Secure and Scalable Data Collection with Time Minimization in the Smart Grid

Journal:	*IEEE Transactions on Smart Grid*
Manuscript ID:	TSG-00726-2014
Manuscript Type:	Transactions
Date Submitted by the Author:	16-Jul-2014
Complete List of Authors:	Uludag, Suleyman; U of Michigan - Flint, Computer Science, Engineering and Physics Lui, King-Shan; The University of Hong Kong, EEE Ren, Wenyu; University of Illinois at Urbana-Champaign, Computer Science Nahrstedt, Klara; University of Illinois at Urbana-Champaign, Computer Science
Technical Topic Area :	Cyber-Physical Systems (cyber and physical security, intelligent monitoring, outage management) < Transactions on Smart Grid, Smart Grid Devices (smart sensors, advanced metering infrastructure, protocol < Transactions on Smart Grid
Key Words:	data collection in the Smart Grid, Secure data collection in the Smart Grid, scalable data collection in the Smart Grid, data collection with time minimization in the Smart Grid

Secure and Scalable Data Collection with Time Minimization in the Smart Grid

Suleyman Uludag[1], King-Shan Lui[2], Wenyu Ren[3], and Klara Nahrstedt[3]

[1]Department of Computer Science, University of Michigan - Flint, MI, USA
[2]Department of Electrical and Electronic Engineering, The University of Hong Kong, Hong Kong
[3]Department of Computer Science, University of Illinois at Urbana-Champaign, IL, USA

***Abstract*—Deployment of data generation devices, such as sensors and smart meters, has been accelerating towards the vision of Smart Grid. With insufficiencies of the legacy power grid communications protocols, increased data generation and communications bring about new challenges in collecting the data securely, efficiently and in a scalable fashion. In this paper, we present a secure and scalable data communications protocol for Smart Grid data collection. Under a hierarchical architecture, relay nodes (aka data collectors) collect and convey the data securely from measurement devices to the power operator. While the data collectors can verify the integrity, they are not given access to the content, which may pave the way for third party providers to deliver value-added services or even the data collection itself. We further present optimization solutions for minimizing the total data collection time.**

I. Introduction

In the Smart Grid, massive number of sensors or measurement devices will be installed to collect real-time information. The generated data should be collected in a secure and scalable manner. A hierarchical data collection framework is usually adopted to make it scalable. For example, in Advanced Meter Infrastructure (AMI), smart meters first report data to data concentrators [1]. Thereby, the power operator does not have to maintain a separate, expensive connection with each smart meter. Besides, data concentrators can aggregate the smart meter data to further reduce the message size. Apart from data collection, this hierarchical communication structure should also allow a delivery of a command or an instruction, issued by the power operator to be delivered to a meter or measurement device securely. In this paper, we develop a comprehensive protocol that allows a power operator to collect data, as well as send commands to measurement devices in a secure, scalable, and efficient manner.

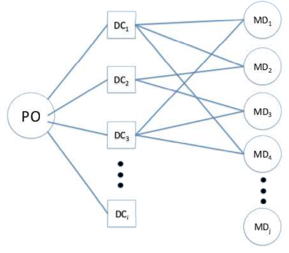


Fig. 1. Hierarchical Data Collection Structure.

Fig. 1 presents the data collection architecture considered in this paper. The Measurement Devices (MDs) are sensors or smart meters that generate power-grid specific data. They are small telemetric devices and computationally constrained. Each MD is connected to at least one Data Collector (DC), and each DC may connect to multiple MDs. The Power Operator (PO) has a direct connection with each DC. PO and DCs are relatively more powerful than MDs. The data are reported to PO via a set of DCs. PO may also issue commands to the MDs via the DCs. Theoretically, a DC is trustworthy if it is within the security domain of the PO.

However, due to the massive number of MDs and their dispersion over a large area, it may not be appropriate to assume DCs can be completely trusted. In addition, one of the seven actors identified by the NIST in the SG Framework [2] is third party service providers which are to furnish value-added services. We assume honest-but-curious model for DCs. Thus, the data collection tasks may be outsourced to third party service providers [3]. Besides, the benefits of cloud computing [4] may be accrued for storage and processing of the data collected. Data sharing to others to provide services like energy management services can be facilitated as well.

In some other applications [5], DCs are mobile and the connections between DCs and MDs are dynamic. Therefore, it would be desirable for MDs to encrypt their data in a way that DCs do not have access to them. In other words, each MD should encrypt its data using an appropriate key to keep its data private to DCs and other possible adversaries. On the other hand, due to limitation in memory and computational capability, the encryption algorithm used should be efficient. PO should also protect its commands appropriately. Apart from ensuring the security of these commands, it is also crucial to deliver these commands promptly because fast actions of MDs are necessary to maintain the stability and health of the smart grid. In this work, we develop a customized key establishment scheme and data collection protocol to protect the data and commands sent between PO and MDs via DCs in a scalable and efficient manner. In particular, our protocol has the following features:

1) Data reported by a certain MD can be accessed by the PO only, even the message is transported by a DC.
2) The protocol is light-weight in the sense that MDs do not have to perform expensive operations to report data and it does not take a lot of memory to remember key information.
3) The protocol allows commands and urgent data to be

delivered promptly and securely.

One or more DCs can be designated to collect data from a certain MD. However, the cost, the delay, and the security of data collection may differ among different DCs. The PO should select DCs according to the performance requirements. Different optimization objectives can be developed. In this paper, we study the time needed for the PO to collect all data from the MDs via the DCs. We study how to assign MDs to DCs such that the time of data collection can be minimized.

The rest of the paper is organized as follows: Sec. II describes existing efforts on data collection in smart grids. We provide the system and protocol overview in Sec. III. The details of the protocol are described in Sec. IV and V. In Sec. VI, we analyze the time performance of our mechanism and present the DC-MD assignment problem as an optimization problem. We conclude our paper in Section VIII.

