RESEARCH ARTICLE

Password Authenticated Cluster-Based Group-Key Agreement for Smart Grid Communication

Hasen Nicanfar* and Victor C.M. Leung

WiNMoS Lab, Department of Electrical and Computer Engineering, The University of British Columbia, Vancouver, BC, Canada V6T 1Z4
Email: hasennic@ece.ubc.ca and vleung@ece.ubc.ca

ABSTRACT

Several multi-party systems supporting group- and cloud-based applications have been proposed in the context of Smart Grid. An important requirement of these systems is that the devices/parties need to communicate with each other as members of a group. In this paper, we present an efficient group-key (GK) management scheme aimed at securing the group communications, for instance, from the utility to appliances and smart meters located in different homes. Our scheme is based on the X.1035 Password Authenticated Key Exchange protocol standard, and also follows the cluster-based approach to reduce the costs of the GK construction and maintenance for large groups. Our protocol enables secure communications utilizing any communication technology. Analysis using one of the best evaluation tools in the technical community shows that our constructed GK is valid and secure against well-known attacks. We also show that the proposed scheme supports forward and backward secrecy, and is more efficient in comparison with other GK mechanisms in the literature. Copyright © 2012 John Wiley & Sons, Ltd.

KEYWORDS
Group Key; Cluster-Based; Password Authenticated; Security; Smart Grid.

*Correspondence
WiNMoS Lab, Department of Electrical and Computer Engineering, The University of British Columbia, Vancouver, BC, Canada V6T 1Z4, Email: hasennic@ece.ubc.ca

1. INTRODUCTION

Improvements in power generation, transmission, distribution and consumption as well as service delivery, operation and market management of the grid are the main benefits that will be realized with the implementation of the Smart Grid (SG) system. SG implementation will require extensive development and adoption of new technologies, especially information and communication technologies (ICT), that were not a part of the traditional grid [1]. This paper addresses secure communications over SG systems.

A key motivation of the SG is that ICT technologies can support the use of dynamic pricing to counteract the inefficiency of engineering and operating a power grid based on the peak demand of consumers. A price increase in the peak-hours of power demand is one of the tools that providers can use to encourage consumers to shift their demand to the off-peak hours [2]. Therefore, different applications and ICT systems are emerging to support the consumers’ needs to manage their energy demand in a smart way and even in real-time. Also, SG will integrate small power producers, which highlights the need for multi-party communications over the SG [3]. Different applications that require multi-party interactions in the SG context to address a variety of the customer needs have been reviewed in [4].

As shown by Figure 1, a typical SG links a group of consumers to a group of producers. The power generated by the producers is sent to the SG to be delivered via the transmission and distribution domains of the SG to the customers. The producers need to communicate with each other as a group, to balance their power generation in order to reach a better Return On Investment (ROI) for their assets. In the customers’ domain, the devices such as smart appliances or smart meters (SMs) need to communicate with each other as part of a group, to balance their demands in order to take advantage of the best/lowest price. For instance, the plug-in electric vehicles in different homes in a Neighbourhood Area Network (NAN) can schedule their charging time to achieve a flat power demand.
Generally speaking, in order to have the benefits of the smart consumption and/or generation, devices/parties are required to communicate with each other as part of a group, to balance their resources and/or demands. While these group-based applications can be centralized or distributed, distributed ones are more efficient since the parties can locally make decisions. Most of these communications are many-to-many, e.g., in [5] and [6], and without any doubt, having a symmetric group-key (GK) is the best solution to secure the communications.

Contribution: In this paper, we propose the Password Authenticated Cluster-based Group-Key Agreement (PACGKA) protocols to manage the security of group communications in SG to support multi-party applications. PACGKA extends the Password Authenticated Key Exchange (PAKE) protocol to construct and manage a GK among a cluster of devices, utilizing a pre-shared password for authentication. We show that key management using PACGKA is more efficient than existing methods without sacrificing security.

The outline of this paper is as follows. Section 2 provides a literature review of related security protocols and mechanisms. We present our proposed PACGKA protocols for key formation and key maintenance, for single cluster in Section 3 and multiple clusters in Section 4. We evaluate our proposal in Section 5 in terms of its security and efficiency. Section 6 concludes the paper.

2. LITERATURE REVIEW

In most of the existing solutions for construction of symmetric keys between two or more parties, the Diffie-Hellman (D-H) protocol [7], or a D-H based protocol is used. Furthermore, some additional features like using a predefined password can be incorporated in order to secure the solutions against attacks like Man-In-The-Middle (MITM). In [8], a PAKE protocol called Simple Authentication and Key Agreement is presented, in which both parties compute a shared number from their shared password, then each one picks a random number and multiplies it to the shared number to be used in the D-H algorithm. In [9], a three steps PAKE protocol is proposed to resist dictionary, password compromise impersonation and ephemeral key compromise impersonation attacks, and to supply forward secrecy. The authors of [10] used a temporary session key as a verifier in the final symmetric key establishment.

The PAKE protocol in the X.1035 standard [11] assumes that a password pw is shared by two parties. The mutual authentication protocol defined in the X.1035 standard utilizes the four phases shown in Figure 2 to form a symmetric cryptographic key by D-H exchange using D-H values g & p and five shared hash functions H1 = H2. In this figure, IDA & IDB are the identities (IDs) of two parties, e.g., Alice and Bob, a & b are random values chosen by the parties, and P = (IDA|IDB|pw). Also, the password based public key cryptographic technique is standardized in IEEE 1363.2 [12]. The standard specifies schemes for password-authenticated key agreement and password-authenticated key retrieval, which utilizes passwords and other low-grade secrets to secure electronic transactions.

