Cumulative Attestation Kernels for Embedded Systems

Michael LeMay and Carl A. Gunter

Department of Computer Science
University of Illinois at Urbana-Champaign

Abstract. [1] There are increasing deployments of networked embedded systems and rising threats of malware intrusions on such systems. To mitigate this threat, it is desirable to enable commonly-used embedded processors known as flash MCUs to provide remote attestation assurances like the Trusted Platform Module (TPM) provides for PCs. However, flash MCUs have special limitations concerning cost, power efficiency, computation, and memory that influence how this goal can be achieved. Moreover, many types of applications require integrity guarantees for the system over an interval of time rather than just at a given instant. The aim of this paper is to demonstrate how an architecture we call a *Cumulative Attestation Kernel (CAK)* can address these concerns by providing cryptographically secure firmware auditing on networked embedded systems. To illustrate the value of CAKs, we demonstrate practical remote attestation for Advanced Metering Infrastructure (AMI), a core technology in emerging smart power grid systems that requires cumulative integrity guarantees. To this end, we show how to implement a CAK in less than one quarter of the memory available on low end AVR32 flash MCUs similar to those used in AMI deployments. We analyze one of the specialized features of such applications by formally proving that remote attestation requirements are met by our implementation even if no battery backup is available to prevent sudden halt conditions.

1 Introduction

Networked embedded systems are becoming increasingly common and important. The networking of these systems often enables updating of firmware in the field to correct flaws or add functionality. This updating also introduces security threats if adversaries are in a position to use it to install malware. A good example of this trend is in the deployment of Advanced Metering Infrastructure (AMI), a centerpiece of “smart grid” technology in which networked power meters are used to collect, process, and transmit electrical usage data, and relay commands from utilities to intelligent appliances. Meters are required to support remote upgrades, since physical service visits are too expensive. Threats to the updates on this infrastructure are severe since meters are a common target of exploits aimed at electrical service theft. This type of threat will arise in many other contexts as well when remote sensing systems become more pervasive.

For such systems one would like something like the Trusted Platform Module (TPM) to provide remote attestation so that the embedded infrastructure can be efficiently and

[1] European Symposium on Research in Computer Security, Saint Malo, France

securely queried for its configuration [2]. This configuration information can be examined to detect intrusions resulting in the installation of malware. However, there are a variety of challenges to extending the concept of remote attestation for personal computers to work for embedded systems. Among these are the cost, power, memory, and computational limitations of embedded systems and the need to provide audit data over an interval of time rather than just at a given point in time. These requirements can be seen in the planned AMI deployments which envision millions of remotely monitored systems based on inexpensive *flash MicroController Units (MCUs)*, which are integrated circuits containing a microprocessor core, integrated flash memory for storing a program, data RAM, and other peripherals. They are required to reliably provide high-integrity billing data over a lifetime of 10-15 years and support remote updates of their firmware.

In this paper we describe an architecture for providing remote attestation on networked embedded systems. The architecture is called a Cumulative Attestation Kernel (CAK), which is implemented at a low level in the embedded system and provides cryptographically secure audit data for an unbroken sequence of firmware revisions that have been installed on the protected system, including the current firmware. The kernel itself is never remotely upgraded, so that it can serve as a static root of trust. Our specific objective is to show that CAKs can be practically achieved on flash MCUs. Only recently have inexpensive flash MCUs possessed the memory capacity and memory protection functions required to properly support a CAK. More expensive MCUs typically rely on external memory. Flash MCUs are also typically distinguished from high-end MCUs by their simple, monolithic firmware images containing a static set of applications that run in a single memory space. High-end MCUs often run a full-featured OS such as Linux. Finally, flash MCUs operate at low clock frequencies, and may not offer many of the features of high-end MCUs such as superscalar execution and a Memory Management Unit (MMU). We account for these characteristics of flash MCUs in our design.

We explore the feasibility of CAKs with respect to the requirements of advanced meters, since they represent an interesting application of flash MCUs. Although meters are connected to the power mains there is concern about their power usage since they may generate an undesirable drain on the power grid. To accommodate this, CAKs only consume energy when they are actually invoked and can be operated with acceptable efficiency. Another interesting peculiarity of the AMI application is that the long deployment lifetime means that it is infeasible to rely on battery backups over the complete lifetime of a typical meter. We demonstrate that CAKs are able to address this and a range of other such requirements using an implementation called Cumulative Remote Attestation of Embedded System Integrity (CRAESI). CRAESI is targeted at a mid-range Atmel AVR32 flash MCU equipped with a Memory Protection Unit (MPU). Our prototype is integrated with a practical advanced meter for illustration purposes. Since the battery backup assumption is unusual we formally verify that the CAK design for CRAESI is resilient to sudden, unexpected power loss.

Our contributions are as follows: *1)* requirements and design for CAKs that are fault-tolerant and respect the constraints of networked embedded systems based on flash MCUs, *2)* a prototype CAK implementation called CRAESI that satisfies these requirements, and *3)* formal proof that CRAESI has certain security and fault-tolerance properties. The paper is organized as follows. Section 2 contains additional background on illustrative security-critical embedded systems. In Section 3, we present the requirements for a CAK. Section 4 presents a design that satisfies those requirements. We present experimental results from CRAESI in Section 5. We formally analyze important properties of CRAESI in Section 6. Additional related work is discussed in Section 7. Finally, we conclude in Section 8.