II. Related Works

Data integrity and confidentiality are the major security concerns. End-to-end data protection has been studied extensively in the Internet. However, most schemes, such as TLS [6], assume the devices have abundant memory and computational power to perform expensive cryptographic operations. In smart grids, on the other hand, reporting devices have limited memory with a slow CPU. Traditional Internet security protocols are thus not suitable for data collection in smart grids. DNP3 [7] is a standard communication protocol used in SCADA (Supervisory Control And Data Acquisition). It assumes all components are within the security perimeter of the operator and is not designed to protect data forwarded by the DC as in our situation. A more recent standard for substation automation is the IP-based IEC 61850 [8]. Yet, IEC 61850 was also initially designed without security mechanisms [9]. It is thus generally agreed by the experts that new security protocols for data collection and command delivery need to be developed.

[10] and [11] study key management in smart grid. Nevertheless, these work do not consider how to hide information from DC. The SAKE protocol [12] allows two neighboring sensor nodes to establish keys using hash chains. However, the authors assume the attackers are of limited computational capability as another sensor. The authors in [13] apply the elliptic curve public key technique to perform key management. Mutual authentication between different entities is studied. Nevertheless, there is no discussion on how to protect the data reported by a sensor.

Some protocols have been developed to establish shared keys when the two parties can establish direct communication. [14] describes how to establish keys and secure unicast and multicast communications. [15] describes how to apply the Diffie-Hellman mechanism to establish a shared key for data authentication between two parties. [16], on the other hand, relies on identity-based cryptography. All these mechanisms cannot be applied in the hierarchical data collection model because the PO and the MDs cannot establish a direct connection. The authors in [17] describe how a device establishes shared keys with different controllers at different hierarchical levels. However, it is assumed that a shared key exists between two adjacent controllers.

Some efforts have been put in studying the transport protocol for data collection among a massive number of MDs. [18] studies how to reduce the storage needed when the control center needs to establish multiple sessions with the MDs. Long-term shared keys are generated by a function so that the control center only needs to memorize the function but not individual keys. Nevertheless, the key developed this way is not very secure. Besides, the protocol is not suitable for the hierarchical data collection architecture. Data collection through a data collector is considered in [19]. The authors propose to maintain two separate TCP connections, and the two connections can be protected using different mechanisms independently. Nevertheless, the data collector is assumed to be trustworthy that it can read the data sent by the MD.

[20] studies how data generators report data to a *honest-but-curious* storage center for a user to retrieve later. To the best of our knowledge, the data collection trust model assumed in this paper is the most related to our scenario. The storage center is similar to the DC in our model that it is semi-trusted, and data should be hidden from it. MDs in our model are the data generators, while PO is a user in their model. However, the paper suggests to use expensive identity-based and public key encryption to protect data to incorporate policy consideration. The experimental computational time for a decryption on a message of size less than 1000 bits in a low-end smart meter (TinyPBC library on a 32-bit ARM XScale PXA271 processor) is around 140ms, while the encryption is supposed to be a few times more expensive. Our protocol, on the other hand, encrypts data using the much more light-weighted symmetric key cryptography. We also perform experiments to study the time performance of our mechanism.

None of the papers mentioned above studies how to assign MDs to DCs. We formulate this assignment problem and develop a fast heuristic algorithm.

III. System and Protocol Overview

A. Operations and their requirements

As mentioned in Section I, our communication architecture supports MDs to report data and PO to deliver commands in a timely and secure manner. Table I describes each operation. Op 1 is a regular call-for-data from the PO which is performed periodically. Op 2 is performed when PO detects something abnormal and would like a data report from a particular MD. Time is more critical than a regular data reporting. Op 3 is done when MD detects something abnormal and would like to report to the PO. OP 4 is issued when PO needs a group of MDs to perform a certain action as soon as possible.

We develop our protocol to be secure from outsider attacks such as eavesdropping, impersonation, and message tampering, etc. There are three types of insiders in the protocol: PO, DCs, and MDs. We assume the PO is always trustworthy because it is the control of the whole system. The DCs, on the other hand, are *honest-but-curious* that they would follow

	Operation	Security Requirement	Time Requirement
Op 1	PO initiates data collection of all MDs or a group of MDs	Data reported should be authenticated and should be read only by the PO, not by other MDs or any DC	The total time to collect all data should be minimized
Op 2	PO requests data from a certain MD	Same as Op 1	The time needed should be kept minimal
Op 3	MD initiates an urgent data report	Same as Op 1	The data should be delivered to the PO as soon as possible
Op 4	PO issues an urgent command to a group of MDs	The command should be authenticated appropriately	Time for each MD to receive and read the command should be minimized

TABLE I
SYSTEM OPERATIONS AND THEIR REQUIREMENTS

the protocol as specified but would like to read the data and share with others if they could. That is, they would not impersonate another entity in the system, nor actively tamper the data, but would like to learn as much as possible based on the information they can access according to the normal operation of the protocol. As the MDs are devices located in the field (for example, on power grid poles), they are not likely to be in a very secure physical environment. We thus assume the MDs may be compromised after installation. In other words, an attacker takes over the MD and is able to read the key information kept in the device. In this situation, the attacker can report fake data to the PO on behalf of the MD. Our protocol cannot identify whether the data reported using a legitimate key is generated by an attacker, but our protocol ensures this compromised MD cannot impersonate others based on the key information it has. To detect whether a certain MD is compromised, intrusion detection techniques can be used, which is beyond the scope of this paper.

B. System Parameters

Before any communication, PO, DCs, and MDs are equipped with a set of system parameters. We assume necessary parameters are configured in a DC or MD before they are installed in the field.

1) *Long-term keys:* We assume there is a key server that can generate a set of public and private keys for each entity in the system. The public/private key pair is configured into a DC or MD before it is installed in the field. PO, on the other hand, apart from keeping its own key pair, it also remembers the public keys of all MDs and DCs in the system. We denote the public key and private key of A as A^+ and A^-, respectively. Under normal circumstances, PO would not publish the public keys of DCs and MDs to the general public. However, our protocol is secure even if the attackers know the public key information of any DC or MD they want to attack.

2) *Diffie-Hellman (DH) parameters:* We adopt the Diffie-Hellman key exchange mechanism to develop shared keys between two parties. Due to space limitation, we refer readers to [21] for the details. Generally speaking, DH allows the two parties to develop a secret shared key even eavesdroppers can read the half keys they exchange with each other. Through forgetting half keys and shared keys appropriately, DH keys also support *perfect forward secrecy*.