The “conference key system” proposed by Burmester and Desmedt [13], known as the BD protocol, is a protocol that addresses the symmetric key construction for a group of users. This protocol consists of three steps. Consider n parties Uᵢ, i = 1, 2, ..., n, forming a cyclic group such that Uᵢ₊₁ = U₁.
1. Each member $U_i$ generates a random value $r_i$, computes $X_i$ via (1) and broadcasts it:

$$X_i \equiv g^{r_i} \mod p$$ (1)

2. After receiving the broadcast values by others in the previous step, each member $(U_i)$ calculates $Y_i$ via (2) and broadcasts it:

$$Y_i \equiv \left( \frac{X_{i+1}}{X_{i-1}} \right)^{r_i} \mod p$$ (2)

3. Then assuming the values of the previous steps are received by all the members, each member $(U_i)$ calculates the shared key $K_i$:

$$K_i \equiv (X_{i-1})^{n-r_i}.Y_{i-1}^{n-2}Y_{i+1} \mod p$$ (3)

As it can be seen by (3), the $K_i$s of the nodes are the same, which is called the shared key $K$.

Research and proposals on GK construction/management are mainly in two categories. In the first category, the GK is generated and managed by a central authority/controller. In the second category, the GK is constructed by participation of all the group members.

The first category is mainly motivated by multicast communications, which may have one source or one core node that handles data distribution to all the other nodes. The central controller generates and distributes the key between the receivers. The main problem in this category is in the efficiency and robustness of the key distribution and refreshment, along with the handling of membership changes (join and leave). There are different solutions in the literature for this category, most of which use the concept of structuring and forming the group in a tree topology [14] [15]. Since these systems use a central entity for the key management, they are vulnerable to a single point of failure. Although they are efficient in managing the join and leave of nodes/members, mostly the data needs to be partially decrypted by each node before being forwarded to the downstream nodes.

On the other hand, the second category is mainly motivated by many-to-many communications. They try to address the key construction in a distributed fashion by having participations of the entire membership. They are based either on the D-H or the BD protocol, with different techniques added to improve the key construction from the security and/or efficiency points of view.

Each one of the aforementioned proposals was designed for a specific application or platform. The group PAKE protocol [16] assumes that each user has an individual password shared with the server. This design dictates having multiple passwords saved in a server, which decreases the efficiency of the system. The BD protocol is extended in [17] to address the failure of the group members as well as the size of the messages that are transferred between the members. The proposal assumes having authenticated links between the members, and constructs the key during two rounds in a ring-based group.

The GK construction in [18] is aimed at small groups of entities. It assumes that each user has a workstation as well as a mobile device. The users meet each other while they carry their mobile devices. The mobile devices setup an initial shared value that they use in the workstations to communicate with each other.

The protocol presented in [19] is an extension of the existing protocol called S-3PAKE, both of which construct the GK assuming the existence of a server. The protocol of [19] increases the number of members of the group from three to $n$. In both protocols, the server plays the main role by receiving messages from all members and then responds to the members. Since the server needs to provide services to the entire membership and is involved in all the steps in the interaction, the protocol is vulnerable to the single point of failure.

By utilizing Exclusion Basis Systems (BES) and Cipher-Text-Policy Attribute-Based Encryption (CP-ABE) techniques in [20], a GK management for large scale systems is proposed. It provides an EBS-Based protocol that supports forward/backward secrecy relative to the join/leave process, and resilience to collusion attacks. Instead of using a clustering approach, it uses CP-ABE to handle large groups, which is more useful for the multicast communications.

Identity-Based Cryptography (IBC) is used in [21] to design a GK agreement for multicast communications. The design maintains forward secrecy and integrity, and is developed for a dynamic environment. The system requires a group leader with whom each member communicates to prepare the shared values for the key construction. Although the process consists of two rounds, in each round communication with the leader is required. The protocol proposed in [22] is based on identities and do not require certificates. The protocol starts by each member choosing a random number and sending it to other members. Then, the results of the second round calculations are broadcast to all the members. The members are able to compute the GK after the second round. Similar to many other proposals, this protocol relies on broadcasting data/messages to others, which may not be robust for large groups.

Several IBC-based GK agreement protocols are evaluated in [23]. Moreover, a survey on security of group communications is presented in [24]. A brief survey on cluster-based GK Agreement (GKA) protocols for wireless sensor networks is presented in [25], differentiated into infrastructure-based and infrastructure-less networks. The infrastructure-based protocols studied include the Hierarchical Key Agreement Protocol, GKA protocol for Circular Hierarchical Group, Password-Based GKA protocol for Hierarchical Group and AP-1 which is a cluster-based GKA protocol based on the constant round multi-party dynamic key agreement protocol. The survey shows that the best performance is delivered in a system with equal cluster size and a small number of layers.

The proposal in [26] provides a GK management for the advanced distribution automation system of SG, which
is based on a three-tier tree structure and decentralized architecture. In [27], firstly a SG gateway constructs a symmetric key with each SM based on a D-H algorithm. Then, the gateway multiplies the symmetric keys to form a GK, and finally, sends the GK using the symmetric keys to the SMs.

The tree concept is used in the key management proposal in [28] to cover unicast, multicast and broadcasting keys for the SG, in which the multicast key is close to our design. The design is based on a binary tree in which each node uses two hash functions to calculate secret values of the tree nodes, which requires knowing the entire tree construction. Due to the high resource consumption and overhead cost, it may not be suitable for the SG with many nodes.

Recently we proposed a multilayer key formation, which deals with the multi symmetric keys between a smart appliances and upper layer controllers [29] as part of an access control proposal. In this paper, we focus on forming a GK, for instance, for different devices located inside and outside of the HANs.

Discussion: Generally speaking, the mechanisms that are based on the BD protocol may suffer from the following weaknesses: (i) Some of them rely on a server, which makes them vulnerable to a single point of failure. (ii) Mostly they use broadcasting to distribute the key construction messages, which lack robustness as the messages may not be received by all the members. Even if they include a verification step to address this issue, it makes the algorithm time consuming and increases the system overhead. The problem is worsened in a large group with a long distance between nodes, or if the Internet is used for the communications.

