# A Model for Security Analysis of Smart Meters

Farid Molazem Tabrizi
*ECE Department University of British Columbia*
*Vancouver, Canada*
*faridm@ece.ubc.ca*

Karthik Pattabiraman
*ECE Department University of British Columbia*
*Vancouver, Canada*
*karthikp@ece.ubc.ca*

***Abstract*—Smart grids are replacing traditional power grids and smart meters are one of the key components of smart grids. Rapid deployment of smart grids has resulted in development of advanced metering infrastructures (AMI) without adequate security planning. In this paper we propose a systematic method for modeling functionalities of smart meters and deriving attacks that can be mounted on them. We apply our method to a real open source meter, implement two of the derived attacks, and measure their performance/memory overheads.**

***Keywords*-Security; Abstract Model; Advanced Metering Infrastructure**

## I. Introduction

Smart grids are poised to replace traditional power grids in North America and Europe. Unlike traditional grids, smart grids use Advanced Metering Infrastructure (AMI) with two-way communication capabilities. AMI meters have a key role in providing monitoring and control capabilities for smart grids. Unfortunately, rapid deployment of smart grids has resulted in developing advanced metering infrastructure without adequate security and reliability planning [8], [14], [19]. Current estimates indicate that in the US alone, $6 billion is lost by providers due to fraud [12]. The financial benefits that would be accrued from tampering with smart metering devices makes security of smart meters an important issue. Currently, to our knowledge, there is no systematic procedure to make smart meter software resilient to attacks and as a result, many vulnerabilities and attacks continue to be discovered against these systems [7], [11], [15].

In this paper, we propose a systematic way of modeling the procedures implemented for the smart meter software and performing security analysis of these procedures. By systematic, we mean a model-based, step by step approach that is amenable to be automated. Building a systematic method for performing the security analysis will remove the errors resulting from ad hoc analysis techniques and provide an accurate way to perform penetration testing, and fix the design flaws found.

Existing work on building models and systematic techniques to provide security for smart meters either target the network communications of smart meters, or testing of the meters based on existing generic attacks. For example, Berthier et al. [3] model the normal behavior of the communication of the meters and propose a specification-based intrusion detection system (IDS) based on their model. But this does not cover the vulnerabilities of the software running on the meter. McLaughlin et al. [13] propose a systematic way to perform penetration testing for the meters, given the attack model. However, there is no well-defined mechanism to build the attack models. *To the best of our knowledge, we are the first to provide a model for security analysis of the software executed by the meter*. In this paper, we make the following contributions:

- We characterize the step-by-step operations of generic smart meter software.
- We propose an abstract model for the meter which provides a systematic way to analyze software attacks against the meter.
- We map the abstract model to the implementation of a real meter based on an open source stack.
- We implement two attacks on the real meter based on our abstract model and measure their CPU and memory overheads respectively to quantify their stealthiness.

In this paper, we assume that an adversary can obtain system level access to the smart meter. This can be obtained by exploiting known vulnerabilities in the meter or by obtaining the meter's root password. The threat model is detailed in Sec. III-C. Also the implications of this assumption are discussed in Sec. V. While we have shown the applicability of our method on one smart meter in Sec. IV, it is based on a generic model and therefore, we believe that it is applicable to other meters as well.

Although we do not propose protection mechanism for smart meters in this paper, we believe that the results of our work can be used to build security mechanisms tailored for advanced metering infrastructures. Unlike general purpose computers, smart meters are low capacity computing devices and are limited in computational resources. On the other hand, generic security mechanisms such as intrusion detection systems incur high performance overheads, making them unsuitable for meters [8], [9]. Therefore security mechanisms for smart meters must be carefully tailored to specifically target the attacks associated with advanced metering infrastructure to comply with the constraints and requirements of these systems. Using our model, through detecting and analyzing attacks against smart meters and performing feature extraction for them, we can build security mechanisms at the operating system or application level that

effectively address the attacks.

Further, our model can be used in the design phase of software production for smart meters. Many detected attacks can therefore be addressed by making corrections in the design decisions and adding security measures and components in the design phase to build more secure software.

