Efficient Authentication and Key Management Mechanisms for Smart Grid Communications

Hasen Nicanfar, *Student Member, IEEE,* Paria Jokar, *Student Member, IEEE,*
Konstantin Beznosov, *Member, IEEE* and Victor C. M. Leung, *Fellow, IEEE*

***Abstract*—Smart Grid (SG) consists of many sub-systems and networks, all working together as a system of systems, many of which are vulnerable and can be attacked remotely. Therefore, security has been identified as one of the most challenging topics in the SG development, and designing a mutual authentication scheme and a key management protocol is the first important step. This paper proposes an efficient scheme that mutually authenticates a Smart Meter (SM) of a Home Area Network (HAN) and an authentication server in SG by utilizing an initial password, by decreasing the number of steps in the Secure Remote Password (SRP) protocol from five to three and number of exchanged packets from four to three. Furthermore, we propose an efficient key management protocol based on our Enhanced Identity-Based Cryptography (EIBC) for secure SG communications using the Public Key Infrastructure. Our proposed mechanisms are capable of preventing various attacks while reducing the management overhead. The improved efficiency for key management is realized by periodically refreshing all public/private key pairs as well as any multicast keys in all the nodes using only one newly generated function broadcasted by the key generator entity. Security and performance analysis are presented to demonstrate these desirable attributes.**

***Index Terms*—Mutual Authentication; Key Management; SRP; Security; Smart Meter; Smart Grid; EIBC.**

I. Introduction

PROVIDING a high level of security is one of the most important and challenging topics in the Smart Grid (SG) design, which has gained substantial attention in the research community [1]. SG is a combination of different systems and sub-systems, and is vulnerable to various attacks that may cause different level of harms to the devices and even society-at-large [2]. Since SG is moving the power grid from a closed control system to one employing open IP networks [3], a variety of threats have been identified in the SG context, e.g., Man-In-The-Middle (MITM), Denial of Service (DoS), impersonation, which can affect the data integrity and authentication of users and devices. Moreover, different viruses or attacks such as brute-force and dictionary attacks can target the data security and confidentiality. The Stauxnet worm is another example that can cause a significant impact on even national security [3]. Once an entry point is found, an intruder or a malicious node may perform different action to compromise the whole system. Since millions of homes are connected to an SG, the impact of such attacks can cause a significant loss or harm on society, e.g., by causing a blackout, changing the customer billing information, or changing the pricing information sent to the customers [2], [3], [4].

Providing an authentication scheme and a key management protocol are the required first steps of designing and implementing system security in SG [5]. The National Institute of Standards and Technology (NIST) of the United States, which is developing SG-related standards and guidelines, suggests using Public Key Infrastructure (PKI) to secure SG communications [1]. PKI [6] is briefly reviewed in Section II. Our proposal facilitates secure and efficient authentication and key management on top of PKI.

The customer's side of an SG consists of Home Area Networks (HANs) in customer premises where smart appliances and controllers are connected to smart meters (SMs), which form the end-points of the Advanced Metering Infrastructure (AMI) that provides two-way data communications between SMs and the utility's Meter Data Management Center. This work is focused on authentication and key management over the AMI. The AMI will likely employ Internet Protocol version 6 (IPv6) technology in a mesh based topology [4]. Although power line communication (PLC) has gained much attention in Europe, in North America wireless mesh networks (WMN) is a more popular and dominant solution for the AMI [3]. Gharavi et. al [7] proposed a mesh based architecture for the last mile SG, in which the Neighbourhood Area Network (NAN) supports communications between SMs and AMI head-end via data aggregation points and mesh-relay-stations if required. This mesh based topology enables easy expansion of the network coverage area using multi-hop communications.

Secure communications generally employ cryptographic keys for encrypting/decrypting data messages. There are different solutions to establish a key between two parties, usually as a part of the authentication process, some of which are tailored for (mutual or one-way) authentication. One well-known solution to form a session (symmetric) key is the Diffie and Hellman (D-H) algorithm [8]. To protect the D-H algorithm from different attacks like MITM, Bellovin et al. proposed a solution [9] that utilizes a password to assure the secrecy of key establishment messages. Later on, Seo et al. developed a two-step Password Authenticated Key Exchange (PAKE) protocol called SAKA [10]. First, both parties obtain a number based on their shared password. Then, each party picks a random number and multiplies it to the shared number

Manuscript received: 4/15/2012; revised: 9/30/2012; accepted: 1/23/2013; date of current version 2/15/2013.

This paper is based in part on a paper appeared in Proc. of the first IEEE PES ISGT Asia conference, as well as a paper in Proc. of the IEEE SysCon.

Hasen Nicanfar, Paria Jokar and Victor C.M. Leung are with WiNMoS Lab, and Konstantin Beznosov is with LERSSE Lab.

All authors are with Department of Electrical and Computer Engineering, The University of British Columbia, Vancouver, BC V6T 1Z4, Canada.

E-mail: {hasennic , pariaj , beznosov , vleung}@ece.ubc.ca.

from the first step to be used in D-H algorithm. In [11], a verifier is utilized for key establishment, with the support of a server as a trusted third party. Each party has an individual password and the server holds the appropriate verifier. The entities establish temporary session keys used to construct the final symmetric key in a protocol with four phases.

The Secure Remote Password (SRP) protocol [12] also utilizes a predefined password and the verifier to construct a key, which delivers most of the characteristics that are expected from an authentication scheme. SRP is a fast mutual authentication scheme that uses the session key in the mechanism and resists the dictionary attacks. Furthermore, in the SRP protocol, compromising the server does not make it easy to find the password as well as compromising the password does not lead to revealing the past session keys (forward secrecy); and finally, compromising the session key does not lead to compromising of the password. More details are provided in Section II.

PKI is preferred for securing data exchanges over SG. Based on the Identity-Based signature scheme [13] proposed by Shamir, Boneh and Franklin [14] proposed ID-Based Cryptography (IBC) for encryption-decryption and key management, which extends PKI by replacing the public key of an entity with a function of the entity's ID to reduce the overhead of public key distribution. More details are provided in Section II.

Contributions: In this paper we propose a secure and efficient Smart Grid Mutual Authentication (SGMA) scheme and Smart Grid Key Management (SGKM) protocol. SGMA provides efficient mutual authentication between SMs and the security and authentication server (SAS) in the SG using passwords; it reduces the number of steps in SRP from five to three and the number of exchanged packets from four to three. SGKM provides an efficient key management protocol for SG communications using PKI as specified by NIST [1]; it employs our Enhanced Identity-Based Cryptography (EIBC) scheme to substantially reduce the overhead of key renewals.

SGMA and SGKM are presented in Sections III and IV, respectively. Security analysis in Section V shows that these schemes are capable of preventing various well-known attacks such as Brute-force, Replay, MITM and DoS. Furthermore, we reduce the network overhead caused by the control packets for key management. The improved efficiency results from our key refreshment protocol in which the SAS periodically broadcasts a new key generation to refresh the public/private key pairs of all the nodes as well as any required multicast security keys. Performance analysis in Section V verifies the overhead reduction.

II. Background Review and Related Works

A. Authentication and Key Management

By definition, authentication means binding an identity (ID) to a subject or principal. It can be accomplished by showing what the subject (i) is capable of doing, e.g., performing a digital signature, or (ii) knows, e.g., a password, or (iii) possesses, e.g., a smart card, or (iv) has biometrically, e.g., fingerprints. Moreover, in a networking environment, network nodes should follow a mutual authentication to establish a certain level of trust [1]. After two parties get authenticated to each other, they need to set-up a secure communication channel, normally by use a security key for data encryption, to protect their data from unauthorized parties. The key can be symmetric, supported by a private key cryptography system, or asymmetric, supported by a public key cryptography system.

B. Secure Remote Password Protocol

SRP [12] (latest version 6a [15]), is an authentication and key-exchange protocol for secure password verification and session key generation over an insecure communication channel. SRP utilizes Asymmetric Key Exchange (AKE) [12], and stores verifiers instead of the passwords. AKE uses a one-way (hash) function to compute the verifier and stores it in the server. Therefore, compromising the server and finding the verifier is not enough to obtain the key, since the password is still required.

In SRP, the client initially enters a password and then the server computes a verifier from the password using a randomly generated salt and stores the client's ID, salt and verifier in the server database. Subsequently, the client is authenticated to the server by providing the password to the server, which computes the verifier again using the salt stored against the client's ID and checking it against the one stored in its database. Furthermore, each party generates a random number, then calculates the session key based on the password, verifier and random numbers as well as verifies the key utilizing a one-way hash function.

C. Public Key Infrastructure

In the PKI [6], two keys, namely the public key and private key, are associated with each entity. The sender uses her private key to sign the message the public key of the recipient to encrypt the message, and the recipient uses her private key to decrypt the message and the sender's public key to authenticate the sender's ID. A Private Key Generator (PKG)/Certificate Authority (CA) issues to each entity an individual certificate consisting of the private key of the entity, and makes the public key of the entity available to the public. The PKG is required to refresh these keys periodically per the system security requirements and informs the relevant parties, which may incur a substantial communication overhead [16]. One solution to reduce this overhead is IBC.