2 Background

Remote Attestation. Remote attestation is the process whereby a remote party can obtain certified measurements of parts of the state of a system. There are a variety of protocols that can be used to accomplish this, but they usually involve at least two messages. The first message is a request from the remote party containing a nonce used to verify the freshness of the attestation results. The second message is from the system being attested to the remote party, containing a certified record of the system's state that incorporates the nonce provided by the remote party. Of course, the system must contain some set of components that is capable of securely recording and certifying the system's state. On desktop PCs, the TPM and supporting components in the system software often fulfill this role.

Flash MCUs. Trends in microcontroller technology have recently made our approach to providing remote attestation practical and useful. In the past, flash MCUs were most commonly available with 8-bit architectures, small memory sizes, and very limited memory protection. For example, the popular 8-bit megaAVR line of MCUs by Atmel contains parts with up to 256KiB of program memory and 8KiB of data memory. The main memory protection provided by those parts is a boot block in which the instructions for modifying the flash memory must reside.

Atmel introduced a line of 32-bit flash MCUs based on the AVR32 architecture that focus on low power consumption and high code density in April 2007. ST Microelectronics introduced the STM32 flash MCU line based on the ARM Cortex-M3 architecture with similar capabilities in June 2007. Certain older flash MCUs had large memories, but they typically did not include fine-grained memory protection hardware. Thus, the introduction of the AVR32 and similar processors illustrates that conditions are finally ripe for low-power processors with large memories and memory protection.

Since many applications originally developed to run on 8-bit MCUs do not yet require the memory protection supported by these new MCU architectures, it can easily be used to implement protected security functions on the MCU itself. Even if memory protection is commonly required by future embedded applications they can still be accommodated using virtualization. The ability to implement security on the MCU itself can eliminate the need for security coprocessors in some applications, particularly those that do not include hardware attacks in their threat models.

Advanced Metering Infrastructure (AMI). Advanced electric meters are embedded systems deployed by utilities in homes or businesses to record and transmit information about electricity extracted from the power distribution network. They arose out of Automated Meter Reading (AMR). Current plans of many utilities call for AMI with new applications envisioned based on bidirectional communications such as the ability to manipulate power consumption at a facility by sending a price signal or direct command to its meter. AMI networks are being deployed on a massive scale. Southern California Edison (SCE) recently filed a plan to deploy 5.3 million residential meters [1]. AMI is a particularly good example of an embedded sensor system and a good benchmark for study because of its nascent but real deployment and rich set of requirements.

The sophisticated functionality of advanced meters creates numerous attack scenarios and increases the likelihood that they will contain security vulnerabilities linked to firmware bugs. An outage of the meters in a region would likely entail a huge financial loss for a utility. The UtiliSec AMI-SEC AMI Task Force System Security Requirements call for code-auditing capabilities that can be provided by remote attestation [4]. In a previous work we further motivated the use of attestation to provide AMI security, but did not address the need for cumulative attestation or provide a design suitable for use on practical flash MCUs [16].

Other embedded systems also could benefit from CAK-supported intrusion detection. Modern Intelligent Electronic Devices (IEDs) used in electrical substations to monitor and control the transmission and distribution of electricity support firmware upgrades. As an example from another area, some car insurance companies are placing data loggers within cars in exchange for lower rates [23]. Those devices are prime targets for all kinds of tampering.

Formal Methods. Formal methods are used to verify correctness and fault-tolerance properties of the integrated CRAESI design in Section 6. Specifically, model checking is a methodology for systematically exploring the entire state space of a model and verifying that specific properties hold over that entire space. Maude is the name of a language as well as a corresponding tool that support model checking based on rewriting logic models and Linear Temporal Logic (LTL) properties [8]. Essentially, rewriting logic provides a convenient technique to express non-deterministic finite automata. Maude is a multi-paradigm language, and supports membership equational logic, rewriting logic, and even has a built-in object-oriented layer. We use Maude for our verification tasks.

3 Threat Model and Requirements

Threat Model. Data integrity on embedded systems can be compromised by malicious application firmware in various ways, as shown in Figure 1. A CAK can detect and report all three types of intrusions, whereas a remote attestation scheme that does not provide cumulative attestation and is invoked only when data is reported can only detect corruption caused by firmware running at that time, being vulnerable to Time-Of-Use-To-Time-Of-Check (TOUTTOC) inconsistencies. Similarly, cumulative attestation can provide assurance that actuator controls have not been abused in the past.

We assume that an attacker is capable of communicating with a protected system over a network and installing malicious application firmware. Well-designed systems include access control mechanisms to prevent unauthorized firmware from being installed, but we assume that those mechanisms can be overcome by attackers. This is in accordance with the principle of defense in depth.

“Ordinary” environmental phenomena must not cause any of the security requirements of the kernel to be violated. An example is an accidental power failure, unless the system has a robust, trusted power supply. On the other hand, a bit flip caused by cosmic radiation would be considered an extraordinary phenomenon in most ground-based embedded systems. These examples make it clear that the definitions of ordinary and extraordinary will vary based on a system’s intrinsic characteristics and environment. In this paper, we only include accidental power failures in our threat model. We also exclude physical attacks on microcontrollers such as fault analysis, silicon modifications, and probing [3, 12]. If such attacks are a concern, as they often are, then tamper-resistance techniques must be incorporated into the device’s packaging.