Thus, to overcome the aforementioned issues, especially the second (ii) problem, we propose to unicast the messages in the PACGKA protocol presented in this paper, which is based on the PAKE protocol in the X.1035 standard. As we will show in Section 5, an approach based on the BD protocol would be less efficient as it requires a larger number of messages in the protocol operation.

3. PACGKA-I PROTOCOL FOR SINGLE CLUSTER

The PACGKA protocol for a single cluster (PACGKA-I) is presented in Algorithm 1 for constructing and verifying a shared value, and calculating the GK. The mechanism constructs the shared value in two rounds involving \(2 \times n - 1\) messages. PACGKA-I consists of the protocol for forming the GK and the auxiliary protocol for key maintenance. As shown in Figure 3, we assume the members’ IDs form a cyclic group.

To describe the protocol based on Figure 3, consider a group with four parties \(ID_1, ID_2, ID_3\) & \(ID_4\), which are preloaded with the \(g, p\) & \(H(.)\) parameters. They also receive a shared password \(pw\) from the system along with the required system parameters such as number of entities \((n = 4)\) in the group (ring/cluster) plus IDs of the neighbours (prior & next). The protocol use a message vector \(M\) that has \((n - 1)\) fields (three in this example).

3.1. Group Key Construction

3.1.1. First round

We run the protocol starting from \(ID_1\).

**Note:** For encryption of the message vector \(M\), the parties simply multiply each field of the \(M\) to the forward session key \(P_{k+}\). Thus for decryption, the parties only need to divide the fields of the received vector \(M\) to the backward session key \(P_{k-}\).

\(ID_1\): First, \(ID_1\) generates random value \(r_1\), computes initial value and loads the \(M[1]\) to begin with. Then, \(ID_1\) calculates the backward and forward session keys with \(ID_4\) and \(ID_2\), which are given by \(P_{1-}\) and \(P_{1+}\), encrypts \(M\) with \(P_{1+}\), and sends it to \(ID_2\).

\[\begin{align*}
    r_1 &= Rand(.) \times ID_1 \equiv g^{r_1} \mod p \\
    P_{1-} &= H(ID_4|pw|ID_1) \\
    P_{1+} &= H(ID_2|pw|ID_2)
\end{align*}\]

\(ID_2\): \(ID_2\) generates random value \(r_2\) and also computes the backward and forward session keys \(P_{2-}\) and \(P_{2+}\). Then, \(ID_2\) receives \(M\) and decrypts it with \(P_{2-}\). Note that \(P_{2-} = H(ID_1|pw|ID_2) = P_{1+}\). Then, \(ID_2\) updates \(M\) and finally, encrypts it with \(P_{2+}\) and sends it to \(ID_3\).

\[\begin{align*}
    r_2 &= Rand(.) \\
    P_{2-} &= H(ID_1|pw|ID_2) \\
    P_{2+} &= H(ID_2|pw|ID_1) \\
    M[2] &\equiv M[1]^{r_2} \mod p \equiv g^{r_2r_1} \mod p \\
    M[1] &\equiv g^{r_2} \mod p
\end{align*}\]

\(ID_3\): Similarly, \(ID_3\) generates random value \(r_3\) and also computes the backward and forward session keys \(P_{3-}\) and \(P_{3+}\). Then, \(ID_3\) receives \(M\) and decrypts it with \(P_{3-}\). Then, \(ID_3\) updates the vector \(M\) and finally encrypts it.

![Figure 3. Single Cluster (Ring-Based) Structure](image-url)
Algorithm 1 PACGKA-I: Group-key formation for a single-cluster group

Define:
n : Total number of members, where "n + 1 \equiv 1" and "1 \equiv n".
g \& k : DH algorithm parameters.
H(.) : Shared Hash function.
pw : Shared password
M : An n - 1 element message vector; M[i] is kth field of the M.
ID_k : ID of the kth party.
Rand(.) : Random number generator function.
                                          r_k \& SV_k : Random value and final shared value of ID_k
P_k \& P_k− : Forward \& backward session keys of the kth party.
E_k(X) \& D_k(X) : Encryption \& decryption of X with the K key.
V_i \& V_i+ : Verifier for the previous and next parties.
P_{GK}(. \& P_{GK}(. : Background functions to send and receive messages.

Algorithm:
First round: ID_1 to ID_{n−1}
\begin{align*}
r_1 & \leftarrow \text{Rand}(.) \\
M[1] & \leftarrow g^{r_1} \mod p \\
P_{1−} & \leftarrow H(ID_{n−1}\{pw\}|ID_1) \\
P_{1+} & \leftarrow H(ID_1\{pw\}|ID_2) \\
MyEncM & \leftarrow E_{P_{1−}}(M) \\
F_2(MyEncM \rightarrow ID_2) & \\
\end{align*}
for i = 2 \to n − 1 do
\begin{align*}
r_i & \leftarrow \text{Rand}(.) \\
P_{1−} & \leftarrow H(ID_{n−1−1}\{pw\}|ID_i) \\
P_{1+} & \leftarrow H(ID_1\{pw\}|ID_{i+1}) \\
F_2(MyEncM \leftarrow ID_1−1) \\
M & \leftarrow D_{P_{1−}}(MyEncM) \\
for j = i \to 2 do \\
M[j] & \leftarrow M[j] \times i \mod p \\
end for \\
end for \\
Second round: ID_{n} and ID_{1} to ID_{n−1}
\begin{align*}
F_2(MyEncM \leftarrow ID_{n−1}) \\
M & \leftarrow D_{P_{1−}}(MyEncM) \\
SV_n & \leftarrow M[n−1] ^ {r_1} \mod p \\
for j = n − 1 \to 1 do \\
M[j] & \leftarrow M[j−1] ^ {r_1} \mod p \\
end for \\
end for \\
V_{n+} & \leftarrow H(pw|M[n−1]|SV_n) \{Verifier for the next party\} \\
MyEncM & \leftarrow E_{P_{n+}}(M) \{n + 1 \equiv 1\} \\
F_2(MyEncM,V_{n+}) & \leftarrow ID_1 \\
end for \\
for i = 1 \to n − 1 do \\
F_3(MyEncM) & \leftarrow ID_{i+1} \\
M & \leftarrow D_{P_{1−}}(MyEncM) \\
SV_i & \leftarrow M[n−1] ^ {r_1} \mod p \\
if V_{i+} \leftarrow H(pw|M[n−1]|SV_i) then \\
GK_i & \leftarrow H(pw|SV_i) \\
else \\
return Error : Verification failed \\
end if \\
for j = n − 1 \to i + 1 do \\
M[j] & \leftarrow M[j−1] ^ {r_1} \mod p \\
end for \\
V_{i+} & \leftarrow H(pw|M[n−1]|SV_i) \\
MyEncM & \leftarrow E_{P_{i+}}(M) \\
F_3(MyEncM,V_{i+}) & \leftarrow ID_{i+1} \\
end for 