D. Identity-Based Cryptography

In IBC [14], the PKG provides a unique one-way function $F(.)$, e.g., a hash function, to all the parties, which can be applied to ID (e.g., email address, phone number or IP address) of each party to obtain the public key of the party using (1). Then, PKG selects a secret random number "s", and applies it to the public key of each party to obtain the private key of the party via (2).

$$\begin{cases} PubK(ID) = F(ID) & (1) \\ PrvK(ID) = s * F(ID) = s * PubK(ID) & (2) \end{cases}$$

The message encryption/decryption and verification processes then follow those of PKI using the private and public keys thus generated.

Key Refreshment: In order to refresh the keys to maintain the system security, PKG periodically reselects "s", recalculates private keys of the entire entities and securely informs them one by one. Since this is time consuming, normally PKG supplies the parties with a Valid Time (VT) value, which presents the starting time of using the new keys.

E. EIBC: Enhanced Identity-Based Cryptography

Our proposed EIBC [16] enhances IBC by making the private key refreshment more efficient and accommodating distribution and refreshment of any multicast key needed in the network. The modifications to IBC are described as follows.

1) One-way/Hash function $F(.)$: The static function $F(.)$ in IBC is made dynamic in EIBC as function $F_i(.)$. Precisely, PKG periodically generates and broadcasts function $f_i(.)$ that is applied to $F_i(.)$ to obtain $F_{i+1}(.)$, which is the new one-way function of the system. In this case, all of the public keys and private keys are being updated. Each party updates the public key of any other party by applying $f_i(.)$ to the current public key of that party. Also, each party uses $f_i(.)$ in the private key refreshment algorithm that will be explained shortly. The index "i" represents the current state (called $live$ in this paper) of the system.

$$\begin{cases} F_{i+1}(.) = f_{i+1}(F_i(.)) & \text{(3a)} \\ PubK_i(ID) = F_i(ID) & \text{(3b)} \end{cases}$$

2) System secret value "s": In IBC, "s" is the product of a True Random Number Generator (TRNG) managed and kept secret by PKG. In EIBC, "s" is replaced by two values: "s_i" from (4a) is a non-shared TRNG value kept by PKG, and "$\widetilde{s}_i$" is obtained from (4b) using a Pseudo Random Number Generator (PRNG) with parameters a, b and modulus q, shared by all entities.

$$\begin{cases} s_{i+1} = f_{i+1}(s_i) & \text{(4a)} \\ \widetilde{s}_{i+1} \equiv (a * \widetilde{s}_i + b) \bmod q & \text{(4b)} \\ s.t.: \quad i, a, b, q \in \mathbb{Z} \ \& \ \widetilde{s}_i \in \mathbb{Z}_q^* \end{cases}$$

3) Seed and End Values: In EIBC, some of the parameters have a Seed Value (SV) as well as an End Value (EV). For instance, PKG has "public key SV" ($\widetilde{PubK}^i_{PKG}$) and "public key EV" ($PubK^i_{PKG}$). Moreover, each entity has a private key SV ($\widetilde{PrvK}^i_A$) and a private key EV ($PrvK^i_A$). PKG produces SVs of the keys via (5a) and via (5b), and all entities perform (6a) and (6b) to obtain the live EVs:

$$Seed\ Values: \begin{cases} \widetilde{PubK}^i_{PKG} = s_i.\breve{P}^i_{PKG} & \text{(5a)} \\ \widetilde{PrvK}^i_A = s_i.F_i(ID_A) & \text{(5b)} \end{cases}$$

$$End\ Values: \begin{cases} PubK^i_{PKG} = f_i(\widetilde{s}_i).\widetilde{PubK}^i_{PKG} & \text{(6a)} \\ PrvK^i_A = f_i(\widetilde{s}_i).\widetilde{PrvK}^i_A & \text{(6b)} \end{cases}$$

4) Key refreshment periods: In EIBC, there are different values that need to be updated or refreshed from time to time, including "$f_i(.)$", "s_i", "$\widetilde{s}_i$", and the PRNG parameters "a & b". EIBC employs three timers for Short, Medium and Long Term Refreshments (STR, MTR and LTR) for the refreshment of these parameters.

a) STR process: PKG generates a new function "$f_{i+1}(.)$" and makes it publicly accessible, along with a VT, which is the start time of moving to a new live ("i" $\rightarrow$ "$i+1$"). At the time of VT, each party refreshes "$\widetilde{s}_i$" following (4b), updates $F_i(.)$ via (3a) in order to have refreshed public keys of others. Also, the party refreshes the public key of PKG as per (7a) and (7b), as well as its own private key based on (7c) and (7d), utilizing the updated values of "$\widetilde{s}_{i+1}$" and "$F_{i+1}(.)$":

$$\begin{cases} \widetilde{PubK}^{i+1}_{PKG} = f_{i+1}(\widetilde{PubK}^i_{PKG}) & \text{(7a)} \\ PubK^{i+1}_{PKG} = f_{i+1}(\widetilde{s}_{i+1}).\widetilde{PubK}^{i+1}_{PKG} & \text{(7b)} \\ \widetilde{PrvK}^{i+1}_A = f_{i+1}(\widetilde{PrK}^i_A) & \text{(7c)} \\ PrvK^{i+1}_A = f_{i+1}(\widetilde{s}_{i+1}).\widetilde{PrvK}^{i+1}_A & \text{(7d)} \end{cases}$$

b) MTR process: PKG renews the PRNG parameters "a & b" along with the required VT, and shares them with all the parties to be used starting at VT.

c) LTR process: PKG reselects the system non-shared secret values, along with the system shared secret values, and updates one-way function "$F_i(.)$", in order to refresh all the keys, i.e., public and private keys of all parties. PKG also updates the private key of each party, and informs the party along with a VT via the secure channel.

Note that the LTR process is similar to the IBC key refreshment process. As it has been analyzed in the [16], EIBC simultaneously improves key management process overhead cost and system security level.

5) Multicast group key support: To support secure multicasting, EIBC incorporates two mechanisms to manage the *multicast group source/receiver key pair*. Each multicast group is identified by a Multicast Group ID (MID), which is used similar to ID of an entity, to obtain Source Multicast Key (SMK) of the group via (1). At the same time each group has a Receiver Multicast Key (RMK) managed by SAS and obtained via (5b) and (6b). Each Multicast Group Source (MGS) entity receives the group's SMK and RMK, and grants membership to a Multicast Group Receiver (MGR) entity by sending RMK to the new MGR. So, MGS encrypts the messages by SMK, and a MGR uses RMK to decrypt the messages. In order to authenticate the source of a multicast packet and because a SMK can be compromised, MGS signs the messages using its own entity (original) private key ($PrvK^i_{ID}$).

Furthermore, EIBC generates "$\widetilde{m}_i$", similar to "$\widetilde{s}_i$", using a Muticast Group Pseudo Random Number Generator with its own setup values "c & d" and initial value "$\widetilde{m}_0$". Receivers use "$\widetilde{m}_i$" to refresh RMK.

F. SG Security Schemes in the Literature

The security scheme in [17] is aimed at data transfer via the PLC technology for SG communications. In this mechanism, the manufacturer of any device, e.g., meter, modem or aggregator, should obtain a certificate for the device from the

SG security server following the PKI approach, and embeds it in the device. Then, each node/device utilizes its own public/private key pair to construct a shared symmetric key with the next node. In this system, the SG security server is involved in authentications of all the nodes in each stage of the mechanism, which can be a heavy workload in the SG environment. Another concern about this proposal is the assumption that all the manufacturers of the devices are fully trusted parties. Also, the shared symmetric key is chosen by one node and transferred to the peer encrypted with the public key of the peer. Therefore, the proposed mechanism is vulnerable to attacks, e.g., by malicious nodes that have obtained a certificate illegally, or devices from a rogue manufacturer.

The use of symmetric keys for SG security is proposed in [18], [19], the former based on the D-H algorithm, and the latter based on the elliptic curve approach of the D-H algorithm; both adds a key verification step to the pairwise key construction. Use of symmetric keys is vulnerable to MITM attacks, despite the verification phase. Furthermore, using symmetric keys for communications over the entire SG system is not scalable due to the large number of devices and nodes. Consequently, PKI is recommended in [1] to secure SG communications.

In order to decrease the cost of key distribution, the proposal in [20] requires all packets to be transferred through a server. Each source encrypts its packet with the public key of the server and sends it to the server. Then, the server uses its private key to decrypt the packet, and uses the public key of the destination to re-encrypt the packet and sends it to the destination, e.g., a service provider. In an SG, this mechanism causes a very high demand on the server to handle the decryption and re-encryption of packets and on the network to route each packet via the server. Thus, the cost of key distribution is lowered at the cost of a highly loaded server and increased data packet communication load. Furthermore, this method does not preserve confidentiality of the packets since all packets are decrypted by the server, which is not the end receiver. The mechanism presented in [21] is also vulnerable to the MITM attack, although the authors mentioned that it is safe against this attack. For instance an authenticated malicious node can perform the MITM attack. This scheme requires two hash functions, and needs a third party in the key construction process, in initializing the key construction as well as the key verification.