The security of our design is dependent upon the fact that application firmware runs at a lower privilege level than the CAK and is not permitted to access security-critical memory and peripherals, to exclude a wide variety of attacks, such as Cloaker [9]. The specific peripherals that are considered security-critical will vary between microcontrollers.

Common operating systems used on embedded systems do not fundamentally rely on memory protection, and their reliance on privileged peripherals can be accommodated through emulation or simple modifications, which makes our design suitable for them.

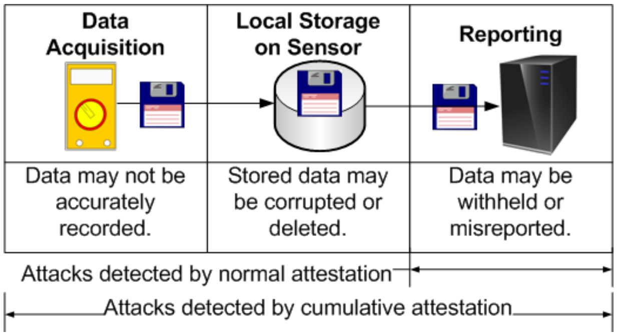

Data Acquisition
Local Storage on Sensor
Reporting
Data may not be accurately recorded.
Stored data may be corrupted or deleted.
Data may be withheld or misreported.
Attacks detected by normal attestation
Attacks detected by cumulative attestation

Fig. 1. *Three modes of attack available to malicious application firmware running during various lifetime phases occupied by sensor data.*

Requirements. The basic security and functional requirements for a CAK are that it maintain an audit log of application firmware revisions installed on an embedded system, and that it make a certified copy of that log available to authorized remote parties that request it. It must satisfy the following properties to provide security: *1) Comprehensiveness:* The audit log must represent all application firmware revisions that were ever active on the system. Application firmware is considered to be active whenever the processor’s program counter falls somewhere within its active code space. *2) Accuracy:* Whenever application firmware is active, the latest entry in the audit log must correspond to that firmware. The earlier entries must be chronologically ordered according to the activation of the firmware revisions they represent.

We define the following requirements for a broadly-applicable embedded CAK based on the characteristics and constraints of many embedded systems. The importance of each requirement varies between systems. *1) Cost-Effectiveness:* Low cost devices in competitive markets are unable to tolerate even the smallest unjustified expense. *2) Energy-Efficiency:* Some embedded systems are critically constrained by limited energy supplies, often provided by batteries. Even embedded systems attached to mains power may be constrained to low energy consumption to reduce energy costs. *3) Suitability for Hardware Protections:* The CAK must be adapted to the protection mechanisms provided by the embedded system's processor.

4 Design

We now present a general design that satisfies the requirements. The persistent memory (NVRAM) conceptually available to the kernel is divided into several regions, and contains the following data: *1)* A list of cryptographic hashes for all application firmware revisions installed, arranged chronologically and with a maximum size dictated by the capacity of the NVRAM. If necessary, it includes a hash value representing a hash chain for the oldest application firmware versions installed that no longer fit in the NVRAM. An entry will also contain an event code if an exceptional event has occurred, such as an aborted upgrade attempt. The specific codes will vary between designs. *2)* A counter to record the number of entries currently represented in the audit log and hash chain. *3)* An asymmetric keypair used to sign the firmware audit log during attestation operations. *4)* An explicit state variable to control transactions. *5)* A master keypair, used to sign the other coprocessor public keys. *6)* A keypair used during Diffie-Hellman key exchanges. *7)* Two counters to record the number of signatures generated by each of the audit log and key exchange private keys. The keys will be automatically refreshed when these counters reach a threshold value.

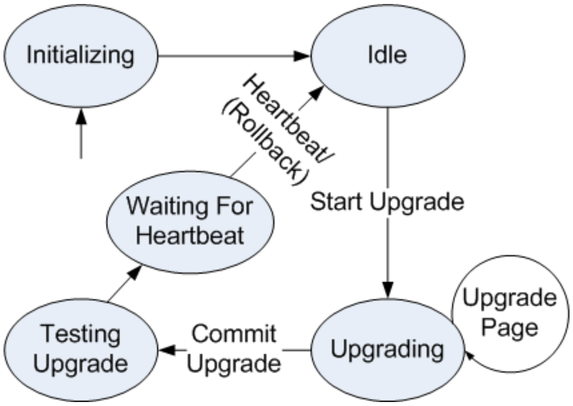


Fig. 2. *A basic state machine representation of CAK operation, in which transitions are generated by the specified commands.*

The master keypair is generated by the CAK using its built-in Random Number Generator (RNG) when it is first started and stored in memory, or burned into fuses at the factory in such a way that no entity, including the manufacturer, can determine its value. The master keypair is only used to sign the other two public keys, to preserve the cryptographic useful lifetime of the master keypair.

Since the audit log can overflow, the remote party performing the attestation must already know the sequence of hashes for those firmware images no longer contained in the audit log. This is a reasonable assumption if the embedded system is used by a group of remote parties that can communicate with all parties that have installed new firmware revisions on the system during the period of time in which the party verifying the attestation is interested, and if that party also knows the value of the hash chain immediately prior to that period. In that case, the party verifying the attestation can request that the updaters provide all the entries represented by the current hash chain after the checkpoint for which the verifier knows the hash chain value. It can then verify the current hash chain.

