Enforcing end-to-end security in scada systems via application-level cryptography

Adrian-Vasile Duka, Bela Genge, Piroska Haller, Bogdan Crainicu

Chapter 8

ENFORCING END-TO-END SECURITY IN SCADA SYSTEMS VIA APPLICATION-LEVEL CRYPTOGRAPHY

Adrian-Vasile Duka, Bela Genge, Piroska Haller and Bogdan Crainicu

Abstract Recent technological advances have had a strong impact on performance optimization and the provisioning of flexible supervisory control and data acquisition (SCADA) systems. However, most SCADA communications protocols, as currently implemented, are extremely vulnerable to cyber attacks. Several international organizations have been developing security standards to alleviate these threats. Nevertheless, investigations reveal that the vast majority of high-end control hardware devices do not incorporate security features (i.e., security protocols). Therefore, the enforcement of data security in end-to-end communications flows must be addressed at the application layer. This chapter evaluates the feasibility of performing cryptographic computations at the application layer of a programmable logic controller. It shows that, despite the modest computational resources of modern programmable logic controllers, it is possible to develop efficient cryptographic applications that enforce several data security properties in the application layer. The experimental evaluations compare the performance of AES, SHA1 and HMAC-SHA1 against the performance of the new Speck and Simon lightweight block cipher algorithms executing on a Phoenix Contact ILC 350 PN controller with the control logic of a real SCADA system used in the Romanian gas transportation network.

Keywords: SCADA systems, cryptography, embedded systems

1. Introduction

The development and integration of complex software modules in industrial equipment are key factors in performance optimization and the provisioning of flexible supervisory control and data acquisition (SCADA) systems. However, most SCADA communications protocols, as currently implemented, are extremely vulnerable to cyber attacks. Since traditional computer system at-

tacks can impact SCADA systems in a significant manner [5, 6], encryption and authentication mechanisms have become mandatory requirements for protecting SCADA communications.

In an attempt to promote encryption and authentication in SCADA system communications, several organizations, including the National Institute of Standards and Technology (NIST) [17], International Electrotechnical Commission (IEC) [9] and American Gas Association (AGA) [7], have been developing security standards. A recent effort by the Object Linking and Embedding (OLE) for Process Control (OPC) Foundation has resulted in the specification of the OPC Unified Architecture (OPC UA) [13]. OPC UA provides a built-in security model, including the establishment of secure communications channels and application-layer client-server sessions.

Despite these efforts, researchers have observed that implementing computationally-intensive cryptographic algorithms in control hardware is not feasible (see, e.g., [8]). This is mainly due to the critical time constraints imposed on control logic and the limited computational resources of control hardware. Studies also suggest that symmetric cryptography (e.g., AES) combined with keyed hash-based message authentication codes (MAC) (i.e., HMAC) can enhance the security of SCADA systems in situations where the provisioning of separate, external cryptographic modules are not feasible. Additionally, despite technological progress and the roll-out of high-end devices with OPC UA support, the vast majority of high-end control hardware devices do not incorporate security features (i.e., security protocols). As a result, over the next 10 to 15 years, numerous industrial control devices with limited security features will continue to be provisioned and deployed in the field. Therefore, data security for end-to-end communications flows must be addressed at the application layer. This would enable the enforcement of security features directly on the exchanged data structures.

This chapter evaluates the feasibility of performing cryptographic computations at the application layer of programmable logic controllers (PLCs). It examines the software architecture of a traditional control application, including the structuring of variables and the planning of execution tasks. It demonstrates that, despite the modest computational resources provided by a programmable logic controller, it is possible to deploy efficient cryptographic applications that enforce various data security properties at the application layer. The chapter also demonstrates that lightweight block ciphers released recently by the National Security Agency (NSA) [1, 2] support the development of practical solutions even for time-constrained applications. To this end, the performance of the AES, SHA1 and HMAC-SHA1 algorithms are compared against the Simon and Speck families of lightweight block ciphers, and insights are provided on the possible integration of cryptographic operations in software applications of programmable logic controllers. Experimental evaluations of the performance of the cryptographic algorithms are performed using an ILC 350 PN controller from Phoenix Contact and a real control application that is deployed in the Romanian gas transportation system.

2. Related Work

Knapp and Langill [10] describe the potential risks and consequences of cyber attacks against industrial control systems; they discuss the protocols and applications underlying industrial control systems and provide general rules for their protection. Stouffer et al. [17] provide guidelines for securing industrial control systems, SCADA systems and distributed control systems, as well as other control devices such as programmable logic controllers. Saxena and Choi [14] review authentication protocols for smart grids; they specifically analyze the mutual authentication, privacy, trust, integrity and confidentiality of communications in smart grid networks.

Nai Fovino et al. [12] have presented an extension of the Modbus protocol for legacy SCADA systems that supports authentication, non-repudiation and replay protection. Although the extended protocol protects against several attacks, performance and packet size overhead may impact real-time operations. Shahzad et al. [15] have presented a methodology for deploying and testing secure communications using the Modbus/TCP protocol. A cryptographic construction deployed in Modbus/TCP provides an inclusive security solution; the algorithms integrated in industrial control network communications include AES, RSA and SHA-2. According to the authors, the approach provides efficient and inclusive security, but no details are provided about the hardware (programmable logic controllers) on which the secure Modbus protocol was implemented. In any case, according to Hohlbaum et al. [8] and confirmed by the experiments presented in this chapter, it is unlikely that the vast majority of modern control hardware can support RSA and SHA-2 computations. Therefore, it is necessary to analyze the practical integration of security solutions very carefully before deploying them in production environments.