Using IBC to secure vehicle-to-grid communications over SG is proposed in [22]. The authors mainly focused on the key management, and they provide a one-way authentication for authenticating the vehicles to the grid. Using biometrics is proposed in [23] for the authentication of users in SG. The author suggested that their proposal addresses the user privacy issue in SG communications [23], although the need to collect users' fingerprint information can raise overall user privacy concerns.

Authors of [24] studied the approaches of having a Unified Key Management Function (UKMF) and Dedicated Key Management Functions (DKMF) or a hybrid of the two for different applications in SG. They showed that using UKMF is more efficient, and furthermore, they suggested an Extensible Authentication Protocol based mechanism to be used in SG.

Our work is built on top of PKI, the preferred method to secure SG communications, and provides secure and efficient mechanisms for initial authentication and key generations and updates.

III. Smart Grid Mutual Authentication

A. System Setup

We concentrate on data communications over the AMI outside of the HAN domain, which includes an SAS that is charged with supporting the required authentication and key management mechanisms. We also cover the key management for unicast, multicast and broadcast communications that may be needed to support any application over SG. Our assumptions are as follows:

- Nodes are connected in a WMN, with requires unicast technology support for the multi-hop communications.
- Each node has a unique ID (most likely an IPv6 address), which may be manually assigned to the node by a technician at set up time.
- Each SM has a unique serial number "SN" embedded by the manufacturer, and an initial secret password "pw" loaded by the installing technician, for authentication purposes. On the other hand, SAS holds the appropriate verifier "ver" and "$salt$" for the SM, in support of the SRP algorithm.
- Each node is initially loaded with the "$H(.)$" function, and values "g & p" to be used in the SRP algorithm, which can be loaded by the technician at set up time, or at manufacturing.
- Nodes are all synchronized in time, and the newly installed SM would be able to synchronize itself with others using a suitable synchronization system, which design is outside of the scope of this paper.
- SAS is responsible for the authentication as well as the key management mechanisms.

The system topology is depicted by Fig. 1, which is based on [7]. Referring to our discussion in Section I, when a new SM is installed, it mutually authenticates itself with the SAS, and receives its private key from the SAS as well.

Definition: Let us define system state "(i, j)":

Dimension "i": Represents the index, also referred as live, of system functions "$f_i(.)$" and "$F_i(.)$" as well as random values "s_i" and "$\widetilde{s}_i$".

Dimension "j": Represents index of PRNG set up values "a_j & b_j" used in (4b), which are shown only by "a & b" for simplicity.

B. Mutual Authentication Scheme

Depicted by Fig. 2, our SRP-6a based mutual authentication scheme consists of following three steps:

1) Step I: New SM, "sm", selects a random value "R_{sm}" and calculates "$G_{sm} = g^{R_{sm}} \bmod p$". Then, SM sends "$G_{sm}$" along with its own "$SN_{sm}$" and "$ID_{sm}$" to the SAS.

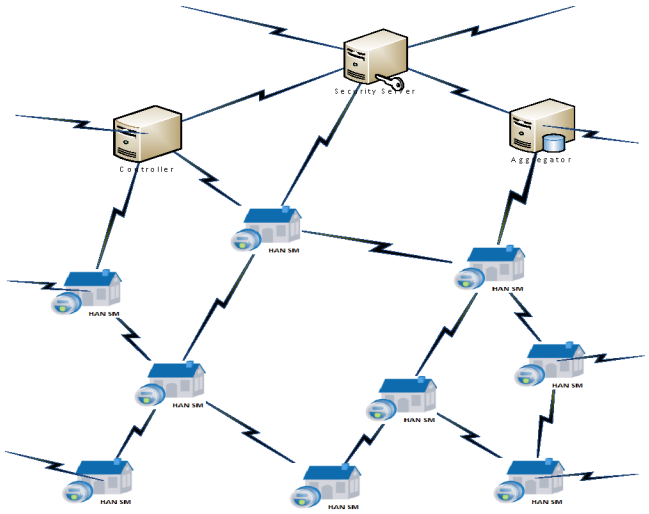

Fig. 1: Smart Grid Topology for AMI

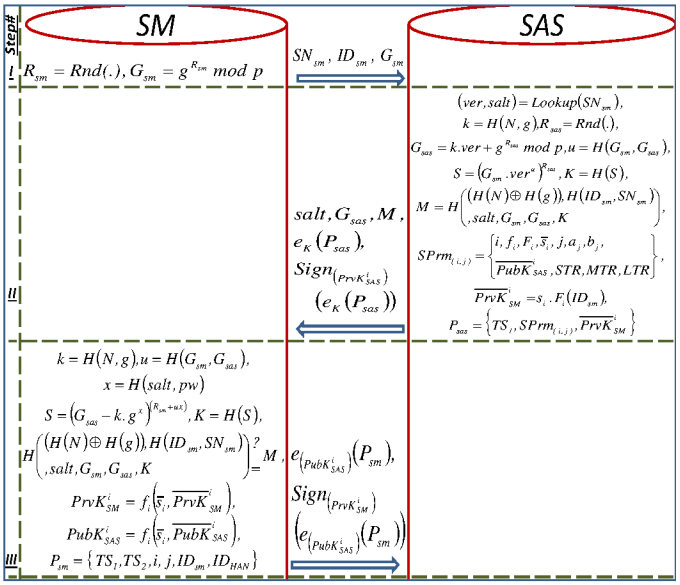

Fig. 2: Smart Grid Mutual Authentication (SGMA)

2) Step II: SAS performs the following steps upon receiving the packet from SM in Step I:

- SAS lookups values "ver & $salt$" associated with "SN_{sm}".
- SAS computes "$k = H(N, g)$", and picks random values "R_{sas}".
- SAS calculates "$G_{sas} = k.ver + g^{R_{sas}} \bmod p$" and "$u = H(G_{sm}, G_{sas})$".
- SAS computes "$S = (A * ver^{u})^{R_{sas}}$" followed by "$K = H(S)$" and verifier value "$M$" as "$M = H((H(N) \oplus H(g)), H(ID_{sm}, SN_{sm}), salt, G_{sm}, G_{sas}, K)$".
- Furthermore, SAS computes the private key SV of SM, "$\widetilde{PrvK}^{i}_{SM}$", and forms the system parameter set for SM.
- Finally, SAS sends values "$salt$, G_{sas} & M" along with the encrypted and signed parameters set of the system to SM.

3) Step III: SM performs the following steps when it receives the packet sent by SAS in Step II:

- SM calculates "$k = H(N, g)$" and "$u = H(G_{sm}, G_{sas})$".
- SM computes "$x = H(salt, pw)$", and then "$S = (G_{sas} - k.g^{x} \bmod p)^{(R_{sm}+u.x)}$".
- Then, SM calculates "K" as "$K = H(S)$", and then verifies "K" based on the received "M" by comparing it with "$H((H(N) \oplus H(g), H(ID_{sm}, SN_{sm}), salt, G_{sm}, G_{sas}, K)$".
- If the verification condition holds, SM is assured that the symmetric key "K" shared with the server is valid. So, SM is able to decrypt received values, as well as is capable of checking the signature.
- Finally, SM obtains its own private key and sends an encrypted and signed acknowledgement to the SAS.

Note that by this point, SM and SAS are mutually authenticated to each other, and SM has received system parameters as well as its own private key.

IV. Smart Grid Key Management Protocol

Our proposed SGKM is based on EIBC. Thus far, nodes have the appropriate private-public keys to be used for unicast and node-to-node secure communications based on PKI. In this section, we introduce our key refreshment mechanism as well as solutions for the required multicast and broadcast keys.

A. Key Refreshment

Referring to the EIBC mechanism presented in Section II and [16], the system needs to set the values of three timers STR, MTR and LTR. Values of these timers are transferred as parts of the system parameter in Step II of the authentication process described above.

1) Short Term Refreshment Process: As depicted by Fig. 3, the system regularly runs this process to move the system state from "(i, j)" to "$(i + 1, j)$" based on the value of STR.

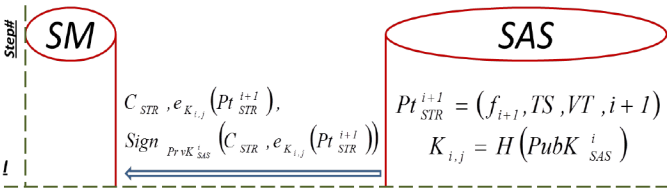

Fig. 3: Broadcasting an encrypted and signed packet in STR

a) SAS duties: SAS first generates a new function "$f_{i+1}(.)$" according to the new system state "$i+1$". Then, SAS prepares a packet "Pt^{i+1}_{STR}" containing the "$f_{i+1}(.)$" function, Time Stamp (TS) of producing the "$f_{i+1}(.)$", Valid Time (VT) of the current system state dimension "i" and its new value "$i+1$". Then, SAS applies the original "$H(.)$" function to its own live public key to obtain a symmetric key "$K_{i,j}$" via (8):

$$K_{i,j} = H(PubK^{i}_{SAS}) \tag{8}$$

Note: We describe more about "$K_{i,j}$" at the end of this section, since we use this technique to handle the broadcasting key in the broadcast key management part (Section IV-C).