C. Cryptographic functions

To provide authentication, confidentiality, integrity, and other security protections, messages have to be encrypted, hashed, or signed. We assume the PO selects appropriate cryptographic algorithms for the purposes, and these functions are installed in the DCs and MDs. For example, PO may use AES for symmetric key encryption and SHA-256 for hash computation. Table II summarizes the functions used in the protocol. In the table K_p is a public key while K_s is a shared key.

Name	Description	Name	Description
$PKE(K_p, M)$	encrypt M using K_p	$PKD(K_p, C)$	decrypt C using K_p
$SKE(K_s, M)$	encrypt M using K_s	$SKD(K_s, C)$	decrypt C using K_s
$SIGN(A, M)$	sign of M by A	$SIGV(A, M)$	verify M signed by A

TABLE II
SYSTEM FUNCTIONS

Some cryptographic functions run much slower than others. As some smart grid operations are time sensitive, it is very crucial to identify efficient cryptographic functions appropriately to protect the communication. To further understand the computational time of the cryptographic functions on computationally constrained devices, we measure the time needed to execute some representative cryptographic functions on Raspberry Pi. Raspberry Pi is a tiny computer with a size similar to a credit card. The CPU is 700MHz and the memory available is 512MB. Due to space limitation, we only present some of the results. More details can be found in [22].

RSA	1024 bits			3072 bits		
Message Size (bits)	Sign. (ms)	Ver. (ms)	sign/ver ratio	Sign. (ms)	Ver. (ms)	sign/ver ratio
128	64.01	3.91	15.12	1048.37	11.49	91.23
256	64.97	4.01	16.19	1033.46	11.65	88.68
512	64.27	4.00	16.08	1047.96	11.69	89.67

TABLE III
RSA COMPUTATIONAL TIME

Table III presents the time needed to create an RSA signature and verify an RSA signature using different key sizes. The time spent on encrypting a message using public key is similar to the time needed in verifying a signature. The time needed on decrypting a message using private key is similar to the time needed on signature creation. It can be observed that the time needed does not grow with message size but with key size. Column *ratio* in the table gives the time ratio of $\frac{\text{signature computation}}{\text{signature verification}}$. The time spent on a private key operation (signing a message) is much longer than that on a public key operation (verifying a signature). An efficient protocol should not require MDs to sign a lot of messages, especially when a long RSA key is used.

We also measured the time needed to generate different Diffie-Hellman keys with different key sizes [22]. A DH shared key generation is more expensive than an RSA signature verification. It implies that it may not be appropriate to re-generate DH shared key for each data collection instance. By adopting different cryptographic functions and techniques carefully based on their security features and computational complexities, our protocol facilitates efficient and secure data collection.

D. Protocol Overview

Because encrypting data using public key cryptography is very expensive, before any data collection, we should first develop shared keys among PO, DCs, and MDs for data protection. To ensure data reported by a certain MD can be decrypted by the PO only, we need to establish a key that is known by PO and that MD. We call a key that is known by exactly two parties a *pairwise shared key*. PO and each DC should also develop a pairwise shared key to protect their conversations. The same applies to DC with each MD it will talk to. Apart from pairwise keys, to facilitate a certain command or instruction to be delivered to a group of MDs in a secure and efficient manner, we also develop a set of *group keys* that each group key is shared between the PO, a DC, and the MDs that connect to that DC. The group keys will also be used to update the pairwise shared keys efficiently. We will describe the details in Section IV-B.

The PO initiates the *Shared Key Generation Process* to establish the necessary pairwise shared keys and group keys. We adopt the Diffie-Hellman key exchange mechanism to develop all pairwise shared keys. We authenticate the DH half keys using the long-term public keys to avoid the man-in-the-middle attack. Once the pairwise shared keys and group keys are established, they will be used for data collection and command delivery.

As DH operations are expensive, we should not re-generate the DH shared keys for every data collection. However, it may not be very secure if we use the same shared keys to encrypt data collected at different times. To strike a balance of computational complexity and security, the data encryption key for each data collection instance depends on both the DH shared key and the timestamp. As the timestamp changes for every data collection instance, the data encryption key will be changed even though we do not re-generate the DH shared key. In the following, we will first describe the Shared Key Generation process in Section IV. The detailed message exchanges of the four operations mentioned in Section III-A will be provided in Section V.

IV. Shared Key Generation

Let the set of MDs be $\mathbb{MD}$ and the set of DCs be $\mathbb{DC}$. Before the PO initiates the process, PO has to assign a set of MDs for DC to connect to. We let $MDLIST_i \subseteq \mathbb{MD}$ be the set of MDs that are assigned to DC_i. Definitely, $\cup_{DC_i \in \mathbb{DC}} MDLIST_i = \mathbb{MD}$. However, $MDLIST_i \cap MDLIST_j$, where $i \neq j$, may not necessarily be $\emptyset$. It is possible that PO would like multiple DCs to collect data from the same MD to enhance reliability. In fact, different assignments between MDs and DCs would differ in data security, cost, and data collection time. In Section VI, we will formulate the assignment problem to minimize the data collection time.

In the rest of this paper, for the ease of discussion, we use *shared key* to refer to *pairwise shared key*. We further denote K^A_B as the shared key between A and B. We refer to the set $\{PO, DC_i\} \cup MDLIST_i$ as group G_i, and the group key of G_i is GK_i. We use $M1||M2$ to represent concatenating messages $M1$ and $M2$. The definitions of the functions used can be found in Table II.