Finally, Mohan et al. [11] have proposed the SNAPE cyber security architecture for microgrids. The architecture isolates the control network from the secure SCADA network and other external networks, provisions bump-in-the-wire devices for integrating legacy equipment that cannot perform cryptographic tasks and uses OPC UA as the communications backbone. The SNAPE architecture also leverages transport layer security (TLS) and end-device authentication. However, Mohan et al. emphasize that trade-offs have to be performed to balance security versus performance, cost and usability.

3. Problem Statement

Several organizations such as the National Institute of Standards and Technology [17], International Electrotechnical Commission [9] and American Gas Association [7] have developed security standards for industrial control systems. These efforts have identified mandatory cipher suites, including modern security protocols such as transport layer security with digital signatures, Diffie-Hellman key exchange and AES with 256-bit keys and SHA. In addition, it is recommended to implement X.509 encoded certificates with certificate management systems and associated protocols.

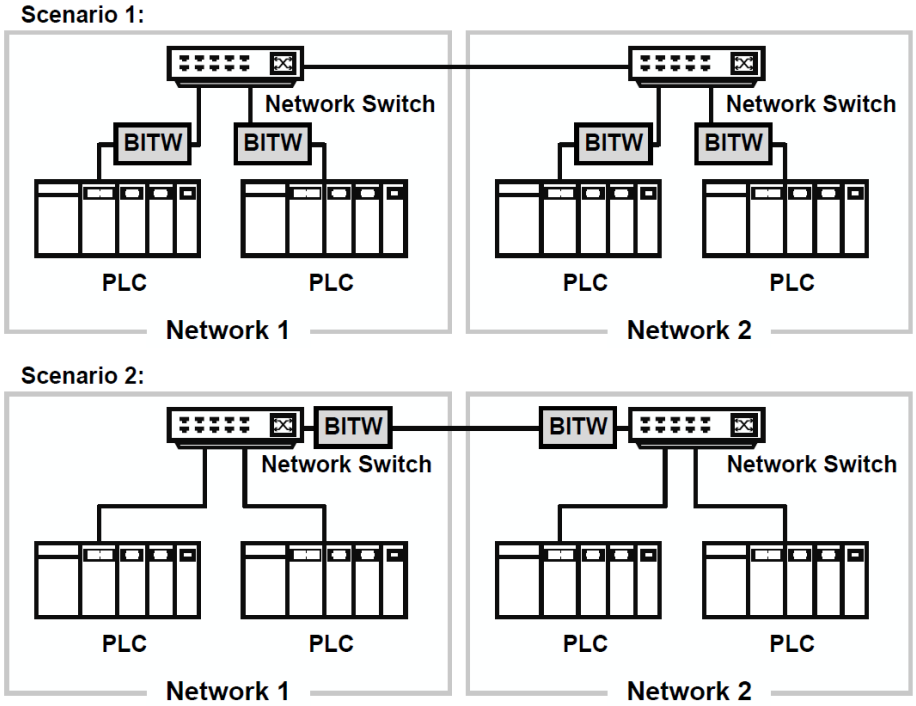


Figure 1. Bump-in-the-wire security scenarios.

While traditional cryptographic systems can indeed enhance the security of SCADA systems, careful analysis is needed to address applications of the recommended cipher suites in deployed control devices. To this end, two main approaches are available: (i) bump-in-the-wire (BITW) devices that are placed separately and in front of control hardware [11, 16]; and (ii) security mechanisms with hardware-accelerated cryptographic support [7–9] that are incorporated in communicating end-devices (e.g., programmable logic controllers).

Bump-in-the-wire solutions are typically used in end-devices with insufficient computing power [11]. This involves the positioning of a separate bump-in-the-wire device in front of each protected end-device. The bump-in-the-wire device establishes a secure communications channel between end-devices that provides a wide range of properties, including confidentiality, integrity, message authenticity and non-repudiation. Figure 1 illustrates two bump-in-the-wire security scenarios. In Scenario 1, a bump-in-the-wire device is positioned in front of each control device. In Scenario 2, a bump-in-the-wire device is positioned at each network entry-point.

From the security perspective, an obvious advantage of Scenario 1 is that, if the network is compromised and the attacker can capture/inject packets, then the internal bump-in-the-wire device would preserve the security of communications. The scenario also leverages the advantages of multiple security

devices that collectively provide defense-in-depth. Another layer of security is provided by deploying hardware and software from multiple manufacturers [17]. However, despite the advantages, Scenario 1 raises significant implementation issues. First, secure multi-peer communications require a separate channel to be established between every pair of end-devices. Unfortunately, this requires more complex routing protocols (e.g., multiprotocol label switching (MPLS)) that have to be aware of the presence of more than two end-devices in order to facilitate the routing and encryption of packets between multiple peers. Second, a significant downside is the cost of purchasing and deploying a large number of bump-in-the-wire devices. This is a major aspect of the analysis because a two-end-point virtual private network (VPN) enabler can cost as much as a programmable logic controller. Therefore, while Scenario 1 yields a highly secure network, the cost of the security devices could well exceed the cost of the SCADA infrastructure.