Finally, SAS encrypts the "Pt_{STR}^{i+1}" packet utilizing the "$K_{i,j}$" key, and broadcasts it along with the STR control command "C_{STR}". SAS also signs these values with its own live private key in order to provide source authentication.

b) SMs duties: As soon as any of the SMs receives the broadcast information identified by "C_{STR}", uses the live public key of SAS to verify the signature. If the signature is valid, SM calculates the symmetric key "$K_{i,j}$" following (8) and decrypts the received packet "Pt_{STR}^{i+1}". Then, SM verifies the received system state "$i+1$" as part of the packet to make sure it is one after the current state. Furthermore, to prevent the replay attack, SM checks that "TS" is more than the "VT" received in the previous STR refreshment command. Finally, prior to "VT", SM utilizes "$f_{i+1}(.)$" to refresh the appropriate keys using (7a)-(7d) by following the steps in the short period refreshment process of EIBC (Section II.E), and starts using them by "VT".

2) Medium Term Refreshment Process: The system runs the medium term refreshment process presented in Fig. 4 to change the system state from "(i,j)" to "$(i,j+1)$" based on the value of MTR.

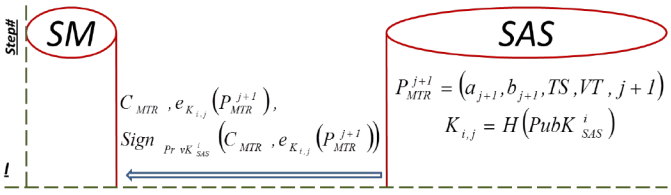

Fig. 4: Broadcasting an encrypted and signed packet in MTR

a) SAS duties: Referring to EIBC, SAS first generates a new pair of PRNG parameters "a_{j+1} & b_{j+1}" for the new system state "$(i,j+1)$". Then, SAS prepares a packet "Pt_{MTR}^{j+1}" containing the "a_{j+1} & b_{j+1}" values, Time Stamp "TS" of the packet, Valid Time "VT" of the new setup values plus the new system state "$j+1$". Then, SAS applies the original "$H(.)$" function to its own live public key to obtain a symmetric key "$K_{i,j}$" (8). Finally, SAS broadcasts the encrypted packet "Pt_{STR}^{j+1}" utilizing the "$K_{i,j}$" key, along with the MTR control command "C_{MTR}". SAS also signs this packet with its own live private key in order to maintain source authentication.

b) SMs duties: When a SM receives the broadcast information identified by "C_{MTR}", it obtains the live public key of SAS to verify the signature. If the signature is valid, SM calculates the symmetric key "$K_{i,j}$" following (8), and decrypts the received packet "Pt_{MTR}^{j+1}". Then, SM makes sure the system state "$j+1$" is one after the current one (j), and checks "TS" to prevent a replay attack. Finally, starting by "VT", SM updates its "$\widetilde{s}_i$" parameters according to (4b).

3) Long Term Refreshment Process: Based on the value of LTR, the system runs the long term refreshment process as shown in Fig. 5 to go from the "(i,j)" state to the "$(0,0)$" state. SAS needs to regenerates the system parameters as well as the private key of each node and inform them one by one.

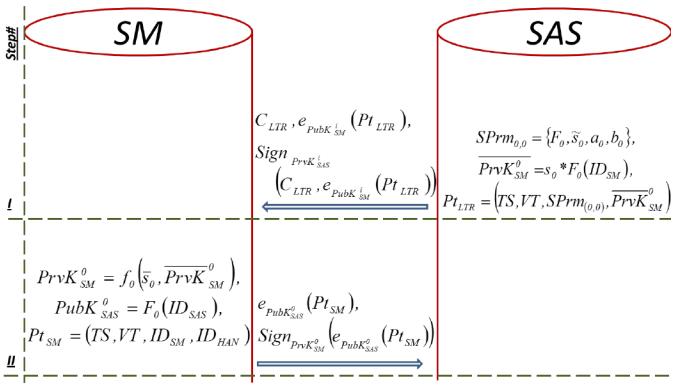

Fig. 5: Unicasting an encrypted and signed packet in LTR

B. Multicast Key Mechanism

SMK is used by a group source to encrypt the multicast packets. Furthermore, RMK is used by all group receivers to decrypt the messages that are encrypted by SMK. Our assumptions are:

- The multicast group is source based, and joining is initiated by the receiver.
- Each group is identified by a unique MID.
- SAS is in-charge of the multicast group key management.

Beside the SMK and RMK keys, each group also has a public/private key pair that is used in the multicast join algorithm. Similarly and by utilizing MID, system manages this key pair based on (5a), (5b), (6a) and (6b).

For the SMK and RMK keys, we define multicast group state "$(k\ \&\ l)$" in a manner similar to the "$(i\ \&\ j)$" state. Furthermore, "$g_k(.)\ \&\ G_k(.)$" similar to the "$f_i(.)\ \&\ F_i(.)$" functions, and finally "$m_k\ \&\ \widetilde{m}_k$" along with "$c_l\ \&\ d_l$" are similar to the "$s_i\ \&\ \widetilde{s}_i$" and "$a_j\ \&\ b_j$" items in our original system design for the unicast communication.

$$\begin{cases} G_{k+1}(.) = g_{k+1}(G_k(.)) & \text{(9a)} \\ m_{k+1} = g_{k+1}(m_k) & \text{(9b)} \\ \widetilde{m}_{k+1} = c_l * \widetilde{m}_k + b_l & \text{(9c)} \\ SMK_k = G_k(MID) & \text{(9d)} \\ RMK_k = (\widetilde{m}_k, (m_k * G_k(MID)) & \text{(9e)} \end{cases}$$

1) Establishing a multicast group: (i) An MGS that wants to form a multicast group sends a request to SAS. (ii) SAS provides MGS with the group initial parameters set consisting of "$\{MID,\ \widetilde{m}_0,\ RMK_0\ \&\ G_0\}$" along with the private key SV of the group per (5b) and (6b) based on MID. (iii) MGS picks "$c_0,\ d_0\ \&\ g_0(.)$" and completes the group parameter set for the multicast group "$(0,0)$" state. Once the multicast group is established by MGS, MID is made publicly accessible by the parties that want to join. Note that MGS is in-charge of generating the "$g_k(.)$" function in each state.

2) Joining multicast group: The join algorithm, as presented in Fig. 6, consists of the following steps:

a) Join request (Step I): The new MGR applies the current system state function "$F_i(.)$" to MID to obtain the public key via (3b). Then, MGR broadcasts its join request encrypted by the public key of the group, including its own

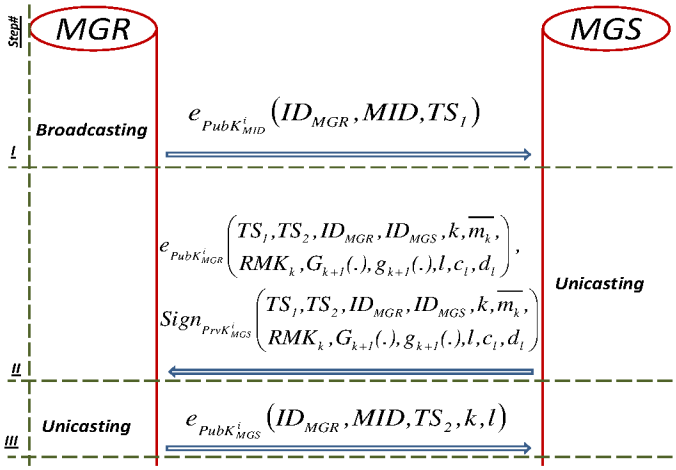

Fig. 6: Joining a Multicast Group

ID.

b) Grant membership (Step II): Since only MGS has private key of the group, only MGS can decrypt the packet and replies with the membership grant, which consists of the group parameter set " $\widetilde{m}_k$, RMK_k, G_{k+1}, g_{k+1}, c_l, d_l", and at the same time, sends the "$g_{k+1}(.)$" to the entire (current) group members to support forward secrecy. For the source authentication purposes, MGS signs this packet with its own private key.

c) Acknowledgement of membership (Step III): Firstly, MGR verifies the signature, and then accepts the information and joins the group if it is a valid one. Then, MGR sends an acknowledgement to the source notifying the source that MGR has successfully joined the group.