## II. Background

Hardware based approaches provide security through special hardware modules, such as a Trusted Platform Module (TPM) [16]. For the case of smart meters, pushing the security down to hardware level has disadvantages. First, TPMs incur high cost, and using them in millions of meters, makes their use an expensive proposition. Further, hardware based solutions are difficult to update. The cost, memory and power limitation of smart meters make the use of TPM-based techniques impractical [8].

Current intrusion detection systems (IDS) for addressing security of smart meters are mostly limited to security of the communication link. Berithier et al. [2], propose some guidelines to build intrusion detection systems for advanced metering infrastructure. In recent work [3], they propose a specification-based intrusion detection system for smart meters that monitors the communication links and detects abnormality in the traffic according to a previously built model. Intrusion detection systems by themselves, cannot fully secure smart meters, as they may have false negatives that allow attackers to bypass the security mechanism by exploiting software vulnerabilities. Therefore, protecting the meter at the software level is a necessary complement to intrusion detection systems.

Software verification techniques such as Pioneer [17] and oblivious hashing [5] verify the integrity of software on a third party machine by executing an instance of the program on a remote server. Considering the scale of smart metering networks, this imposes a high overhead on the server which makes these techniques impractical for smart metering infrastructure. LeMay et al. [10] propose a virtualization-based architecture for remote attestation of smart meters to ensure the integrity of the meter software and hardware. Remote attestation techniques do not ensure that the software running on the meter does not have any vulnerabilities. Even if the software running on the system is legitimate, it can still have vulnerabilities that let an adversary attack the smart meter. Hence, this work does not address the problem of studying the smart meter software to analyze attacks applicable to it.

Unlike the other prior work, in this paper, we focus on security analysis of the software running on the smart meters. We propose an abstract model that provides a systematic method to analyze security attacks against the meters. A similar approach is taken by [13] in which a technique for performing penetration testing based on generic attacks is presented. The technique uses generic attack trees to find attacks on the meter. However, there is no well-defined procedure to build the generic attack trees. In contrast, we propose a generic model and technique for deriving a comprehensive set of attacks on the smart meters.

## III. Approach

### A. Terms

In this section, we define the terms "abstract model" and "concrete model" that we use in the paper.

**Abstract model**: Abstract model represents the behavior of the software running on the meter. It consists of a collection of interconnected blocks where each block represents a set of software procedures of the smart meter and the connections between the blocks represent the execution flow among those procedures.

**Concrete model**: Concrete model is an instantiation of the abstract model for the software running on a specific meter. In the concrete model, the blocks of the abstract model are decomposed into the code level procedures that implement the functionality of that block.

### B. Overview

In this section, we propose a systematic approach for analyzing security of smart meters. Our systematic approach extracts all the important components of the smart meter and highlights the vulnerabilities and attacks that target those components. Compared to ad hoc techniques, the systematic approach is a consistent way to analyze a smart meter's security and is less likely to miss the important and critical components of the meter with undetected vulnerabilities. Further, it is amenable to automation.

Our approach to building a systematic method of analyzing security of smart meters is based on building an abstract model of the meter. Smart meters are computing devices and can be considered as small general purpose computers. However, unlike general purpose computers, smart meters are low-powered, low-capacity and are designed to carry out a specific set of operations. As a result, we build an abstract model for the activity of the smart metering device based on its architecture and use cases. We design our model to be general enough to be able to represent a wide range of smart meters, while at the same time be detailed enough to identify specific vulnerabilities and possible attacks against functionality of the meter.

To perform security analysis of the meter, we derive an attack table from the abstract model. We traverse the model step by step and according to the procedures defined for each block of the model we define the attacks that target or affect those procedures (Table I). Building the attack table based on the abstract model obviates the need for coming up with generic attack trees as done in [13]. Later in Sec. IV, we will show how to derive a concrete model of the meter's implementation from the abstract model.