In contrast, in Scenario 2, a bump-in-the-wire device is positioned at the edge of each network. A secure channel is thus established between two bump-in-the-wire devices that control the access points to the two networks. While this scenario significantly reduces infrastructure costs, it does not provide end-to-end security. For example, an attacker who can inject valid packets into one of the networks can easily reach the other network by forging packets, despite the bump-in-the-wire devices positioned at the network edges.

Conversely, integrating security mechanisms in end-devices is clearly superior to using bump-in-the-wire devices because it can enforce end-to-end security requirements on each device. However, this only holds when the end-devices have adequate computing power to establish a secure communications channel. If this is the case, then application-layer authentication and data security can be achieved using a number of protocols, including OPC UA. Still, despite technological advancements and the roll-out of high-end devices with OPC UA support, investigations performed by the authors of this chapter in cooperation with Romanian automation companies reveal that deployed end-devices such as programmable logic controllers do not support security protocols. Furthermore, programmable logic controller vendors, at least those based in Romania, are only just beginning to integrate OPC UA support in their products – this means that, in the short-term, only the next generation of programmable logic controllers will implement security protocols. Unfortunately, this also means that the available high-end controllers will not support security protocols, including OPC UA. As a result, the adoption of OPC UA, at least in Romania, will only occur in the long term.

Based on these observations, it can be concluded that, although technological advancements have enhanced controller performance, high-end devices that are being integrated in SCADA systems do not support modern security protocols such as the highly-recommended OPC UA. As a result, research suggests that bump-in-the-wire solutions should be used to implement security measures. Nevertheless, the authors of this chapter believe that an increase in the computing power of programmable logic controllers would enable the in-

tegration of cryptographic algorithms and security protocols in programmable logic controller applications. However, despite improvements in the performance of modern programmable logic controllers, the devices still have limited capabilities. Therefore, it is necessary to carefully examine the computations that can be implemented in time-critical applications. The remainder of this chapter describes the research involved and the insights gained in integrating cryptographic computations in programmable logic controller applications. The analysis is based on experiments performed on a real programmable logic controller with control logic that is used in the Romanian gas transportation network.

4. Cryptographic Applications

This section discusses cryptographic applications for programmable logic controllers.

4.1 PLC Architecture

The main unit of execution in a controller (e.g., programmable logic controller) is a task. Tasks are scheduled to run periodically and may trigger the execution of blocks of code called program units; these are simply referred to as "programs." Tasks that run programs are usually called user tasks. Another key task is communications (e.g., a task that runs the communications with an OPC server/client). Each user task can be assigned a priority level and can run several programs.

Data exchanges between end-points involve variables that can be selected for reading and/or writing. The types of variables vary according to the supported data types. However, typical values include scalar variables (one to four bytes of data) along with arrays, strings, structures and user-defined data types. User-defined data structures are commonly employed to group data. Variables of this type can be selected for reading and/or writing via protocols such as OPC. A user-defined data structure is a common element of a programmable logic controller program and an important feature that is exploited in the proposed security architecture. When a different protocol is used for data transfer (e.g., Modbus/TCP), the variable view is transformed to a memory register map. In this case, a dedicated program copies variables from/to Modbus packets.

4.2 Secure Application Architecture

The proposed application architecture is designed to enforce certain security properties for data exchanges between end-points. The architecture is founded on three main assumptions: (i) symmetric key cryptography is used due to the limited computing resources; (ii) each end-point (e.g., programmable logic controller, OPC client/server or Modbus client/server) is bootstrapped via a unique secret key K_i that is used to establish session keys; and (iii) end-points i and j share the session key K_{ij}.

When the three assumptions hold, it is possible to enforce the security properties of confidentiality, data integrity and authentication on data exchanges. The confidentiality property is achieved by encrypting data using the session key K_{ij}. The data integrity property is achieved by applying a hashing algorithm. The data authentication property is achieved via authenticated hashing, where a hash function is applied to the data and secret key; this operation yields the message authentication code (MAC).

The aforementioned security properties can be enforced on data exchanges (i.e., program variables) in a number of ways. As in any security architecture, it is imperative to apply cryptographic operations before data is transmitted and after it is received. As mentioned earlier, the data exchanges are performed by the communications tasks (e.g., via the OPC protocol). Unfortunately, programmable logic controller applications usually do not have access to the transmission/reception routines of communications modules such as OPC. These vendor-specific modules are usually inaccessible to applications even when a programming license has been purchased. Therefore, cryptographic operations on the exchanged variables must be performed by the programs running in user tasks. Obviously, if the moment of transfer is deterministic and controllable by user code (as in the case of the Modbus protocol), then the cryptographic operations can be performed by the programs dedicated to these communications.

Because the unit of execution is a program, the proposed application architecture is designed to ensure that, although programs are unaware of the exact times when variable exchanges are performed, they can still enforce the security properties. Accordingly, the architecture involves two aspects. First, programs are extended with cryptographic operations on the variables that are exchanged between end-points. Second, the user-defined data structures that are exchanged are modified to include the computed security data. Thus, programs perform decryption and integrity/authenticity verification operations at the very beginning and terminate their execution with encryption and integrity/authenticity enforcement operations. The results of these computations are stored in data structures and are included in the data transfer.