3) Key refreshment process: The reasons for the key refreshments in case of multicasting situation are different than the aforementioned unicast situation and consist of two cases: (i) a member joining or leaving causes the system to refresh the keys in order to maintain forward and backward secrecy, and (ii) providing overall multicast key secrecy. However, we define and use a similar algorithm in both cases. To be more precise, each multicast group has timers similar to the unicast case, which are set by the system administrator as per group establishment purposes and application requirements. Referring to the unicast timer refreshment processes, we only describe relevant points of the multicast timers refreshment.

- For multicasting forward and backward secrecy concerning the receivers join/leave situation, we follow the short term refreshment process.
- MGS is in-charge of generating and distributing the new "$g_{k+1}(.)$" in a manner similar to the short term key refreshment, and proceeding from the "(k, l)" to "$(k+1, l)$" state.
- MGS is in-charge of distributing the " $\widetilde{m}_k$" set up values "c_{l+1} & d_{l+1}" addressing in a manner similar to the medium term key refreshment, moving from the "(k, l)" state to the "$(k, l+1)$" state.
- SAS is in charge of the long term key refreshment process, moving from the "(k, l)" state to the "$(0, 0)$" state. SAS provides appropriate parameters including keys to the MGS, and then MGS unicasts them to the members utilizing their unicast public/private pair key.

C. Broadcast key mechanism

Referring to our unicast medium term key refreshment process, we apply the system original "$H(.)$" function to the public key of SAS to obtain a symmetric key. Since the public key of SAS is dynamic and changes periodically according to the "$f_i(.)$" function and state of the system, only the parties authenticated by the SAS, who receive their key management service from the SAS, have the live public key of SAS.

V. Security and Performance Analysis

In this section, we evaluate the security of our proposed SGMA and SGKM mechanisms using the Automated Validation of Internet Security Protocols and Application (AVISPA) security analyzer [25]. Furthermore, we review the adversary models including adversary interests and capabilities to attack the system, following the Dolev and Yao approach [26]. Then, we review the system security against attacks. At the end of this section, we verify the overhead cost reduction of our proposal.

A. Formal Validation Using Software Tool

AVISPA is a software tool for the automatic verification and analysis of Internet security protocols that is currently considered by the research community as one of the most trusted evaluation tools to analyze the ability of a scheme or protocol to withstand different attacks. AVISPA integrates automatic security analysis and verification back-end servers like On-the-Fly Model-Checker (OFMC) and Constraint-Logic-based Attack Searcher (Cl-AtSe). First of all, the mechanisms under examination by AVISPA must be coded in the High Level Protocol Specifications Language (HLPSL) to be evaluated by the back-end servers. HLPSL is an expressive, role-based formal language used to describe the details of the protocol in question. Our HLPSL codes (see Appendix A) includes different sections used to model the roles of SM and SAS entities, as well as the role of the environment and the security goals that have to be achieved. We started with the original model already existing in the AVISPA library, and then developed our HLPSL codes based on the proposed mechanism. The results of the evaluation presented in Figs. 7a and 7b show that our proposed mechanism is secure and safe from attacks. To be more precise, the symmetric key that we prepare at the end of our authentication to be used by SAS to send the system parameters to SM is a valid and safe key. The system parameters consists of the PRNG and its setup values "a & b", as well as the private key SV of SM. Furthermore, SM is capable of finding the public key of SAS, and sends acknowledgement back to SAS, which is secure as well.

B. Adversary Models

Since we may have different situations for an adversary, we describe two scenarios addressing the adversary's different objectives and initial knowledge. In the first scenario, the

```
% OFMC
% Version of 2006/02/13
SUMMARY
 SAFE
DETAILS
 BOUNDED_NUMBER_OF_SESSIONS
PROTOCOL

/ubc/ece/home/vl/grads/hasennic/Deskto
p/avispa-1.1/testsuite/results/SGAS6.if

GOAL
 as_specified
BACKEND
 OFMC
COMMENTS
STATISTICS
 parseTime: 0.00s
 searchTime: 23.32s
 visitedNodes: 0 nodes
 depth: 1000000 plies
```

(a) OFMC

```
SUMMARY
 SAFE

DETAILS
 BOUNDED_NUMBER_OF_SESSIONS
 TYPED_MODEL

PROTOCOL

/ubc/ece/home/vl/grads/hasennic/Deskto
p/avispa-1.1/testsuite/results/SGAS6.if

GOAL
 As Specified

BACKEND
 CL-AtSe

STATISTICS

 Analysed  : 1956 states
 Reachable : 1956 states
 Translation: 1.16 seconds
 Computation: 7285.36 seconds
```

(b) ATSE

Fig. 7: AVISPA Results

adversary does not have control on any party; however in the second scenario, the adversary has full control on one of the SMs (i.e., there exists a malicious SM).

1) First scenario:

Objectives: The adversary wants to gain access to the system resources, like SAS or any of the SMs, and wants to be able to decrypt and encrypt the messages. Other possible objectives of the adversary are performing a DoS attack against SAS, or compromising the SAS.

Initial capabilities: The adversary knows the IDs of all of the parties, as well as initial "$H(.)$" function and "g & p" values in SGMA. Also, the adversary knows in detail the design of SGMA, and can make or have a valid serial number.

Capabilities during the attack: During the attack, the adversary is able to receive the entire SMs and SAS communications, encrypted and unencrypted packets. If the adversary is able to steal the private key of any victim SM, it will be able to decrypt the encrypted packets sent to the SM, and impersonate the SM in sending packets with forged signatures of the SM. Therefore, the adversary will be able to send incorrect pricing information to the SM, take control of the smart appliances attached to the SM, modify billing information, etc. The adversary will also be able to mount a DoS attack by sending multiple authentication requests to the SAS.

Discussion: An adversary forging an SM's signature to mount a DoS attack on the SAS by sending multiple authentication requests (Step III in Fig. 2) to SAS. As soon as SAS receives the requests, it checks its database for the $(ver, salt)$ pair associated with each request. Incorrect or missing values of $(ver, salt)$ cause the SAS to drop the request and ignore subsequent requests from the SM once a number of requests have been dropped.

If the adversary initiates the request with valid ID & SN that have been stolen from a SM, SAS may find the $(ver, salt)$ values and process the request by sending the response back to SM, and goes to the next step of SGMA. Since the adversary does not have the appropriate password, s/he is not able to obtain the key and decrypt the packets. However, SAS will leave the session open. Note that SAS sends a time stamp (TS_1) among other information in Step II of SGMA. SAS can close the session if the appropriate acknowledge is not being received within a certain time period (e.g.,session expiry time). Furthermore, to prevent DoS attack in Step I, SAS can limit the number of the authentication requests it process within a given time frame. So, sending a large number of requests does not harm the SAS.

The adversary may try to perform a replay attack by forwarding a previous acknowledgement from the SM to the server. This solution does not help the adversary since the acknowledgement should be encrypted and signed utilizing the valid and appropriate system public and private keys. Also, the acknowledgement consists of the time stamp and ID of SM, which is not the valid one for the authentication session of the adversary.

The next option for the adversary is performing a brute-force attack and obtaining access to the encrypted packets. Normally, brute-force attack is time consuming, based on size of the key that packets are encrypted with. If the attacking time takes more than the session expiry time, the attack will not cause any issue. In the worst situation, the adversary can move to the on-line dictionary attack to speed up, or performs an off-line dictionary attack and find the session key, and finally obtain an expired private key for a not valid SM. However, the adversary would gain access to the system parameters, and if SAS has not run the key refreshment process yet, the adversary can keep going making the system parameters valid and fresh. In summary, by using any of the aforementioned attacks, the adversary is not able to compromise the server, since the adversary can only communicate with others, and only if the other parties send information to the malicious node, the adversary would be able to decrypt the packets. Furthermore, since SGMA uses a hash function, our authentication provides forward secrecy, and the adversary is not able to find out the original password.

To perform a MITM attack as another option for the adversary, the adversary may receive the first packet generated by a victim SM and change the value of "G_{sm}". However, the adversary is not able to decrypt the second packet coming from the server, because the adversary needs the password of the victim to obtain the symmetric key "K".

The other option is compromising the server by social engineering. Compromising the server does not give the adversary access to the passwords of SMs since SAS only keeps the verifier (and salt). However, if SAS records and keeps the private keys of the nodes (to be more precise, the private key SVs), the adversary will have private keys of the entire SMs. This attack is costly and unfortunately works in almost most of the situations. If SAS only generates the private keys and does not log them, to some extent this will prevent the attack from harming the previous generated keys. However, the adversary will be able to attack the new SMs. The best solution to prevent this attack is improving the server security well enough, for instance by changing the server password more often.

2) Second scenario:

Objectives: Similar to the previous scenario, the adversary wants to gain access to the system resources, like SAS or any of the SMs. The adversary would like to decrypt and encrypt the messages. Other objectives of the adversary

may include performing a DoS attack against the SAS, or compromising the server or any of SMs.

Initial capabilities: Similar to the previous scenario, the adversary knows the IDs of all the parties, the system parameter "$H(.)$" function and "g & p" values regarding SGMA, as well as the detail design of the SGMA protocol. Furthermore, the adversary has a valid password to start SGMA, and by proceeding with the SGMA protocol, the adversary has a valid private key and all of the system parameters like "$F_i(.)$".

