information in the block is loaded into the SMM using the $Load_{GCM}$ policy, necessitating the inclusion of an IV and a TS specifically for the SMM. Following the configuration of the SMM, the steps for secure hardware application loading, described in Section 4.1, are followed to complete system load. In the case of an $Exec_{GCM}$ policy architecture, the TS memory size, the AC tag memory size, and the number of memory segments supported by the SMM must be large enough to support the storage requirements of a target application. This approach is used to avoid the need to develop a new FPGA bitstream for each protected application.

5. EXPERIMENTAL APPROACH

An FPGA-based system including an architecture based on a Xilinx Microblaze processor [Xilinx Corporation 2009] was developed to validate our approach. Our security core and associated memory was implemented in FPGA logic and embedded memory and interfaced to the processor via a 32-bit processor local bus (PLB). In separate sets of experiments, the Microblaze was first allocated instruction and data caches of 512 bytes and then 2 kilobytes (KB). The widely-used MicroC/OS-II [LaBrosse 2002] embedded operating system was used to validate our approach. MicroC/OS-II is a scalable, preemptive and multitasking kernel. The OS can be configured by the designer during application design to use only the OS features that are needed. A priority-based scheduling approach is used to evaluate which one of up to 64 tasks runs at a specific point in time. MicroC/OS-II uses a hardware timer to produce ticks which force the scheduler to run.

To explore the impact of the security management on performance and area, four multi-task applications were used.

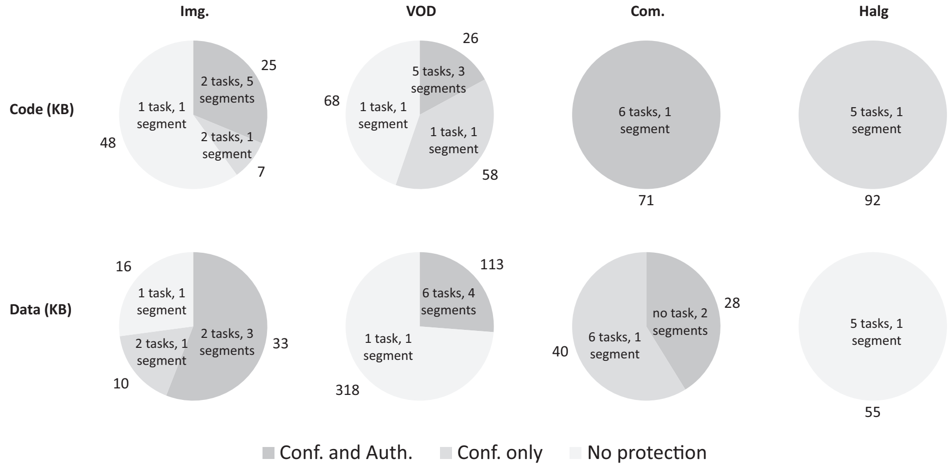

Fig. 8. Application memory protection details by protection level. The three protection levels include confidentiality and authentication (Conf. and Auth.), confidentiality-only (Conf. only), and no protection.

—*Image processing (Img).* This application selects one of two values for a pixel and combines the pixels into shapes. Pixel groups that are too small are removed from the image. This process is sometimes called morphological image processing [Dougherty and Lotufo 2003].

—*Video on demand (VOD).* This application includes a sequence of operations needed to receive transmitted encrypted video signals. Specific operations include Reed Solomon (RS) decoding, AES decryption, and Moving Picture Experts Group 2 (MPEG-2) decoding with artifact correction.

—*Communications (Com).* This application includes a series of tasks needed to send and receive digital data. Specific operations include Reed Solomon decoding, AES encryption, and Reed Solomon encoding.

—*Halg.* This application can perform selective hashing based on a number of common algorithms. Supported hash algorithms include Message Digest (MD5), secure hash algorithm (SHA-1), and SHA-2.

