Long Term Key Management Architecture for SCADA Systems

Hendra Saputra
Singapore Management University, Singapore
hsaputra@smu.edu.sg

Zhigang Zhao
Institute for Infocomm Research, Singapore
zzhao@i2r.a-star.edu.sg

***Abstract*—A SCADA key management is required to provide a key management protocol that will be used to secure the communication channel of the SCADA entities. The SCADA key management scheme often uses symmetric cryptography due to resource constraints of the SCADA entities. Normally the use of symmetric cryptography mechanism is in the form of pre-shared keys, which are installed manually and are fixed. Then, these pre-shared keys or long term keys are used to generate session keys. However, it is important that these long term keys can be updated and refreshed dynamically. With the nature of SCADA systems which may consist of hundreds of nodes deployed in different locations, manually updating and refreshing these long term keys is time consuming. Thus, this paper proposes an automatic long term key management method that updates and refreshes these keys efficiently. The proposed method provides a flexibility to refresh the long term keys and reduces the number of keys stored in the SCADA entities.**

***Keywords*-key management; long term key; SCADA;**

I. Introduction

Industrial control systems that include a supervisory control and data acquisition (SCADA) system have been widely used in many industrial facilities for decades. Before the era of the Internet, these SCADA systems operated in a closed environment. Thus, there are no security features applied in the SCADA systems, except the existence of physical access controls. With the advances of the computing and networking technologies, as well as the increase of the SCADA area usage, many of these industrial facilities are moving their SCADA systems to an open environment, which is connected to their corporate and internet network.Especially with the commercial availability of cloud computing, SCADA systems have increasingly adopted Internet of things (IoT) technologies and will continue to play an important role in Industrial Internet of Things (IIoT). Consequently, current SCADA systems face new vulnerabilities from outsider adversaries.

The security of industrial control systems is facing real threats. Several security attacks and incidents have happened in the past few years. In June 2010, a computer worm called Stuxnet caught the attention of many security experts, it attacked and infected at least 14 industrial sites in Iran, including a uranium-enrichment plant [1]. Another security breach happened in 2011. According to the deputy assistant director of FBI's Cyber Division, the infrastructure of three cities were accessed by hackers through the SCADA systems [2]. In 2014, Dragonfly hacking group successfully looted valuable information from multiple US and European energy companies [3]. Hence, it is necessary to provide security solutions to address these new security challenges.

A SCADA system generally consists of several entities that are connected in a hierarchical structure. These entities include a master terminal unit (MTU), several sub-master terminal units (subMTUs), remote terminal units (RTUs), and programmable logic controllers (PLCs). The standard communication protocols that have been widely used in the SCADA system normally do not provide any communication protection, for instance the Modbus and DNP3 protocols [4] [5]. Since SCADA systems are also used in critical infrastructures, unprotected communication channels are easily vulnerable to attacks. Hence, one of the security features that need to be added to the SCADA systems is the implementation of secure communications to provide authenticity, confidentiality and integrity.

Secure communications are built based on cryptography methods. One can use a symmetric cryptography, an asymmetric cryptography or a combination of both symmetric and asymmetric cryptography. Depending on the computing power and resources of the RTUs, asymmetric cryptography, such as public key cryptography, may not be suitable for the system. However, using symmetric cryptography requires secure key distribution methods. Normally the pre-shared keys are installed manually before the system implementation. These pre-shared keys are usually constants and fixed. However, it is very important that these pre-shared keys can be updated and refreshed. With the nature of a SCADA system that may consist of several to hundreds of nodes in different locations, manually updating these pre-shared keys is time consuming.

This paper proposes an automatic long term key management method that updates and refreshes these pre-shared keys efficiently. The proposed method provides a flexibility to refresh the long term keys and reduces the number of keys stored in the SCADA entities. Although subMTUs and MTU can have high computing power, it is still beneficial to reduce the number of keys for management purposes.