A. Initial Shared Key Generation

Figure 2 presents a summary of the initial shared key generation process. When the procedure starts, the only keys an MD or a DC knows are its own public/private keys and the public key of the PO. After the procedure, MD_j should have established $K^{PO}_{MD_j}$, $D^{DC_i}_{MD_j}$, and GK_i if $MD_j \in MDLIST_i$. Through the procedure, DC_i knows GK_i, $K^{PO}_{DC_i}$ and $K^{DC_i}_{MD_j}$ for all $MD_j \in MDLIST_i$. The detailed procedure is as follows:

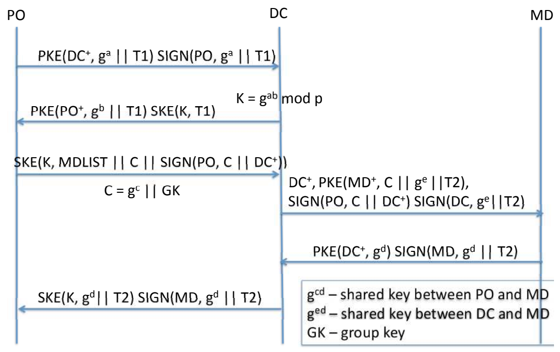

PO
DC
MD
PKE(DC⁺, gᵃ || T1) SIGN(PO, gᵃ || T1)
K = gᵃᵇ mod p
PKE(PO⁺, gᵇ || T1) SKE(K, T1)
SKE(K, MDLIST || C || SIGN(PO, C || DC⁺))
C = gᶜ || GK
DC⁺, PKE(MD⁺, C || gᵉ ||T2), SIGN(PO, C || DC⁺) SIGN(DC, gᵉ||T2)
PKE(DC⁺, gᵈ) SIGN(MD, gᵈ || T2)
SKE(K, gᵈ|| T2) SIGN(MD, gᵈ || T2)
gᶜᵈ – shared key between PO and MD
gᵉᵈ – shared key between DC and MD
GK – group key

Fig. 2. Initial Shared Key Generation.

1) PO starts the key generation process. It first generates a DH secret a to talk to the DCs. It is possible for PO to use different a's for different DCs. If so, PO has to remember the a used for each DC. Apart from a, PO also captures the current timestamp $T1$ and sends the following message to DC_i.

$$PO \rightarrow DC_i\text{: } PKE(DC_i^+, g^a||T1), SIGN(PO, g^a||T1)$$

This message is secure from message tampering and impersonating because an attacker cannot sign $g^a||T1$ correctly.

2) When DC_i receives the message, it retrieves g^a and $T1$ using DC_i^- and PO^+. It verifies whether $T1$ is reasonable. If so, it generates its DH secret b and computes K as $g^{ab} mod\ p$. K is then the shared key between PO and DC_i ($K^{PO}_{DC_i}$). It also replies PO its public DH key.

$$DC_i \rightarrow PO\text{: } PKE(PO^+, g^b||T1), SKE(K, T1)$$

An attacker, who does not know DC_i^-, cannot develop K and produce a correct message.

3) When PO receives the message, it can retrieve g^b using PO^- to compute K. It also verifies $SKE(K,T1)$ to ensure it was DC_i who sent the message. It then sends DC_i the list of MDs, together with the MDs' public keys, that it assigns DC_i to talk to. It also creates C for DC_i to talk to the MDs in the list. C contains g^c, which is used for establishing shared keys between PO and MDs, and GK_i, which is the group key of G_i. The public keys of the MDs should also be sent (We assume they are included in $MDLIST_i$ in Figure 2). It is worth noting that PO also sends $SIGN(PO,C||DC_i^+)$ to DC_i. This signature is to avoid messages from being tampered.

$$PO \rightarrow DC_i\text{: } SKE(K,MDLIST_i||C||SIGN(PO,C||DC_i^+)) \text{ where } C = g^c||GK_i$$

4) After verifying $SIGN(PO,C||DC_i^+)$ to ensure the message has not been tampered, DC_i can then generate its DH half key, g^{e_i} for establishing shared keys with the MDs. DC_i also captures the current timestamp $T2$ and sends the information to MD_j in $MDLIST_i$ using the public keys provided. DC_i also needs to send its public key. To protect DC_i^+ and $C||g^{e_i}||T2$ from being tampered, $SIGN(PO,C||DC^+)$ and $SIGN(DC_i,g^{e_i}||T2)$ are sent as well.

$$DC_i \rightarrow MD_j\text{: } DC_i^+,\ PKE(MD_j^+,C||g^{e_i}||T2),\ SIGN(PO,C||DC^+),\ SIGN(DC_i,g^{e_i}||T2)$$

5) Upon receiving the message, MD_j retrieves $C||g^{e_i}||T2$ and verifies the signatures. It then generates a DH secret key d to establish the shared key between itself and PO ($K_{MD_j}^{PO}$), which is g^{cd}, and the shared key with DC_i($K_{MD_j}^{DC_i}$), which is $g^{e_i d}$. It sends the information of g^d to DC_i. As DC_i sends the same g^{e_i} and $T2$ to all other MDs in $MDLIST_i$, MD_j has to sign g^d to authenticate the reply.

$$MD_j \rightarrow DC_i\text{: } PKE(DC_i^+,g^d),\ SIGN(MD_j,g^d||T2)$$

6) When DC_i receives the message, it retrieves g^d and verifies the signature. If so, it sends PO the key information.

$$DC_i \rightarrow PO\text{: } SKE(K,g^d||T2),\ SIGN(MD_j,g^d||T2)$$

7) If $g^d||T2$ encrypted using K and signed by MD_j are the same, the message has not been tampered. PO can then compute the $K_{MD_j}^{PO}$ to be g^{cd}. Note that as DC_i can only read g^c and g^d but neither c nor d, it cannot compute g^{cd}. g^{cd} is thus a key shared by PO and MD_j only.

We now analyze the memory needed for each entity to keep the shared keys. The PO needs to keep a shared key for each DC, a shared key for each MD, and a group key for each group. The total number of keys is 2x$|\mathbb{DC}|$ + $|\mathbb{MD}|$. DC_i has to keep $K_{DC_i}^{PO}$, a shared key with each MD belongs to its group, and a group key. The total is 2 + $|MDLIST_i|$. For MD_j, for each group G_i it belongs to, it has to keep a shared key with DC_i and the group key GK_i. It is worth noting that MD_j can establish different shared keys with PO through different DCs. If PO provides different g^c's for different DCs, the shared keys developed via different DCs must be different. Even when PO provides the same g^c through different DCs, MD_j can also establish different shared keys by replying different g^d's for different DCs. Therefore, MD_j has to keep at most *3 x number of groups it belongs to* keys in total. PO decides how many groups an MD is associated with and can thus establish keys according to the memory available in different MDs.