Algorithm 2 PACGKA-II: Group-key formation for a multi-cluster group

with the forward key P_{3+} and sends it to ID_4.
\begin{align*}
r_3 & \leftarrow \text{Rand}(.) \\
P_{3−} & \leftarrow H(ID_{n}|pw|ID_4) \\
P_{3+} & \leftarrow H(ID_4|pw|ID_4) \\
M[3] & \equiv M[2] ^ {r_3} \mod p \equiv g^{r_3r_2} \mod p \\
M[2] & \equiv M[1] ^ {r_3} \mod p \equiv g^{r_3r_2} \mod p \\
M[1] & \equiv g^{r_3} \mod p \\
\end{align*}

3.1.2. Second round

This round starts with ID_1.
ID_4: Similar to ID_3, firstly ID_4 generates random value r_4 and computes the backward and forward session keys P_{4−} and P_{4+}. Then, ID_4 receives M and decrypts it with P_{4−}. ID_4 (last member of the cyclic group) now is able to calculate its shared value SV_4. Then, ID_4 updates M and computes the GK as well as a verifier for the next party (ID_1). Finally, ID_4 encrypts M with P_{4+} and sends it along with the verifier to ID_1.
\begin{align*}
r_4 & \leftarrow \text{Rand}(.) \\
P_{4−} & \leftarrow H(ID_3|pw|ID_4) \\
P_{4+} & \leftarrow H(ID_4|pw|ID_4) \\
SV_4 & \equiv M[3] ^ {r_4} \mod p \equiv g^{r_4r_2r_1} \mod p \\
M[3] & \equiv M[2] ^ {r_4} \mod p \equiv g^{r_4r_2} \mod p \\
M[2] & \equiv M[1] ^ {r_4} \mod p \equiv g^{r_4} \mod p \\
M[1] & \equiv g^{r_4} \mod p \\
G_{K_4} & \leftarrow H(pw|SV_4) \\
\end{align*}

ID_1: First of all, ID_1 receives M and decrypts it with P_{1−}. Then, ID_1 calculates its shared value (SV_1 = SV_2) and then verifies it versus the received verifier VR_{F_1−}(=VR_{F_1+}) from ID_4. Assuming the verification holds positive, ID_1 is assured that its shared value is the same as the one that ID_4 has. Then, ID_1 updates M and also calculates a verifier VR_{F_1+} for the next party. ID_1 finally encrypts M with P_{1+} and sends it along with the verifier to ID_2.
\begin{align*}
SV_1 & \equiv M[3] ^ {r_1} \mod p \equiv g^{r_4r_3r_2r_1} \mod p \\
V_{1−} & \leftarrow H(pw|M[3]|SV_1) \\
M[3] & \equiv M[2] ^ {r_1} \mod p \equiv g^{r_4r_3r_1} \mod p \\
M[2] & \equiv M[1] ^ {r_1} \mod p \equiv g^{r_4} \mod p \\
V_{1+} & \leftarrow H(pw|M[3]|SV_1) \\
G_{K_1} & \leftarrow H(pw|SV_1) \\
\end{align*}

ID_2: Similarly, ID_2 receives M and decrypts it with backward session key P_{2−}. Then, ID_2 calculates its shared value (SV_2 = SV_1) and then verifies it versus the received verifier VR_{F_2−}(=VR_{F_2+}) from ID_1. If the verification holds positive, ID_2 is assured that its shared value is the same as the one that ID_1 has, which is the same as the shared value of ID_4. Then, ID_2 updates M.
and also calculates a verifier \( V_{F_2+} \) for the next party. \( ID_2 \) finally encrypts \( M \) with \( P_{2+} \) and sends it along with the verifier to \( ID_3 \).

\[
\begin{align*}
SV_2 &\equiv M[3]^{r_2} \text{ mod } p \equiv g^{4r_3r_2r_1} \text{ mod } p \quad (8) \\
V_{2-} &\leftarrow H(pw|M[3]|SV_2) \\
M[3] &\equiv M[2]^{r_2} \text{ mod } p \equiv g^{4r_3r_2r_1} \text{ mod } p \\
V_{2+} &\equiv H(pw|M[3]|SV_2) \\
GK_2 &\leftarrow H(pw|SV_2) \quad (9)
\end{align*}
\]

\( ID_3 \) receives \( M \) and decrypts it with \( P_{3-} \). Then, \( ID_3 \) calculates its shared value and then verifies it versus the received verifier \( V_{F_3-} (= V_{F_2+}) \) from \( ID_2 \).