The rest of the paper is organized as follows: In section II, we discuss the related work. Section III covers the SCADA system we used and the proposed key management model. The key management algorithms are explained in detail in Section IV. Section V discusses the performance and security analysis of the proposed method. Finally the conclusions and future

work are covered in section VI.

II. Related Work

The first key management scheme for SCADA systems was proposed back in 2002 by Sandia National Laboratories (SKE) [6]. In the SKE method, the secure communication protocol uses both symmetric and public key cryptography. In this technique, the pre-shared keys are installed manually before system implementation. In 2006, an AGA 12 team (American Gas Association) published a secure communication standard for SCADA systems [7]. However, this standard did not include any key management scheme. Originally the AGA 12 would publish 4 different reports, but currently only two parts are available. The 2nd AGA 12 standard was adopted by IEEE as IEEE Standard 1711-2010, which published early 2011 [8]. In this standard, the pre-shared keys are installed manually in each device. Then, the session key is generated using these pre-shared keys. Nonetheless, this standard was withdrawn in 2014.

Researchers at Queensland University of Technology proposed the SKMA method that improves the efficiency of the SKE technique [9]. SKMA uses a modified ISO 11770-2 Mechanism 9 as its secure key protocol. Furthermore, SKMA only uses symmetric cryptography techniques. The pre-shared keys in this method are also installed manually. Assuming that each entity in the SCADA system has its own unique pre-shared key, the total number of the pre-shared keys stored in each RTU, subMTUs and MTU are *1*, *r*, and *m(r+1)* respectively, where *r* is the number of RTUs in every subMTU, and *m* is the number of subMTUs.

In 2009, ASKMA [10] and ASKMA+ [11] were proposed to improve the efficiency of SKMA in terms of the number of keys stored in a device and to address broadcast/multicast issues. In ASKMA, the logical key structure is organized as a binary tree/n-ary tree, while in ASKMA uses Iolus framework and a binary tree. However, these protocols have fixed pre-shared keys.

There have been other related work that uses asymmetric cryptography instead of symmetric cryptography, for instance the ID-based key management system (IKMS) [12], and the LiSH key management system [13].

III. System Model

We consider a SCADA system that consists of a MTU, subMTUs and RTUs or PLCs as depicted in Figure 1 which all entities have synchronized clocks. There are four types of communication: MTU-subMTUs, subMTUs-RTUs, MTU-RTUs, and RTUs-RTUS communications. Each entity in the SCADA system has several unique IDs, where the subMTU has a list of its RTU's IDs, and the MTU has the list of all entities' IDs. The MTU and subMTUs have reasonable computing power while RTUs have less computing power. The main Key Distribution Center (KDC) in our system is located at the MTU. In every subMTU there is a local KDC. When the SCADA system is starting up, there is a sequence of operations that needs to be performed. The first operation

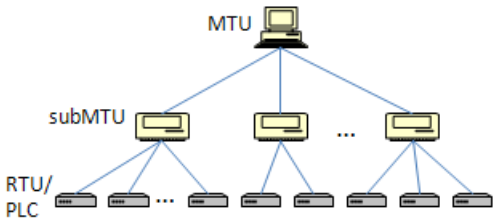

Fig. 1. SCADA Structure

is the key distribution process between subMTUs and RTUs, and between MTU and subMTUs. The second operation is the KDC update from the subMTUs to the MTU so that the MTU can communicate with all RTUs.

We assume that the MTU and subMTUs are located in the secure areas which are protected through some physical and logical access control. We modeled our SCADA system similar to the SCADA system in a Metro Rail Architecture, where the subMTUs are located in the Metro Rail Stations and the MTU is located in the Operating Control Room (OCR). Thus, we only consider adversaries that are able to compromise several RTU nodes and try to access other uncompromised RTUs' critical data.

The proposed method only considers the SCADA key management, which requires pre-shared keys such as *Long Term Key* (LTK) that are usually installed manually before the system implementation. This paper only focuses on how to generate, distribute, and update these LTKs efficiently. The detail of the key management itself, for instance the session key establishment and the broadcast key protocol, can use other existing methods, such as SKMA, ASKMA/ASKMA+.