To satisfy the Comprehensiveness and Accuracy properties, it is most likely necessary for the kernel to control all access to the low-level firmware modification mechanisms in the system for the application firmware memory region. Figure 2 depicts the state machine that manages the application firmware upgrade process within the CAK. The transition labels not in parentheses are commands that can be issued by the application to cause itself to be upgraded. The explicit state variable records the current state. The "Waiting for Heartbeat" state causes the application firmware to be reverted to its previous revision if no heartbeat command is received within a certain period of time. Any unexpected command received by the CAK will be ignored.

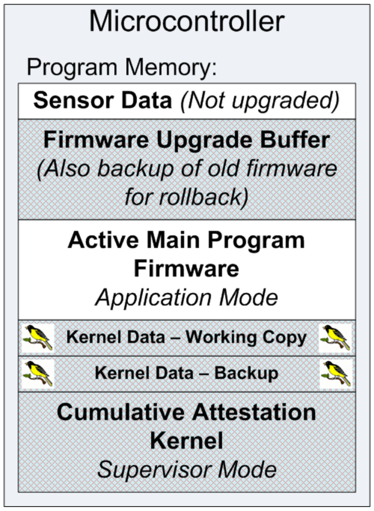

Microcontroller
Program Memory:
Sensor Data (Not upgraded)
Firmware Upgrade Buffer
(Also backup of old firmware for rollback)
Active Main Program Firmware
Application Mode
Kernel Data – Working Copy
Kernel Data – Backup
Cumulative Attestation Kernel
Supervisor Mode

Fig. 3. *The general CAK program memory layout. The birds represent canary values.*

Three additional commands not shown in the figure can be executed by an application to: "quote" the audit log by digitally signing and transmitting a copy including a nonce for freshness (*Quote*), retrieve the public keys signed using the master private key (*Retrieve Public Keys*), and perform a Diffie-Hellman key exchange (*Handshake*). The Handshake command demonstrates how the asymmetric cryptography implemented within the kernel can be used to perform operations directly useful to the application (establish a symmetric key with a remote entity, in this case), to defray the memory space that the CAK requires. More general access could be provided in future designs, but would complicate the security analysis of the API.

Transactional semantics must be provided for all the persistent data used by the kernel. This design accomplishes that by maintaining redundant copies of all persistent data in a static "filesystem" containing a fixed set of files that are referenced using absolute addresses. Both copies of the filesystem have canary values placed before and after the file data to support standard fault-tolerance techniques. The application firmware upgrade process is also fault-tolerant. The basic memory layout of the system, including conceptual canary locations, is depicted in Figure 3. Both fault-tolerance processes are analyzed in Section 6 to ensure that the particular memory manipulations we use correctly recover from accidental power failures.

Every time the embedded system boots, the processor immediately transfers control to the CAK. The CAK first initializes the memory protections, performs filesystem recovery if necessary, and completes the application firmware upgrade transaction if one was interrupted by a power failure. It then generates a cryptographic hash of the firmware and compares it to the latest audit log entry. If they differ, it extends the log with a new entry. Finally, it transfers control to the application.

Whenever a remote entity requests the audit log of application firmware revisions, the main program receiving the command sends a Quote command to the kernel, which then returns the audit log of firmware and a signature over it to link the audit log to the embedded system that generated it.

This design does not provide forward integrity, as an attacker that compromises either the master or attestation key can forge logs to indicate arbitrary system histories. A design providing deletion-detecting forward integrity would prevent attackers from undetectably modifying or deleting past entries [5]. However, this would require additional overhead such as a Message Authentication Code (MAC) per entry, additional entry data, and associated infrastructure. This would reduce the number of entries that the log could store, and is of questionable utility in some embedded system applications. In our AMI example, even a recent compromise can result in arbitrary data corruption. However, it is possible that forward integrity could be useful in certain applications, and our architecture could easily be modified to provide it.

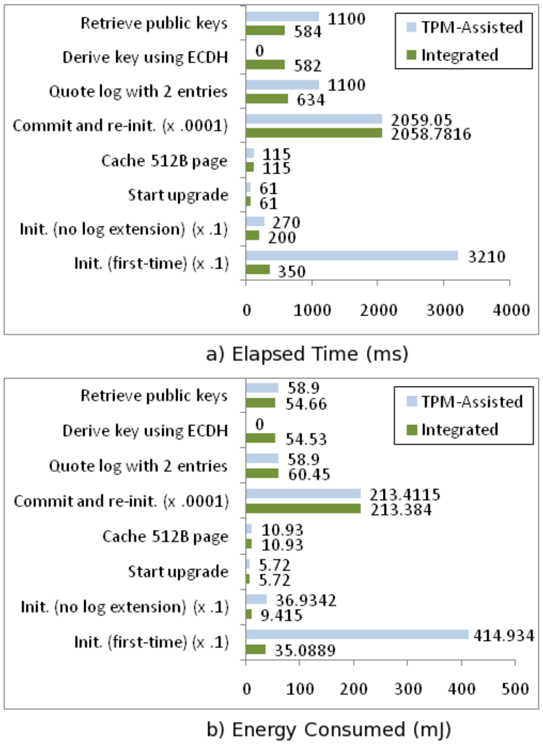
5 Implementation and Evaluation

In this section we present CRAESI, a prototype integrated CAK. The purpose of this prototype is to demonstrate that our design satisfies the practical requirements put forth in Section 4, and to obtain preliminary performance, cost, and power-consumption measurements. However, these preliminary measurements do not indicate the parameters that will be exhibited by commercial implementations, since our prototype relies heavily on unoptimized software.

