An Implementation of a High Assurance Smart Meter Using Protected Module Architectures

Jan Tobias Mühlberg, Sara Cleemput, Mustafa A. Mustafa, Jo Van Bulck, Bart Preneel, Frank Piessens

An Implementation of a High Assurance Smart Meter using Protected Module Architectures

Jan Tobias Mühlberg[1], Sara Cleemput[2], Mustafa A. Mustafa[2], Jo Van Bulck[1], Bart Preneel[2], and Frank Piessens[1]

[1] iMinds-DistriNet, KU Leuven, Celestijnenlaan 200A, B-3001 Belgium
[2] KU Leuven, ESAT-COSIC and iMinds, Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium

Abstract. Due to ongoing changes in the power grid towards decentralised and highly volatile energy production, smart electricity meters are required to provide fine-grained measurement and timely remote access to consumption and production data. This enables flexible tariffing and dynamic load optimisation. As the power grid forms part of the critical infrastructure of our society, increasing the resilience of the grid's software components against failures and attacks is vitally important.
In this paper we explore the use of Protected Module Architectures (PMAs) to securely implement and deploy software for smart electricity meters. Outlining security challenges and an architectural solution in the light of security features provided by PMAs, we evaluate a proof-of-concept implementation of a security-focused smart metering scenario. Our implementation is based on Sancus, an embedded PMA for low-power microcontrollers. The evaluation of our prototype provides strong indication for the feasibility of implementing a PMA-based high assurance smart meter with a very small software Trusted Computing Base, which would be suitable for security certification and formal verification.

Keywords: smart meter security, smart grid, protected module architectures, distributed embedded computing, Sancus

1 Introduction

The smart grid is an extension of the traditional electricity grid. It includes smart appliances, renewable energy resources and smart electricity meters, facilitating bidirectional communication between these components and stakeholders, e.g., between a smart meter and the grid operator [9]. This is needed to deal with the volatility of renewable energy sources and new appliances such as electric vehicles, and to increase the reliability and sustainability of electricity delivery – one of the most critical resources of our time.

Electricity Smart Metering Equipment (ESME), i.e., smart meters, have three main responsibilities. Firstly, ESME measure the consumption of electricity and essential grid parameters, such as voltage or frequency, and timely provide this data to the grid operator. Secondly, they operate a Load Switch, which can

disconnect a premise from the electricity grid. The grid operator may use this in emergency cases to avoid a black-out. Finally, ESME communicate consumption data to smart appliances or an In-Home Display (IHD) present at the premise for local inspection and micro-management by the client. Relying on ESME involves security risks that range from privacy infringements to full-scale black-outs [12,29]. Physically isolating an ESME's critical software components, e.g., by using multiple microprocessors, can mitigate attacks against the grid's digital infrastructure. However, this can be prohibitively expensive and the use of trusted computing to achieve logical separation has been proposed [29,16]. An architecture based on trusted computing to implement High Assurance Smart Meters (HASMs) – smart meters with high security guarantees, certification and verification as design goals – poses a potential solution [4].

In this paper we implement a simplified scenario for smart meter deployment and evaluate its security aspects. We refer to components from the British Department of Energy & Climate Change's "Smart Metering Implementation Programme (SMIP)" [6]. However, our implementation may divert substantially with respect to details specified in the SMIP documents. We simplify communication protocols and we adopt architectural changes suggested in [4], relying on Protected Module Architectures (PMAs) [24] to implement security features. Our choice of technologies suggests a number of architectural changes relative to [6,4], which we discuss in detail. Importantly, the goal of this paper is *not* to accurately implement the SMIP specification but to provide a security-focused reference implementation that illustrates the use of embedded PMAs to achieve a notion of *high assurance smart metering* by means of logical component isolation, mutual authentication and by minimising the software Trusted Computing Base (TCB).

Our scenario contains software components that implement a HASM to be installed at a client's premises, and a Load Switch that can enable or disable power supply to the premises. We further implement components to represent the grid operator's Central System and an IHD. The HASM and the Load Switch communicate with the Central System via a Wide Area Network (WAN) Interface. In our case, the WAN Interface supports periodic access to the HASM's operational data, as well as control of the Load Switch. The HASM and the IHD communicate via the Home Area Network (HAN) Interface. Only consumption data is periodically sent from the HASM to the IHD via this interface. All components are implemented in software only and are meant to be deployed as Protected Modules (PMs) on microcontrollers or larger systems that facilitate software component isolation and authenticated and secure communication between PMs. Essentially, we model a smart metering scenario as a distributed reactive system, relying on security features provided by modern PMAs.

In the remainder of this paper we describe the basic architecture of a smart metering environment according to the SMIP documents [6]. We outline security challenges and an architectural solution as described in [4], in the light of security features provided by PMAs. The key contribution of this paper is to provide a proof-of-concept implementation of a security-focused smart metering scenario, based on Sancus [19], an embedded PMA for low-power TI-MSP430 microcontrollers.

We discuss the security objectives and architectural considerations of our implementation and evaluate its performance and the impact on the size of the system's deployed and trusted code base. Our prototype and evaluation provide a strong indication for the feasibility of implementing a secure, high assurance ESME and communicating controllers for smart home appliances, based on our architecture. The Sancus core, infrastructure software and our implementation are available at `https://distrinet.cs.kuleuven.be/software/sancus/wistp16/`.