A. Key Management Model

Instead of using and installing a fixed LTK (*Long Term Key*) in every RTU or subMTU before the system implementation, the proposed scheme installs a Setup-Key generation program in each SCADA entity. The Setup-Key program generates a setup key (K_{setup}) dynamically based on the device's identifiers (*RTU IDs / subMTU IDs*), *time*, and the previous LTK as shown in Figure 2. The setup key is used to encrypt the new LTK that is generated by the KDCs. The setup-key program has two variants: Setup-Key$_{RTU}$ program and Setup-Key$_{MTU}$ program. The Setup-Key$_{RTU}$ program is only installed in RTUs while the Setup-Key$_{MTU}$ program is installed in all subMTUs and MTU. The difference between these two programs is that in the Setup-Key$_{RTU}$ program the previous LTK value is stored in the device while the Setup-Key$_{MTU}$ program has two options in generating the previous LTK value. The first option is for a device that acts as a subordinate, in which the previous LTK value is already available in the device, because the device stores its LTK. The second option is for a device that acts as a controller. When the device receives an incoming request for a new LTK from its subordinate, it needs to generate the previous LTK value of that subordinate dynamically, based on the data stored in

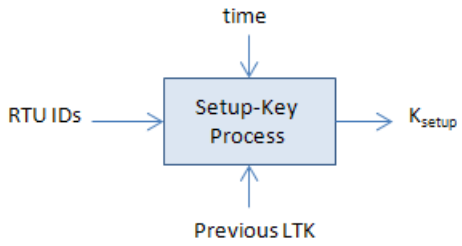

Fig. 2. Setup-Key Generation Code

Fig. 3. Device Initialization

its local KDC.

The device IDs can be the serial number of the device, MAC address, or other types of device identifier. The serial number of the device can be in the form of the CPU serial number, board serial number, or hard disk serial number. How to retrieve this serial number depends on the firmware provided by each device manufacturer. For instance, the RDREC command can be used to retrieve the serial number of the CPU in SIEMENS PLCs running on the firmware V2.0.x or greater. Note that this serial number also needs to be given to the system engineer of the company who wants to implement the SCADA system with the proposed key distribution method. The system engineer should be trusted and he is responsible to keep the serial number from leaking to the public.

The *time* represents the time period when the communication is initiated. It can be in term of hour or every several minutes. For instance when the communication starts at 10:13:20am and the *time* is described as every 1 minute, the communication time will be at 10:13am.

In the initial system setup the previous LTK of a device is not available. Thus the Setup-Key generation program only takes 2 inputs: the device IDs and time. Note that the initial system setup is done in a safe environment by the trusted system engineer as shown in Figure 3. After the Setup-Key program is installed in the new device, the system engineer connects to the appropriate KDC securely and stores the device IDs. Then the new hardware is connected to the KDC so that it can requests for LTK^0. Such setup procedure is usually a common practice in a company which a new hardware should be initially setup and checked by the system administrator before being used and operated. After the initial system setup, the device can be installed on the actual device's location.

To support the proposed key management method, the KDC stores two tables: Master Key Table and Entity Table as shown

Master Key Table

Master Keys
Master $Keys_0$
Master $Keys_1$
....
Master $Keys_{p+1}$

Entity Table

IP	MAC	Serial No.	Login	Version
...	....	.....	-1	0
...	....	.....	-1	0
...	....	.....	-1	0
...	....	.....	-1	0

Fig. 4. Master Key and Entity Tables

in Figure 4. The Master Key table contains a list of Master Keys. A Master Key is used as a seed for generating an LTK. The Master Key for RTUs in subMTU is called K_{M-SM} while the Master Key for subMTUs in MTU is called K_{M-MTU}. Note that each subMTU has its own K_{M-SM}. The MTU has all of the latest K_{M-SM} of its subMTUs. Thus, the number of Master Keys stored in each subMTU and MTU are $1 + p$ and $1 + p + m$ respectively, where p is the duration how long a device can be offline and m is the number of subMTUs. The Entity table contains the IDs, Login and Version parameters of its subordinates. The Login parameter identifies which Master Key is currently used by a particular SCADA entity. The Version indicates the latest LTK version currently used in that entity.