Hardware Components. Our prototype implementation comprises five distinct devices. The first is an Atmel ATSTK600 development kit containing an AVR32 AT32UC3A0512 microcontroller with a 3.3V supply voltage. The second device is a Schweitzer Engineering Laboratories SEL-734 substation electrical meter. The SEL-734 has a convenient RS-232 Modbus data interface. We could have used any similar device in our experiments since it simply serves as a realistic data source connected to the AVR32 microcontroller. Third, we use a standard desktop PC to communicate with the AVR32 microcontroller over an RS-232 serial port from a Java application that issues Modbus commands. The final two devices are paired ZigBee radios that relay RS-232 data between the PC and AVR32 microcontroller.

Application Firmware. We prepared two application firmware images for our experiments. They both implement Modbus master and slave interfaces, where the master communicates with the meter over an RS-232 serial port, and the slave accepts commands from the PC over the ZigBee link and either passes them to the kernel or handles them directly if they are requesting data from the meter. The first image accurately relays meter data, whereas the second halves all meter readings, as might be the case with a malicious firmware image installed on an advanced meter by an unethical customer.

Kernel Firmware. The kernel is invoked whenever the processor resets, and by the application firmware when required. The AVR32 `scall` instruction is used to implement a simple syscall-style interface between the application and the kernel. TinyECC provides software implementations of SHA-1 hashing and Elliptic Curve Cryptography (ECC) [18]. They are not significantly optimized for AVR32. Note that the algorithms and key lengths used here may not be suitable for production use in systems with extended lifetimes during which the algorithms may be compromised. However, they are useful to illustrate the principles of our system. Pseudo-random numbers are generated by Mersenne Twister [19]. A commercial implementation would require a true RNG. Excluding the cryptography and the drivers provided by Atmel, the kernel comprises around 1,620 lines of C++, which includes 13 lines of inline assembly.



a) Elapsed Time (ms)

b) Energy Consumed (mJ)

Fig. 4. *A performance comparison of TPM-assisted and integrated CRAESI.*

The kernel consumes 81,312 bytes of program memory. We reserved 88KiB of flash memory to store the kernel code, and another 40KiB to store the persistent data manipulated by the kernel. 10,872 bytes of SRAM is used to store static data, 392B is dedicated to the heap, and 1KiB is dedicated to the stack. Thus, a total of 12KiB of SRAM is set aside for the kernel. Obviously, the memory consumed by the kernel is unavailable to the application, which does impose an added cost if it becomes necessary to upgrade to a larger microcontroller than would have been required without the kernel. In this prototype, the maximum application firmware image size is 191.5KiB. However, commercial kernel implementations will be significantly more compact in both flash and SRAM than our unoptimized prototype, and clever swapping schemes could effectively eliminate the SRAM consumption of the kernel when it is not active. The audit log in this implementation can record up to 107 upgrades and events before overflowing.

Performance Results. We now compare the energy and time consumed by our firmware-only prototype (integrated CRAESI) to that consumed by an Atmel AT97SC3203 TPM performing comparable operations (TPM-assisted CRAESI), since TPMs are currently popular devices used to implement remote attestation and could in fact be used by CRAESI to perform its cryptographic functions with some minor modifications to the design of CRAESI. We have not actually implemented TPM-assisted CRAESI, and used a TPM installed in a PC instead to perform comparable operations. The TPM has a supply voltage of 3.3V and relies on an LPC bus connection. We used Digital Multi-Meters (DMMs) that have limited sampling rates (100-300 ms between samples) to measure the energy consumption of both systems. This introduces some error into our calculations, so we have presented an upper-bound on the energy consumed by integrated CRAESI and a lower-bound on the energy consumed by TPM-assisted CRAESI. The time and energy consumed for a variety of operations is presented in Figure 4.

The TPM uses a 2048-bit RSA key to sign the PCRs, which provides security equivalent to a 224-bit ECC key, superior to the security of the 192-bit ECC keys used in integrated CRAESI. Due to the use of hardware, the TPM RSA signature generation mechanism is roughly as energy consumptive as the ECC software implementation in the integrated design. The Elliptic-Curve Diffie-Hellman key exchange supported by integrated CRAESI would not be supported by TPM-assisted CRAESI, although it could potentially be replaced with equivalent functionality.

The most significant efficiency drawback of the TPM is that it consumes 10.6mW when sitting idle. It may be possible to place the TPM into a deep sleep state to reduce this constant burden, but that is not done in practice in our test system, and may have unexamined security consequences.

Practical Implications of Experiments. As stated in Section 2, SCE is planning to deploy 5.3 million advanced meters in the short term. If AT97SC3203 TPMs were installed in all of those meters, they would consume 492,136 kWh per year, even if they sat idle at all times. In contrast, if integrated CRAESI were used instead, no energy would be consumed by CRAESI until a reset occurred or it was actually used. At $0.07/kWh, powering 5.3 million TPMs would cost around $34,450 per year.