Assuming the verification holds positive, \( ID_3 \) is assured that its shared value is the same as the one that \( ID_2 \) has, which is the same as the shared value of \( ID_1 \) and \( ID_4 \). \( ID_3 \) is the last party that was supposed to calculate the shared value. The only step left is verifying it for the \( ID_4 \). Therefore, \( ID_3 \) calculates a verifier \( V_{F_3+} \) and sends it to \( ID_4 \).

\[
\begin{align*}
SV_3 &\equiv M[3]^{r_2} \text{ mod } p \equiv g^{4r_3r_2r_1} \text{ mod } p \\
V_{3-} &\leftarrow H(pw|M[3]|SV_3) \\
V_{3+} &\equiv H(pw|M[3]|SV_3) \\
GK_3 &\leftarrow H(pw|SV_3) \quad (11)
\end{align*}
\]

\( ID_4 \): Finally, \( ID_4 \) only needs to check the verification value. The positive verification result assures that \( ID_4 \) has the same shared value that \( ID_3 \) has.

\[
V_{4-} \leftarrow H(pw|M[3]|SV_4)
\]

**Note:** That the nodes have the same shared value can be seen by (4), (6), (8) and (10). Therefore, the \( GK_i \)s are the same, which are shown by (5), (7), (9) and (11).

### 3.2. Key Maintenance

#### 3.2.1. Key Refreshment

To improve and guarantee/increase the secrecy of the GK, PACGKA-I refreshes the key periodically. In order to do this, we propose setting up a timer to initiate and trigger the refreshment process. Note that the timer value that determines how often the key is refreshed depends on the application as well as the size of the group. Therefore, we propose the following Group Key Reconstruction (GKR) process for PACGKA-I: the system controller distributes a new password along with the start and expiry times to the entire group members to construct the new GK.

#### 3.2.2. Join and leave process

A new node should not gain access to the past information (forward secrecy), and a leaving node should not gain access to the future information (backward secrecy). In the case of a new node joining the existing group, or an existing node leaving the group, the controller performs GKR to support the forward and backward seccreces.

### 3.2.3. Malicious behaviour of a node

In case one of the group members begins behaving maliciously, the malicious node is removed from the group. In PACGKA-I, the system controller relies on both peer neighbours of the malicious node to vote jointly to identify the misbehaving member. In this case, they directly send a unicast message via the secure channel to the system controller. Subsequently, the system controller invokes the GKR algorithm for the group while excluding the malicious one.

### 4. CLUSTER-BASED MECHANISM: PACGKA-II PROTOCOL

In this section, we present an efficient multi-cluster GK management mechanism, PACGKA-II, which is based on the single cluster algorithm presented in Section 3. This scheme is motivated by the fact that in the case of a large group, the PACGKA-I protocol becomes time consuming as the nodes should perform many polynomial and arithmetic operations. Although SG systems are mostly static with low occurrences of node joining or leaving, security considerations dictate running GKR every so often for key refreshment. To overcome the latency issue, we propose using a clustering approach.

#### 4.1. Clustering Scheme

We define our clustering scheme following the presentations in Section 2 and [25]. Consider a group with \( N \) members. We divide the group into \( n \) clusters with no more than \( m \) members in each cluster (12):

\[
N \leq m \times n \quad (12)
\]

An example of the clustering scheme is depicted by Figure 4. We identify each cluster by \( Clstr_u, u = 1, ..., n \). Furthermore, members of the \( u \)th cluster are denoted by \( MC_k^u, k = 1, ..., m \). One of the cluster members acts as
heads, given by $C$ form a core ring/sub-group consisting of all the cluster heads, given by $C = Clstr$.

Note that finding the right value for $m$ (or $n$) is an optimization problem and can be formed based on the criteria that are important for the system and the application, and we leave this task to the administrator of the system. For instance, the problem can be formed to minimize the number of operations, or the delay of the key formation process, or any other system parameter or security measures. Indeed, the problem should address the application, system, resources and security aspects. After presenting our protocol, we will give an example of this problem to find the optimum values of $n$ and $m$.

### 4.2. The logic of the multi-cluster group key mechanism

Overall, PACGKA-II follows a similar concept as PACGKA-I. In fact, PACGKA-II can be considered as an extended version of PACGKA-I. The main steps of the PACGKA-II protocol are as follows:

I. Dividing the main group to clusters $Clstr_u$.

II. Nominating one party per cluster as the cluster head $HC_u$ to represent the cluster.

III. Forming the core cluster consisting of all the cluster heads. Note that each cluster head is a member of two sub-groups: the cluster it is representing and the core cluster.

IV. The protocol starts by sending a password $pw$ to the core cluster/sub-group.

V. Each cluster head picks a password $pw_{u,m}$, sends it to its own cluster members to construct a GK using PACGKA-I within the cluster. Note that, since the cluster head is a member of the cluster it is representing, at the conclusion of PACGKA-I, it has the GK of the sub-group (sub-GK) $SG_u$. Also, the cluster head has the shared value that the members of the sub-group used to obtain sub-GK. Let us call this shared value the "sub-group shared value" $G_u$.

\[
G_u \equiv g^{\prod_{i=1}^{n} r_u^i} \mod p \quad (13)
\]
\[
SG_u = H(G_u|pw_u) \quad (14)
\]

VI. Using the password received in Step IV, the members of the core cluster run PACGKA-I to construct the shared value $HSV_u$ and GK $K_{Group}$, taking the sub-group shared value (from Step V) as the random value of the cluster head ($G_u$).

\[
HSV_u \equiv g^{\prod_{i=1}^{n} G_i} \mod p \quad (15)
\]
\[
K_{Group} = H(HSV_u|pw) \quad (16)
\]

VII. The cluster head distributes the GK $K_{Group}$ to the cluster utilizing sub-GK $SG_u$ for encrypting the GK.

### 4.3. Key maintenance

All of the situations that require key maintenance are explained in Section 3 regarding single cluster GK formation are applicable to the multi-cluster GK formation as well. To handle the key refreshment, we need to rerun the complete PACGKA-II protocol. However, for situations such as a member joining or leaving, and detection of a malicious node, we propose a different solution. If a member joins the group, the new member should join one of the clusters, so it can be considered as a sub-group event. If one of the cluster members becomes malicious or leaves the cluster, again it can be considered as a sub-group event, unless the malicious node, or the node that is leaving the group is a cluster head, in which case we call it a cluster head event.

