Egyptian Informatics Journal 22 (2021) 245–255

Contents lists available at ScienceDirect

Egyptian Informatics Journal

journal homepage: www.sciencedirect.com

A proposed authentication and group-key distribution model for data warehouse signature, DWS framework

Mayada AlMeghari [a,b,*], Sanaa Taha [a], Hesham Elmahdy [a], Xuemin(Sherman) Shen [c]

[a] Cairo University, Egypt
[b] Palestine Technical College-Deir ElBalah, Palestine
[c] Department of Electrical and Computer Engineering, University of Waterloo, Canada

ARTICLE INFO

Article history:
Received 23 January 2020
Revised 7 June 2020
Accepted 17 September 2020
Available online 16 October 2020

Keywords:
Modular symmetric polynomial
Authentication
Group key
Data warehouse
Parallel computing
Middleware

ABSTRACT

In the virtual world, the authentication process is used to prove the identity for both of the server and the clients. Therefore, the authentication is an urgent issue for sharing data between users in the data warehouse systems. This paper presents the enhancement of Data Warehouse Signature (DWS) framework through proposing a novel authentication and group-key distribution model, based on the modular symmetric polynomials, to distribute a secure group-key between the manager and the chosen executors. To the best of our knowledge, we are the first to propose a group-key based on the symmetric polynomial and the modular arithmetic. This leads to increase the security level of the proposed model from t, where t is the symmetric polynomial degree, to $\sum_{i=1}^{c}\binom{c}{i} \times t$, where c is the number of executors in the DWS system. In this paper, enhancing the DWS achieves the high performance using a parallel computing in our middleware. The proposed model has the lowest cost of computation, communication and complexity overheads among the existing security models. Additionally, security analysis shows that the proposed model achieves the key security, prevents insider and outsider attacks, and provides forward and backward secrecy.

1. Introduction

A Data Warehouse (DW) is a collection of integrated different data sources used to support decision-making processes [1]. For this reason, there are a set of challenges associated with building data warehouses. These challenges are Data Quality, Understanding Data, Testing, Performance, Designing the DW, Cost, and Security and Privacy Concerns [2] [3]. Thus, it is necessary to consider data security in the DWs. The DW contains a huge amount of business and financial data that is considered as the main goal for hackers [4]. Therefore, it is important to keep these data in a secure state. For data security, there are mainly three security issues: confidentiality, integrity, and availability known by the acronym CIA [5]. These are the main topics used in the practical DW field. Many researchers have discussed the security issues in the DWs as presented in the work of [6]. One of these approaches is the Data Warehouse Signature (DWS) framework, as shown in Fig. 1. This framework achieves the three security issues, CIA, in a DW system [7], where the client requests a huge amount of data as Query Result Table, QRT from the DWServer. The data of the QRT should be encrypted before being sent to the client.

The DWSerever sends the QRT to a middleware after dividing it into blocks of records $(B_1, B_2, B_3, \ldots\ldots, B_m)$. In this middleware, the manager chooses a group of executors to be used in encrypting the blocks in parallel in order to save the encryption time. The manager sends these blocks along with the shared key (K_{SK}), used with the AES encryption algorithm, to the executors. However, regarding the above DWS framework communication, all transmitted blocks along with the key, K_{SK}, are sent in a clear-text, which in turn is a chance for any intermediate attacker to intercept the data before encrypting it. On one hand, the problem of sending the data blocks in a clear-text may be solved by compressing the QRT and dividing it to blocks, then keeping some blocks at the owner (DWServer/Client) (i.e., the owner is working as one of the executors). Therefore, the attacker could not be able to identify the

* Corresponding author.
E-mail addresses: mayada@teachers.org (M. AlMeghari), staha@fci-cu.edu.eg (S. Taha), ehesham@fci-cu.edu.eg (H. Elmahdy), sshen@uwaterloo.ca (Xuemin (Sherman) Shen).