The security requirements of portions of the application were estimated based on their function. Other security assignments than the ones listed could also be possible, although they are not explored in this work. For the image processing application, image data and application code used to filter data is protected, but data and code used to transfer information to and from the system is not. For the video on demand application, deciphered image data and AES specific information (e.g. the encryption key) is considered critical. Also, the MPEG algorithm is considered proprietary and its source code is encrypted, while MPEG data and RS code and data are left unprotected. For the communications application, all data is considered sensitive and worthy of protection. In order to guarantee correct operation, the code must not be changed, so confidentiality and authentication checking is applied to all code. Application data is only protected for confidentiality. Halg application code is only encrypted (confidentiality) and application data has no protection. For example, a company may wish to protect its code from visual inspection. Since there is no need for authentication checking for this application, no storage for TS or tag values is needed.

Figure 8 summarizes the tasks, external memory count, and number of memory segments for the applications. As noted in Section 3.1, memory segments may be of variable sizes. All four applications were successfully implemented on a Spartan-6

SP605 evaluation platform ([Xilinx Corporation - DS160 2010; Xilinx Corporation - UG526 2010) containing a XC6SLX45T FPGA device, 128 MB of external DDR3 memory, and 32 MB of flash memory. Area and embedded memory counts were determined following synthesis with Xilinx Platform Studio 12.2.

6. EXPERIMENTAL RESULTS FOR MAIN MEMORY SECURITY

For comparison purposes, a Microblaze-based system without a security core was synthesized to a Spartan-6 based FPGA. An XC6SLX45T FPGA contains 43,661 look-up tables (LUTs), 54,576 flip-flops (FFs), 116 18-kilobit (Kb) block RAMs (BRAMs), and 58 digital signal processing (DSP) slices. Our base configuration includes data and instruction caches, a timer, flash memory controller, DDR-SDRAM memory controller, and a Joint Test Action Group (JTAG) interface. After synthesis with XPS 12.1 it was determined that the base configuration with 512 byte caches consumes 3,610 LUTs and 2,647 FFs, and operates at a clock frequency of 75 megaHertz (MHz). A base configuration with 2 KB caches requires 3,335 LUTs, 2,538 FFs, and 4 additional 18-Kb BRAMs. It operates at 86 MHz.

As stated in Section 3, the availability of a security core which allows for different security levels for different memory segments provides for security versus resource tradeoffs. In our analysis we consider three specific scenarios.

—*No protection (NP)*. This is the base Microblaze configuration with no memory protection.
—*Programmable protection (PP)*. This Microblaze and security core configuration provides exactly the security required by each application memory segment (Section 5).
—*Uniform protection (UP)*. This Microblaze and security core configuration provides the highest level of security required by a memory segment to all memory segments. Since all segments use the same security level, the SMM size is reduced.

The logic overhead of the security core in the programmable protection case is not constant since the size of the SMM depends on the number of defined security areas. For the uniform protection case, logic overhead variations result from differences in the control circuitry required for the AC tag storage.

6.1. Area Overhead of Security

As shown in Table II for configurations with 512 byte caches, in most cases the hardware security core (HSC) logic required for programmable protection is similar to uniform protection. A detailed breakdown of the size of individual units in the HSC is provided in Table III for both uniform and programmable protection. It is notable that the resources required for AC tag storage for the programmable protection version of **VOD** is reduced versus uniform protection since the amount of AC tag storage is reduced. For the Halg application, authentication checking is not performed for either uniform or programmable protection so no additional hardware is needed. The used percentage of total FPGA logic resources for each unit is also included in the table.

The $Exec_{GCM}$ implementation includes two 128-bit AES units with a single 128-bit key (as shown in Figure 5). The AES blocks (labeled E_{UKey} in Figure 5) are implemented using a balance of BRAMs, DSP slices, and logic slices [Drimer et al. 2010]. Although not shown in Table III, 16 BRAMs and 32 DSP slices are necessary for the two required 128-bit AES cores. Memory overheads associated with the approach are discussed in Section 6.3.