2 High Assurance Smart Metering

Smart Electricity Metering. The British SMIP working documents [6,5] specify the physical requirements, functional requirements, interface and data requirements of an ESME. According to [6], an ESME minimally includes the following physical components: a clock, data storage, an electricity meter (i.e., metrology component), a Load Switch, a random number generator, a display, a HAN Interface, and a physical interface to connect to an independent Communication Hub [5], which is responsible for communicating with the grid operator's Central System over a dedicated WAN Interface. The ESME can communicate with the Communication Hub via its HAN Interface, which is furthermore used to connect to two types of local devices. *Type 1* devices store security credentials and can exchange authenticated and encrypted commands and data with the ESME, whereas *Type 2* devices do not store any security credentials and can only receive commands or data from the ESME.

As for the ESME's functional requirements, three main categories can be distinguished. First, an ESME should be capable of establishing and maintaining confidentiality- and integrity-preserving communication channels to receive command from and send data to the Central System (via the Communication Hub) and pre-defined local devices. Second, the ESME should be able to calculate bills, based on up-to-date tariffs, in credit as well as in prepayment mode. Finally, the ESME should be able to disconnect the household from the electricity grid by operating a Load Switch.

In addition, the ESME should store the following types of data: constant data (e.g., identifiers, model type, variant), internal data (e.g., installation credentials), configuration data (e.g., billing calendar, device log, security credentials, electricity quality thresholds), and operational data (e.g., import/export energy registers, cumulative and historical consumption data, power event log, security log).

Smart Metering Using Trusted Computing. However, ESME might be insufficiently secure, since *(1)* there is little isolation between the different modules that run on it, *(2)* it is possible to influence the ESME via the HAN Interface, and *(3)* it is easy to fill up the security log with non-critical events. Also, it might be impractical to have the Communication Hub physically separated from the ESME. In fact, the ESME should be a high assurance system, since the safety and security requirements are critical, due to the potentially huge physical impact of an attack. The German BSI has decided to introduce a specific component,

the smart meter gateway, to their smart metering architecture [2]. This gateway, which is installed at a client's premise, acts as communication unit between the client's local devices (including the ESME) and the Central System, and provides the necessary security properties by deploying several layers of protection using a Public Key Infrastructure (PKI). Although this solution provides a sufficient level of security, as also pointed out by von Oheimb [20], it is prohibitively expensive in terms of computational costs (mainly due to the use of PKI). Yan et al. [29], as well as Metke and Ekl [16] proposed using trusted computing in the smart grid to provide system, process and data integrity. However, they gave no details on how to implement this. Petrlic [22] proposed that each ESME must have a trusted platform module which acts as a tamper-resistant device and calculates users' bills based on the metering data measured at the ESME and the pricing data provided by the Central System. Jawurek et al. [11] proposed to use a plug-in component – placed between the ESME and the Central System – to calculate users' bills. LeMay et al. [14] described an implementation of a smart meter using Trusted Platform Modules and Virtual Machine Monitors. Unlike our work however, they did not give details on the internal architecture of the smart meter.

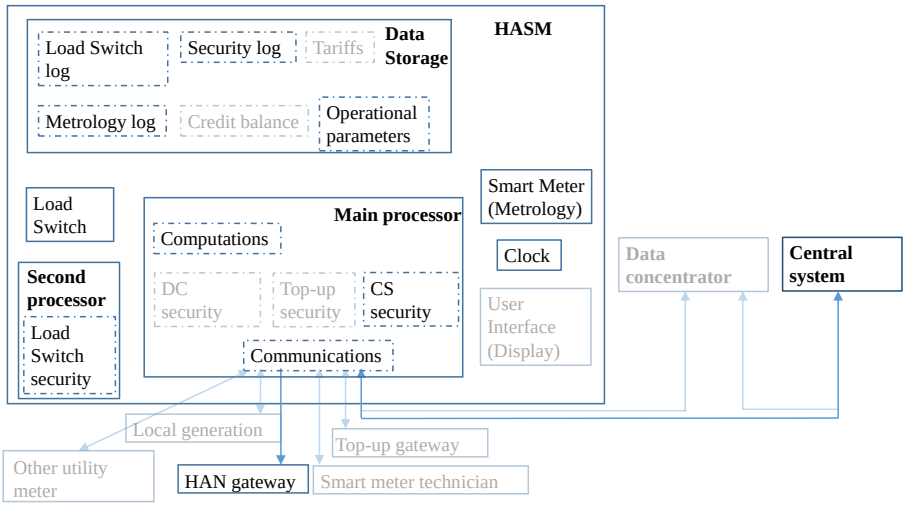

Fig. 1. Simplified version of a HASM according to [4].

The HASM proposed by Cleemput et al. (shown in Figure 1) [4] contains roughly the same physically separate components as the ESME: data storage, the Load Switch, the main processor, the metrology component, the display, and the clock. However, it also features a second processor for the Load Switch security module. The HASM has six different communication interfaces: an interface to the other-utility meter, an interface to the local generation, the HAN Interface, an interface to the top-up gateway, a local maintenance interface and an interface to a data concentrator.