Obviously, a principal concern regarding these changes to the architecture of a program is the impact on execution time. The following notation is introduced to evaluate the possible limitations of the approach. Let t_b^P and t_e^P be the computational times of the cryptographic operations performed at the beginning and at the end of the execution of a program P, respectively. Let t_u^P be the execution time of the program code.

As mentioned above, tasks are scheduled to run periodically. When a task is scheduled to run, it executes all its associated programs. Let T_α be the task scheduling time. Then, for the proposed scheme to work, it is necessary to ensure that the total execution time of all the programs P associated with task α does not exceed T_α:

$$\sum_P (t_b^P + t_u^P + t_e^P) < T_\alpha \tag{1}$$

Equation (1) ensures that the total execution time of all the programs, including the cryptographic operations, do not exceed the task scheduling period. Note that a programmable logic controller also incorporates a fail-safe mechanism implemented using a watchdog timer. Specifically, an error is signaled when the program execution exceeds a preset watchdog time T_α^W. A watchdog timer is usually configured on a per-task basis. Let T_α^W denote the watchdog timer configured for task α. Then, in addition to enforcing Equation (1), the following equation must hold:

$$\sum_P (t_b^P + t_u^P + t_e^P) < T_\alpha^W \tag{2}$$

Equations (1) and (2) demonstrate that it is necessary to reduce the values of t_b^P and t_e^P as much as possible in order to ensure the feasible application of the proposed methodology. A significant issue is that the value of T_α in a time-critical application can be very small. In fact, T_α can be a few milliseconds, which means that all the cryptographic operations must be completed in less than 1 ms.

The following recommendations help ensure practical applications of the proposed architecture:

- **Recommendation 1:** Applications should use efficient cryptographic algorithms to reduce their computational times as much as possible. Standard cryptographic algorithms such as AES and SHA-1 are recommended. Novel lightweight cryptographic algorithms designed for devices with limited computational resources have been released recently. As shown later in this chapter, block ciphers from the Speck and Simon family [1] can execute several times faster than AES while offering a similar level of security. Furthermore, these ciphers can be used to construct lightweight MAC functions that significantly outperform the HMAC standard.

- **Recommendation 2:** Applications should select variables that are secured (i.e., encrypted or hashed) to reduce their computational times. This step requires knowledge about the specific application in order to identify sensitive variables and the required security properties. A careful examination, for example, may show that it is necessary to ensure only data authenticity for a reduced set of variables that are deemed to be the most critical (e.g., variables used in control loop computations).

5. Use Case Assessment and Results

The use case assessment employed a Phoenix Contact ILC 350 PN controller running the control logic from a real SCADA system. The ILC 350 PN is a high-end controller that runs ProConOS (Programmable Controller Operating System) and is based on Windows CE technology and the .NET 4.2 framework. The control logic was provided by a local automation company that deploys SCADA systems for gas transportation system operators.

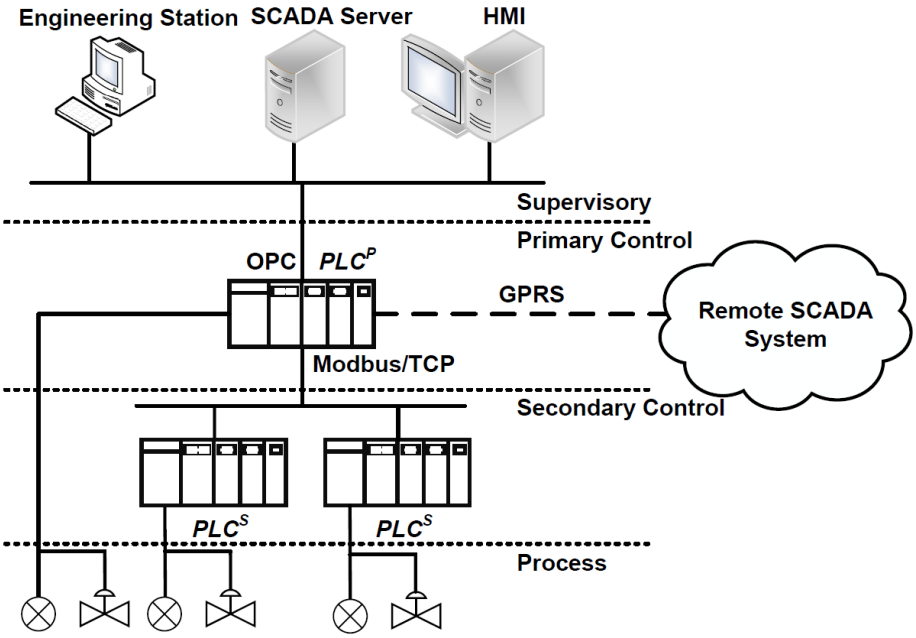


Figure 2. Simplified representation of the analyzed system.