A Master Key is used in single operation duration. Operation duration starts when a new Master Key is generated and ends when this Master Key is replaced by another new Master Key. These Master Keys can be renewed whenever the SCADA system restarts or at any particular time using a specific command. When a Master Key is renewed while the system is in operation, all LTKs that were generated from the previous Master Key should be updated accordingly. If the updated Master Key is K_{M-SM}, the new K_{M-SM} should be passed to the MTU.

B. Long Term Key Distribution and Update Processes

The complete long term key distribution process is shown in Figure 5. When the SCADA system starts up every entity except MTU requests for a new LTK, LTK^N where $N \geq 1$, to its controller. This request is encrypted by the individual K_{setup}. Every RTU and subMTU generates its K_{setup} based on its IDs, the time period when the request is initiated, and its previous LTK, LTK^{N-1}. When the controller receives this request, it needs to retrieve the data belongs to the subordinate (RTU or subMTU) indicated by the source IP address from the local KDC: IDs, Login, and Version. Then the controller can generate the previous LTK of the subordinate based on the Master Key_{Login}, IDs, and Version. Finally the K_{setup} of that subordinate can be generated. After decrypting the incoming request, the controller validates the request. If the request contains the correct IDs of the subordinate then LTK^N can be generated from the controller's latest Master Key. The LTK^N is sent back to the subordinate. It decrypts the LTK^N package, retrieves the LTK^N and stores it. Finally the subordinate sends a confirmation message to the controller that the LTK^N is

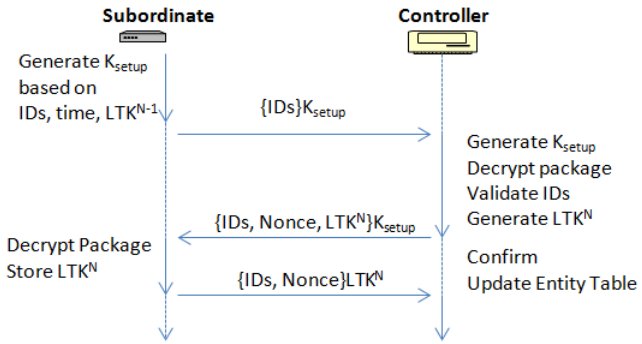

Fig. 5. LTK Distribution

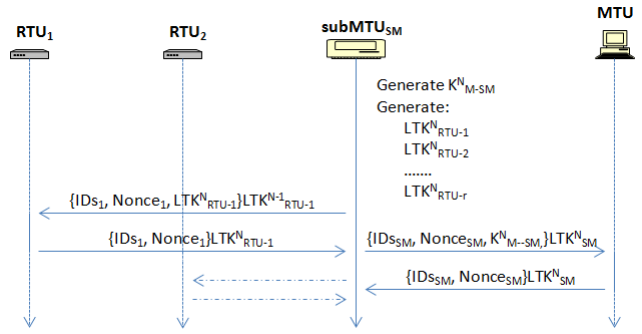

Fig. 6. Key Update Establishment

successfully received and stored. Then the controller updates the Login and Version fields in its local KDC. When every entity in the SCADA system has received its LTK, every subMTU can send its current Master Key, Login and Version fields to the MTU.

The LTK update process is performed according Figure 6. When a new Master Key K_{M-SM} is generated in a subMTU, the subMTU generates a list of new LTKs for its subordinates based on this new K_{M-SM}. Then the subMTU sends each of these LTKs to the appropriate subordinate encrypted by the previous LTK. After receiving the LTK update, each subordinate sends a confirmation message encrypted by the new LTK. Then the subMTU can store the new K_{M-SM} in the Master Key table in FIFO order and sends the new K_{M-SM} to the MTU. The LTK update process can also be performed without renewing the Master Key. In this case, the LTK is renewed based on the value of Version field in the Entity Table. The subMTU can increase the Version values in order to update the LTK of its subordinates. The difference of these LTK update processes is that in the case of renewed Master Key, the LTKs of all subordinates should be updated while in the case of increased Version field, only the subordinates whose the Version fields got renewed that need to be updated.