B. Shared Key Update

The shared keys generated are expected to be used in the subsequent data collections and command deliveries. As this shared key generation process would not be launched for every single data collection, time and computational complexities are thus not a very major concern, especially for the first time the shared keys are generated. On the other hand, a secure system should periodically change the shared keys. If the keys have to be frequently changed, the procedure in Fig. 2 might be too expensive as an MD has to handle several expensive public key and DH operations. To reduce the complexity, if the current shared keys are still secure (remain secret to attackers), we can replace the public key encryptions and signatures by symmetric key encryptions as shown in Fig. 3.

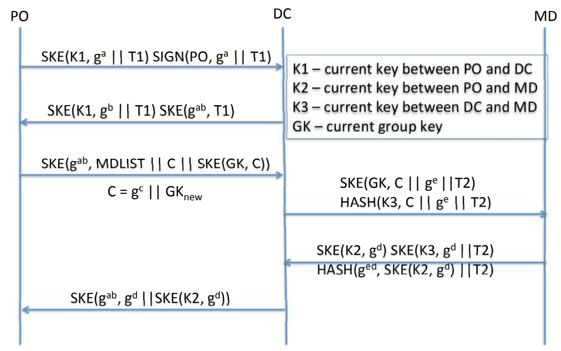

PO
DC
MD
SKE(K1, g^a || T1) SIGN(PO, g^a || T1)
K1 – current key between PO and DC
K2 – current key between PO and MD
K3 – current key between DC and MD
GK – current group key
SKE(K1, g^b || T1) SKE(g^ab, T1)
SKE(g^ab, MDLIST || C || SKE(GK, C))
C = g^c || GK_new
SKE(GK, C || g^e ||T2)
HASH(K3, C || g^e || T2)
SKE(K2, g^d) SKE(K3, g^d ||T2)
HASH(g^ed, SKE(K2, g^d) ||T2)
SKE(g^ab, g^d ||SKE(K2, g^d))

Fig. 3. Shared Key Update.

In the figure, $HASH(K,M)$ means producing a keyed-hash on message M using key K. It is worth noting that in the key update procedure, it is possible for a *malicious* DC to trick an MD to accept a fake C created by the DC. This should not happen if all DCs are *honest-but-curious*. In case it is necessary to protect a previously honest but now malicious DC to issue the attack, the PO can sign C. That is, instead of sending $SKE(GK,C)$, PO should send $SIGN(PO,C)$ to the MDs via the DC. PO can start collect data after the shared keys are established. We will describe the details of each operation in the next section.

V. Data Collection and Command Delivery

A. PO initiates Data Collection of a Group of or All MDs

It is a regular data collection initiated by the PO. To ensure the data reported can be read by the PO only, MD_j should encrypt the data using $K_{MD_j}^{PO}$. The protocol should not request MDs to perform a lot of expensive operations as well. To reduce the total time needed, we should reduce the number of messages PO or DC has to create. In the following, we first present the data collection procedure in a step by step manner. Fig. 4 shows the whole process. In the figure, $K1$, $K2$, and $K3$ are $K_{DC_i}^{PO}$, $K_{MD_j}^{PO}$, and $K_{MD_j}^{DC_i}$, respectively.

1) PO first identifies all the DCs to talk to according to a certain optimization criterion. It captures the current timestamp

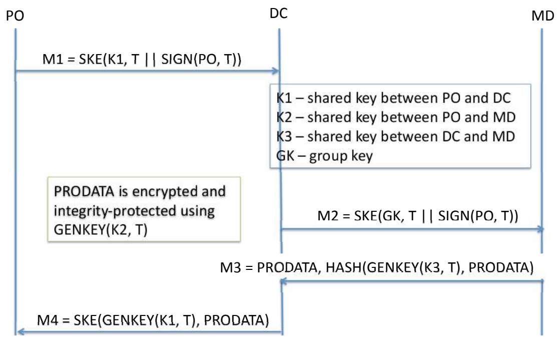

PO
DC
MD
M1 = SKE(K1, T || SIGN(PO, T))
K1 – shared key between PO and DC
K2 – shared key between PO and MD
K3 – shared key between DC and MD
GK – group key
PRODATA is encrypted and integrity-protected using GENKEY(K2, T)
M2 = SKE(GK, T || SIGN(PO, T))
M3 = PRODATA, HASH(GENKEY(K3, T), PRODATA)
M4 = SKE(GENKEY(K1, T), PRODATA)

Fig. 4. Data Collection.

T and sends a message to DC_i. Note that it is possible that PO does not want to collect data from some MDs in $MDLIST_i$. If so, PO should also include the list of intended MDs. We omit that in our protocol to simplify the discussion.

$$PO \rightarrow DC_i\text{: } SKE(K^{PO}_{DC_i}, T||SIGN(PO,T))$$

2) DC_i verifies the signature and checks whether T is reasonable. It then sends T to $MD_j \in MDLIST_i$ (or only the MDs PO wants to collect data from).

$$DC_i \rightarrow MD_j\text{: } SKE(GK_i, T||SIGN(PO,T))$$

By encrypting the message using the group key GK_i, DC_i only needs to create a single message for all MDs in its group. However, the group key cannot authenticate it was PO who requested the data collection because it is a key shared by many entities. We thus need to include a signature of PO to facilitate authentication. This message should work fine if DC_i has to collect data from every MD in $MDLIST_i$. However, when some MDs are not supposed to report data, those are not reporting can also read T in the message. If this is a serious concern, DC_i can send $SKE(K^{DC_i}_{MD_j}, T||SIGN(PO,T))$ to the involved MDs instead. The disadvantage of this approach is DC_i needs to create a different message for different MD and possibly incurs more delay in the data collection process.