5.1 System Analysis

The programmable logic controller application examined in this work executed the primary control logic used to automate a gas transportation node in the Romanian gas network. Figure 2 shows a simplified view of the network architecture. The control system is structured into two layers: (i) primary control layer; and (ii) secondary control layer. The primary controller PLC^P communicates with secondary controllers PLC^S via Modbus/TCP and is connected to the rest of the SCADA system (that manages information from other gas transportation nodes) via GPRS. PLC^P performs two main functions: (i) implementation of the process control loops based on measurements from PLC^S and information from transducers directly connected to it; and (ii) information exchange with the supervisory layer via OPC.

The remainder of the analysis focuses on the application architecture implemented as part of PLC^P. PLC^P executes five main tasks:

- **Task 1 ($\alpha = 1$):** This task runs two main control loop programs, P_1 and P_2, with a cycle time of $T_1 = 100\,\text{ms}$ and watchdog timer $T_1^W = 200\,\text{ms}$.

- **Task 2 ($\alpha = 2$):** This task reads the digital inputs (alarm and sensor limits) and sets the digital outputs (to open or close control relays according to the sensor alarms) every 50 ms ($T_2 = 50\,\text{ms}$) with watchdog timer $T_2^W = 100\,\text{ms}$.

- **Task 3 ($\alpha = 3$):** This task reads the analog sensor values every 50 ms ($T_3 = 50$ ms) and sets the alarm variables if the limits are exceeded ($T_3^W = 100$ ms).

- **Task 4 ($\alpha = 4$):** This task ensures communications with the secondary programmable logic controllers via Modbus/TCP. It is invoked every 10 ms ($T_4 = 10$ ms) with watchdog timer $T_4^W = 100$ ms.

- **Task 5 ($\alpha = 5$):** This task implements communications with the remote SCADA system. It runs a program unit that periodically verifies the status of the communications line every 10 ms ($T_5 = 10$ ms) with watchdog timer $T_5^W = 100$ ms.

The following data structures are transferred between the components:

- **DATAS1:** This data structure, which is 288 bytes long, is read and written by the remote SCADA system (Task 5), by Task 1 and via OPC.

- **DATAS2:** This data structure, which is 198 bytes long, is received from the secondary controllers (Task 3) and is forwarded via OPC.

- **DATAS3:** This data structure, which is 8 bytes long, is written by Task 2 and is read via OPC.

- **DATAS4:** This data structure, which is 546 bytes long, is written by Task 3 and is forwarded via OPC.

- **DATAS5:** This data structure, which is 202 bytes long, is received from the remote SCADA system and is forwarded via OPC.

Certain observations can be made based on this analysis. `DATAS1` includes measurement variables and setpoints that are changed by three sources. Therefore, the application of the proposed scheme on `DATAS1` requires careful reorganization to separate the values modified by the remote SCADA system from the values modified by PLC^P so that each end-point enforces the security properties on the data it generates. On the other hand, the security measures for `DATAS2` must be applied by the secondary controllers; this is because these controllers take the measurements. Similarly, the security properties for `DATAS5` must be enforced by the sender, which is the remote SCADA system. The security properties for `DATAS3` and `DATAS4` must be enforced by PLC^P.

5.2 Cryptographic Algorithms

This section evaluates the performance of the AES block cipher, SHA1 hash function, HMAC-SHA1 message authentication code and Speck and Simon family of block ciphers. The `IT_Security_1_00` library from Phoenix Contact, which includes AES implementations with 128-, 192- and 256-bit keys, SHA1 and HMAC-SHA1 implementations, was employed. The Speck and Simon family of block ciphers was also implemented on the ILC 350 PN controller. Speck

Table 1. Execution times of the Phoenix Contact cryptographic algorithms.

Algorithm	**Number of Calls**	**Total Time (ms)**	**Time per Call (ms)**
AES-128	500	391	0.782
AES-192	500	463	0.926
AES-256	250	266	1.064
SHA1/512 bytes	100	168	1.68
SHA1/256 bytes	100	108	1.08
SHA1/128 bytes	100	78	0.78
SHA1/64 bytes	100	63	0.63
SHA1/32 bytes	100	47	0.47
HMAC-SHA1/512 bytes	44	115.28	2.62
HMAC-SHA1/256 bytes	34	67.32	1.98
HMAC-SHA1/128 bytes	28	45.92	1.64
HMAC-SHA1/64 bytes	25	36.75	1.47
HMAC-SHA1/32 bytes	22	28.60	1.30

and Simon offer great flexibility and a range of configurations in terms of block and key sizes. Versions are available with block sizes ranging from 32 bits to 128 bits and key sizes ranging from 64 bits to 256 bits. The complete set of configurations for the two ciphers was implemented and tested.

Performance differences seen in the implemented variants of Speck and Simon are due to the data types supported by the controller. The programmable logic controller natively supports data types of 8, 16 and 32 bits as part of the IEC 61131-3 elementary data types, as well as user-defined data types that can be derived from them in the form of data structures and arrays. The programmable logic controller can perform the operations required by Simon and Speck for data sizes of n = 8, 16 and 32 bits. For the remaining cases (i.e., n = 24, 48 and 64 bits), the arithmetic for left/right rotation and modulo 2^n addition was implemented.

5.3 Computational Time

The execution times of the cryptographic algorithms implemented in the ILC 350 PN controller were evaluated by recording the number of elapsed system ticks, where 1 tick = 1 ms. The initial evaluations revealed that single calls to most of the assessed algorithms were executed in less than 1 ms. Since the controller resolution was 1 ms, all the measurements were made using multiple calls to each algorithm. The performance evaluations were performed by a dedicated task and program; the watchdog timer was set to 500 ms.