Peer review under responsibility of Faculty of Computers and Information, Cairo University.

https://doi.org/10.1016/j.eij.2020.09.002
1110-8665/

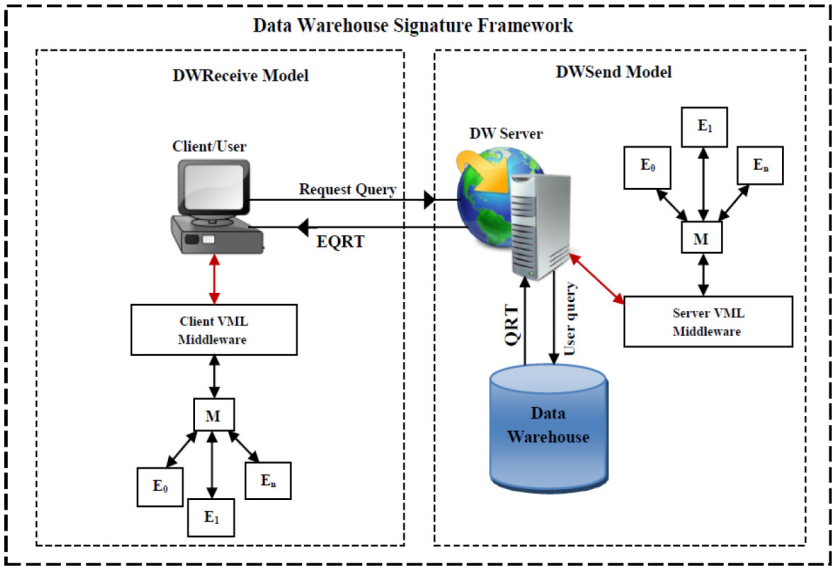

Fig. 1. The architecture of DWS framework.

whole blocks. For example, for 100 data blocks $(B_1, B_2, B_3, \ldots\ldots, B_{100})$, the manager chooses 10 executors, each of which encrypts 10 blocks. Those ten executors are in practice nine $(E_1, E_2, E_3, \ldots\ldots, E_9)$ and the owner is the tenth one. On the other hand, the problem of sending the shared key in a clear-text is still a challenge. According to Kerckhoffs principle [8], the security of the system depends on securing the shared keys.

In this paper, the problem of sending the shared key (K_{SK}) of encryption algorithm in a clear-text is solved by proposing a novel authentication and group-key distribution model. The goal of the model is to mutually authenticate the manager and the executors. Also, it generates a secure group-key among the manager and the chosen executors to encrypt/decrypt the data. Specifically, the contribution of this paper is in three folds:

- A novel authentication model between the manager and the executors in the DWS environment. This model thwarts any intermediate attacks, such as denial of service, replay, and man-in-the middle attacks.
- A group-key generation model that is distributed between the manager and the chosen group of executors employed to transmit the key, K_{SK} in a secure way. The model is based on the idea of modular symmetric polynomial and to the best of our knowledge, we are the first to use the modular symmetric polynomial in generating and distributing a group key. Therefore, we increase the achieved security level from t to $\sum_{i=1}^{c} \binom{c}{i} \times t$, where t is the symmetric polynomial degree. Note that, not all the executors know the generated key, only the executors chosen by the manager.
- A detailed security and performance analyses are performed to measure the security and performance of the proposed models. In the performance evaluation, the implementation of DWS achieves the high performance using the parallel computing through the VML middleware connecting 12 nodes, owner (DWServer/Client), manager, and 10 executors. This middleware is developed on Alchemi.Net framework to increase the security among the network nodes through the proposed model.

The rest of this paper is organized as follows. Section 2 reviews the related work. Section 3 presents some preliminaries. The system model is described in Section 4. In Section 5, an Authentication and Group-Key Distribution model is proposed. Section 6 offers the performance evaluation. Finally, the conclusion and future work are given in Section 7.

2. Related work

According to Kerckhoffs principle [8], the security of the system depends on securing the shared key among the network entities. Additionally, securing the shared keys in the DWS systems requires authenticating the clients and securing the key distribution.