Of course, the security coprocessors will not sit idle at all times. Let us assume that attestation is performed once per day per meter. In this case, TPMs would consume at least 31,651 kWh per year performing the quotation operations in addition to their idle energy consumption. Integrated CRAESI would consume less than 32,489 kWh per year performing comparable operations in addition to its negligible idle energy consumption.

6 Correctness and Fault-Tolerance Analysis

We used the Maude model checker to ensure that our design actually satisfies critical aspects of the security requirements put forth at the beginning of Section 4 [11]. First, we converted our design into a rewriting logic model, which represents transitions between states using rewrite rules. Then, we expressed aspects of the requirements for the design as theorems, which we converted into LTL formulas that were checked using a model checker. We discuss the outcome of this process in this section. The model checker did not discover any errors in the aspects of our implementation that we modeled, and thus increased our confidence that those aspects of the implementation are correct.

The model comprises several objects within modules that roughly correspond to the modules of functionality in the implementation. When the model is being used to check high-level properties, such as the correctness of the application firmware upgrade operations, the model assumes that any operation invoked on an object runs until completion without interruption. Without such an assumption, the state space that must be checked becomes intractable. However, that assumption does not necessarily hold in the real world, since power failures can occur and cause the processor to reset in the middle of any operation. Thus, we define rewrite rules that model power failures that can occur at arbitrary times in separate modules. We then use those modules to check that the system is fault-tolerant in the presence of power failures in representative scnarios.

A wide variety of theorems could be important, but we have selected the ones that deal with the parts of our design that have the most complex interactions, since it is most helpful to gain increased confidence in the correctness of those parts.

The first theorem is concerned with the correctness and auditability of application firmware upgrade procedures:

Theorem 1. *At the conclusion of any operation that modifies the active application firmware image, the audit log is updated to accurately reflect the new state. Additionally, the previous active application firmware image is cached if an elective upgrade is performed (not a rollback).*

Proof. We must check that all possible upgrade and rollback operations are correct, and that the firmware audit log is properly updated after each operation. We examine six distinct cases for upgrade and rollback operations in the following five lemmata. Taken together, these six cases are representative of all possible upgrade and rollback operations. In Lemma 6, we show that the firmware audit log is properly updated after every operation. □

We now discuss lemmata that the preceding theorem depends upon. To limit the state space, we are concerned with three distinct application firmware images, referred to as *image #0*, *image #1*, and *image #2*. The images are installed in order, and it is possible to jump directly from image #0 to image #2, or to halt without performing any upgrades. Since we are not concerned with the semantics of each image, but rather its identity, the upgrade transitions between these three images represent all possible upgrade operations.

Lemma 1 ensures that the initial application firmware on the device is not modified until a specific command to do so is received from the application.

Lemma 1. *If no upgrade operations are performed, then image #0 is active whenever the application is active.*

Lemma 2. *If image #1 has been installed, and no other upgrade or rollback operation has yet been performed, then image #1 is active and image #0 is cached whenever the application is active.*

This specifies that the image #0 is cached when replaced, and image #1 can be successfully activated at the proper time, and remains unmodified until the application firmware is upgraded to image #2, or it fails to send a heartbeat and is automatically rolled back to image #0.

Lemma 3 is similar, but handles transitions to image #2 from either image #0 or image #1.

Lemma 3. *If image #2 has been installed, replacing image #N, and no other upgrade or rollback operation has yet been performed, then image #2 is active and image #N is cached whenever the application is active.*

Lemma 4. *If image #0 is cached at the time that a rollback occurs, then whenever the application is active after the rollback until another upgrade operation occurs, image #0 is active.*

This specifies that the application firmware rollback action always operates as expected when rolling back to image #0.

Lemma 5 is similar, but handles rollback operations that restore image #1. If a rollback restores image #1, then it must be rolling back from an upgrade to image #2, which means that no further upgrades are possible within our model. Thus, this lemma does not include an allowance for further upgrade operations, as is the case in the previous lemma.

Lemma 5. *If image #1 is cached at the time that a rollback occurs, then image #1 is active whenever the application is active after the rollback.*

Lemma 6. *The current audit log entry corresponds to the active application firmware whenever the application is active.*

This states that the latest entry in the audit log is accurate whenever the application is running, ensuring that no undetected actions can be performed by the application. It does not verify the mechanism that is responsible for actually inserting new entries into the log and archiving old entries when the log overflows. That mechanism is consolidated into a short, isolated segment of code in the implementation that can be manually verified. The primary value of the model checker is in verifying portions of the implementation that interact in complex ways with other portions of the implementation and the environment.

The following theorem is used to ensure the fault-tolerant application firmware upgrade mechanism operates as expected. We modeled non-deterministic power failures, and allowed them to occur at any point in the upgrade process. The model checker exhaustively searched all combinations of power failures, and verified that the application firmware upgrade process always eventually succeeds as long as the power failures do not continually occur forever. Only one upgrade operation is modeled, because all upgrade operations are handled similarly regardless of identity and content. We tested this theorem on real hardware by pressing the reset button repeatedly during an upgrade and verifying that it still eventually succeeded, but of course we were not able to exhaustively test all possible points of interruption as the model checker did.