Capabilities during the attack: During the attack, the adversary is able to receive the entire SMs and SAS communications, encrypting and decrypting packets. Since the adversary has a valid private key of a SM, the adversary is able to decrypt and encrypt packets to and from the SM. For instance, the adversary can change the HAN commands, price list, or meter/billing information.

Discussion: In this situation, the adversary has full control of a malicious SM, in other words the adversary is a valid SM. Therefore, the adversary can rerun SGMA to be authenticated, and some-how perform a DoS attack. However, the adversary has only one password, and can resend the same ID and SN of victim SM to initiate a session, and in the worst case causes one open session.

The previous discussion about analyzing the adversary behaviour is valid in this scenario as well. The only differences are having valid system parameters like PRNG. Generally speaking, being in this scenario does not help an adversary to improve the chance of a successful attack. For instance, the adversary can run a brute-force attack by having a valid private key and communicate with others to obtain their private keys by brute-force. In this case, off-line dictionary can work because the adversary has the system parameters, like $f_i(.)$ and PRNG, and can find the live private key. However, just by performing one LTR process by SAS, the system can prevent the adversary from continuing the successful attack.

C. Other security characteristics

Recall in our discussion in Section III, a mutual authentication is performed since SAS needs to know the password verifier, and on the other side, SM needs to know the password. Both ends require one of these values to calculate the session key. In terms of attacks resilience, we refer to the discussion in the previous subsection, about the most well-known attacks such as brute-force, DoS, replay, on-line and off-line dictionary and MITM attack, which cover parts of the attacks resilient summary as presented by TABLE I. We also refer to the above section about the social engineering attack that may work partially on the server; however, compromising one SM does not help the adversary to attack the whole system. In TABLE I we compare our mechanism with five of the schemes described in Section II-F, which include mechanisms for authentication and/or key construction. Since [20] proposed using PKI and aimed at reducing the number of certificates (or issued private keys), [23] suggested using users' biometric parameter (fingerprint) for authentication and [24] does not have detail design of the authentication and/or key construction, therefore we did not include them in this table.

TABLE I: Summary of resilience to the attacks

Attack	[17]	[18]	[19]	[21]	[22]	Ours
Social engineering	✖	✖	✖	✖	✖	✔ & ✖
Brute-force	✖	✔	✔	✔	✖	✔
Replay	✔	✔	✔	✔	✔	✔
DoS	✖	✖	✖	✖	✖	✔
MITM	✔	✖	✖	✖	✔	✔
On-line dictionary	✔	✔	✖	✖	✖	✔
Off-line dictionary	✖	✔	✔	✔	✖	✔
Unknown key share	✔	✔	✔	✔	✔	✔
Compromised impression	✔	✔	✔	✔ & ✖	✔	✔
Denning-Sacco	✖	✔	✔	✔	✖	✔
Key privacy & insider	✔	✖	✔	✔	✖	✔
Ephemeral key compromise impersonation	✖	✔	✖	✖	✖	✔

Unknown key-share attack: The second packet of the authentication scheme presented in Fig. 2 is encrypted by symmetric key "K". Encryption of this packet by SAS shows SAS has the key, and decryption the packet by SM and acknowledging the SAS proves that SM has the key as well.

Compromised impression resilience: Referring to our analysis at the beginning of this section, finding the private key of any SM does not help an intruder to obtain the private key of any other node or SAS.

Denning-Sacco attack resilience: If an intruder somehow finds a symmetric key used in the authentication scheme, since the key is the product of a hash function, which is a one-way function, the intruder would not be able to find the original password or the verifier. Furthermore, finding a private key does not help the adversary to find a symmetric key of the authentication session.

Privacy and insider attack resilience: Since our scheme is based on PKI, each private key is known only by the owner (and maybe the server). Other nodes know only the public keys of all the nodes, which in fact is required by them to communicate with each other. Even if other nodes in between relay the packets, since the packets are encrypted and signed, they cannot have access to the private key of the source or destination nodes.

Ephemeral key compromise impersonation: Suppose an adversary performs an off-line dictionary attack or brute-force or even social engineering attack and obtains the password of a SM. Because the password is only one of the values required for the session key construction, the adversary still is not able to find the session key, or the private key.

D. Performance analysis

Consider the topology shown by Fig. 1. Suppose SAS wants to refresh the keys of all the SMs. Compared to the original PKI, the IBC approach yields a better performance in the overhead cost, as we have discussed in Section II. Therefore, we only compare our proposal with an SG that uses the IBC approach to secure data exchanges.

We assume that on average, each SM is connected to "$H_{sm} > 1$" neighbours (dimension of SM), and the average hop counts between SAS and any SM is equal to "L_{sas}"

TABLE II: F_D and F_{BM} based on H_{sm} and L_{sas}

H_{sm}	L_{sas}	$F_D(L_{sas}, H_{sm})$	$F_{BM}(L_{sas}, H_{sm})$
3	5	54.6	10.13
3	10	14024	21.37
3	20	8.45E+08	43.875
3	40	3.00E+08	88.87
4	5	159.2	12.45
4	10	1.69E+05	25.78
4	20	1.50E+11	52.44
4	40	2.00E+23	105.78
5	5	371	14.84
5	10	1.19E+06	30.47
5	20	1.18E+13	61.72
5	40	1.13E+27	124.22

(Length of SAS network). Moreover, we define "bw_l" as the bandwidth (BW) of each link required per key distribution while the total network BW to refresh all the keys is "BW_{net}". To compare the delay, we define "d_h" as the delay/time required by each hop (or link) to deliver/process a packet, and "D_{net}" to be the total system delay/time to refresh all the keys. For simplicity, we assume SAS generates same packet sizes in STR, MTR and LTR. Since the LTR process is similar to the key refreshment process in the original IBC, we use it as our bench mark in this study. In order to show the improvement of SGKM employing EIBC, we assume the following relations exists between values of the timers:

$$\begin{cases} MTR = ms * STR\ ,\ ms > 1 & \text{(10a)} \\ LTR = lm * MTR\ ,\ lm > 1 & \text{(10b)} \\ LTR = ls * STR\ ,\ ls > 1 & \text{(10c)} \\ ls = lm * ms & \text{(10d)} \end{cases}$$

The total network required BW and applicable delay by each key refreshment process are as follow:

$$\begin{cases} D_{net}(LTR) = d_h.(H_{sm} + \sum_{v=2}^{L_{sas}} v.H_{sm}^{v-1}) & \text{(11a)} \\ BW_{net}(LTR) = bw_l.\sum_{v=1}^{L_{sas}} (v.H_{sm} + v - 1).H_{sm}^{v} & \text{(11b)} \\ D_{net}(STR) = d_h.(1 + 2.d_h) & \text{(11c)} \\ BW_{net}(STR) = 2.bw_l.H_{sm}.\dfrac{H_{sm}^{L_{sas}} - 1}{H_{sm} - 1} & \text{(11d)} \end{cases}$$

In (11a)-(11d), we assume that in each STR (and MTR) process, 50% of the nodes broadcast concurrently, and in the LTR process, SAS processes "H_{sm}" SMs at the same time.

By a reasonable estimation, we have:

$$F_D(L_{sas}, H_{sm}) = \frac{D_{net}(LTR)}{D_{net}(STR)} \approx \frac{\sum_{v=2}^{L_{sas}} v.H_{sm}^{v-1}}{2.L_{sas}} \quad (12)$$

$$F_{BW}(L_{sas}, H_{sm}) = \frac{BW_{net}(LTR)}{BW_{net}(STR)} \approx \frac{\sum_{v=1}^{L_{sas}} (v.H_{sm}^{v+1})}{2.H_{sm}^{L_{sas}}} \quad (13)$$

F_D in (12) represents the relationship between the delays of the key refreshment processes, while F_{BW} in (12) demonstrates their required network bandwidth. Although these two quantities depend on the network topology, they are always greater than one.

TABLE II illustrates a few examples of F_D and F_{BM} based on H_{sm} and L_{sas}. As the table shows, the values increase with H_{sm} and L_{sas}. Note that STR (and MTR) processes are run more frequently in our mechanism compared to LTR, whereas in the original IBC (and PKI), the key renewal (similar to LTR) process are run at almost the same rate as STR in our mechanism. For example if "$H_{sm} = 4$" and "$L_{sas} = 40$", the system requires less than 1% bandwidth to distribute the private keys following SGKM, compared with IBC/PKI. The time required for key distribution is reduced to "$5E - 24$" of the LTR delay. The data in TABLE II along with the above examples clearly shows that the proposed mechanism is much more efficient and greatly reduces the key refreshment delays compared to the original IBC or PKI mechanisms.

Overall analysis: In our design, we take advantage of the SRP, PKI and IBC approaches. Each one brings some benefits to our proposed mechanisms. Besides, our enhancement of each mechanism has improved the overall benefits to the system.