6.2. Performance Cost of Security

The run time of each Microblaze-based system for each application was determined using counters embedded within the FPGA hardware. Table IV shows the run time of

Table II. Architecture Synthesis Results and Overall XC6SLX45T Device Logic Resources Usage for Different Security Levels

	Uniform protection				Programmable protection			
Arch.	μB + HSC		HSC		μB + HSC		HSC	
	LUTs	FFs	LUTs	FFs	LUTs	FFs	LUTs	FFs
Img. 512	7095	3769	3485	1122	7237	3796	3627	1149
	16.3%	6.9%	8.0%	2.1%	16.6%	7.0%	8.3%	2.1%
Img. 2k	6820	3660	3485	1122	6962	3687	3627	1149
	15.6%	6.7%	8.0%	2.1%	15.9%	6.8%	8.3%	2.1%
VOD 512	7229	3796	3619	1149	7209	3792	3599	1145
	16.6%	7.0%	8.3%	2.1%	16.5%	6.9%	8.2%	2.1%
VOD 2k	6954	3687	3619	1149	6934	3683	3599	1145
	15.9%	6.8%	8.3%	2.1%	15.9%	6.7%	8.2%	2.1%
Com. 512	7080	3768	3470	1121	7120	3776	3510	1129
	16.2%	6.9%	7.9%	2.1%	16.3%	6.9%	8.0%	2.1%
Com. 2k	6805	3659	3470	1121	6845	3667	3510	1129
	15.6%	6.7%	7.9%	2.1%	15.7%	6.7%	8.0%	2.1%
Halg 512	6186	3598	2576	951	6153	3596	2543	949
	14.2%	6.6%	5.9%	1.7%	14.1%	6.6%	5.8%	1.7%
Halg 2k	5911	3489	2576	951	5878	3487	2543	949
	13.5%	6.4%	5.9%	1.7%	13.5%	6.4%	5.8%	1.7%

Table III. Detailed Breakdown of Hardware Security Core (HSC) Logic Resource Usage

Uniform protection										
App.	Total		AES_{GCM}		AC Tag Storage		SMM		Ctrl.	
	LUTs	FFs	LUTs	FFs	LUTs	FFs	LUTs	FFs	LUTs	FFs
Img.	3485	1122	2065	798	473	154	23	3	924	167
	8.0%	2.1%	4.7%	1.5%	1.1%	0.3%	0.1%	0.0%	2.1%	0.3%
VOD	3619	1149	2065	798	604	177	29	3	921	171
	8.3%	2.1%	4.7%	1.5%	1.4%	0.3%	0.1%	0.0%	2.1%	0.3%
Com.	3470	1121	2065	798	470	153	20	3	915	167
	7.9%	2.1%	4.7%	1.5%	1.1%	0.3%	0.0%	0.0%	2.1%	0.3%
Halg	2576	951	2065	798	0	0	21	3	490	150
	5.9%	1.7%	4.7%	1.5%	0.0%	0.0%	0.0%	0.0%	1.1%	0.3%
Programmable protection										
App.	Total		AES_{GCM} Core		AC Tag Storage		SMM		Ctrl.	
	LUTs	FFs	LUTs	FFs	LUTs	FFs	LUTs	FFs	LUTs	FFs
Img.	3627	1149	2065	798	473	153	214	31	875	167
	8.3%	2.1%	4.7%	1.5%	1.1%	0.3%	0.5%	0.1%	2.0%	0.3%
VOD	3599	1145	2065	798	473	153	161	27	900	167
	8.2%	2.1%	4.7%	1.5%	1.1%	0.3%	0.4%	0.0%	2.1%	0.3%
Com.	3510	1129	2065	798	475	154	61	10	909	167
	8.0%	2.1%	4.7%	1.5%	1.1%	0.3%	0.1%	0.0%	2.1%	0.3%
Halg	2543	949	2065	798	0	0	12	1	466	150
	5.8%	1.7%	4.7%	1.5%	0.0%	0.0%	0.0%	0.0%	1.1%	0.3%

Note: Percentage values indicate the used fraction of available XC6SLX45T FPGA resources.