Table 1 shows the execution times of the cryptographic algorithms implemented as part of the `IT_Security_1_00` library. The results reveal that 500 calls to the AES algorithm with a 128-bit key length were executed in 391 ms.

Table 2. Execution times of the Speck and Simon cryptographic algorithms.

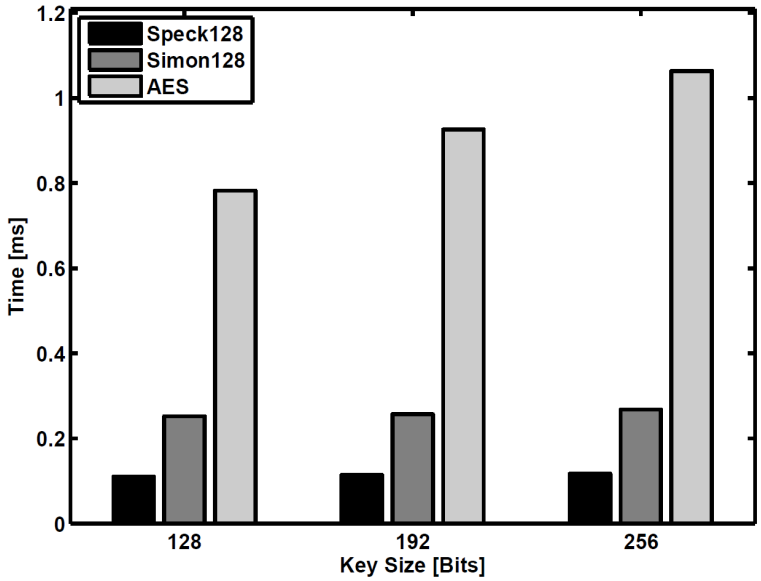
Algorithm	Total Time (ms)	Time per Call (ms)	Algorithm	Total Time (ms)	Time per Call (ms)
Speck32/64	17	0.017	Simon32/64	34	0.034
Speck48/72	27	0.027	Simon48/72	68	0.068
Speck48/96	29	0.029	Simon48/96	68	0.068
Speck64/96	19	0.019	Simon64/96	42	0.042
Speck64/128	20	0.020	Simon64/128	44	0.044
Speck96/96	92	0.092	Simon96/96	197	0.197
Speck96/144	95	0.095	Simon96/144	205	0.205
Speck128/128	110	0.110	Simon128/128	252	0.252
Speck128/196	114	0.114	Simon128/196	257	0.257
Speck128/256	117	0.117	Simon128/256	268	0.268

Note that, in this case, it can be inferred that a single call to the AES algorithm with a 128-bit key completes in 0.782 ms. Given a block size of 128 bits, a single call to AES-128 would encrypt 16 bytes of data. However, when the key size was increased to 256 bits, the program execution time exceeded the value of the watchdog timer. Hence, the number of calls was reduced to 250, for which the algorithm completed its computations in 266 ms.

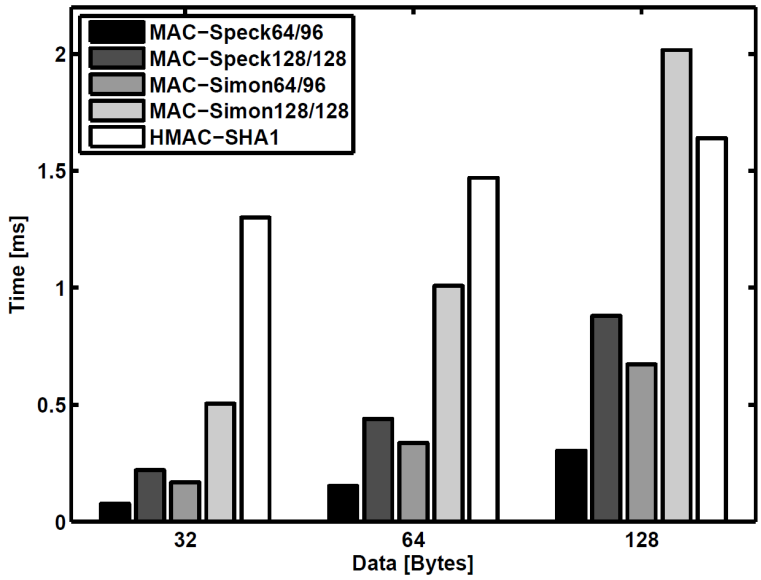
The results reveal that the SHA1 algorithm hashed 512 bytes of data in just 1.68 ms and 32 bytes of data in 0.47 ms. However, the computational times of HMAC-SHA1 increased to 2.62 ms for 512 bytes of data and to 1.30 ms for 32 bytes of data. This is because the HMAC standard includes multiple calls to the SHA1 function as well as several concatenation and arithmetic operations that affect its overall performance.