#### 4.3.1. Sub-group event

Let us assume that the event occurs inside the $u^{th}$ cluster. In this case, the cluster head $HC_u$ reselects a password $pw_{u,m}$ and shares it with its cluster members. Then, the cluster members of the $Clstr_u$ performs PACGKA-I to form sub-GK. Then, the cluster heads perform Steps VI and VII of PACGKA-II. In fact, the other sub-groups do not need to reconstruct their sub-GK, and the cluster heads can still use the prior values. Finally, the cluster heads inform their cluster members about the new GK.

#### 4.3.2. Cluster head event

In this case, either a cluster head is malicious or it leaves the group. Firstly, a new cluster head for that cluster needs to be chosen, and secondly, the GK should be constructed by performing PACGKA-II completely.

### 4.4. Size of the clusters

As shown above, PACGKA-II involves running PACGKA-I in two rounds, once around each cluster and then around the core cluster. Here we illustrate the optimization of the size of the clusters with respect to the delay, by formulation the delay expression and then minimizing it. Table I shows our parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$</td>
<td>Total number of the members in the group</td>
</tr>
<tr>
<td>$m$</td>
<td>Number of the members per cluster</td>
</tr>
<tr>
<td>$n$</td>
<td>Number of the clusters (sub-groups)</td>
</tr>
<tr>
<td>$d$</td>
<td>Party processing time including message delivery</td>
</tr>
<tr>
<td>$\hat{D}$</td>
<td>Delay of the GK construction</td>
</tr>
</tbody>
</table>

We assume equal “party processing time including message delivery” values ($\hat{d}$) for each party, and equal size of the clusters. Our problem is minimizing the “delay of the GK construction” ($\hat{D}$), which can be formulated as...
follows:

\[
\begin{align*}
\text{Min} & \quad \hat{D} = \dot{d} + \ddot{d} + (2m - 1)d + (2n - 1)\dot{d} + \dot{d} \\
\text{S.t.} & \quad m \times n \geq N
\end{align*}
\]

In the above problem formulation, each term in the right hand side of the delay equation respectively represents the delay of:

- Distributing password within the core cluster.
- Distributing password within each sub-group.
- Sub-GK construction.
- GK construction within the core cluster.
- Distributing the GK inside the clusters.

To solve this problem, we simplify it and rewrite as follows:

\[
\begin{align*}
\text{Min} & \quad \hat{D} = (2m + 2n + 1)d \\
\text{S.t.} & \quad m \times n \geq N
\end{align*}
\]

\(\hat{D}\) is a convex function, which has a minimum. We solve the problem in the border of \(m \times n = N\), and calculate the first derivative respect to \(n\) in order to find the optimal value:

\[
\begin{align*}
\frac{m \times n = N}{m = \frac{N}{n}} \\
\frac{\dot{D}}{n} = (2m + 2n + 1)\dot{d} \\
\frac{\partial \dot{D}}{\partial n} = (2m + 2n + 1)\dot{d} = 0 \implies m = n = \sqrt{N}
\end{align*}
\]

Therefore, the best performance of the protocol and the minimum delay happens when \(m = n = \sqrt{N}\).

5. SECURITY AND PERFORMANCE ANALYSIS

To analyze and evaluate the security of the PACGKA protocols, we consider the Dolev-Yao approach [30].

5.1. Formal Validation using Software Tool

To evaluate the validity of the generated key, we use the Automated Validation of Internet Security Protocols and Application (AVISPA) security analyzer package [31]. AVISPA is one of the best and the most trusted tools for the automatic verification and analysis of Internet security protocols in the literature. AVISPA analyzes the scheme and protocols with respect to different attacks by combining automatic security analysis and verification back-end servers like On-the-Fly Model-Checker (OFMC) and Constraint-Logic-based Attack Searcher (CL-AtSe). To perform the key validation using AVISPA and evaluate it using the back-end servers, the protocols and mechanisms are required to be coded by the High Level Protocol Specifications Language (HLPSL). More information about the AVISPA, its features and HLPSL language can be found in [31].

We develop our analysis for a group consisting of four members, corresponding to the example in Section 3. The simulation results presented by Figure 5a and Figure 5b show that the GK constructed by the PACGKA mechanism is secure and safe to be used by the members of the group. Although the system controller has provided the shared password, it does not have access to the GK. We assume this entity is trusted and does not perform any attacks like MITM. Figure 6 presents the evaluation program and AVISPA related HLPSL codes for the session, environment and goal sections. Also, HLPSL codes of the four entities A, B, C, and D roles are shown in Figure 7, 8, 9 and 10, respectively.

5.2. Adversary model

Objective

- Gaining access to the system resources, like a SM or an appliance.
- Performing a MITM attack to gain access to the GK, or a sub-GK.

Initial capabilities

- The adversary has complete knowledge about the topology and the exact address/ID of each party.
- The adversary has access to the system hash function \(H(.)\) and \(g \& p\) used in our protocol.
- The adversary knows the detail design of the PACGKA mechanism (PACGKA-I and PACGKA-II protocols).

Capabilities during the attack

- The adversary receives the entire encrypted and unencrypted (plain) data in different stages of the key formation, or later on and during the using of the GK.
• If the adversary gains access to any password (core cluster, or any other sub-group cluster), she/he will attempt to perform a MITM attack.
• If the adversary gains control to a malicious node, she/he can perform DoS by joining and leaving continuously.