Firstly, we have reduced the required number of packets in our authentication scheme. To be more precise, we reduced the number of packets needed for mutual authentication from four to three. Furthermore, in the three packets, the entire set of system parameters are delivered as well as the private key of the new SM. Our analysis shows that SGMA is fast and robust and secure.

Secondly, implementing the private key cryptography system in a distributed environment causes providing a symmetric key between every two nodes that need to communicate to each other. Moreover, increasing the number of nodes that want to communicate with a single node requires that the node keeps and manages a large number of keys (one per peer node), which is the case in the SG context. However, PKI requires only one key pair per entity in spite of a larger key size. In fact, while a node has its own private/public key pair, it is sufficient for the node and others to exchange secure communications.

Also, since IBC reduces the public key distribution overhead in PKI, we take advantage of this technique in our design. Furthermore, we have designed EIBC, an improved version of the IBC, and utilized it in SGKM. The most important benefit of using EIBC in this design is reduction of the private key distribution and refreshment overhead. In EIBC, most of the key refreshments are accomplished by the PKG broadcasting a packet to all nodes instead of unicasting one packet to each node, which yields substantial reduction in the system overhead cost. Indeed, broadcasting is used in two out of three key refreshment processes (STR and MTR), while unicasting is used in the LTR refreshment process, which is run much less frequently than the STR and MTR processes.

Also, our mechanism can be easily implemented in any system and platform. Since nodes are only required to have their own private keys, and to know only the public keys of the nodes that they want to communicate with, the mechanism is scalable. Based on the nodes population and application that are going to be run on the system, the system administrator can tune the security and overhead by changing the values of the timers as well as sizes of the keys. Furthermore and referring to our EIBC design [16], the system administrator can even

turn any of the features off, like PRNG. All an administrator requires to do is for instance to set "$a = 0$ & $b = 1$". On the other hand, if the administrator wants to turn the periodic distribution function $f_i(.)$ off, s/he can set "$f_i(x) = x$". This flexibility makes our mechanisms applicable to a variety of systems and platforms.

VI. Conclusion

In this paper, we have presented novel mutual authentication and key management mechanisms tailored for the SG communications. The proposed mechanism addresses the required security aspects by the SG system, and at the same time, manages the process in an efficient fashion. The savings in resource consumption as the result of our mechanism can be used to handle more data delivery, and/or to increase the security of the system by refreshing the keys more often, which brings to SG the opportunity to utilize keys of smaller sizes, further reducing resource consumption in the system. In order to enjoy the security benefits of PKI, SG has to endure the inefficient resource utilization due to the large key sizes as well as the large key distribution overhead. We have shown that our proposed SGMA and SGKM mechanism have successfully addressed both of these efficiency concerns of the PKI approach, while retaining the security strength of PKI.

Appendix A
Implementation of proposed mechanism in AVISPA

The HLPSL codes for AVISPA to define the SM and SAS roles are presented in Figs. 8 and 9, respectively. Also, the required session and environment HLPSL codes are shown in Fig. 10. Note that since AVISPA does not support arithmetic operations, we have used instead the "xor & exp" (raise to power) operators besides other security functions. The "xor" operator is used for the mod 2 "$+$ & $-$" (addition and subtraction) operations required for the authentication algorithms.

Acknowledgement

This work was supported in part by the Natural Sciences and Engineering Research Council (NSERC) of Canada through grant STPGP 396838.

References

[1] NIST Smart Grid, Cyber Security Working Group, "Introduction to NISTIR 7628 Guidelines for Smart Grid Cyber Security," Guideline, Sep. 2010. [Online]. Available: www.nist.gov/smartgrid

[2] P. McDaniel and S. McLaughlin, "Security and privacy challenges in the smart grid," *Security & Privacy, IEEE*, vol. 7, no. 3, pp. 75–77, 2009.

[3] Z. Fan, P. Kulkarni, S. Gormus, C. Efthymiou, G. Kalogridis, M. Sooriyabandara, Z. Zhu, S. Lambotharan, and W. H. Chin, "Smart Grid Communications: Overview of Research Challenges, Solutions, and Standardization Activities," *IEEE Commun. Surveys & Tutorials*, vol. 15, no. 1, pp. 21–38, 2013.

[4] J. Wang and V. Leung, "A survey of technical requirements and consumer application standards for ip-based smart grid ami network," in *International Conference on Information Networking (ICOIN)*. IEEE, 2011, pp. 114–119.

[5] H. Nicanfar, P. Jokar, and V. Leung, "Smart grid authentication and key management for unicast and multicast communications," in *IEEE PES Innovative Smart Grid Technologies Asia (ISGT)*. IEEE, 2011, pp. 1–8.

[6] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk, "Internet X. 509 public key infrastructure certificate and certificate revocation list (CRL) profile," 2008.

[7] H. Gharavi and B. Hu, "Multigate communication network for smart grid," *Proceedings of the IEEE*, vol. 99, no. 6, pp. 1028–1045, 2011.

[8] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[9] S. Bellovin and M. Merritt, "Encrypted key exchange: Password-based protocols secure against dictionary attacks," in *IEEE Computer Society Symposium on Research in Security and Privacy*. IEEE, 1992, pp. 72–84.

[10] D. Seo and P. Sweeney, "Simple authenticated key agreement algorithm," *Electronics Letters*, vol. 35, no. 13, pp. 1073–1074, 1999.

[11] Z. Zhang and Q. Zhang, "Verifier-based password authenticated key exchange protocol via elliptic curve," in *IEEE International Conference on Information Theory and Information Security (ICITIS)*. IEEE, 2010, pp. 407–410.

[12] T. Wu *et al.*, "The secure remote password protocol," in *Internet Society Symposium on Network and Distributed System Security*, 1998.

[13] A. Shamir, "Identity-based Cryptosystems and Signature Schemes," in *Advances in CryptologyCRYPTO 1984*. Springer, 1984, pp. 47–53.

[14] D. Boneh and M. Franklin, "Identity-based encryption from the Weil pairing," in *Advances in CryptologyCRYPTO 2001*. Springer, 2001, pp. 213–229.

[15] T. Wu *et al.*, "SRP-6: Improvements and Refinements to the Secure Remote Password Protocol," P1363.2 working group,.

[16] H. Nicanfar and V. C. Leung, "EIBC: Enhanced Identity-Based Cryptography, a Conceptual Design," in *IEEE InternationalSystems Conference (SysCon)*. IEEE, 2012, pp. 1–7.

[17] S. Kim, E. Kwon, M. Kim, J. Cheon, S. Ju, Y. Lim, and M. Choi, "A Secure Smart-Metering Protocol Over Power-Line Communication," *IEEE Transactions on Power Delivery*, vol. 26, no. 4, pp. 2370–2379, 2011.

[18] J. Kamto, L. Qian, J. Fuller, and J. Attia, "Light-weight key distribution and management for advanced metering infrastructure," in *IEEE GLOBECOM Workshops (GC Wkshps)*. IEEE, 2011, pp. 1216–1220.

[19] F. Zhao, Y. Hanatani, Y. Komano, B. Smyth, S. Ito, and T. Kambayashi, "Secure authenticated key exchange with revocation for smart grid," in *IEEE PES Innovative Smart Grid Technologies (ISGT)*. IEEE, 2012, pp. 1–8.

[20] X. He, M. Pun, and C. Kuo, "Secure and efficient cryptosystem for smart grid using homomorphic encryption," in *IEEE PEIS nnovative Smart Grid Technologies (ISGT)*. IEEE, 2012, pp. 1–8.

[21] J. Xia and Y. Wang, "Secure Key Distribution for the Smart Grid," *IEEE Transactions on Smart Grid*, vol. 3, no. 3, pp. 1437–1443, 2012.

[22] H. Tseng, "A secure and privacy-preserving communication protocol for v2g networks," in *IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 2012, pp. 2706–2711.

[23] Q. Gao, "Biometric authentication in Smart Grid," in *International Energy and Sustainability Conference (IESC)*. IEEE, 2012, pp. 1–5.

[24] S. Das, Y. Ohba, M. Kanda, D. Famolari, and S. Das, "A key management framework for ami networks in smart grid," *IEEE Communications Magazine*, vol. 50, no. 8, pp. 30–37, 2012.

[25] "AVISPA-Automated Validation of Internet Security Protocols." [Online]. Available: http://www.avispa-project.org

[26] D. Dolev and A. Yao, "On the security of public-key protocols," *IEEE Transactions Information Theory*, vol. 29.

Hasen Nicanfar (S'11) is a PhD student in the department of Electrical and Computer Engineering, the University of British Columbia. He received his BASc. degree in Electrical Engineering from Sharif University of Technology in 1993, and his MASc. degree in Computer Networks from Ryerson University in 2011. From 1993 to 2010, he has worked in different positions such as IT/ERP manager, project manager, production engineer/manager, and business & system analyst. His research interests are in the areas of the security, privacy, cryptography, system & network design and management for wireless communication and Smart Grid system.