There are several differences between this HASM and the standard ESME:

- A HASM contains an additional processor which houses a dedicated security module with exclusive access to the Load Switch. This is the most critical module in the smart meter, since hacking this module allows an attacker to disconnect households from the electricity grid.
- A separation kernel is used to logically isolate the rest of the modules on the main processor. These modules are: security modules, a computations module and a communications module. The latter corresponding to the Communication Hub in the ESME architecture. However, rather than being a separate component, we consider it as a part of the HASM.
- The log files are divided into three different logs: a metrology log, a security log and a Load Switch log. This ensures that the attacker cannot flush an event out of the Load Switch log by generating less critical security events.
- The three security modules implement separate communication components, one for the Data Concentrator, one for the Top-Up component, and one for the Central System.
- All incoming communication must first go through the communications module and from there on directly to one of the security modules.
- The HAN Interface must be a one-way interface, such that only communication from the HASM to the HAN gateway is possible. This is important, since the smart appliances and the IHD are considered untrusted components, so they should not be able to influence the HASM. Note that this means, we only consider *Type 2* devices. We believe this is a practical assumption, since the in-home devices are not under the control of the grid operator, thus it is difficult to have pre-established keys between these devices and the HASM. Furthermore, we cannot think of convincing use cases where communication from the in-home devices to the HASM is strictly necessary. The main difference between the HAN Interface and the WAN Interface is that the latter is an interface to a trusted entity. Thus, the WAN Interface is a bidirectional interface, where all data is authenticated and encrypted.

Our Scenario. In this paper we present a simplified version of a HASM, as illustrated in Figure 1. We do not consider the display present on the smart meter itself, and we only consider two communication interfaces: a WAN Interface to the Central System and a HAN Interface to the HAN gateway. Moreover, we implement only the following representative subset of ESME use cases.

Billing. For our implementation we only consider non-prepaid billing in Credit-Mode. Prepaid billing is an interesting use case in itself that exhibits security and privacy aspects different from non-prepaid billing. We are confident that our prototype could easily be extended to support prepaid billing. There are two main threats to the billing process: fraud and privacy infringements.

Load switching. As mentioned above, the Load Switch can be used to remotely enable or disable power supply at the client's premises. The main threat for this use case consists of an adversary who manages to cause a large-scale

black-out by triggering the Load Switch. This is the most critical threat to our architecture, since concurrently disconnecting many consumers may cause a cascading instability, eventually bringing down large parts of the electricity grid. Ultimately, even public facilities that rely on the electricity grid, such as sewer operations, traffic lights and the telephone network, will be affected.

Consumer feedback. The goal of providing the user with fine-grained consumption data through the HAN Interface is to realise energy savings, as well as to allow them to connect smart appliances. The main threat in this use case is an adversary influencing the smart meter via the HAN Interface.

3 Authentic Execution with PMAs

PMAs [24] are a new brand of hardware security architectures, the main objective of which is to support the secure and isolated execution of critical software components with a minimal, hardware-only TCB. Implementations of PMAs for higher-end systems – Intel's Software Guard Extensions (SGX) [15] or ARM's TrustZone [1] – as well as several prototypes for embedded application domains [8,19,13] have been proposed. Software components that are specifically designed and implemented to leverage PMA features are provided with strong confidentiality and integrity guarantees regarding their internal state, and can mutually authenticate each other. More specifically, modern PMAs offer a number of security primitives to *(1)* configure memory protection domains, *(2)* enable or disable software module protection, and *(3)* facilitate key management for secure local or remote inter-module communication and attestation.

A Notion of Authentic Execution. PMAs allow us to securely implement authentic execution of distributed event-driven applications that execute on a heterogeneous shared infrastructure with a small TCB. These applications are characterised by consisting of multiple components that execute on different computing nodes and for which program flow is determined by events such as sensor inputs or external requests. As an example, consider the HASM with its sensors (electricity measuring element), communication interfaces (WAN and HAN), and actuators (Load Switch).

Roughly speaking, our notion of authentic execution is the following: if the application produces a physical output event (e.g., disabling supply via the Load Switch), then there must have happened a sequence of physical input events such that that sequence, when processed by the application (as specified by the application's source code), produces the output event.

This notion of authentic execution does provide strong integrity guarantees: it rules out both spoofed events as well as tampering with the execution of the program. Informally, if the executing program produces an output event, it could also have produced that same event if no attacker was present. Any physical output event can be explained by means of the untampered code of the application, and the actual physical input events that have happened.

Building Blocks. Authentic execution relies on well-defined application components that are encapsulated as PMs and that are protected by the PMA. PM software components consist of a contiguous code and data section in the shared address space. Hardware-level PMAs such as Sancus [19] rely on a lightweight program counter based access control mechanism [26] to enforce that a PM's private data section is exclusively managed via its corresponding code section, which can only be entered via a few predefined entry points. The latter impedes code abuse attacks (e.g., Return Oriented Programming [3]) against PMs.