3) After verifying T, MD_j generates a key $MK = GENKEY(K^{PO}_{MD_j}, T)$. An encryption key and an integrity key developed based on MK are used to protect the data. The protected data is denoted as $PRODATA$. As MK depends on T, different MK's will be used for different data collection instances even $K^{PO}_{MD_j}$ is not changed. MD_j also generates $DK = GENKEY(K^{DC_i}_{MD_j}, T)$. The hash of $PRODATA$ using DK is computed and sent to DC_i.

$$MD_j \rightarrow DC_i\text{: } PRODATA, HASH(DK, PRODATA)$$

4) DC_i verifies the hash to ensure $PRODATA$ was generated by MD_j even it cannot decrypt $PRODATA$. It then forwards $PRODATA$ to PO.

$$DC_i \rightarrow PO\text{: } SKE(GENKEY(K^{PO}_{DC_i}, T), PRODATA)$$

Alternatively, DC_i can encrypt all the replies from MDs in a single message. In this case, only a single symmetric key encryption is needed, but PO may receive some data later.

5) Finally, PO develops MK on its own to extract the data from $PRODATA$.

It can be observed that each MD, each DC, and the PO need to perform one public key operation only no matter how many messages it has to handle. Besides, the signature verification that MDs and DCs have to perform is not very expensive when compared with signature creation. Our protocol is thus very light-weight and scalable.

B. PO requests data from MD_j

1) PO first identifies a certain DC_i such that $MD_j \in G_i$. T is the timestamp. Apart from signing the timestamp, PO also encrypts the timestamp using $K^{PO}_{MD_j}$.

$$PO \rightarrow DC_i\text{: } SKE(K^{PO}_{DC_i}, T||SIGN(PO,T)||SKE(K^{PO}_{MD_j}, T))$$

2) DC_i sends the information to MD_j after verifying the signature on T.

$$DC_i \rightarrow MD_j\text{: } SKE(K^{DC_i}_{MD_j}, T||SKE(K^{PO}_{MD_j}, T))$$

Steps 3 - 5 are the same as in Section V-A.

Similar mechanism can be used for PO to issue an urgent command to MD_j. MD_j should respond with an acknowledgement instead of $PRODATA$.

C. MD_j initiates an urgent data report

1) MD_j first identifies a certain DC_i to relay the message and records the current timestamp T. $PRODATA$ and DK are generated as in Step 3 in Section V-A.

$$MD_j \rightarrow DC_i\text{:}$$
$$SKE(K^{DC_i}_{MD_j}, T||PRODATA||HASH(DK, PRODATA))$$

2) DC_i verifies the hash and forwards $PRODATA$ to PO.

$$DC_i \rightarrow PO\text{: } SKE(K^{PO}_{DC_i}, T||PRODATA)$$

3) PO can then extract T using $K^{PO}_{DC_i}$ to develop the appropriate keys to decrypt $PRODATA$.

In reporting emergency information, latency and reliability are very important. In the protocol, MD_j does not need to perform any expensive public key operation before sending the data report. The latency is thus very small. To enhance reliability, MD_j can send the data to PO via multiple DCs. It has to compute $HASH(DK, PRODATA)$ and encrypt $T||PRODATA||HASH(DK, PRODATA)$ using different keys for different DCs in Step 1. As both operations are not expensive, MD_j can send out the reports promptly.

D. PO issues an urgent command to a group of MDs

1) Similar to requesting data, PO should first identify the DCs that cover all the MDs that it wants to send the urgent command to. Let the command be $COMD$. $MDLIST_i$ contains the MDs that DC_i should talk to.

$$PO \rightarrow DC_i\text{:}$$
$$SKE(K^{PO}_{DC_i}, SIGN(PO, COMD)||MDLIST_i||COMD)$$

2) DC_i sends to each MD_j in $MDLIST_i$ the urgent command.

$$DC_i \rightarrow MD_j\text{: } SKE(GK_i, SIGN(PO, COMD)||COMD)$$

The signature of the command by the PO provides authentication check to all MDs and DCs. By using a group key in Step 2, we share the same issue as in Step 2 of Section V-A. The administrator can thus select the most appropriate way to strike a balance of security and efficiency.

VI. Grouping Optimization

We now consider how to minimize the time to perform data collection from a group of MDs by selecting a single appropriate DC to collect data from each MD. To compute the total time needed for PO to collect the data, we first define some notations to represent the time needed to perform a single cryptographic operation defined in Table II. Theoretically speaking, the time needed for a cryptographic operation depends on the size of the message. As we only perform public key operations on small-sized messages, we ignore this factor and denote $T^p(OP,A)$ as the time needed for A to execute public key cryptographic operation PKE, PKD, $SIGN$, and $SIGV$. For example, the time for PO to sign a message is $T^p(SIGN,PO)$. To capture the effect of message size on the computational time of symmetric key and hash operations, we denote the time needed as $T^s(OP,A,SIZE)$. As symmetric key encryption and decryption take roughly the same time, we use SK to represent both SKE and SKD. We also use $HASH$ to denote both hash computation and verification. To simplify our discussion, we assume the size of $T||SIGN(PO,T)$ in Section V-A as 1 unit. That is, the time needed for DC_i to develop message $SKE(GK_i,T||SIGN(PO,T))$ is $T^s(SK,DC_i,1)$. The one-way network delay between A and B is $T^n(A,B)$. We also let $x_{ij}=1$ if MD_j belongs to G_i.

To simplify our discussion, we use $M1$, $M2$, $M3$, and $M4$ to represent the four messages exchanged between PO, DCs, and MDs as shown in Fig. 4. We only consider the situation where a DC reports all data collected in a single message to PO. To illustrate the process of time analysis, we present Fig. 5 to explain the different time components in the whole data collection process. In the picture, we assume there are only two MDs.