Discussion
We assume cluster heads and cluster members receive the appropriate password via a secure channel. Therefore, if the adversary finds out the password of a cluster after completion of the initial sub-GK formation (PACGKA-I protocol), the adversary cannot gain any further information since the password is not being used any more. Similarly, if the adversary by performing any attack like brute-force or off-line dictionary obtains the shared password of the core cluster after the GK construction, this information is useless for the adversary since the key is formed and the password is like a one-time password. Thus there is resilience against Ephemeral key compromise impersonation. However, if the adversary finds/steals the password before the key formation process starts in any level such as in a cluster (PACGKA-I & PACGKA-II) or in the core cluster (PACGKA-II), she/he can take advantage of this password by performing a MITM attack. As long as the GK is valid without any changes, the adversary can use it. However, the GKR process changes the key completely. Thus, key refreshment by GKR periodically should be considered as a requirement for the system.

Another opportunity of the adversary is compromising the server by for instance social engineering attack. Consequently, the adversary can send the new password to the cluster head and dictates them to re-construct the GK. Although we improved the process of the key formation by using the clustering approach, it can harm the system resources. On the other hand, the adversary can participate in the key formation and gain access to the GK easily. Performing social engineering attack against the server is possible in any system and environment. The only solution to prevent this attack is having a strong system security management procedure. Generally speaking, although technically feasible, the social engineering attack should be a very expensive attack. Therefore, the best solution is increasing the cost of the attack, in order to make it unattractive for the adversary.

5.3. Attack analysis
Based on aforementioned discussion about the adversary, plus the PACGKA assumptions in Section 3 (i.e., parties are already authenticated to the system and have valid security system and key management to be able to have a secure communication), Table II analyzes the resilience of PACGKA against different well-known attacks.

Unknown key-share attack In our proposed mechanism, all of the parties should participate in the key formation and the verification steps. Indeed, the key is formed in a consensus manner with commitments of the entire membership. Thus, our protocol guarantees that if one of the members has the key, its neighbours have it as well.

Denning-Sacco attack resilience Due to using hash functions in the final key calculation steps in the sub-GK or in the final step of the GK (and verification steps), finding a sub-GK or a GK does not help adversary to gain access to the cluster or cluster head initial passwords.

5.4. Overhead analysis
Following our discussions in Section 2, in a BD-based mechanism, the messages are supposed to be distributed to the entire membership. The original concept is to broadcast the messages to the group members, although it may not be possible in all cases. One may consider broadcasting the message in the overlay layer; however, in the lower layer the messages are transferred by unicast communications. Moreover, it may be possible to broadcast the messages only to a small group within a short distance. Thus, making sure that the messages reach the destinations can cause extra overheads. Missing any message by any member causes failure on the algorithm.

Let us assume that we have a group with $n$ members, all in one cluster. We assume the following scenarios:

1. BD protocol based model: The messages of each member in the first round should be delivered to two members ($2 \times n$ messages), and in the second round, to all the members ($n \times (n - 1)$ messages), which totally is $n \times (n + 1)$ messages.

2. PACGKA: We require $2 \times n - 1$ message delivery (including the verification).

Regardless of the $n$ value, the second scenario has a smaller number of message deliveries. If we increase the $n$ value to a high value, the second scenario requires about $2/n$ times the number of message deliveries in the first scenario

5.5. Implementation considerations
Any application that requires a GK can use the PACGKA protocol. The method is scalable and can be easily

Table II. PACGKA attacks resilience summary

<table>
<thead>
<tr>
<th>Attack</th>
<th>Resilience</th>
</tr>
</thead>
<tbody>
<tr>
<td>Social engineering attack</td>
<td>✔️ ✗ ✗</td>
</tr>
<tr>
<td>Brute-force attack</td>
<td>✔</td>
</tr>
<tr>
<td>Replay attack</td>
<td>✔</td>
</tr>
<tr>
<td>DoS attack</td>
<td>✔</td>
</tr>
<tr>
<td>MITM attack</td>
<td>✔</td>
</tr>
<tr>
<td>On-line dictionary attack</td>
<td>✔</td>
</tr>
<tr>
<td>Off-line dictionary attack</td>
<td>✔</td>
</tr>
<tr>
<td>Unknown key share attack</td>
<td>✔</td>
</tr>
<tr>
<td>Denning-Sacco attack</td>
<td>✔</td>
</tr>
<tr>
<td>Ephemeral key compromise impersonation</td>
<td>✔</td>
</tr>
</tbody>
</table>
implemented based on different system specifications. Same as any other security system, the strength of the key required to achieve certain the security/confidentiality level depends on its size. We do not specify the key size or the time period of the key refreshment process, and leave them to be defined by the system administrator. Furthermore, while we propose that a group can be divided to the clusters, the number of clusters and size of each cluster are also parameters to be determined by the system administrator. For instance, the administrator may define each NAN as a cluster, and choose the NAN controller to act as the cluster head. Indeed, these set up values are driven by the application and system conditions. The detail analysis of the application and system resources helps the administrator of the system to identify the key size, as well as the size of the $g$ & $p$ parameters used in the PACGKA key construction mechanism.

6. CONCLUSION

In this paper, we have presented PACGKA, a group key management scheme aimed at securing communications by the multi-player and group-based applications in the SG context. The mechanism, which includes an auxiliary protocol for the key maintenance, is useful in a variety of applications such as managing power consumption and power generation. Indeed, in any situation where multiple parties are required to be part of a group that makes an agreed group decision in a consensus manner, they can use the PACGKA protocol. This protocol is flexible and is designed as an extension to the PAKE protocol and Diffie-Hellman algorithm. To reduce the key size, one can consider using the elliptic curve cryptography approach. The PACGKA protocols can be used with any group sizes, with PACGKA-I aimed at smaller groups while PACGKA-II targets larger groups by dividing each group into clusters. We have presented extensive analysis to show that the GK generated by the PACGKA protocol is valid and safe, and the protocol in secure against different attacks. Furthermore, the PACGKA protocol is efficient in system overheads compared with the BD-based protocol mechanisms.