In distributed event-driven applications, each PM consumes *input events* and produces *output events*. The types of these events and communication channels are to be defined at compile-time. Components that directly interact with sensors or actuators are further able to claim *exclusive access* to interrupt vectors or device registers that are accessed through memory-mapped I/O on low-end microcontrollers [19]. Program code and access permissions form the identity of a PM, which can be *attested* cryptographically by remote parties, including other components of the distributed application. Intuitively, successful attestation implies that a software component is deployed with no modifications on a specific computing node, obtained exclusive access over its desired hardware resources, and that protection has been enabled through the PMA.

Trusted Computing Base. A key strength of hardware-level PMAs such as Sancus is that they reduce the TCB up to the point where a remote stakeholder only has to trust the Sancus-enabled microcontroller, and the implementation of his own modules to be guaranteed authentic execution. Only the software components that implement the actual application logic of a distributed application have to be deployed as PMs. Of course, the compilation and deployment processes, running on the remote stakeholder's infrastructure, still need to be trusted. Yet, supporting software such as (embedded) Operating Systems (OSs), network stacks, module loaders, and even components implementing event management and distribution are explicitly untrusted as to authentic execution security guarantees. However, as we outline further on, such infrastructure software components may be trusted with regard to *availability* guarantees.

4 Implementation

In this section we present and discuss our proof-of-concept implementation of a HASM's software stack. Relying on Sancus [19] as the underlying architecture and the approach to authentic execution outlined in Section 3, we implement a representative subset of the HASM architecture as a reactive system, which is illustrated in Figure 2: The core of our implementation is formed by three distributed PMs that realise the ESME's metering component, the Load Switch, and the grid operator's Central System. These PMs communicate bidirectionally over the untrusted WAN interface, where authenticated encryption is used to guarantee confidentiality and authenticity of messages, and to attest module

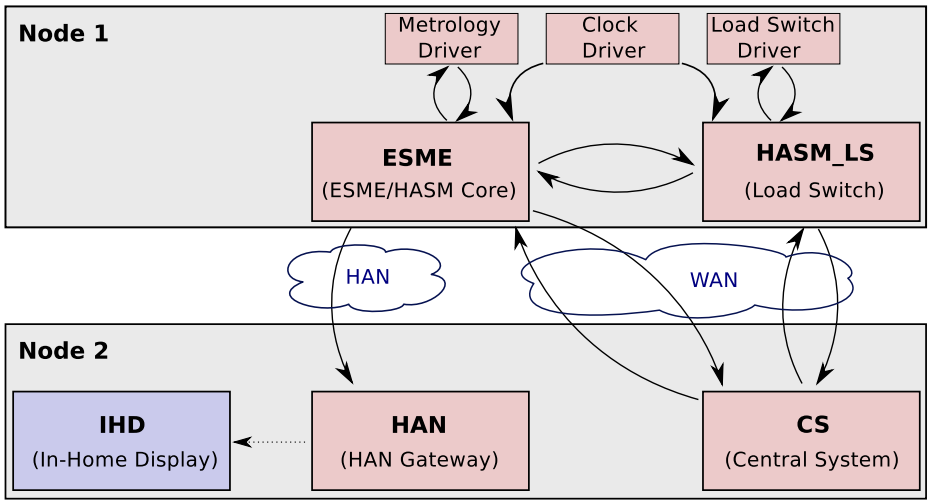

Fig. 2. Our implementation of a HASM's software stack using distributed PMs. Boxes shaded in red represent PMs and continuous arrows denote secure communication channels between these PMs. The IHD executes without PMA protection and must rely on alternative mechanisms to secure its communication with the HAN Interface.

integrity. A fourth PM implements the HAN Gateway, which acts as a unidirectional security gateway to relay consumption data to in-house appliances such as the IHD. For completeness we add such an IHD as an untrusted software component. The PMs are deployed and configured according to a Deployment Descriptor that defines which modules are to be loaded on which computing nodes and which module outputs are to be linked to which inputs. A section of the Deployment Descriptor, focusing on the Load Switch PM is given in Listing 1.

Our implementation runs on two TI MSP430 microcontrollers that implement the Sancus extensions; we rely on the Contiki OS [7] for untrusted supporting software such as the scheduler and the network stack. Figure 2 mentions three driver PMs that are meant to securely produce low-level I/O events (i.e., clock ticks and electricity consumption readings) and to operate actuators (the Load Switch). As we do not have all these hardware components available, we have left the implementation of these driver PMs for future work. Key features of Sancus and other PMAs are hardware-based isolation and integrity protection of PMs, and the built-in mechanisms for deriving, storing and managing cryptographic keys. These features naturally lead to a number of changes in the overall design of a HASM, specifically with respect to the system's communication infrastructure. We discuss these design choices below.