Paria Jokar (S'11) received the B.Sc and M.Sc. degree with distinction in electrical engineering from the Iran University of Science and Technology. She is currently working toward Ph.D. in the Department of Electrical and Computer Engineering, The University of British Columbia, Canada.

She has more than five years of practical and working experience in the industry as network and security engineer & analyst. Currently, she holds the research assistant position in WiNMoS Lab. Her Research interests include computer networks, wireless networks and network security.

Konstantin Beznosov (M'98) is an Associate Professor at the Department of Electrical and Computer Engineering, University of British Columbia, where he directs the Laboratory for Education and Research in Secure Systems Engineering. His research interests are usable security, distributed systems security, secure software engineering, and access control. Prior UBC, he was a Security Architect at Hitachi Computer Products (America) and Concept Five. Besides many academic papers on security engineering in distributed systems, he is also a co-author of "Enterprise Security with EJB and CORBA" and "Mastering Web Services Security" books, as well as XACML and several CORBA security specifications.

Victor C.M. Leung (S'75-M'89-SM'97-F'03) is a Professor of Electrical and Computer Engineering and holder of the TELUS Mobility Research Chair at the University of British Columbia (UBC). He has contributed more than 600 technical papers and 25 book chapters in the areas of wireless networks and mobile systems. He was a Distinguished Lecturer of the IEEE Communications Society. He has been serving on the editorial boards of IEEE Transactions on Computers, IEEE Wireless Communications Letters and several other journals, and has served on the organizing and technical program committees of numerous conferences. Dr. Leung was a winner of the 2012 UBC Killam Research Prize, and the IEEE Vancouver Section Centennial Award.

```
role sgas_Init (SM,SAS :      agent,
                PW :          symmetric_key,
                Hsh :         hash_func,
                G,N :         text,
                Snd,Rcv :     channel(dy))
played_by SM
def=
 local          State :              nat,
        Rsm :                text,
        Salt :               protocol_id,
        PubKsm, PubKsas:     public_key,
        FFi, Ffi :           hash_func,
        STi, SNsm, Gsm, Gsas, Ver, K0, K, M1, M2, M, Ru, X, S0, S1, S2, S :        message

 const          sec_init_Si, sec_init_K :      protocol_id

 init  State := 0

 transition

 1. State = 0   /\ Rcv(start) =|>                    % start
   State'= 2    /\ Rsm' := new()                     % R_sm = Rnd(.)
                /\ SNsm' := new()                   % SN_sm = Rnd(.)
                /\ Gsm' := exp(G,Rsm')                    % Gsm = g^R_sm
                /\ Snd(SM.Gsm'.SNsm')               % Sending ID_sm, g^R_sm, SN_sm

 2. State = 2   /\ Rcv(Salt'.Gsas'.FFi'.STi') =|>   % Receiving Salt, B, Encrypted F_i(.) & State i with K
   State'= 4    /\ K0'= Hsh(N.G)                    % k= hash(N,g)
                /\ Ru'= Hsh(Gsm.Gsas')              % u= hash(A,B)
                /\ X'= Hsh(Salt'.PW)                      % x= hash(salt, pw)
                /\ Ver' := exp(G,X')                % ver = g^x
                /\ S0' := xor(Gsas',Hsh(K0'.Ver'))  % g^b + k.g^x - k.g^x = g^b
                /\ S1' := exp(S0',Rsm)                    % (g^b)^a = g^ab
                /\ S2' := exp(exp(S0',Ru'), X')           % ((g^b)^u)^x = g^bux
                /\ S' := xor(S1',S2')               % S = g^ab xor g^bux
                /\ K' := Hsh(S')                          % K = hash(S)
                /\ witness(SM,SAS,k1,K')            % Checking K
                /\ secret(K',sec_init_K,{SM,SAS})   % Checking K
                /\ M1' := xor(Hsh(N),Hsh(G))              % M1 = hash(N) xor hash(g)
                /\ M2' := Hsh(xor(SM,SNsm))               % M2 = hash(ID xor SN)
                /\ M' := Hsh(M1'.M2'.Salt.Gsm.Gsas'.K')   % M = hash(M1,M2,salt,A,B,K)
                /\ STi' := {STi'}_inv(K')                 % Decrypting i with K
                /\ FFi' := {FFi'}_inv(K')                 % Decrypting F_i with K
                /\ PubKsas' := FFi(SAS)             % PubK_sas = F_i(ID_SAS)
                /\ Snd({STi'}_inv(PubKsas'))              % sending i uncrypted by PubK_sas
                /\ witness(SM,SAS,si1,STi')         % Checking state i
                /\ secret(STi',sec_init_Si,{SM,SAS})% Checking state i

 2. State = 4   /\ Rcv(M) =|>                             % receiving hash(M)
   State' := 6
                /\ request(SM,SAS,k2,K)             % Checking K
                /\ request(SM,SAS,si2,STi)          % Checking state i

end role
```

Fig. 8: Smart Meter (SM) HLPSL codes

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

role sgas_Resp (SAS, SM:     agent,
                Ver :        message,
                Salt :       protocol_id,
                Hsh :        hash_func,
                G,N :        text,
                Snd, Rcv :   channel(dy))


played_by SAS

def=
 local State :  nat,
                Rsas :               text,
                FFi, Ffi :           hash_func,
                PubKsm, PubKsas :    public_key,
                Resi, Resj, STi, SNsm, Ru, M1, M2, M, K0, K, Gsm, Gsas, X, S0, S1, S :       message

 const          sec_resp_Si, sec_resp_K :      protocol_id

 init  State := 1

 transition

 1. State = 1   /\ Rcv(SM.Gsm',SNsm') =|>                % A and g^a
   State'= 3    /\ K0' := Hsh(N.G)                       % k = hash(N,g)
                /\ Rsas' := new()                        % b = Rnd()
                /\ Gsas' := xor(exp(G,Rsas'),Hsh(K0'.Ver))  % B = g^b + k.g^x
                /\ Ru' := Hsh(Gsm'.Gsas')                % u = hash(A,B)
                /\ S0' := exp(Gsm',Rsas')                % (g^a)^b = g^ab
                /\ S1' := exp(exp(Ver,Ru'),Rsas')        % ((g^x)^u)^b = g^bux
                /\ S' := xor(S0',S1')                    % S = g^ab xor g^bux
                /\ K' := Hsh(S')                         % K = hash(S)
                /\ M1' := xor(Hsh(N),Hsh(G))             % M1 = hash(N) xor hash(g)
                /\ M2' := Hsh(xor(SM,SNsm'))             % M2 = hash(ID xor SN)
                /\ M' := Hsh(M1'.M2'.Salt.Gsm'.Gsas'.K') % M = hash(M1,m2,salt,A,B,K)
                /\ STi' := new()                         % State i
                /\ PubKsas' := FFi(SAS)                  % PubK_sas = F_i(ID_sas)
                /\ Snd(Salt.Gsas'.{FFi}_K'.{STi'}_K')    % sending salt,B
                /\ witness(SAS,SM,k2,K')                 % Checking K
                /\ secret(K',sec_resp_K,{SM,SAS})        % Checking K

 2. State = 3   /\ Rcv({Resi'}_inv(PubKsas)) =|>         % receiving state i
   State'= 5    /\ witness(SAS,SM,si2,Resi')             % Checking state i
                /\ secret(Resi',sec_resp_Si,{SM,SAS})    % Checking state i
                /\ Snd(M)                                % sending M
                /\ request(SAS,SM,si1,Resi')             % Checking state i
                /\ request(SAS,SM,k1,K)                  % Checking K


end role

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Fig. 9: Smart Meter (SAS) HLPSL codes

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

role session(SM,SAS :          agent,
             PW :              symmetric_key,
             Salt :            protocol_id,
             Hsh :             hash_func,
             G,N :             text)
def=

  local SndSM, RcvSM, SndSAS, RcvSAS :   channel (dy)

  composition
       sgas_Init(SM,SAS,PW,Hsh,G,N,SndSM,RcvSM)  /\
       sgas_Resp(SAS,SM,exp(G,Hsh(Salt.PW)),Salt,Hsh,G,N,SndSAS,RcvSAS)    % x = hash(Salt, pw) & Ver = g^x

end role

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

role environment()

 def=
  const si1, si2, k1, k2 :       protocol_id,
        sm, sas, intruder :      agent,
        kab, kai, kbi :          symmetric_key,
        s_ab,s_ai,s_bi :         protocol_id,
        hsh :                    hash_func,
        g, n :                   text

 intruder_knowledge = {i, kai, kbi, s_ai, s_bi}

  composition
          session(sm,sas,kab,s_ab,hsh,g,n)
        /\ session(sm,intruder,kai,s_ai,hsh,g,n)
        /\ session(sas,intruder,kbi,s_bi,hsh,g,n)

end role

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

goal

   secrecy_of   sec_init_Si, sec_init_K, sec_resp_Si, sec_resp_K

   authentication_on k1
   authentication_on k2

   authentication_on si1
   authentication_on si2

end goal

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

environment()
```

Fig. 10: AVISPA Session and Environment HLPSL codes