We first develop the time needed for DC_i, after having prepared $M2$, to send message $M2 = SKE(GK_i,T||SIGN(PO,T))$ to MD_j and verify the hash of MD_j's reply, which is denoted as T_{ij}. T_{ij} is the sum of the following components:

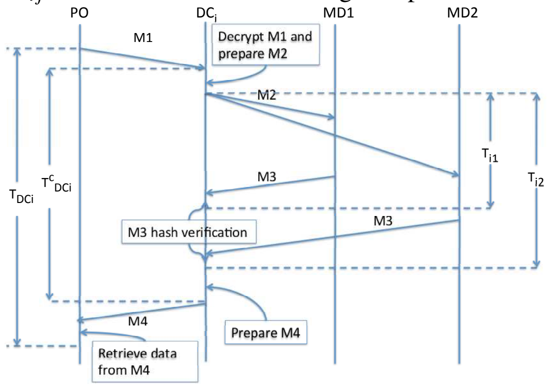


Fig. 5. Time for Data Collection

1) round-trip network delay between DC_i and MD_j: $2T^n(DC_i,MD_j)$
2) time needed for MD_j to generate reply $M3$ (Step 3): $T^s(SK,MD_j,1)+T^p(SIGV,MD_j)+T^s(SK,MD_j,size)+2T^s(HASH,MD_j,size)$ where $size$ is the size of the data in terms of number of units.
3) time needed for DC_i to verify the hash: $T^s(HASH,DC_i,size)$

Before DC_i can send message $M2 = SKE(GK_i,T||SIGN(PO,T))$ to MD_j, DC_i needs to decrypt $M1$ and prepare $M2$. As described in Step 2 in Section V-A, DC_i has to spend $2T^s(SK,DC_i,1)+T^p(SIGV,DC_i)$ time to prepare $M2$. We now study the time needed for DC_i to prepare the reply ($M4$) to PO after verifying the hashes of the replies from all MDs. Let N_i be $\sum_j x_{ij}$. That is, N_i is the number of MDs in G_i. The total amount of data received by DC_i is $N_i \times size$. The time to prepare $M4$ is $T^s(SK,DC_i,N_i \times size)$. Therefore, the total time needed for DC_i from the moment it receives $M1$ from PO to the moment it sends out $M4$ to the PO is:

$$\begin{aligned} T^c_{DC_i} &= 2T^s(SK,DC_i,1)+T^p(SIGV,DC_i)+max_j\{x_{ij}T_{ij}\} \\ &\quad +T^s(SK,DC_i,N_i \times size) \end{aligned}$$

We now study the time from the moment that PO sends out $M1$ until the moment that PO successfully decrypts and verifies the data carried in $M4$ sent by DC_i. We denote this time as T_{DC_i}. To retrieve the raw data from $M4 = SKE(GENKEY(K^{PO}_{DC_i},T),PRODATA)$, PO first needs to decrypt $M4$ using $GENKEY(K^{PO}_{DC_i},T)$. It then needs to decrypt and verify the hash carried in $PRODATA$. Therefore, T_{DC_i} is

$$\begin{aligned} T_{DC_i} &= 2T^n(PO,DC_i)+T^c_{DC_i}+ \\ &\quad 2T^s(SK,PO,N_i \times size)+T^s(HASH,PO,N_i \times size) \\ &= f(i)+max_j\{x_{ij}T_{ij}\} \qquad (1) \end{aligned}$$

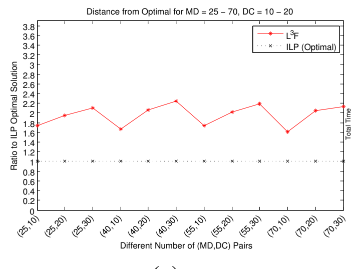
where

$$\begin{aligned} f(i) &= 2T^s(SK,DC_i,1)+T^p(SIGV,DC_i)+2T^n(PO,DC_i) \\ &\quad +2T^s(SK,PO,N_i \times size)+T^s(HASH,PO,N_i \times size) \end{aligned}$$

When PO wants to collect all data as soon as possible, we should assign each MD to an appropriate DC such that the maximum T_{DC_i} over all $i \in \mathbb{D}$ is minimized. Such an objective leads to what is known in the literature as a *minimax* problem using Integer Linear Programming (ILP). As solving ILP is NP-hard by reduction from a 3-Partition problem, we develop a greedy heuristic, Least Loaded DC First (L^{3}F), to solve the problem. We find the largest time for data collection for any (DC,MD) pair, say δ,μ. We assign MD μ to a DC that will complete in the least time. Next, we pick the next largest time and assign it to the least loaded DC for the corresponding MD. We iterate until we deplete unassigned MDs. It is obvious that the complexity of the algorithm is $O(d)$. Due to the page limitation, we omit the details of the full algorithm and refer the reader to [22].

VII. Performance Evaluation

We have used CPLEX to solve the ILP formulations and implemented our approaches in C++. Since the problem is NP-Hard, the ILP formulation that can be solved by CPLEX hits a wall rather quickly: After about 70 MDs and 35 DCs, CPLEX started taking very long to yield any results. Thus, we have run some simulations up to 70 MDs and 35 DCs each



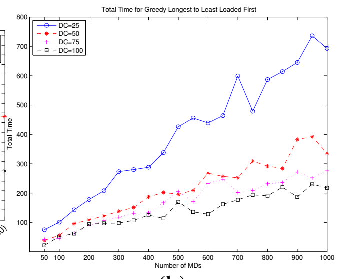


Fig. 6. (a) Ratio of total data collection time for L^{3}F to Optimal ILP, (b) Performance of L^{3}F in terms of total data collection time over changing the number of MDs with 4 different number of DCs.

with 30 runs to get an idea of the comparative performance results. The time for collecting data from MDs by DCs are randomly generated from a uniform probability distribution in the range of 10 to 100. The number of DCs took the values of 10, 20, and 30 while the number of MDs were assigned 25, 40, 55, and 70. All possible combinations were run for 30 times for statistical significance.

Figure 6a shows the performance of ILP and L^{3}F for all 12 combinations of the number of MDs and DCs. It plots the total time values returned by the ILP from CPLEX as the reference point and hence shows it as a straight line on bottom. L^{3}F, being a greedy algorithm, performed worse with an average distance ratio to the optimum of approximately 1.96.