IV. Key Management Algorithm

The algorithms of the setup-key generation programs are shown in Figure 7 and in Figure 8. Figure 7 is the algorithm for Setup-Key$_{RTU}$ program which is only installed in RTUs.

Setup-Key$_{RTU}$ Algorithm
begin
1. time = get the current time
2. ids = get the device id
3. if LTK_{device} exists then
   3.1. return hash(ids + time + LTK_{device})
4. else
   4.1. return hash(ids + time)
end

Fig. 7. Setup-Key$_{RTU}$ Program

Figure 8 is the algorithm for Setup-Key$_{MTU}$ program which is installed in every subMTU and MTU.

The Login and Version fields in the Entity Table are used for the LTK generation. In the beginning, the Login field and the Version field of all entities are set to -1 and 0. These values represents that all entities are new devices that have not been initially set up. In the initial stage, all KDCs should have at least 1 Master Key.

The Login field indicates which Master Key is used to calculate the K_{setup} of a subordinate dynamically while the Version field is used to multiply the hash function in generating the LTK. The complete algorithm of the LTK generation code is shown in Figure 9. The MK[] contains the list of Master Keys which index *0* points to the latest confirmed Master Key. Index p shows the oldest Master Key that the controller still maintains. For instance, if an RTU has been offline longer than the period permitted such that the Master Key used in its LTK is already discarded by the subMTU, the RTU will not be able to connect the subMTU. Consequently, the system operator will be alarmed due to the failure in accepting a new LTK request from that RTU. This RTU can work normally if it is reconfigured again as a new device and its entry in the Entity Table is reset by the trusted system engineer. If an RTU is successful in requesting a new LTK from its subMTU, then the RTU is considered as "logged in" and its entry in the Entity Table is updated. Every time a device logs in to its controller, its entry in the Entity Table is updated as follows:

- The Login Field is set to 0
- If the new LTK is produced by a new Master Key then the Version field is set to 1
- If the new LTK is produced by the existing Master Key then the Version field is set to Version + 1

When a new Master Key is generated in a subMTU, all LTKs of the logged-in subordinates need to be updated. When it has confirmed that these LTKs have been updated, this Master Key will be stored at index 0 in the Master Key List. Then the previous Master Keys will be shifted to the right. Consequently the oldest Master Key is discarded. All Login fields of the logged-in subordinates will be set to 0, while the login fields of the offline subordinates will be increment by 1. This new Master Key along with the IDs of the updated RTUs are sent to the MTU. The key distribution and update process for MTU-SubMTU are performed similarly.

```
Setup-Key_MTU Algorithm
begin
  1. time = get the current time
  2. if the device acts as a subordinate then
    2.1. ids = get the device id
    2.2. if LTK_device exists then
      2.2.1. return hash(ids + time + LTK_device)
    2.3. else
      2.3.1. return hash(ids + time)
  3. else
    3.1. IP = retrive the source IP address
    3.2. if device_IP has previously logged in
      3.2.1. params = get the entry of device_IP in
                      the Entity Table
      3.2.2. ltk = generate LTK based on params
      3.2.3. return hash(ids + time + ltk)
    3.3. else
      3.3.1. return hash(ids + time)
  end
end
```

Fig. 8. Setup-Key$_{MTU}$ Program

```
LTK Generation Algorithm(ids, MK[], req, version):
    begin
        1. seed = RS[req]
        2. for i = 2 to version do
            2.1. seed = hash(seed)
        3. return hash(seed + ids)
    end
```

Fig. 9. LTK Generation Algorithm in controller sides (subMTUs, MTU)

V. Performance and Security Analysis

In this section we give detailed security and performance analysis of the proposed method. The proposed method provides efficiency and flexibility of the key distribution procedure without sacrificing security level and incurs small performance overhead.