Communications. In our implementation, the *Communications* component described in Section 2 is represented by an Event Manager which is an untrusted software component running on every node that is responsible for routing events from a PM's outputs to (another) PM's inputs. The Event Manager cannot decrypt and inspect these events. Instead, PMs themselves maintain keys for

```
{
  "nodes": [
    {
      "type": "sancus",
      "name": "node1",
      "ip_address": "...",
      "vendor_id": 4660,
      "vendor_key": "..."
    }
  ],
  "modules" : [
    {
      "type": "sancus",
      "name": "HASM_LS",
      "files": ["hasm_ls.c"],
      "node": "node1"
    }
  ],
  "connections": [
    {
     "from_module": "CS",
     "from_output": "ls_out",
     "to_module": "HASM_LS",
     "to_input": "cs_in"
    },
    {
     "from_module": "HASM_LS",
     "from_output": "cs_out",
     "to_module": "CS",
     "to_input": "ls_in"
    },
    {
     "from_module": "ESME",
     "from_output": "ls_out",
     "to_module": "HASM_LS",
     "to_input": "esme_in"
    },
    {
     "from_module": "HASM_LS",
     "from_output": "esme_out",
     "to_module": "ESME",
     "to_input": "ls_in"
    }
  ]
}
```

Listing 1: Deployment descriptor for the Load Switch PM: `HASM_LS`

each communication channel. Decrypting events and verifying authenticity and freshness is implemented by each module, based on the cryptographic primitives provided by the PMA hardware. In consequence, PMs such as our Load Switch component and the Central System are easier and more securely implemented by defining bidirectional communication channels that use communication media and the Event Manager transparently, relying on purpose-specific keys.

In Listing 1 we present the section of the Deployment Descriptor that is relevant for deploying and configuring the Load Switch PM. As can be seen, the Deployment Descriptor specifies which node the PM is to be deployed on, and how input and output channels are to be linked together. Intuitively, a `connections` entry defines a unidirectional channel between a `from_module` PM and a `to_module` PM. The entries `from_output` and `to_input` correspond with module-specific handles for the connection that can be referred to in the source code of each PM. At deployment time, when configuring the channel, a symmetric key is securely transferred to each of the two PM endpoints, using hardware-level module keys provided by the PMA implementation.

We illustrate the use of module-specific channel handles in Listing 2. The Sancus compiler ensures that only successfully authenticated and decrypted events will ever be received at the input handles, and the PM's source code defines how to react upon these events. E.g., the Load Switch PM implements an access control policy by defining that only the Central System may issue commands to change the system's supply state. The ESME PM may only query the supply state. In a more realistic implementation, changing the supply state must then result in using a driver PM to operate an actual load switch peripheral (i.e., an electrical relay).

```
#include <sancus/reactive.h>

#include "commands.h"

SM_OUTPUT(HASM_LS, cs_out);
SM_OUTPUT(HASM_LS, esme_out);

static uint8_t SM_DATA(HASM_LS)
  sply_state;

SM_INPUT(HASM_LS, cs_in, data,
  len)
{
  if (data[2] & ENABLE_SPLY) {
    sply_state = SPLY_ENABLED;
    cs_out(ENABLE_SPLY, 1);
  }
  if (data[2] & DISABLE_SPLY) {
    sply_state = SPLY_DISABLED;
    cs_out(DISABLE_SPLY, 1);
  }
  if (data[2] & GET_SPLY_STAT) {
    cs_out(sply_state, 1);
  }
}

SM_INPUT(HASM_LS, esme_in, data,
  len)
{
  if (data[2] & GET_SPLY_STAT) {
    esme_out(sply_state, 1);
  }
}
```

Listing 2: Simplified C source code of the Load Switch PM: `hasm_ls.c`

Use of Separate CPUs. Another important aspect of using a PMA is that strong isolation and integrity protection of PMs guarantee that a PM's code and data can only be accessed through well-defined entry point functions. This effectively rules out attacks from the OS or any other software on a computing node. As a result, two PMs can securely co-exist on the same computing node without risking interactions that lead to manipulation of a PM's state in a way that is not defined by the source code of the PM, which is why we decided to deploy the Load Switch on the same node as the ESME. However, as we discuss in Section 5 guaranteeing availability and system progress may require further changes to the configuration. That is, availability and real-time requirements must be reflected by the hardware configuration. As evident from our deployment mechanism, module configurations and deployment details are easily adapted to different hardware configurations.

Persistent Storage. Strong component isolation further weakens the requirement for implementing a dedicated data storage component. Instead individual PMs can securely store operational data in the modules' secret data section and manage access to this data directly. This is particularly true in the case of size-bounded circular log buffers as specified in [6]. Methods to persist this operational data can be implemented with hardware support by the PMA [25]. Alternatively, secure resource sharing for persistent storage can be implemented as described in [27], via an intermediate PM that implements an access control policy for the module that "owns" the data.

5 Evaluation

In this section we evaluate the TCB and security properties of our HASM implementation. Our prototypic implementation is based on a developmental version of Contiki 3.x running on a Sancus-enabled openMSP430 [10,19] that is programmed on a Xilinx Spartan-6 FPGA. We do not provide a detailed

performance evaluation as this does not yield interesting results beyond what is published in related work [19,18,27]: module loading, enabling protection, initial attestation and key deployment is relatively slow and may prolong startup of a HASM by a few milliseconds. The performance of cryptographic operations at run-time does not incur prohibitive overheads and the relatively relaxed real-time constraints specified in [6] (in the order of tens of seconds or minutes) can easily be met by our implementation. These results are in-line with a previous implementation of a smart meter for the MSP430 [17], where cryptography is implemented in software. A discussion of availability and real-time guarantees of our approach in the presence of adversaries concludes this section.