For more MDs and DCs, ILP cannot yield results. Thus, we only report L^{3}F in extensive simulations with the number of MDs going up to 1000 in increments of 50 starting from 50 and number of DCs at 25, 50, 75, 100. We had a total of 80 unique (MD, DC) pairs. Again, in order to attain statistical significance, each combination pair was run 30 times. The time values for the data collection from MDs by DCs were generated using a uniform density function in the range of 10 to 100. Figure 6b displays the total time of data collection for L^{3}F over the number of MDs from 50 to 1000 for 25, 50, 75, and 100 DCs as separate lines. Except for when the number of DCs was equal to 25, the total time increases with respect to larger number of MDs is with moderate slope. When DC is equal to 25, the increase is rather steep but still linear. This behavior might indicate that when there is significant imbalance between the number of DCs and MDs the total time to collect data may adversely affected. This point of operating overload is hard to have a threshold value to associate with but nevertheless should be considered.

VIII. Conclusion

The bidirectional power and information flow of the Smart Grid vision has led to the proliferation of a variety of measurement devices. These devices generate unprecedented amounts of data. The existing, legacy protocols are not capable of addressing this new phenomenon. In order to address this challenge, we propose a comprehensive and secure communications protocol to enable a power operator to collect data from measurement devices in a practical, scalable, and efficient manner under a hierarchical data collection model. Intermediary nodes are assumed to follow the honest-but-curious model in relaying the data. Thus, our protocol paves the way for third party service provisioning, as envisioned by the NIST Smart Grid Framework. Examples of such services include outsourcing data collection by third party DCs, utilizing cloud computing services for data storage and processing, etc. We formulate an optimization problem for associating the intermediary relay nodes with measurement devices for data collection in order to minimize the total data collection time. The problem is intractable and thus we present a heuristic algorithm with good approximation and fast convergence.

References

[1] N. Kayastha, D. Niyato, E. Hossain, and Z. Han, “Smart grid sensor data collection, communication, and networking: a tutorial,” *Wireless Communications and Mobile Computing*, pp. n/a–n/a, 2012.
[2] National Institute of Standards and Technology. (2013, October) NIST Framework and Roadmap for Smart Grid Interoperability Standards, Release 3.0. Smart Grid Interoperability Panel (SGIP).
[3] X. Fang, S. Misra, G. Xue, and D. Yang, “Managing smart grid information in the cloud: opportunities, model, and applications,” *Network, IEEE*, vol. 26, no. 4, pp. 32–38, July 2012.
[4] S. Bera, S. Misra, and J. Rodrigues, “Cloud computing applications for smart grid: A survey,” *IEEE Tran. on Par. and Dist. Sys.*, no. 99, 2014.
[5] R. Tabassum, K. Nahrstedt, E. Rogers, and K.-S. Lui, “SCAPACH: Scalable password-changing protocol for smart grid device authentication,” in *Proc. of Third International Workshop on Privacy, Security, and Trust in Mobile and Wireless Systems (MobiPST)*, 2013.
[6] *RFC 5246*, “The transport layer security (tls) protocol version 1.2,” 2008.
[7] *IEEE 1815-2012*, “Dnp3 secure authentication version 5,” 2011.
[8] International Electrotechnical Commission’s (IEC) Technical Committee 57 (TC57). IEC 61850, Power Utility Automation .
[9] W. Wang and Z. Lu, “Cyber security in the smart grid: Survey and challenges,” *Computer Networks*, vol. 57, no. 5, pp. 1344 – 1371, 2013.
[10] X. Long, D. Tipper, and Y. Qian, “An advanced key management scheme for secure smart grid communications,” in *Proc. of IEEE SmartGridComm*, 2013.
[11] N. Liu, J. Chen, L. Zhu, J. Zhang, and Y. He, “A key management scheme for secure communications of advanced metering infrastructure in smart grid,” *IEEE Tran. on Ind. Elect.*, vol. 60, no. 10, 2013.
[12] A. Seshadri, M. Luk, and A. Perrig, “Sake: Software attestation for key establishment in sensor networks,” in *Proc. of International Conference on Distributed Computing in Sensor Systems*, 2008.
[13] D. Wu and C. Zhou, “Fault-tolerant and scalable key management for smart grid,” *IEEE Transactions on Smart Grid*, vol. 2, no. 2, June 2011.
[14] Y. Law, G. Kounga, and A. Lo, “WAKE: Key management scheme for wide-area measurement systems in smart grid,” *IEEE Comm. Mag.*, Jan. 2013.
[15] M. M. Fouda, Z. M. Fadlullah, N. Kato, R. Lu, and X. Shen, “A lightweight message authentication scheme for smart grid communications,” *IEEE Transactions on Smart Grid*, vol. 2, no. 4, Dec. 2011.
[16] C. Bekara, T. Luckenbach, and K. Bekara, “A privacy preserving and secure authentication protocol for the advanced metering infrastructure with non-repudiation service,” in *Proc. of ENERGY*, 2012.
[17] H. Nicanfar and V. Leung, “Multilayer consensus ecc-based password authenticated key-exchange (mcepak) protocol for smart grid system,” *IEEE Transactions on Smart Grid*, vol. 4, no. 1, 2013.
[18] Y.-J. Kim, V. Kolesnikov, H. Kim, and M. Thottan, “SSTP: a scalable and secure transport protocol for smart grid data collection,” in *Proc. of IEEE SmartGridComm*, 2011.
[19] T. Khalifa, K. Naik, M. Alsabaan, A. Nayak, and N. Goel, “Transport protocol for smart grid infrastructure,” in *Proc. of IEEE International Conference on Ubiquitous and Future Networks*, 2010.
[20] J. Hur, “Attribute-based secure data sharing with hidden policies in smart grid,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 11, 2013.
[21] G. Dan, K.-S. Lui, R. Tabassum, Q. Zhu, and K. Nahrstedt, “SELINDA: A secure, scalable and light-weight data collection protocol for smart grids,” in *Proc. of IEEE SmartGridComm*, 2013.
[22] S. Uludag, K.-S. Lui, W. Ren, and K. Nahrstedt, “Secure and Scalable Communications Protocol for Data Collection with Time Minimization in the Smart Grid,” University of Illinois at Urbana-Champaign, Tech. Rep., July 2014. [Online]. Available: http://hdl.handle.net/2142/49985