Theorem 1: The proposed method has the same security level as the manual pre-shared keys method.

Proof: (1) The difference between the manual pre-shared key and the proposed method is that in the manual pre-shared key, the LTKs shared between the RTUs and MTU/ subMTUs need to be installed in the RTU and MTU/ subMTUs before the system implementation, while in the proposed method, we install the setup-key generation program. In the manual pre-shared key method, the pre-shared key must be stored securely both in the RTUs and MTU/subMTUs, for instance by encrypting it. Similarly, the setup-key generation code in our scheme is also protected. The protections can be through code obfuscation, code encryption, or using a secure hardware token.
(2) In the manual pre-shared keys scheme, compromising an RTU will not make other RTUs vulnerable because their pre-shared keys are unknown. Similarly, in our approach, compromising an RTU does not make other RTUs vulnerable as well. When an RTU is compromised, adversaries will have the setup-key generation code, and the previous LTK of the compromised RTU. Based on these two properties, they are not able to deduce and predict LTKs of other uncompromised RTUs. They need to have access to the K$_{M-SM}$, LTK generation program, and other RTUs' IDs, Login and Version parameters. ■

Theorem 2: The proposed method provides authentication mechanism and simplifies the key management operations.

Proof: (1) The initial setup in the proposed method is performed in a safe environment. Together with the device's IDs and the safe environment, the authentication of a device can be guaranteed. (1) When an RTU starts up, it needs to request for a new LTK to its subMTU based on its identifiers and the previous LTK value. This request can be considered as a *login* process. If this RTU is requesting from a different location, the subMTU will not accept the request unless the local KDC is updated accordingly. Similarly, when a different RTU is placed on the existing location, the subMTU will not accept the request. Thus it provides an authentication capability based on the location and the hardware dynamically.
(2) In the manual pre-shared keys method, the pre-shared keys are fixed. There is no mechanism to modify these pre-shared keys at the RTU sides, unless the modification is done manually. In the proposed method, the LTKs can be easily updated from the subMTUs or the MTU. This simplifies the key management operations.
(3) When an RTU is offline and not in operation longer than the time permitted, the subMTU will not be able to create the same K$_{setup}$ as the one created by the RTU. Since the identifiers and the login histories of this RTU are stored in the local KDC, the subMTU/MTU can alert the system operators about the failure of the login request. Thus an appropriate action can be taken by the system operator or engineer. ■

Theorem 3: The proposed method incurs small performance and size overhead.

Proof: The performance of the setup-key generation code does not affect the system performance in general since it is only executed once at the system start up before the normal operation starts. When a node needs to communicate with another node, it needs a session key. This session key is usually valid for a period of time. The generation of the session key may depend on the value of the LTK, for instance in the SKMA method. Since the RTUs store their LTKs, the generation of a session key can be performed normally. On the other hand, a subMTU/MTU needs to run LTK generation code to generate a session key. Since the subMTU/MTU has reasonable computing power, it will not affect the communication performance. The comparison between the proposed method and the manual pre-shared keys method in term of number of keys stored in a device is shown in table I. ■

Table II represents the size and performance overheads of the proposed method. The simulations are performed as follows:

TABLE I
NUMBER OF KEYS TO BE STORED IN A DEVICE

Node	SKE	SKMA	*ASKMA+	Ours
MTU	m(1+r)	m(1+r)	mr	2+p+m
subMTU	1+r	1+r	r	2+p
RTU	1	1	1	1

(*)without considering the group keys [14]

TABLE II
SIZE AND PERFORMANCE OVERHEADS OF THE PROPOSED METHOD

Node	Size	*Setup-Key Generation	*LTK Generation
MTU	12 Kbytes	<400usec(subordinate); <200usec(controller)	<50usec
subMTU	12 Kbytes	<400usec(subordinate); <200usec(controller)	<50usec
RTU	35 Kbytes	0	-

(*)without considering database accesses and LTKver = 10

- subMTUs and MTU: It is based on VMWare Player running Ubuntu 12.04.05 with the host computer running on Windows 7. The Ubuntu is configured to have 4GB RAM and 4 processors. The host computer has 8GB RAM and Intel Xeon Processor @3.00 Hz. The applications are written in C and compiled using gcc.
- RTUs: It is simulated on Vxworks 5.5 and Tornado 2.2 running on Windows XP 32 bits. The Windows XP is configured to have 4GB RAM and 4 processors as Virtual Machine. The simulation Vxsim uses PowerPC processor.