Table 1. Size of the software for running the evaluation scenario. The shaded components are part of the TCB.

Component	Source LOC	Binary Size (B)	Deployed
Contiki	38386	16316	per node
Event manager	598	1730	per node
Module loader	906	1959	per node
ESME/HASM Core	119	2573	once
Load Switch	79	2377	once
HAN Gateway	30	1599	once
Central System	63	2069	once
Deployment Descriptor	90	n/a	n/a

TCB Size and Implications. Table 1 shows the sizes of the different software components deployed on nodes. As can be seen, the majority of the code running on a node – about 40 kLOC – is untrusted in our model. A total of only 291 LOC comprising of the actual application code is compiled to PMs and needs to be trusted, together with 90 LOC of the deployment descriptor. That is, only 1% of the deployed code base is part of the software TCB. When looking at the binary sizes of the these software components, the difference between infrastructure components (19.5 KiB) versus TCB (8.4 KiB, 43.1%) appears less prominent, which is due to a large number of conditionally compiled statements in Contiki as well as compiler generated entry points and stub code in the PMs.

For a full implementation of a HASM that provides trusted paths from sensors to the Central System, one also has to consider driver code. Without having the actual physical components for building a smart meter available, we conjecture that the sizes of such driver PMs are probably on par with our HASM implementation. Nevertheless, the reduction of the TCB when using our approach is substantial in comparison with other implementations that focus on security and attestation [14], which leads to a reduced attack surface on each node. The application owner, i.e., the grid operator, does not have to trust *any* infrastructure software but the driver modules that his application uses. As shown in related work, embedded programs of the size of our HASM PMs can

be formally verified at acceptable efforts [23] and are more manageable in safety and security certification than the entire deployed code base. Of course, the TCB of our scenario also includes compilers and hardware. We aim to address the security of these parts of the TCB by means of secure compilation [21].

Security Evaluation. As explained in Section 3, the security property offered by our approach is that any physical output event can be explained by means of the untampered code of the application, and the actual physical input events that have happened. For the operator of a smart grid, this is a valuable property: it means that the response to a request to disable supply at a client's premises implies that the request was received and processed, down to the level of the Load Switch driver. To give another example, the guarantee also means that received consumption data is indeed based on the measurement of a specific metering element and the chain of untampered PMs that process the measurement. Together with the use of timestamps and nonces (at application level) and the built-in cryptographic communication primitives, our approach provides further confidentiality and freshness guarantees for the system's outputs.

From our discussion of design choices it can be seen that the use of PMAs must be considered early in the software development cycle since component isolation will affect the way in which components interact with one another. In particular, different protection domains cannot easily communicate through shared memory but must rely on cryptography and authenticated method invocation. Software developers will require tool support to isolate security critical code in protected modules, to design communication mechanisms and to to assess the reliability, performance and security characteristics of the resulting software system.

Software that is executing in a PM can still be subject to low-level attacks that exploit implementation details. Such attacks can cause memory corruption within the module and may even allow the attacker to control the internal execution of the module. This is because a software component encapsulated in a PM may offer a richer API than just input and output of primitive values. Methods or functions callable from the malicious context might also accept references to mutable objects or function pointers as parameters, or produce those as return values. Ongoing research addresses this by means of secure compilation, formal verification and the use of safe programming languages.

Furthermore, while our approach and the use of PMAs in general offer strong confidentiality and integrity guarantees for software modules, they offer no availability, let alone real-time guarantees, which we discuss below.

Availability and Real-Time Guarantees. The HASM/ESME reference implementation presented and evaluated above shows the feasibility of encapsulating high assurance smart metering functionality in isolated PM software components. Such an approach provides a grid operator with strong guarantees regarding the internal state of the smart meter and the authenticity of its measurements, while the underlying infrastructure software remains explicitly untrusted. However, as the timely execution of the smart metering PMs cannot be ensured, these

guarantees do not extend to *availability*. Consider for example the scenario where an adversary exploits a remote vulnerability in the network stack or dynamic software loader. Our approach prevents such an attacker from operating the Load Switch peripheral or altering the security logs, but currently does not protect against various denial-of-service attacks where a malicious or buggy application for example overwrites crucial OS data structures or monopolises CPU time.

In the context of high assurance smart metering architectures availability properties cannot be considered out of scope. From the SMIP requirements document [6], we identified at least the following three real-time properties:

1. The HAN gateway shall receive information updates from the ESME at least every 10 seconds, and send them out to the IHD for visualisation purposes.
2. When operating in prepaid mode, the ESME shall be capable of monitoring the leftover credit balance, and disabling the power supply when a certain "disablement threshold" has been exceeded.
3. The ESME shall include measures to prevent physical tampering with the device. Upon detection of an unauthorised physical break-in event, the ESME shall establish a "locked state" whereby the power supply is disabled.

A challenging aspect of our proposed architecture is how to incorporate such hard real-time constraints. While non-trivial, we believe our reference implementation can serve as a base for an enhanced architecture that preserves the timely execution of security- and safety-critical functionality, even on a partially compromised smart meter. In the following, we outline several required extensions that allow the above real-time criteria to be met, without enlarging the TCB for the grid operator's security guarantees.