### C. Threat Model

In this paper, we consider the adversary to be a malicious user who has obtained system level access to the meter, even for a small period of time. We discuss the implications of this assumption in Sec. V. The attack vectors that our abstract model considers include all software attacks against the procedures running by the meter including man in the

middle attacks, buffer overflow attacks, etc. In this paper, we show how our adversary model results in performing stealthy attacks that do not change the normal behavior of the meter, but result in the meter transmitting incorrect data to the utility server. These attacks are difficult to be detected by the utility server as they do not disrupt or change the services provided by the meter. We do not consider privacy issues of smart meters and the meter's availability, i.e., denial of service attacks in our model.

### *D. Abstract Model*

We classify the procedures in a smart meter into two broad categories, namely control procedures and communication procedures. Control procedures involve receiving data from sensors, calculating consumption information, and processing data and commands coming from the utility server. Communication procedures involve sending and receiving data to/from the server and passing the data to the processes in charge of control operations. To build an abstract model of the smart meter, we create the building blocks of the operations defined for the control and communication procedures and model the execution flow among these operations. The abstract model of the smart meter is presented in Fig. 1. Below, we discuss each block. These operations correspond to our complete abstract model presented in Fig. 1. Operations 1 through 6 represent control procedures and operations 7 through 13 represent communication procedures:

1) Initializing: When starting, the meter initializes the sensors, communication interfaces, variables, etc.
2) Reading input commands that have been sent by the utility server: The meter periodically checks for any input command from the utility server.
3) Processing the input commands: Any incoming command from the server is parsed and checked for validity and destination. If the command is approved, it will be executed.
4) Reading data from sensors: Processing sensor data is done using a microcontroller. The main procedure in charge of processing consumption data periodically collects input data from all the channels associated with the sensors by scanning the input pins.
5) Producing consumption information: At each scan, for each channel, a number of samples are read and averaged. Based on the time between two consecutive scanning of each channel, the consumption data is added together.
6) Passing consumption data to the communication processes: After specific number of cycles of collecting data from channels and computing consumption data, the data are passed to another procedure to be transmitted to the server.
7) Receiving consumption information from the controller process: This procedure checks for the availability of new consumption information.
8) Checking for the accessibility of the server: Depending on the network interface installed on the meter, network communication is done through 3G, LAN, etc. Based on this, network connectivity is verified through different means such as checking with a DNS.
9) Saving data to the physical storage: The meter can be equipped with a physical storage such as flash memory. The meter writes data on a physical storage so that it does not have to send data to the server every cycle. This also provides tolerance against network disconnections.
10) Reading data from the physical storage: Before data is transmitted to the utility server, all the unsent old data is also retrieved from the physical storage and transmitted to the server.
11) Submitting data to the utility server: The meter transmits data to the utility server through its network interface.
12) Checking for incoming commands from the server: The meter periodically checks the input buffer for incoming information from the utility server.
13) Passing the incoming commands to the controller processes: incoming commands are passed to the process in step 2 to be processed.

| Attack | Target Block | Example |
|---|---|---|
| Data spoofing | 6, 7, 10 | Creating a middle process to provide fake data to the communication procedures on behalf of the control procedures |
| Physical memory | 9, 10 | Modifying/erasing data on the flash memory |
| Physical | 4, 6, 7, 8 | Tampering with network connectivity, heat attack to flash memory |
| Buffer | 6 | Filling receiver buffer with dummy data so that the legitimate command is dropped |
| Eavesdropping | 11, 12 | Recording outgoing/incoming traffic |

Table I
ATTACK TABLE. COLUMN1: CATEGORY OF THE ATTACK, COLUMN2: BLOCKS OF THE ABSTRACT MODEL TO WHICH THE ATTACK IS APPLIED, COLUMN3: AN INSTANCE OF THE ATTACK.

To derive the attack table from the abstract model, we traverse the model step by step and according to the functionality of each block, define the attacks that target those functionalities. For instance, there is a message passing step (for passing consumption data) between blocks 6 and 7 of our model (Fig. 1). This procedure is vulnerable to man in the middle attack and data spoofing. Therefore, we create an entry for "Data Spoofing" in the first column of the attack table (Table I). In the second column, we list the 'id' of the blocks of the abstract model that this attack targets (6, 7, 10), and in the third column, list instances of the attacks that can perform data spoofing. Table I is compacted due to space limitations. In Sec. IV we present more details on implementing two attacks based on the concrete model of the meter built based on our abstract model.