Theorem 2. *Executing any application firmware upgrade operation eventually results in the expected application firmware images being cached and active when the application is subsequently activated, regardless of how many times the processor is reset during the upgrade process, if the processor does not continually reset forever.*

The initial state for the model checking run of Theorem 2 represents the system running application firmware image #0 after an upgrade to image #1 has been cached and is about to be committed.

The following theorem is used to verify that the fault-tolerant persistent configuration data storage mechanism used by the kernel exhibits correct behavior. As in the previous theorem, non-deterministic power failures are modeled at every transition point in the model. We model only a single store-commit sequence, because all persistent data is handled identically regardless of identity and content. We tested this theorem on real hardware by setting breakpoints at critical locations in the filesystem code and forcing the processor to reset at those locations. Again, the model checker provides exhaustive testing, which is superior to our manual tests.

Theorem 3. *The filesystem correctly handles any transaction, regardless of how many times the processor is reset during a transaction, as long as the processor does not continually reset forever.*

Proof. We must show that transactional semantics are provided whether or not the transaction is interrupted prior to a critical point. The critical point occurs when the processor executes the instruction that invalidates the first canary in the redundant copy of the filesystem. Lemma 7 checks transactions that are interrupted prior to the critical point and Lemma 8 checks all other transactions. □

Lemma 7. *Executing any filesystem transaction eventually results in the original filesystem state if the transaction is interrupted prior to the critical point.*

Lemma 8. *Executing any filesystem transaction results in the filesystem state that is expected following the successful completion of the transaction if it is first interrupted after the critical point or is not interrupted at all.*

7 Related Work

The Linux Integrity Measurement Architecture (Linux-IMA) supports remote attestation of Linux platforms. It uses the TPMs that are being deployed in many modern desktop and laptop computers to record the configurations of those systems and provide a signed copy of that configuration information to authorized remote challengers [20]. It only maintains information about the configuration of a system since it was last reset.

The reference model provided by the Mobile Phone Working Group within the Trusted Computing Group deals with both configuration control and integrity measurement for mobile devices [21]. It recommends the use of a Mobile Local-owner Trusted Module (MLTM) to implement the functions of a TPM, although many of the TPM's operations are made optional to accommodate the resource constraints of mobile devices. It also recommends the use of a Mobile Remote-owner Trusted Module (MRTM) that is based on the design of the MLTM and also controls what code can run in certain regions of the system based on certificates. Such modules can be implemented in software, as has been shown using the ARM TrustZone hardware security extensions [24].

Terra synthesizes virtualization and attestation to provide application isolation and support for "closed-box" VMs that are observable via remote attestation [13]. We believe that such an architecture can be extended with cumulative attestation and is useful on embedded systems, as we have shown.

SWATT is an approach to verify the memory contents of embedded systems [22]. Its basic operating model assumes the existence of an external verifier that knows the precise type of hardware installed in the embedded system to be verified and that is connected to that system over a low-latency communications link, which is not available in many embedded system installations. It provides no intrinsic assurances of the continuous proper operation of embedded systems and requires that the system being verified not be able to offload computation to an external device (proxy attacks). Embedded systems often operate on networks where this assumption is not valid.

The ReVirt project has shown that it is feasible to maintain information on the execution of a fully-featured desktop or server system running within a virtual machine that is sufficient to replay the exact instruction sequence executed by the system prior to some failure that must be debugged [10]. DejaView uses a kernel-level approach to process recording to allow desktop sessions to be searched and restarted at arbitrary points [15]. It is conceivable that these techniques could support a CAK for desktops and servers, although it may not be feasible to store cumulative information for a long enough period of the system's life to be useful.

Attested Append-only Memory (A2M) maintains a cumulative record of logged kernel events in an isolated component to provide Byzantine-fault-tolerant replicated state machines [7]. Their architecture proposals are oriented towards server applications, but the paper provides examples of how attested information besides application firmware identity can be useful. The Trusted Incrementer project showed that the TCB for A2M and many other interesting systems can be reduced to a simple set of counters, cryptography, and an attestation-based API implemented in a trusted hardware component known as a "trinket" [17]. Our design could be adapted to provide similar functionality in firmware with a potentially different threat model.

One of the primary factors leading to the security issues in hardware security coprocessors is the complexity of their APIs [14]. To ease analysis and reduce the incidence of vulnerabilities our proposed design exports a very simple API. We have analyzed the security of that design using a model checker.

A previous methodology for explicitly modeling faults that can occur in systems and verifying that the systems tolerate those faults using a model checker only gives examples of logical faults, such as dropped messages [6]. We analyze the tolerance of our system against physical faults, such as power failures.

8 Conclusion

We present requirements for cumulative attestation kernels for embedded systems with flash MCUs to audit application firmware integrity. Auditing is accomplished by recording an unbroken sequence of application firmware revisions installed on the system in kernel memory, and providing a signed version of that audit log to the verifier during attestation operations. We have shown that this model of attestation is suitable for the applications in which sensor and control systems are used, and proposed a design for an attestation kernel that can be implemented entirely in firmware.

Our prototype cumulative attestation kernel is cost-effective and energy-efficient for use on mid-range 32-bit flash MCUs, and can be implemented without special support from microcontroller manufacturers. We used a model checker to verify that the prototype satisfies important correctness and fault-tolerance properties.