Secure Interrupt Handling. In a real-time computing system interrupts are commonly used to notify the processor of some asynchronous outside world event that requires immediate action. As an example, to meet requirement 3 above, a push button connected to the smart meter's case could raise an Interrupt Request (IRQ) when detecting physical tampering with the device. In response to such an IRQ, the PM operating the Load Switch should be activated so as to establish the locked state and disable the power supply.

Importantly, while the SMIP smart meter specifications document [6] does not provide a specific timing constraint for establishing the locked state, this real-time deadline can be considered *hard*. That is, severe system damage (e.g., damage to the grid or critical infrastructure, large-scale fraud) may occur when an adversary succeeds in physically accessing the smart meter's internals without the locked state being established.

The idea to enable secure interrupt handling in our ESME reference implementation is to register the entry point of the Load Switch PM as the interrupt handler for the intrusion detection IRQ. There are multiple ways in which an adversary, after having gained code execution on the smart meter, can prevent the IRQ handler from being (timely) executed. First, an attacker may overwrite the system-wide Interrupt Vector Table (IVT) that records interrupt handler

addresses. This can be prevented by mapping the IVT memory addresses into the immutable text section of a dedicated PM. Second, an adversary may hold on to the CPU by disabling interrupts for arbitrary long times. To prevent such a scenario, and to establish a deterministic interrupt latency, running applications should not be allowed to unconditionally disable interrupts. For this, a hardware/software co-design has been proposed [28] to make PMs fully interruptible and reentrant, without introducing a privileged software layer that enlarges their software TCB, and while preserving secure compilation guarantees [21] via limited-length atomic code sections in a preemptive environment.

Preemptive Multitasking. Requirements 1 and 2 above necessitate the periodic execution of the ESME PM to monitor the client's power consumption and outstanding prepaid credit. In our current event-driven prototype, the Event Manager might schedule a periodic event that updates power consumption measurements in the ESME PM. However, when all input and output events have run-to-completion semantics, the Event Manager cannot be guaranteed to be timely executed, if at all. We will therefore explore *preemptive* scheduling of event handlers where a lightweight protected scheduler PM configures a timer interrupt before passing control to the untrusted event handler thread. Such an approach enables the protected scheduler (or Event Manager for that matter) to multiplex CPU time between multiple mutually distrusting application threads, while remaining responsive to asynchronous external events.

Importantly, in line with the notion of authentic execution introduced in Section 3, the protected scheduler should solely encapsulate the scheduling policy. A compromised scheduler PM should affect CPU availability only, and should not change the property that a grid operator can explain all physical output events by means of the observed physical input events and the application's source code. However, after successful attestation of the scheduler PM, the grid operator will be provided with additional availability guarantees, as defined by the scheduling policy. This ensures that, even in the case of a network failure or compromised infrastructure software, the smart meter's vital functionality will continue to execute as expected: power consumption will be monitored, and the supply will be disabled when the accumulated debt exceeds the pre-set threshold.

6 Conclusions

We have implemented and discussed a proof-of-concept prototype of a security-focused software stack for a smart metering scenario. Our implementation includes a High Assurance Smart Meter, a Load Switch, a HAN Gateway with In-Home Display, and a simplified Central System. Relying on the security guarantees of an embedded Protected Module Architecture, our approach and prototype guarantee that all outputs of the software system can be explained by the system's source code and the actual physical input events. We further guarantee integrity and confidentiality of messages while relying on a very small software Trusted Computing Base. For scenarios that involve critical infrastructure, such

as the smart grid, we believe that our approach has the strategic advantage of enabling formal verification and security certification of small, isolated software components while maintaining the strong security guarantees of the distributed system that is formed by the interaction of these components.

In future work we will extend the prototype to support real electricity metering hardware and work towards providing strong real-time and availability guarantees in distributed event-driven smart sensing scenarios. We believe that our approach for implementing the smart meter can be reused to implement similar applications in the Internet of Things or in sensor networks.

Acknowledgements. The authors would like to thank B. Defend, K. Kursawe and C. Peters from ENCS for their many useful suggestions and help. This work has been supported in part by the Research Fund KU Leuven, by the Research Council KU Leuven: C16/15/058i, by the Research Foundation – Flanders (FWO), through KIC Innovation Project SAGA supported by KIC InnoEnergy SE and by the EU FP7/2007-2013 programme under grant 610535 – AMADEOS.