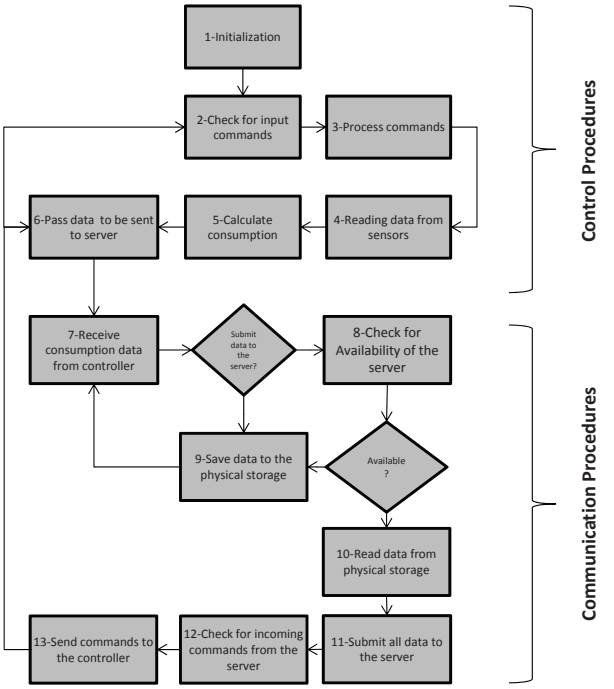


Figure 1. Abstract model for the smart meter

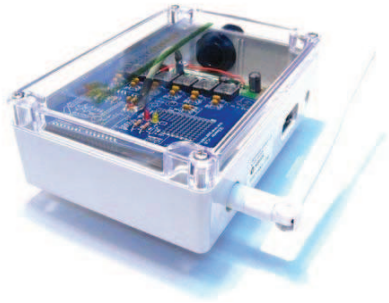

Figure 2. SEGMeter: our open source meter testbed

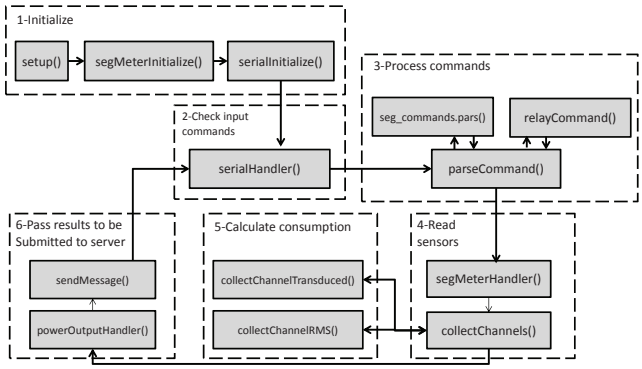


Figure 3. Concrete model for portion of the abstract model

## IV. Evaluation

### A. Testbed

We implement our solution for SEGMeter, an open source smart meter from smart energy groups [18] (Fig. 2). SEGMeter consists of two main boards: 1) an Arduino board [1] with ATMEGA32x series microcontroller which is connected to a set of sensors, receives sensor data, and calculates consumption information and, 2) a gateway board which has LAN and wifi network interfaces, and communicates with the utility server. The gateway board runs the Linux Kernel to which, we have obtained the root password (see Sec. III-C). The boards communicate with each other through a serial interface. The meter software is split between the two boards, with the communication procedures running on the gateway board and the control procedures on the Arduino board

### B. Implementation

To build the concrete model of the meter, we map each block in the abstract model to the corresponding functions in the meter software that implement the functionality of that block. This process is done manually. But since the abstract model contains the information regarding the functionality of each block, we complete the mapping and build the concrete model in a quick and effortless fashion. For most cases, we were able to identify the corresponding functions for each block by only considering the name of the functions without knowing their details. After building the concrete model of the meter, we walk through the attack table derived from the abstract model. According to the components each attack is targeting in the abstract model, we map the attack to the corresponding procedures in the concrete model of the meter. We then implement the attack targeting those procedures and evaluate the vulnerability of the meter against the attack.