VI. CONCLUSION

In this paper we develop an automatic long term key management method for SCADA systems that provides better efficiency and flexibility on the distribution of the long term keys. The proposed method allows the updating and refreshing of the long term keys at run time and reduces the number of keys stored in the subMTUs and the MTU without sacrificing the security and performance in general. Furthermore, the solution helps the system operators to manage and to detect abnormalaties in the deployed RTUs.

In future work, we will improve the current solution to address the possibility of any compromised subMTUs in the system by introducing a diversified LTK generation code. We will also look into how to detect compromised RTUs so that the solution can prevent and stop the communication between the subMTU/MTU and the compromised RTUs.

ACKNOWLEDGMENT

This work was supported by the National Research Foundation (NRF), Prime Minister's Office, Singapore, under its National Cybersecurity R&D Programme (Award No. NRF2014NCR-NCR001-31) and administered by the National Cybersecurity R&D Directorate.

REFERENCES

[1] D. Kushner, "The Real Story of Stuxnet," in *IEEE Spectrum*, 2013.
[2] H. Hodson, "Hackers Accessed City Infrastructure via SCADA - FBI," in *www.information-age.com*, 2011.
[3] J. Langill, E. Zambon, and D. Trivellato, "Cyberespionage Campaign Hits Energy Companies ," in *www.secmatters.com*, 2014.
[4] K. Curtis, "A DNP3 protocol primer, Technical report, DNP Users Group," DNP Users Group, 2005.
[5] MODICON, "Modicon Modbus Protocol Reference Guide," MODICON, 1996.
[6] C. Beaver, D. Gallup, W. Neumann, and M. Torgerson, "Key Management for SCADA, Technical report, Sandia," 2002.
[7] *Cryptographic Protection of SCADA Communications Part 1: Background, Policies and Test Plan (AGA 12, Part 1)*, American Gas Association Std., 2006.
[8] T. Amaio and T. Van, "IEEE 1711-2010 Security For Legacy SCADA Protocols," SEQUI Inc, 2011.
[9] R. Colin, C. Boyd, J. Manual, and G. Nieto, "SKMA – A key management architecture for SCADA systems," in *Proc. 4th Australian Inf. Security Workshop*, vol. 54, 2006, pp. 138–192.
[10] D. Choi, H. Kim, D. Won, and S. Kim, "Advanced Key Management Architecture for Secure SCADA Communications," in *IEEE Transactions on Power Delivery*, vol. 24, 2009, pp. 1154–1163.
[11] D. Choi, S. Lee, D. Won, and S. Kim, "Efficient Secure Group Communications for SCADA," in *IEEE Transactions on Power Delivery*, vol. 25, 2010, pp. 714–722.
[12] Y. H. Lim, "IKMS An ID-based Key Management Architecture for SCADA System," in *IEEE International Conference on Networked Computing*, 2011, pp. 139–144.
[13] R. Jiang, R. Lu, C. Lai, J. Luo, and X. Shen, "Robust Group Key Management with Revocation and Collusion Resistance for SCADA in Smart Grid," in *Communication and Information System Security Symposium*, 2013, pp. 802–807.
[14] O. Pal, S. Saiwan, P. Jain, Z. Saquib, and D. Patel, "Cryptographic Key Management for SCADA System: An Architectural Framework," in *Proc. International Conference on Advances in Computing, Control, and Telecommunication Technologies*, 2009, pp. 169–174.