References

1. Alves, T., Felton, D.: Trustzone: Integrated hardware and software security. ARM white paper 3(4), 18–24 (2004)
2. BSI: Protection profile for the gateway of a smart metering system (smart meter gateway PP) (2014), https://www.commoncriteriaportal.org/files/ppfiles/pp0073b_pdf.pdf
3. Checkoway, S., Davi, L., Dmitrienko, A., Sadeghi, A.R., Shacham, H., Winandy, M.: Return-oriented programming without returns. In: 17th Conf. on Computer and Communications Security (CCS'10). pp. 559–572. ACM (2010)
4. Cleemput, S., Mustafa, M.A., Preneel, B.: High assurance smart metering. In: 17th Int. Symposium on High Assurance Systems Engineering (HASE'16). pp. 294–297. IEEE (2016)
5. Department of Energy and Climate Change: Smart metering implementation programme – communications hub technical specifications; version 1.46 (2014), https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/381536/SMIP_E2E_CHTS.pdf
6. Department of Energy and Climate Change: Smart metering implementation programme – smart metering equipment technical specifications; version 1.58 (2014), https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/381535/SMIP_E2E_SMETS2.pdf
7. Dunkels, A., Gronvall, B., Voigt, T.: Contiki – a lightweight and flexible operating system for tiny networked sensors. In: 29th Annual Int. Conf. on Local Computer Networks. pp. 455–462. IEEE (2004), http://www.contiki-os.org/
8. Eldefrawy, K., Francillon, A., Perito, D., Tsudik, G.: SMART: Secure and Minimal Architecture for (Establishing a Dynamic) Root of Trust. In: 19th Annual Network and Distributed System Security Symposium (NDSS'12), (2012)
9. Farhangi, H.: The path of the smart grid. IEEE Power and Energy Magazine 8(1), 18–28 (2010)
10. Girard, O.: openMSP430 (2009), http://opencores.org

11. Jawurek, M., Johns, M., Kerschbaum, F.: Plug-in privacy for smart metering billing. In: PETS '11. LNCS, vol. 6794, pp. 192–210. Springer (2011)
12. Kalogridis, G., Sooriyabandara, M., Fan, Z., Mustafa, M.A.: Toward unified security and privacy protection for smart meter networks. IEEE Systems Journal 8(2), 641–654 (2014)
13. Koeberl, P., Schulz, S., Sadeghi, A.R., Varadharajan, V.: TrustLite: A security architecture for tiny embedded devices. In: EuroSys '14. p. 14 pages. ACM (2014)
14. LeMay, M., Gross, G., Gunter, C.A., Garg, S.: Unified architecture for large-scale attested metering. In: HICSS '07. IEEE (2007)
15. McKeen, F., Alexandrovich, I., Berenzon, A., Rozas, C.V., Shafi, H., Shanbhogue, V., Savagaonkar, U.R.: Innovative instructions and software model for isolated execution. In: HASP'13. p. 8 pages. ACM (2013)
16. Metke, A.R., Ekl, R.L.: Security technology for smart grid networks. IEEE Transactions on Smart Grid 1(1), 99–107 (2010)
17. Molina-Markham, A., Danezis, G., Fu, K., Shenoy, P., Irwin, D.: Designing privacy-preserving smart meters with low-cost microcontrollers. In: FC'12. LNCS, vol. 7397, pp. 239–253. Springer (2012)
18. Mühlberg, J.T., Noorman, J., Piessens, F.: Lightweight and flexible trust assessment modules for the Internet of Things. In: ESORICS '15. LNCS, vol. 9326, pp. 503–520. Springer (2015)
19. Noorman, J., Agten, P., Daniels, W., Strackx, R., Herrewege, A.V., Huygens, C., Preneel, B., Verbauwhede, I., Piessens, F.: Sancus: Low-cost trustworthy extensible networked devices with a zero-software trusted computing base. In: 22nd USENIX Security Symposium. pp. 479–498. USENIX (2013)
20. von Oheimb, D.: IT security architecture approaches for smart metering and smart grid. In: SmartGridSec'12. LNCS, vol. 7823, pp. 1–25. Springer (2013)
21. Patrignani, M., Agten, P., Strackx, R., Jacobs, B., Clarke, D., Piessens, F.: Secure compilation to protected module architectures. ACM Trans. Program. Lang. Syst. 37(2), 6:1–6:50 (2015)
22. Petrlic, R.: A privacy-preserving concept for smart grids. In: Sicherheit in vernetzten Systemen. DFN Workshop. pp. 1–14. Books on Demand GmbH (2010)
23. Philippaerts, P., Mühlberg, J.T., Penninckx, W., Smans, J., Jacobs, B., Piessens, F.: Software verification with VeriFast: Industrial case studies. Science of Computer Programming (SCP) 82, 77–97 (2014)
24. Strackx, R., Noorman, J., Verbauwhede, I., Preneel, B., Piessens, F.: Protected software module architectures. In: Securing Electronic Business Processes. pp. 241–251. Springer (2013)
25. Strackx, R., Piessens, F.: Ariadne: A minimal approach to state continuity. In: 25th USENIX Security Symposium. USENIX (2016), to appear.
26. Strackx, R., Piessens, F., Preneel, B.: Efficient isolation of trusted subsystems in embedded systems. In: Security and Privacy in Communication Networks, LNICST, vol. 50, pp. 344–361. Springer (2010)
27. Van Bulck, J., Noorman, J., Mühlberg, J.T., Piessens, F.: Secure resource sharing for embedded protected module architectures. In: WISTP'15. LNCS, vol. 9311, pp. 71–87. Springer (2015)
28. Van Bulck, J., Noorman, J., Mühlberg, J.T., Piessens, F.: Towards availability and real-time guarantees for protected module architectures. In: 15th International Conference on Modularity. pp. 146–151. ACM (2016)
29. Yan, Y., Qian, Y., Sharif, H., Tipper, D.: A survey on cyber security for smart grid communications. IEEE Communications Surveys Tutorials 14(4), 998–1010 (2012)