Following the abstract model, we create the complete concrete model of SEGMeter. The model is complex, and hence, we discuss only a portion of it here (Fig. 3) which corresponds to the control procedures (blocks 1 to 6) in the abstract model.

For blocks 1 through 6, we identify the procedures in the code that implement the functionality of the procedures associated with each block. We model the flow of the functions based on the way they are called in the code. We present the concrete model for the control procedures in Fig. 3 and detail the content of each block in the following:

1) In the first block, the functions *setup()*, *segMeterInitialize()*, and *serialInitialize()*, are initialization procedures. These functions define the input and output pins, settings of the serial interface, type of sensors for each channel and also initialize the variables.
2) In the second block, *serialHandler()* handles input commands. It reads data from the serial interface and stores them in a buffer.
3) In the third block, *parseCommand()*, *seg_command.parse()*, and *relayCommand()*, process the input commands. These functions verify the

structure of the commands, validate if the commands are intended for the meter (comparing the name of the meter with the node name in the command string), and execute the command.

4) In the fourth block, *segMeterHandler()* and *collectChannels()* handle sensor data. These functions go through all the channels for a specific number of cycles (currently set to 30) and according to the type of sensor associated with each channel, call the appropriate functions to calculate consumption data at each cycle based on a predefined formulation.
5) In the fifth block, *collectChannelTransduced()* and *collectChannelRMS()* perform consumption calculation tasks. These functions are called according to the type the sensor for each channel and calculate the average energy for one cycle. They return the results to *collectChannels()*.
6) In the sixth block, *powerOutputHandler()* and *sendMessage()* format the consumption data in a buffer and pass the buffer to serial output.

We note that there is not necessarily a one-to-one mapping between the abstract model and the concrete model of the meter. For instance, "Reading data from sensors" and "calculating consumption data" (blocks 4 and 5), are two distinct blocks with a directed edge from data-reading block to consumption-calculation block. When we map these blocks to the corresponding functions in the implemented code, the flow is from *CollectChannel()* to *CollectChannelTransduced()*/*CollectChannelRMS()* and back to *collectChannels()* again (from block 4 to block 5 and then back to block 4 again). It means that the consumption is calculated in *collectChannelTransduced()* and then the result is returned to *collectChannels()*. This is slightly different from the abstract model in which the flow is from block 4 to block 5 and then to block 6.

### C. Attacks

We implement two attacks based on the concrete model of SEGMeter. These attacks are data spoofing attacks from our attack table that affect blocks 6, 7, and 10 of the abstract model. We call these attacks communication interface attack and physical memory attack. Both attacks attempt to fraudulently tamper with energy consumption recorded by the meter.

**Communication interface attack**: This attack is associated with blocks 6 and 7 of the abstract model and targets the communication link between these blocks. As presented in Fig. 4, "pass data" block consists of two functions of *powerOutputHandler()* and *sendMessage()*. These functions are part of the process that is running on a microcontroller installed on the Arduino board. The Arduino board is equipped with a serial port to which the *sendMessage()* function passes the consumption data. The serial port is connected to the gateway board. A process called ser2net runs on the gateway board which acts as a proxy between the serial port and other processes on the gateway board. It opens a network port to pass the data from serial port to the

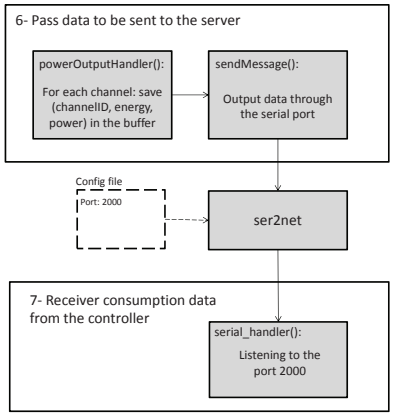


Figure 4. Concrete model for blocks 6 and 7 of the abstract model

communication processes. The port through which ser2net process is communicating is specified in a config file inside SEGMeter and is set to 2000. The function corresponding to "receive consumption data" block (block no. 7 in the abstract model) is called *serialHandler()* which connects to port 2000 and communicates with it to receive consumption data. We wrote a fake ser2net process which, at startup, replaces the original ser2net process and starts listening to port 2000. We reverse engineered the communication protocol between the gateway board and Arduino board and discovered how consumption data and time stamps are arranged in a string and passed through the serial port. Through this, the fake ser2net process produces fake consumption data, and through port 2000, passes it to *serialHandler()*.