The Speck and Simon block ciphers were implemented and the correctness of all their variants was verified as documented in [1, 2]. Table 2 shows the execution times of the Speck and Simon algorithms where the total time corresponds to 1,000 calls of each algorithm. The results reveal that the Speck and Simon algorithms provide significant computational advantages over AES. For example, while AES-128 encrypted 16 bytes in 0.782 ms, Speck with 128-bit blocks and a 128-bit key (denoted by Speck128/128) encrypted the same number of bytes in just 0.110 ms. The Simon algorithm with the same configuration encrypted 16 bytes of data in 0.252 ms. In fact, Speck executed seven times faster than AES and Simon executed three times faster than AES (see Figure 3(a)). Thus, significant reductions in computational times can be obtained by adopting the lightweight cryptographic algorithms with 128-bit keys.

While the Speck and Simon algorithms may be used to enforce the confidentiality property, they can also be employed to implement lightweight MAC functions. Black et al. [3, 4] have shown that certain constructions involving block ciphers and simple XOR operations may yield collision-resistant MAC



(a) Computational times of the Speck, Simon and AES algorithms.



(b) MAC computational times based on the Speck, Simon and AES algorithms.

Figure 3. Comparison of the algorithms implemented on the ILC 350 PN controller.

Table 3. Comparison of the MAC-Speck, MAC-Simon and HMAC-SHA1 functions.

Algorithm	Execution Time (ms)			Improvement over HMAC-SHA1		
	32 B	**64 B**	**128 B**	**32 B**	**64 B**	**128 B**
MAC-Speck64/96	0.076	0.152	0.304	17.1	9.67	5.39
MAC-Speck128/128	0.220	0.440	0.880	5.9	3.34	1.86
MAC-Simon64/96	0.168	0.336	0.672	7.73	4.37	2.44
MAC-Simon128/128	0.504	1.008	2.016	2.57	1.45	**0.81**
HMAC-SHA1	1.3	1.47	1.64	–	–	–

functions. Therefore, a Merkle-Damgard construction was employed to produce lightweight MAC functions with different variants of Speck and Simon.

Figure 3(b) and Table 3 show the computational times for the MAC-Speck, MAC-Simon and HMAC-SHA1 functions on the ILC 350 PN controller. Note that when using a smaller block size (64 bits) and key size (96 bits), the MAC function based on Speck is more than 17 times faster than HMAC-SHA1 for 32 bytes of data and it is 5.39 times faster for 128 bytes of data. Similarly, the Simon64/96-based MAC function implementation is 7.73 times faster than HMAC-SHA1 for 32 bytes of data and 2.44 times faster for 128 bytes of data.

Smaller block sizes yield smaller hash sizes that decrease the collision resistance [4] of the MAC function. Therefore, a larger block size should be chosen to increase the security of the MAC function. However, in this case as well, Speck128/128 is 5.9 times faster than HMAC-SHA1 for 32 bytes of data, 3.34 times faster for 64 bytes of data and 1.86 times faster for 128 bytes of data. Simon128/128 also outperforms HMAC-SHA1 for 32 and 64 bytes of data. These results indicate that it is better to use variants of the MAC-Speck and HMAC-SHA1 functions for larger data blocks.

5.4 Security Properties in Control Applications

The analysis reveals that good performance can be achieved using lightweight cryptography for confidentiality and data authenticity. For the industrial applications under discussion, it is possible to estimate the computational overhead involved in applying the MAC functions to the data structures. In particular, the authenticity properties may be applied to the data generated by PLC^P.

The analysis above identified five data structures of different sizes that are transferred within the system and that PLC^P needs to enforce the security properties on a subset of these data structures – `DATAS3`, `DATAS4` and partly on `DATAS1`.

Since `DATAS3` is eight bytes long, one call to Speck128/128 or Simon128/128 would be sufficient to compute the MAC function. As such, the computational overhead is 0.110 ms for Speck128/128 and 0.252 ms for Simon128/128. On the

other hand, when HMAC-SHA1 is used, the computational overhead increases up to 1.23 ms.

The two larger data structures, `DATAS4` and `DATAS1`, require a different approach. `DATAS4` includes the analog/digital measurements of PLC^P performed by Task 3. In this case, using Speck128/128 would add a computational overhead of 3.73 ms and Simon128/128 would add 8.50 ms. On the other hand, using HMAC-SHA1 would reduce the computational overhead to 2.68 ms.

A similar analysis revealed that HMAC-SHA1 may be more appropriate for `DATAS1`. Nevertheless, despite its better performance, the computational overhead of HMAC-SHA1 may be too high for real-time tasks scheduled at a rate of 10 ms. The computational overhead can be reduced using a different configuration for the Speck cipher. For example, when using Speck64/96, the computational overhead for `DATAS1` is reduced to 0.684 ms, the overhead for `DATAS3` is reduced to 0.019 ms and the overhead for `DATAS4` is reduced to 1.28 ms.

Discussions with industry personnel revealed that, in general, the computational overhead of cryptographic operations should not exceed 10% of the task periodicity. Accordingly, the computational overhead for `DATAS3` (under 1 ms) can be handled by Task 2 because $T_2 = 50$ ms. On the other hand, the computational overhead for `DATAS4` in Task 3 ($T_3 = 50$ ms) would be handled best by HMAC-SHA1 or Speck64/96. Conversely, because `DATAS1` is changed by different tasks, enforcing the authenticity property for a single task (as proposed in this work) is not feasible. Therefore, in this case, the control application requires further restructuring to identify the most appropriate way to rearrange `DATAS1` without impacting the performance of applications. This topic will be investigated in future research.