Table IV. Application Execution Time and Performance Reduction

	No protection	Uniform protection		Programmable protection	
Arch.	Time (ms)	Time (ms)	Overhead	Time (ms)	Overhead
Img. 512	150.5	188.0	24.9%	173.4	15.2%
Img. 2k	131.3	156.9	19.5%	146.9	11.9%
VOD 512	13691.5	16806.4	22.8%	15619.8	14.1%
VOD 2k	11940.3	13751.2	15.2%	13453.5	12.7%
Com. 512	69.1	84.1	21.6%	78.7	14.0%
Com. 2k	60.2	66.7	10.8%	65.4	8.6%
Halg 512	8.6	10.2	18.9%	9.9	15.1%
Halg 2k	7.5	8.7	15.9%	8.6	14.4%

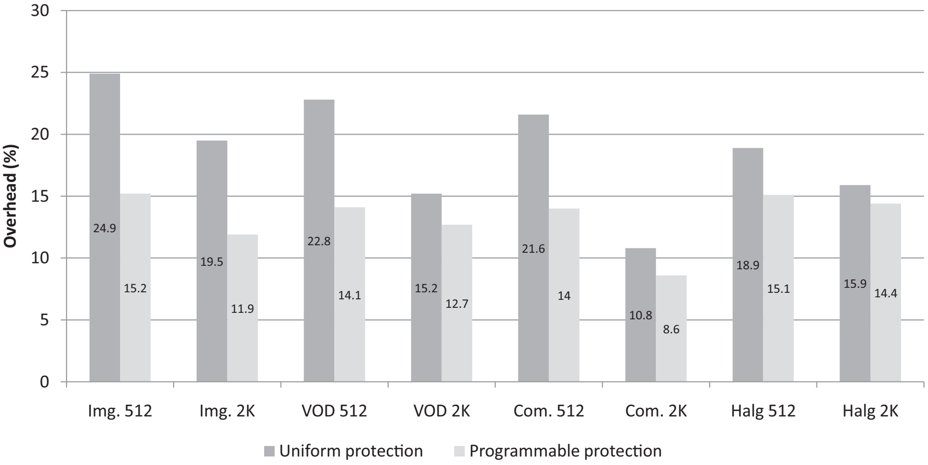

Fig. 9. Performance overhead of uniform and programmable protection versus no protection for four applications.

each application in each configuration and an assessment of performance loss versus the base configuration. Experiments were performed for all three security approaches using both 512 bytes and 2 KB caches. The 32-bit PLB bus requires six 75 MHz cycles for both reads and writes. The extra latency caused by our security approach for the prototype implementation is 7 cycles for a 256-bit cacheline read and 13 cycles for a cacheline write. The cacheline write overhead is primarily due to the 10-cycle 128-bit AES operation in $Exec_{GCM}$. The read overhead is reduced due to an overlap in $Exec_{GCM}$ and bus read operations. The percentage performance loss due to security is higher for configurations which include smaller caches. This is expected, since smaller caches are likely to have a larger number of memory accesses, increasing the average fetch latency. Some per-application variability is seen. Both image processing and VOD applications show a substantial performance reduction (25% and 23%, respectively) with uniform protection even though both contain data segments which require no protection. The use of programmable protection allows these data segments to have less of an impact on application performance. More modest performance reductions (15% and 14%, respectively) are reported for these configurations. The overall effects of our approaches on performance are summarized in Figure 9.

Note that for the 2 KB cache versions of the communications application, the performance loss for the programmable protection version is only 2% less than the uniform protection version. Figure 8 shows that all data and code for this application must be protected with either confidentiality or confidentiality and authentication, so the benefit of programmability is limited.

6.3. Memory Cost of Security

As stated in Section 5, the memory overhead of main memory security is the result of on-chip storage of TS and authentication tags. Equation (1) (Security memory equations) provides the formulae needed to obtain the required amount of on-chip memory to store these values.

For our experimentation, the cacheline size is 256 bits, the AC tag size is 64 bits, and the TS size is 32 bits. Using the values from Figure 8, it is possible to determine the size of required on-chip memory based on the selected security policy. An example of TS and tag overhead calculation is shown for the image processing application with programmable protection. Figure 10 assesses the on-chip memory overhead of security. For the VOD application, 150 KB of on-chip memory are saved by using programmable protection rather than uniform protection. The large savings primarily result from the presence of a large unprotected memory segment in the VOD application which does not require protection. Note that the programmable protection version of the Halg application does not require any memory storage since no data values require authentication protection and TS values are not needed for read-only application code.