**Physical memory attack**: This attack takes advantage of the fact that the meter writes consumption data to the flash memory whenever the network is not available. In case of network disconnection, the status of the meter changes to 'disconnected'. The next time that the connectivity check is successful, the status of the meter will be changed to 'reconnected'. During the time period from 'disconnected' to 'reconnected', the consumption data is stored on flash memory in the file "seg_node_name.dat", where "node_name" is replaced with the name of the meter. The meter continues working in 'disconnected' mode for 10 minutes and then it is automatically restarted. We wrote a script that deactivates the network interface, and overwrites the data file with fake data during the 'disconnected' period. Our script reactivates the network interface before the 10 minute time period is over to prevent the meter from restarting and enabling the meter to transmit the fake data. Activating the script periodically can significantly modify the power consumption.

**CPU and memory overhead**: In order to evaluate the detection potential of the attacks, we measure their overheads in terms of CPU and memory. Running the top command on the meter, which provides real-time status of processor activities, shows CPU overhead of 0% for both of the attack processes. We extract the information regarding the resident set in physical memory (VmRSS) for each of the attack

processes. This information is obtained from the status file inside the *proc/pid* folder, where *pid* is the process id of each attack process. For the communication interface attack, memory consumption is 564KB which is 4% of the total memory. This number for the physical memory attack is 656KB which is 4.7% of the total memory. These numbers show that the implemented attacks impose low CPU and memory overhead on the meter which makes it difficult for a generic security mechanism to detect them.

## V. Conclusion and Discussion

In this paper, we proposed an abstract model for smart meter software that allows us to systematically extract and analyze the attacks that can be applied against the meter. We mapped the abstract model to an implementation of a real open-source smart meter and implemented two attacks against the meter based on our model. Below, we discuss some of the directions in which we can expand this work.

**Applicability to other meters**: Commercial smart meters do not make their code available and we did not have access to any other meters to do further experiments. However, it is important to note that since our model is built in a workflow fashion, it is very flexible in terms of adding functionalities to the model. One of our future directions is to evaluate our model using smart meters from different vendors.

**Threat model**: The threat model in the paper (Sec. III-C) assumes that the attacker has system level access to the meter. System level access can be obtained through recovering the root password or exploiting a vulnerability in the system. For example [13] has shown the applicability of password recovery attack on a smart meter. Also, similar to any other computing systems, smart meters are potentially vulnerable to exploits such as buffer overflow attacks. In another example, [4] shows that an attacker can gain system level access to a modern car and control it by mounting a buffer overflow attack. Relaxing the assumption of system-level access on the threat model is a direction for future work.

**Characteristics of attacks**: It is important to note that the attacks presented in this paper are stealthy attacks. This means that it is difficult for the utility server to notice anything out of ordinary on the meter side. Further, the system level operations involved in these attacks (reading from/writing to data file in physical memory, interprocess communication, etc.) are all operations expected from legitimate processes running on the meter. Hence, detecting these attacks is a future challenge for intrusion detection systems.

*Acknowledgment*: This research was supported in part by the NSERC Strategic Networks Grants programme for Developing next generation Intelligent Vehicular Networks and Applications (DIVA) and Nokia Canada.

## References

[1] Arduino home page. http://www.arduino.cc.

[2] R. Berthier, W.H. Sanders, and H. Khurana. Intrusion detection for advanced metering infrastructures: Requirements and architecture directions. In *Smart Grid Communications (SmartGridComm), 2010*, pages 350 – 355, 2010.