Finally, according to the recommendations presented in the previous section, the computational overhead can be further reduced by carefully examining the variables for which security properties have to be enforced. This work has assumed the worst case scenario in which the security properties need to be enforced in all data exchanges. However, discussions with industry personnel revealed that the variables can be ranked based on their significance because only a subset of variables is actually used by the control loops. Therefore, the proposed protection scheme can be applied to enforce security properties in control applications. However, it should be done on a per-application basis in order to minimize the computational overhead and satisfy the control requirements.

6. Conclusions

Recent technological advances in cyber security and SCADA systems can facilitate the integration of lightweight cryptographic mechanisms in industrial control applications. This enables the enforcement of security properties such as confidentiality, integrity and data authenticity on program variables exchanged between end-points in SCADA networks. However, the efforts undertaken in this research must be extended to integrate key exchange and key distribution

protocols. Note that this research does not advocate the replacement of existing network/transport layer security protocols such as SSL/TLS and IPSec. Instead, the work complements these protocols and other related approaches by ensuring that SCADA system security properties can be enforced and verified by end-points.

Acknowledgement

This research was supported by a grant from the Romanian National Authority for Scientific Research and Innovation: CNCS/CCCDI-UEFISCDI, Project No. PN-III-P2-2.1-BG-2016-0013 within PNCDI III.

References

[1] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks and L. Wingers, The Simon and Speck Families of Lightweight Block Ciphers, National Security Agency, Fort Meade, Maryland (`eprint.iacr.org/2013/404.pdf`), 2013.

[2] R. Beaulieu, S. Treatman-Clark, D. Shors, B. Weeks, J. Smith and L. Wingers, The Simon and Speck lightweight block ciphers, *Proceedings of the Fifty-Second ACM/EDAC/IEEE Design Automation Conference*, 2015.

[3] J. Black, P. Rogaway and T. Shrimpton, Black-box analysis of the block-cipher-based hash-function constructions from PGV, *Proceedings of the Twenty-Second Annual International Cryptography Conference*, pp. 320–335, 2002.

[4] J. Black, P. Rogaway, T. Shrimpton and M. Stam, An analysis of the block-cipher-based hash functions from PGV, *Journal of Cryptology*, vol. 23(4), pp. 519–545, 2010.

[5] T. Chen and S. Abu-Nimeh, Lessons from Stuxnet, *IEEE Computer*, vol. 44(4), pp. 91–93, 2011.

[6] A. Cherepanov, BlackEnergy by the SSHBearDoor: Attacks against Ukrainian news media and electric industry, *WeLiveSecurity*, January 3, 2016.

[7] M. Hadley, K. Huston and T. Edgar, AGA-12, Part 2 Performance Test Results, PNNL-17117, Pacific Northwest National Laboratory, Richland, Washington, 2007.

[8] F. Hohlbaum, M. Braendle and F. Alvarez, Cyber security: Practical considerations for implementing IEC 62351, presented at the *Protection, Automation and Control World Conference*, 2010.

[9] International Electrotechnical Commission, IEC/TS Technical Specifications 62351-1 to 62351-7, Power Systems Management and Associated Information Exchange – Data and Communications Security, Geneva, Switzerland, 2012.

[10] E. Knapp and J. Langill, *Industrial Network Security: Securing Critical Infrastructure Networks for Smart Grid, SCADA and Other Industrial Control Systems*, Syngress, Waltham, Massachusetts, 2015.

[11] A. Mohan, G. Brainard, H. Khurana and S. Fischer, A cyber security architecture for microgrid deployments, in *Critical Infrastructure Protection IX*, M. Rice and S. Shenoi (Eds.), Springer, Heidelberg, Germany, pp. 245–259, 2015.

[12] I. Nai Fovino, A. Carcano, M. Masera and A. Trombetta, Design and implementation of a secure Modbus protocol, in *Critical Infrastructure Protection III*, C. Palmer and S. Shenoi (Eds.), Springer, Heidelberg, Germany, pp. 83–96, 2009.

[13] OPC Foundation, Unified Architecture – The Universal Communication Platform for Standardized Information Models, V1.1 GB, Scottsdale, Arizona (`opcfoundation.org/wp-content/uploads/2014/05/OPC-UA_CollaborationOverview_EN.pdf`), 2015.

[14] N. Saxena and B. Choi, State of the art authentication, access control and secure integration in smart grid, *Energies*, vol. 8(10), pp. 11883–11915, 2015.

[15] A. Shahzad, M. Lee, Y. Lee, S. Kim, N. Xiong, J. Choi and Y. Cho, Real-time Modbus transmissions and cryptography security designs and enhancements of protocol sensitive information, *Symmetry*, vol. 7(3), pp. 1176–1210, 2015.

[16] Siemens, SICAM/SIPROTEC: System Hardening for Substation Automation and Protection, Guideline (Best-Practice Guide), V1.11, Release 12.2012, Nuremberg, Germany, 2012.

[17] K. Stouffer, J. Falco and K. Scarfone, Guide to Industrial Control Systems (ICS) Security, NIST Special Publication 800-82, Revision 1, National Institute of Standards and Technology, Gaithersburg, Maryland, 2011.