On one hand, many authentication and group key protocols have been proposed. Velumadhava et al. [9] propose a group key exchange protocol based on Key Generation Center (KGC) and unique prime numbers. That KGC calculates the group pair for each member and then publishes these pairs to all group members. The protocol is efficient in the key computation, but it is not a support in dynamic and hierarchical groups. Additionally, in [10], a key distribution protocol is proposed to generate the group key only one time using random bits string and changing it using check bits. This approach applied the Von Neumann Corrector that takes pairs of bits from random bit streams, and used the hash function to get the final group key. This work is simple and non-complicated implementation, but it is not support the dynamic system as backward and forward secrecy. The approach of [11] uses a group authentication scheme to ensure the authentication of all the group members as devices using Paillier Threshold Cryptography, in which the group key is generated from the hash code of concatenating different parameters, such as device identity, gateway identity, random secret of device, time-stamps, and hash code of the system setup information. This approach requires a suitable computation time and it supports the dynamic system, but it needs a large storage size and more communications.

On the other hand, approaches related to authentication and group key generation based on polynomial functions are divided into three types: symmetric polynomial bases, pseudo-random generator bases, and linear congruence (modular arithmetic) bases.

First, for symmetric polynomial function, Banaie et al. [12] develop a scheme based on random symmetric polynomial functions, where the key is generated from computing the vector of shared matrices' values between two group members. The shared values are defined from the symmetric polynomials in the initial setup phase. This scheme achieves a high level of security and a full connectivity in the network, but it needs a huge storage for gener-

ating keys and matrix attributes. Additionally, Shi et al. [13] propose a group key distribution scheme based on solving polynomial linear equations and randomly generating matrixes. The KGC generates a unique column vector of matrix for creating the group key and then broadcasts it to all members. Although this approach supports a hierarchical structure and key confidentiality without using cryptographic operations, it requires a lot of communication and a storage cost.

Second, for pseudorandom generator based polynomial functions, In [14], they propose a polynomial-based key management scheme in which the group controller generates the polynomial function using the pseudorandom number and broadcasts it to all the members to calculate the group key. Although the scheme reduces storage and communication overheads, it does not support the authentication method. In [15], a group key distribution scheme is proposed based on Identity Based Broadcast Encryption (IBBE) and Bilinear maps as mathematical polynomial function. It randomly generates the public security parameters and the encapsulation header to establish the group key. This scheme achieves the confidentiality for group key using encryption with efficient time. But, it needs a large size of storage for many public parameters. In [16], the authors proposed a key distribution scheme in hierarchical structure using elliptic curve as a polynomial function of differential cryptography. They generate the group key by using a pseudorandom number and substituting it in an elliptic curve function. This approach ensures that the key collision is infeasible through using the elliptic curve function. However, this approach does not support the dynamic system for updating the key and it has a high computation overhead.

Finally, for modular polynomial function, the approach of [17] propose a group key distribution system for Enterprise Digital Rights Management (E-DRM). It generates the group key from different algorithms, such as Asmuth-Bloom secret sharing scheme, and Aryabhata Remainder Theorem (ART) for solving the linear congruence's equations. Although it achieves key freshness, confidentiality, authentication; and forward and backward secrecy, it requires a high computation time and a large storage. Liu et al. [18] develop a simple authenticated group key distribution protocol, which is based on the Chinese Remainder Theorem (CRT), to generate the group key by solving the linear congruence equations. Also, it used Asmuth-Bloom scheme to generate pairwise and coprime integer sequences at the KGC. Although it ensures confidentiality and authentication of key, it requires several transactions between the group members and KGC, which in turn causes high communication overheads. In [19], the authors develop an authenticated key transfer protocol based on secret sharing. This approach depends on sharing generation and secret reconstruction algorithms. The KGC randomly selects a group key, and then generates an interpolated polynomial function to pass it for all members. This approach ensures key freshness, confidentiality and authentication. But, it requires a high communication overhead and a large storage for all key confirmations.

3. Preliminaries

This section introduces the preliminaries of the proposed model, that are the traditional symmetric polynomials and the modular symmetric polynomials.

3.1. Symmetric polynomials

The symmetric polynomial is defined as "A symmetric polynomial is a polynomial if it is invariant to all permutations of the variables" [20](i.e., when interchanging any pair of the variables, the polynomial is still the same, F(x, y) = F(y, x)). The general form for generating a symmetric polynomial equation is $\sum_{i,j=0}^{t} a_{ij}x^i y^j$, where t is the degree of the polynomial. For instance, $a_{00} = 10, a_{11} = 1, a_{22} = 1, F(x,y) = x^2y^2 + xy + 10$ is a symmetric polynomial, while $F(x,y) = x^2y^2 + x + 10$ is not a symmetric polynomial because $F(y,x) = y^2x^2 + y + 10 \neq F(x,y)$ [21].