[3] Robin Berthier and William H. Sanders. Specification-based intrusion detection for advanced metering infrastructures. *Pacific Rim International Symposium on Dependable Computing, IEEE*, 0:184–193, 2011.

[4] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, and Tadayoshi Kohno. Comprehensive experimental analyses of automotive attack surfaces. In *Proceedings of the 20th USENIX conference on Security*, SEC'11, pages 6–6, Berkeley, CA, USA, 2011. USENIX Association.

[5] Yuqun Chen, Ramarathnam Venkatesan, Matthew Cary, Ruoming Pang, Saurabh Sinha, and Mariusz H Jakubowski. Oblivious hashing: A stealthy software integrity verification primitive. *Work*, pages 400–414, 2003.

[6] Eleazar Eskin, Salvatore J. Stolfo, and Wenke Lee. Modeling system calls for intrusion detection with dynamic window sizes. *DARPA Information Survivability Conference and Exposition,*, 1:0165, 2001.

[7] K. Fehrenbacher. Smart meter worm could spread like a virus., 2010. http://earth2tech.com/2009/07/31/smart-meter-worm-could-spread-like-a-virus/.

[8] Himanshu Khurana, Mark Hadley, Ning Lu, and Deborah A. Frincke. Smart-grid security issues. *IEEE Security & Privacy*, pages 81–85, 2010.

[9] Ricardo Koller, Raju Rangaswami, Joseph Marrero, Igor Hernandez, Geoffrey Smith, Mandy Barsilai, Silviu Necula, S. Masoud Sadjadi, Tao Li, and Krista Merrill. Anatomy of a real-time intrusion prevention system. In *Proceedings of the 2008 International Conference on Autonomic Computing*, ICAC '08, pages 151–160, Washington, DC, USA, 2008. IEEE Computer Society.

[10] Michael LeMay, George Gross, Carl A. Gunter, and Sanjam Garg. Unified architecture for large-scale attested metering. In *Proceedings of the 40th Annual Hawaii International Conference on System Sciences*, HICSS '07, pages 115–, Washington, DC, USA, 2007. IEEE Computer Society.

[11] N. Lewson. Smart meter crypto flaw worse than thought, 2010. http://rdist.root.org/2010/01/11/smart-meter-crypto-flaw-worse-than-thought.

[12] P. McDaniel and S. McLaughlin. Security and privacy challenges in the smart grid. *IEEE Security & Privacy*, pages 75–77, 2009.

[13] Stephen McLaughlin, Dmitry Podkuiko, Sergei Miadzvezhanka, Adam Delozier, and Patrick McDaniel. Multi-vendor penetration testing in the advanced metering infrastructure. In *Proceedings of the 26th Annual Computer Security Applications Conference*, ACSAC '10, pages 107–116, New York, NY, USA, 2010. ACM.

[14] Miguel Correia Paulo Veríssimo, Nuno Ferreira Neves. Crutial: The blueprint of a reference critical information infrastructure architecture. In *Proceedings of the 1st International Workshop on Critical Information Infrastructures @ ISC06*, August 2006.

[15] U. Greveler B. Justus D. Lhr C. Wegener S. Brinkhaus, D. Carluccio. Smart hacking for privacy. In *28th Chaos Communication Congress*, Berlin, Germany, DEC. 2011.

[16] Dries Schellekens, Brecht Wyseur, and Bart Preneel. Remote attestation on legacy operating systems with trusted platform modules. *Sci. Comput. Program.*, 74:13–22, December 2008.

[17] Arvind Seshadri, Mark Luk, Elaine Shi, Adrian Perrig, Leendert van Doorn, and Pradeep Khosla. Pioneer: verifying code integrity and enforcing untampered code execution on legacy systems. In *Proceedings of the twentieth ACM symposium on Operating systems principles*, SOSP '05, pages 1–16, New York, NY, USA, 2005. ACM.

[18] Smart energy groups home page. http://smartenergygroups.com.

[19] Berthier R. Zonouz, S. and P. Haghani. A fuzzy markov model for scalable reliability analysis of advanced metering infrastructure. In *IEEE PES Innovative Smart Grid Technologies Conference (ISGT)*, 2012.