Equations 1 - ***Security Memory Equations***

Size of AC Tag memory for code:
$1 - AC\ overhead = \frac{AC\ tag\ size}{cacheline-size}$
$2 - AC\ code = total\ code \times AC\ overhead$

Size of AC Tag memory for data:
$3 - AC\ data = total\ data \times AC\ overhead$

Size of TS memory for data:
$4 - TS\ overhead = \frac{TS\ size}{cacheline\ size}$
$5 - TS\ data = total\ data \times TS\ overhead$

Example for image processing with programmable protection:
$AC\ overhead = \frac{256}{64} = 0.25$
$AC\ code = 25KB \times 0.25 = 6.25\ KB$
$AC\ data = 33KB \times 0.25 = 8.25\ KB$
$TS\ overhead = \frac{32}{256} = 0.125$
$TS\ data = (33KB + 10KB) \times 0.125 = 5.4KB$

6.4. Comparison to Previous Approaches

In general, the performance of our approach compares favorably to previous security approaches shown in Table I. Although AEGIS [Suh et al. 2003] exhibited a smaller TS storage overhead of 6.25%, its integrity check storage overhead of 28% is similar to ours. A significant difference between the two approaches is the integrity checking latency. While AEGIS relies on SHA-1 which has a latency approaching 80 cycles, our new approaches uses recent AES-GCM authentication which requires 3 cycles for 256-bit inputs. The PE-ICE (Parallelized Encryption and Integrity Checking Engine) approach [Elbaz et al. 2006] has a reported 33% memory overhead and a 15% performance

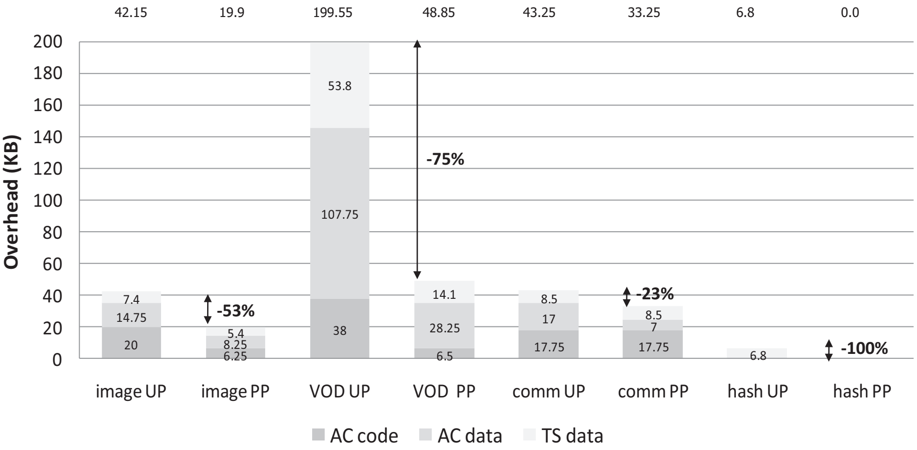

Fig. 10. On-chip security memory footprint for timestamp (TS) and authentication check (AC) tags for uniform protection (UP) and programmable protection (PP). The arrows indicate the percentage on-chip memory savings for programmable versus uniform protection.

penalty for a reduced brute force security level of $\frac{1}{2^{32}}$. Our brute force security level of $\frac{1}{2^{64}}$ (discussed in Section 3.3), although less than XOM (eXecute Only Memory) and AEGIS, is still appropriate for a number of embedded applications, as indicated in Appendix C in National Institute of Standards and Technology [2007]. Our approach requires on-chip AC tag storage which may be limiting for some embedded platforms where most segments must be protected. XOM, PE-ICE, Yan-GCM, and AEGIS allow for a combination of on-chip and off-chip storage of security information. However, the recent expansion in on-chip memory for FPGAs limits the impact of this issue. The brute force security level of our approach could be doubled to $\frac{1}{2^{128}}$ if on-chip AC storage is doubled, although that option was not explored in this work.

7. EXPERIMENTAL RESULTS FOR APPLICATION LOADING SECURITY

The following results consider the area and performance costs related to securely loading and executing a given application. The two memory loading cases considered in Section 4, application code only and application code and SMM loading, are described for systems requiring both uniform and programmable protection.