Symmetric polynomials are widely used for authentication and key distribution algorithms [12]. The Server with an integer identity,*S*,generates different keys for n clients with n different integer identities, $C_1, C_2, \ldots, C_n$ by first generates a symmetric polynomial $F(x,y)$, then keeps this polynomial secret and evaluates it for each client's ID individually, i.e,$F(C_1,y), F(C_2,y), \ldots\ldots\ldots\ldots\ldots F(C_n,y)$. The server then secretly sends the evaluated functions to each individual client, each of which evaluates the received function for the server's ID, to get $F(C_1,S)$ at client 1, $F(C_2,S)$ at client 2,$\ldots\ldots\ldots\ldots$ $F(C_n,S)$ at client n. For example, for 10 clients, each of which has an identity from 1 to 10, and a server's ID equals to 11, the server chooses the symmetric polynomial as $F(x,y) = x^3y^3 + x^2y^2$, Then this symmetric polynomial is evaluated for each client's ID, to get $F(1,y) = y^3 + y^2, F(2,y) = 8y^3 + 4y^2, \ldots\ldots\ldots, F(10,y) = 1000y^3 + 100y^2$ generates a different shared key with each client individually. Finally, after receiving each individual evaluated function, each client evaluates the received evaluated function for the server's ID. for instance, client 5 receives $F(5,y) = 125y^3 + 25y^2$ from the server and then evaluates it for the server's id, to get $F(5,11) = 125 * (11)^3 + 25 * (11)^2 = 169,400$. Therefore, 169,400 is the shared key between the server and client 5. Note that each client has different shared key with the server. For instance, client 4 has a shared key $F(4,11) = 64 * (11)^3 + 16 * (11)^2 = 87,120$. However, it is proven that those kind of polynomials supports only t security level, where t is the degree of the polynomial. Therefore, for t-degree symmetric polynomial, a collusion of t users can easily disclose the symmetric polynomial. For the previous example, if at least 3 clients collude with each others,they can regenerate the secret symmetric polynomial that is generated by the server, $F(x,y)$, and hence disclose all system's keys. Therefore, to increase the security level of the proposed model, we introduce the modular symmetric polynomial to be used for generating not only a pairwise shared key, but also a group-key among the server and a group of clients.

3.2. Modular symmetric polynomial

The modular symmetric polynomial [12] is a symmetric polynomial defined over a finite field F_m, for instance $F(x,y) = x^3y^3 + x^2y^2 \bmod m$. This type of polynomials is used to employ it when proposing a group key between the server and a group of clients, as follows: the server first generates the modular symmetric polynomial $F(x,y)$, evaluates for each client's ID, and transmits the evaluated functions to each individual client. By this step, each client creates a shared key with the server, as depicted in the previous section. However, to create a group key K_{gr} between the server and a group of the clients, the server adds the following equations and solves them for the values of $K_{gr}, m_1, m_2, \ldots\ldots\ldots m_n$:

$$\begin{aligned} K_{gr} &\equiv F(C_1,S) \bmod m_1 \\ K_{gr} &\equiv F(C_2,S) \bmod m_2 \\ &\ldots\ldots \\ &\ldots.. \\ &\ldots.. \\ K_{gr} &\equiv F(C_n,S) \bmod m_n \end{aligned} \tag{1}$$

Since those are n equations in $n+1$ unknowns, the server assumes a value for m_1 and calculates the values of $K_{gr}, m_2, \ldots\ldots, m_n$ accordingly. Finally, the server securely

transmits each modular value of $(m_1, m_2, \ldots\ldots m_n)$ to each client individually. Therefore, each client uses its own modular value m_i to calculate the same group key K_{gr}.

4. System models

In this section, the DWS system models are explained, including DWS framework, threat and trust models, and enhancing of DWS.