Acknowledgments

This work was supported in part by NSF CNS 07-16626, NSF CNS 07-16421, NSF CNS 05-24695, ONR N00014-08-1-0248, NSF CNS 05-24516, NSF CNS 05-24695, DHS 2006-CS-001-000001, and grants from the MacArthur Foundation and Boeing Corporation. Michael LeMay was supported on an NDSEG fellowship from AFOSR for part of this work. Musab AlTurki assisted us in developing our formal model. We thank Samuel T. King, Nabil Schear, Ellick Chan, the researchers in the Illinois Security Lab, and the anonymous reviewers for their feedback. We are grateful to the TCIP Center for its support of our efforts. We thank Schweitzer Engineering Laboratories for providing a substation meter that we used in our experiments. The views expressed are those of the authors only.

References

1. Southern california edison achieves key advanced metering goal. *Electric Energy Online.* `http://electricenergyonline.com/IndustryNews.asp?m=1\&id=71649`, Aug. 2, 2007.
2. TCG specification architecture overview. *Trusted Computing Group, http://www.trustedcomputinggroup.org/developers/trusted_platform_module/specifications*, Aug. 2, 2007.
3. R. Anderson and M. Kuhn. Low cost attacks on tamper resistant devices. In *Security Protocols*, volume 1361, pages 125–136. Springer, 1998.
4. B. Brown, et.al. AMI system security requirements. `http://osgug.ucaiug.org/utilisec/amisec/default.aspx`, Dec. 2008.
5. M. Bellare and B. Yee. Forward integrity for secure audit logs. *ACM Transactions on Information and Systems Security*, 1997.

6. C. Bernardeschi, A. Fantechi, and S. Gnesi. Model checking fault tolerant systems. *Software Testing, Verification & Reliability*, 12(4):251–275, 2002.
7. B. Chun, P. Maniatis, S. Shenker, and J. Kubiatowicz. Attested append-only memory: making adversaries stick to their word. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles*, pages 189–204. ACM Press New York, NY, USA, 2007.
8. M. Clavel, F. Duran, S. Eker, P. Lincoln, N. Martı-Oliet, J. Meseguer, and C. Talcott. Maude Manual (Version 2.1). *SRI International, Menlo Park*, Apr. 2005.
9. F. David, E. Chan, J. Carlyle, and R. Campbell. Cloaker: Hardware Supported Rootkit Concealment. In *Proceeedings of the 29th IEEE Symposium on Security and Privacy*, pages 296–310, 2008.
10. G. Dunlap, S. King, S. Cinar, M. Basrai, and P. Chen. ReVirt: enabling intrusion analysis through virtual-machine logging and replay. *ACM SIGOPS Operating Systems Review*, 36:211–224, 2002.
11. S. Eker, J. Meseguer, and A. Sridharanarayanan. The Maude LTL Model Checker. *Electronic Notes in Theoretical Computer Science*, 71:162–187, 2004.
12. K. Gandolfi, C. Mourtel, and F. Olivier. Electromagnetic analysis: Concrete results. *Lecture notes in computer science*, pages 251–261, 2001.
13. T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: a virtual machine-based platform for trusted computing. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, pages 193–206. ACM Press New York, NY, USA, 2003.
14. J. Herzog. Applying protocol analysis to security device interfaces. *IEEE Security and Privacy*, 4(4):84–87, 2006.
15. O. Laadan, R. Baratto, D. Phung, S. Potter, and J. Nieh. DejaView: a personal virtual computer recorder. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles*, pages 279–292. ACM Press New York, NY, USA, 2007.
16. M. LeMay, G. Gross, C. A. Gunter, and S. Garg. Unified architecture for large-scale attested metering. In *Proceedings of the 40th Hawaii International Conference on System Sciences*, Big Island, Hawaii, January 2007. IEEE.
17. D. Levin, J. R. Douceur, J. R. Lorch, and T. Moscibroda. TrInc: Small trusted hardware for large distributed systems. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*, 2009.
18. A. Liu and P. Ning. TinyECC: Elliptic Curve Cryptography for Sensor Networks. `http://cdl.csc.ncsu.edu/software/TinyECC/`, Sept. 2005.
19. M. Matsumoto and T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1):3–30, 1998.
20. R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn. Design and implementation of a TCG-based integrity measurement architecture. In *Proceedings of the 13th USENIX Security Symposium*, pages 233–238. USENIX Association, Aug. 2004.
21. A. Schmidt, N. Kuntze, and M. Kasper. On the deployment of Mobile Trusted Modules. In *Proceedings of the 9th IEEE Conference on Wireless Communications and Networking*, pages 3169–3174.
22. A. Seshadri, A. Perrig, L. van Doorn, and P. Khosla. SWATT: software-based attestation for embedded devices. In *Proceedings of the 25th IEEE Symposium on Security and Privacy*, pages 272–282, 2004.
23. C. Troncoso, G. Danezis, E. Kosta, and B. Preneel. Pripayd: privacy friendly pay-as-you-drive insurance. In *Proceedings of the 2007 ACM Workshop on Privacy in Electronic Society*, pages 99–107. ACM New York, NY, USA, 2007.
24. J. Winter. Trusted Computing building blocks for embedded Linux-based ARM TrustZone platforms. In *Proceedings of the 2008 ACM Workshop on Scalable Trusted Computing*. ACM New York, NY, USA, 2008.