7.1. Example Hardware and Delay Costs for Application Code Loading from Flash

The Xilinx board described in Section 5 was used to validate our secure system approach. In our developed prototype, application code is read from the flash, decrypted using the $Load_{GCM}$ policy, and reencrypted using the $Exec_{GCM}$ policy. The performance and flash memory cost of $Load_{GCM}$ policy varies on a per-application basis. The load time is directly related to the size of the protected application. In this experiment, all data fetches and operations from flash take place at 85 MHz. A total of 580 cycles are required to read 256 bits from flash memory in 8-bit chunks. Table V shows a breakdown of the time needed to perform application loading from the flash up to the point the code is sent to the DDR-SDRAM. Results are shown for no protection, uniform protection, and programmable protection for the four designs evaluated in Section 6. Code fetch delays from flash memory are shown for the no protection case. $Load_{GCM}$ policy decryption and $Exec_{GCM}$ encryption delays are also shown for the other two cases. In our implementation, the same AES cores are multiplexed between $Load_{GCM}$

Table V. Secure Loading Time for Application Code Transferred from Flash to the Output of the $Exec_{GCM}$ Core

	No protection	Uniform protection				Programmable protection			
App.	Time (ms)	Time (ms)				Time (ms)			
		Load	Exec	Total	Overhead	Load	Exec	Total	Overhead
Img.	19.84	20.36	1.51	21.87	10.21%	20.36	1.05	21.41	7.92%
VOD	37.32	38.30	2.87	41.17	10.32%	38.30	2.41	40.71	9.07%
Com.	17.46	17.92	1.34	19.26	10.33%	17.92	1.34	19.26	10.33%
Halg	22.48	22.91	1.73	24.64	9.62%	22.91	1.73	24.64	9.62%

Note: Load indicates the amount of time to load the application from flash and decrypt/authenticate it using the $Load_{GCM}$ policy. Exec indicates the amount of time needed to reencrypt and generate authentication tags for the application using the $Exec_{GCM}$ policy.

Table VI. Flash Application Header Overheads

	Application header and SMM
Application	Size (bytes)
Img.	128
VOD	112
Com.	64
Halg	48

and $Exec_{GCM}$ operations, and for security needs, two distincts 128-bits keys are used, one for each policy circuitry. A total of 1024 LUTs are needed for $Load_{GCM}$ control and AES-GCM multiplexing. The combined time of a pipeline of these steps is limited by the fetch time from flash, so the overall application loading time is roughly equivalent to the flash load time for each application. There is a 9% load time penalty on average for the applications due to the need for memory protection.

7.2. Example Hardware and Delay Costs for Application Code and SMM Loading from Flash

The area overhead required for application-specific SMM loading is minimal compared to the cost of securing memory. The extra flash storage required to hold application header information, including the SMM configuration, for each application is shown in Table VI. Compared to the target applications, the load time and decryption for this additional information is negligible. Since the SMM must be writable to support configuration, an 18-Kb internal FPGA BRAM is used to hold the 15 SMM memory segments needed by the largest application of our testbench. The LUT count of the SMM control module is increased from 214 (Table III) to 252.

8. CONCLUSIONS AND FUTURE WORK

In this paper we present a security approach for external memory in embedded systems. The approach provides a low-overhead implementation of authenticated encryption for embedded systems based on the NIST-approved AES-GCM policy. The approach minimizes logic required for authentication and takes advantage of increased internal FPGA memory storage. Selective confidentiality and encryption is provided for different parts of an application without the need for additional microprocessor instructions or extensive operating system modifications. The benefits of our security core are demonstrated and quantified using four embedded applications implemented on a Spartan-6 FPGA. The size and performance penalties of the lightweight circuitry included to support secure application loading from external memory are also quantified.

Several opportunities exist for future work. Our approach could be evaluated for authentication-only main memory security in addition to the three levels described here. The work could be extended to target ASICs by considering secure key distribution techniques and fixed size tag and SMM storage. A detailed analysis of appropriate tag and SMM storage size for a variety of applications would be required. An additional optimization would be to store some or all TS and tag values off-chip. A complicated mechanism would be required to ensure that this information is not compromised by an attack.

REFERENCES

Altera Corporation 2008. FPGA Design Security Solution Using a Secure Memory Device Reference Design. Altera Corporation.

Alves, T. and Felton, D. 2004. TrustZone: Integrated Hardware and Software Security. ARM White Paper.