ACKNOWLEDGEMENT

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

REFERENCES


Password Authenticated Cluster-Based Group-Key Agreement for Smart Grid Communication

H. Nicanfar and V.C.M. Leung

EX PROTOCOL: SGGM

HLPSL:

def:

role sgsk_2 (A,B,C : agent, 
    G  : text, 
    Hsh : hash_func, 
    Kb_da : symmetric_key, 
    Smw,Nov : channel(dy), 
    Pa : symmetric_key)

played_by D

- role sgsk_2 (A,B,C : agent, 
    G  : text, 
    Hsh : hash_func, 
    Kb_da : symmetric_key, 
    Smw,Nov : channel(dy), 
    Pa : symmetric_key)

played_by D

const sec_GK_DC, sec_GK_BA : protocol_id

init State := 0

transition

1. State := 0
   / Rxv(A.{{Gda'}_Kab.{Gcda'}_Kab}_Pw.B) => % receive: g^a, g^bc, g^abc
      / Gx' := exp(Gx,Pa) % g^xa
      / Gb := exp(Ga,Pa) % g^ab
      / St_A := Hsh(Pw,Gdab) % send: g^abcd
      / witness(B,C,{{Gdab'}_Kbc.{Gabc'}_Kbc}_Pw.D) % Checking group key with B
      / secret(GK,sec_GK_BA,{A,B}) % Checking group key with A

2. State := 1
   / Rxv(A.{{Gda'}_Kab._Pw.B} => % receive verifier from A
      / St_B := Hsh(Pw,Gbc) % Checking group key with B
      / witness(C,D,{{Gbc'}_Kcd._Pw.B}) % Checking group key with C
      / request(B,C,{{Gdab'}_Kbc._Pw.D}) % Checking group key with C

3. State := 2
   / Rxv(A.{{Gda'}_Kab._Pw.B} => % receive verifier from A
      / St_C := Hsh(Pw,Gab) % Checking group key with C
      / witness(D,A,{{Gabc'}_Kda._Pw.B}) % Checking group key with D
      / request(B,C,{{Gdab'}_Kbc._Pw.D}) % Checking group key with C

end role

---

Figure 8. Second Entity HLPSL Codes

---

EX PROTOCOL: SGGM

HLPSL:

def:

role sgsk_3 (B,C,D : agent, 
    G  : text, 
    Hsh : hash_func, 
    Kb_da : symmetric_key, 
    Smw,Nov : channel(dy), 
    Pa : symmetric_key)

played_by C

const sec_GK_DA, sec_GK_BC : protocol_id

init State := 0

transition

1. State := 0
   / Rxv(C.{{Gbc'}_Kdc._Pw.C} => % receive: g^a, g^bc, g^dab
      / Gx' := exp(Gx,Pa) % g^xa
      / Gb := exp(Ga,Pa) % g^ab
      / Sx := Hsh(Pw,Gcdab) % send: g^abcd
      / witness(B,D,{{Gdab'}_Kdc._Pw.C}) % Checking group key with B
      / secret(GK,sec_GK_DA,{B,C}) % Checking group key with B

2. State := 1
   / Rxv(B.{{Gdb'}_Kbc._Pw.B} => % receive: g^dab
      / Gx' := exp(Gx,Pa) % g^xa
      / Gb := exp(Ga,Pa) % g^ab
      / Sx := Hsh(Pw,Gdab) % send: g^abcd
      / witness(C,D,{{Gdab'}_Kbc._Pw.B}) % Checking group key with C
      / secret(GK,sec_GK_DA,{A,C}) % Checking group key with A

3. State := 2
   / Rxv(B.{{Gdb'}_Kbc._Pw.B} => % receive: g^dab
      / Gx' := exp(Gx,Pa) % g^xa
      / Gb := exp(Ga,Pa) % g^ab
      / Sx := Hsh(Pw,Gdab) % send: g^abcd
      / witness(C,D,{{Gdab'}_Kbc._Pw.B}) % Checking group key with C
      / secret(GK,sec_GK_DA,{A,D}) % Checking group key with A

end role

---

Figure 9. Third Entity HLPSL Codes

---

EX PROTOCOL: SGGM

HLPSL:

def:

role sgsk_4 (C,D,A : agent, 
    G  : text, 
    Hsh : hash_func, 
    Kb_da : symmetric_key, 
    Smw,Nov : channel(dy), 
    Pa : symmetric_key)

played_by D

const sec_GK_DA, sec_GK_DC : protocol_id

init State := 0

transition

1. State := 0
   / Rxv(C.{{Gbc'}_Kdc._Pw.C} => % receive: g^a, g^bc, g^dab
      / Gx' := exp(Gx,Pa) % g^xa
      / Gb := exp(Ga,Pa) % g^ab
      / Sx := Hsh(Pw,Gcdab) % send: g^abcd
      / witness(B,D,{{Gdab'}_Kdc._Pw.C}) % Checking group key with B
      / secret(GK,sec_GK_DC,{A,B}) % Checking group key with A

2. State := 1
   / Rxv(D.{{Gdb'}_Kbc._Pw.D} => % receive verifier from C
      / Gx' := exp(Gx,Pa) % g^xa
      / Gb := exp(Ga,Pa) % g^ab
      / Sx := Hsh(Pw,Gdab) % send: g^abcd
      / witness(C,D,{{Gdab'}_Kbc._Pw.D}) % Checking group key with C
      / secret(GK,sec_GK_DC,{B,D}) % Checking group key with D

3. State := 2
   / Rxv(D.{{Gdb'}_Kbc._Pw.D} => % receive: g^dab
      / Gx' := exp(Gx,Pa) % g^xa
      / Gb := exp(Ga,Pa) % g^ab
      / Sx := Hsh(Pw,Gdab) % send: g^abcd
      / witness(C,D,{{Gdab'}_Kbc._Pw.D}) % Checking group key with C
      / secret(GK,sec_GK_DC,{B,D}) % Checking group key with D

end role

---

Figure 10. Fourth Entity HLPSL Codes