Anderson, R. 2001. *Security Engineering*. John Wiley & Sons, Inc., New York, NY.

Arbaugh, W., Farber, D., and Smith, J. 1997. A secure and reliable bootstrap architecture. In *Proceedings of the IEEE Symposium on Security and Privacy*. 65–71.

Badrignans, B., Elbaz, R., and Torres, L. 2008. Secure FPGA configuration architecture preventing system downgrade. In *Proceedings of the International Conference on Field-Programmable Logic and Applications*. 317–322.

Dietrich, K. and Winter, J. 2008. Secure boot revisited. In *Proceedings of the International Conference for Young Computer Scientists*. 2360–2365.

Dougherty, E. R. and Lotufo, R. A. 2003. *Hands-on Morphological Image Processing*. SPIE Press, New York, NY.

Drimer, S., Güneysu, T., and Paar, C. 2010. DSPs, BRAMs, and a pinch of logic: Extended recipes for AES on FPGAs. *ACM Trans. Reconfigurable Technol. Syst. 3,* 1, 1–27.

Elbaz, R., Torres, L., Sassatelli, G., Guillemin, P., Bardouillet, M., and Martinez, A. 2006. A parallelized way to provide data encryption and integrity checking on a processor-memory bus. In *Proceedings of the IEEE/ACM International Design Automation Conference*. 506–509.

Gassend, B., Suh, G. E., Clarke, D., van Dijk, M., and Devadas, S. 2003. Caches and Merkle trees for efficient memory integrity verification. In *Proceedings of the International Symposium on High Performance Computer Architecture*. 295–306.

Heath, C. and Klimov, A. 2006. A foundation for secure mobile DRM embedded security. *Wireless Design Magazine*, 32–34.

LaBrosse, J. 2002. *MicroC/OS-II: The Real-Time Kernel*. CMP Books, San Francisco, CA.

Lee, K. and Orailoglu, A. 2008. Application specific non-volatile primary memory for embedded systems. In *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis*. 31–36.

Lie, D., Thekkath, C., and Horowitz, M. 2003. Implementing an untrusted operating system on trusted hardware. In *Proceedings of the ACM Symposium on Operating Systems Principles*. 178–192.

McGrew, D. and Viega, J. 2004. The Galois/Counter Mode of Operation (GCM). Submission to NIST Modes of Operation Process.

National Institute of Standards and Technology 2007. Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and (GMAC). National Institute of Standards and Technology. Special publication 800-38D.

Pasotti, M., Sandre, G. D., Iezzi, D., Lena, D., Muzzi, G., Poles, M., and Rolandi, P. L. 2003. An application specific embeddable flash memory system for non-volatile storage of code, data and bit-streams for embedded FPGA configurations. In *Proceedings of the Symposium on VLSI Circuits*. 213–216.

Suh, G. E., Clarke, D., Gassend, B., van Dijk, M., and Devadas, S. 2003. Efficient memory integrity verification and encryption for secure processors. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*. 339–350.

Suh, G. E., O'Donnell, C. W., Sachdev, I., and Devadas, S. 2005. Design and implementation of the AEGIS single-chip secure processor using physical random functions. In *Proceedings of the International Symposium on Computer Architecture*. 25–36.

Vaslin, R., Gogniat, G., Diguet, J.-P., Tessier, R., Unnikrishnan, D., and Gaj, K. 2008. Memory security management for reconfigurable embedded systems. In *Proceedings of the IEEE Conference on Field Programmable Technology*. 153–160.

Xilinx Corporation 2005. Lock Your Designs with the Virtex-4 Security Solution. Xilinx Corporation.

Xilinx Corporation 2009. Microblaze Processor Reference Guide. Xilinx Corporation.

Xilinx Corporation - DS160 2010. Spartan-6 Family Overview. Xilinx Corporation - DS160.

Xilinx Corporation - UG526 2010. SP605 Hardware User Guide. Xilinx Corporation - UG526.

Yan, C., Rogers, B., Englender, D., Solihin, Y., and Prvulovic, M. 2006. Improving cost, performance, and security of memory encryption and authentication. In *Proceedings of the International Symposium on Computer Architecture*. 179–190.

Received November 2010; revised May 2011; accepted September 2011