4.1. DWS framework

The Data Warehouse Signature, DWS framework is designed by using the Client-Server model. For the data flow between the DWServer and the Client, there are two process models in the DWS framework [7]. These models are DWSend model and DWReceive model that use View Manager Layer (VML) middleware to achieve high performance, as shown in Fig. 1. The high performance is the main idea to achieve data availability in the DW system by eliminating the query response timed-out. In the two DWS models(DWSend and DWReceive), the security processes are executed in the parallel computing using the VML middleware to reach the high performance. The VML middleware is supported by a.NET-based grid computing framework called Alchemi [22]. This middleware uses the network resources, such as the shared data and the CPU to become a virtual supercomputer. The VML middleware consists of three elements of the network nodes, which are the owner (DWServer or Client), the manager and the executors. Each of these elements performs a set of functions and operations concatenated together in distributed systems applied in the network architecture [23].

The two models is used to ensure the three security issues, such as Confidentiality, Integrity, and Availability (CIA). The DWSend model performs two main security processes at the DWServer side, the encryption process to ensure data confidentiality and the hash computing process to ensure data integrity in the DW systems. In this model, the owner (DWServer) in the VML middleware separates the Query Result Table(QRT) into Block Objects (BOs) with a fixed size of number of records. These blocks are sent to the manager to distribute them into a set of executors. The manager is the main node in the VML middleware, which distributes these blocks to a set of chosen executors. Also, this manager returns the computed hash value of each block to the owner to get the final hash code value. Then, the manager recollects all encrypted blocks, EBOs to become EQRT to be sent into the owner. The executor is the worker that performs two security processes on its block: the encryption process using AES algorithm with the Shared Key (K_{SK}) generated by using Diffie-Hellman (D-H) algorithm and the hash computing process using SHA-1 algorithm. After the executor finishes these processes, it sends its block with the hash code value to the manager. The output of this model includes the Encrypted Query Result Table(EQRT) and the encrypted final hash code, which are sent to client with the shared key, KSK. This key is distributed in a secure way to participate it with the client for decryption process at the DWReceive model.

The DWReceive model performs three security processes at the Client side, the hash computing process, the hash comparing process, and the decryption process using the VML middleware. Through the VML middleware, the owner (Client) divides the EQRT into Encrypted Block Objects, EBOs of the same size and then sends them to the manager. The manager distributes these blocks to a set of chosen executors. Each of these executors computes the hash code value for its encrypted block to get the final hash code of the EQRT at the owner to compare it with the received final hash code. Also, each executor decrypts its encrypted block object, EBO to get the decrypted BO. Then, the manager recollects all decrypted blocks, to become QRT to be sent into the owner (Client). In the traditional systems, the security processes are executed serially. However, the serial execution leads to network overhead and query response timed-out. In the DWS framework, executing the security processes on a query result of a huge number of records as blocks using parallel computing saves more time than serial computing.

4.2. Threat and trust models

Threat modeling is defined as "it is a process of a hypothetical attacker's point of view by which the possible threats can be identified, enumerated, and prioritized" [24]. The DWS framework is analyzed through different threat models to trust it in DW organizations [25]. From the evaluation of DWS threat analysis, there are a set of threats available for attacking process. These threats are classified in two types, insider attacks and outsider attacks.

1. Insider Attacks: Colluder attacks in which a group of executors collude to getter in order to reveal the AES key and the data of the sent blocks. Moreover, the executors connected in the VML middleware can exploit their authorities to damage or capture the encryption key and the blocks, which are sent by the manager.
2. Outsider Attacks: This attack refers to anyone who deceives in order to join into the VML middleware and can impersonate the executor node. The threat analysis of the DWS approach shows that there are three types of outsider attack.
   - Man-in-the Middle Attack: The attacker can snoop in the network to get the key K_{SK}. This key is used in the encryption/decryption processes with AES algorithm. Also, this type of attacker can eavesdrop transmitted messages between the manager and the executors to reach all of the blocks, and then obtains the QRT.
   - Denial of Service Attack (DoS): The connecting of each node in the VML middleware depends on the network structure as IP and MAC address. Thus, the attacker may send a set of requests from many different sources to be connected in the VML. This leads to a traffic flood and disrupt the system as denial of services attacks.
   - Replay Attack: The attacker can obtain a copy of the data of the blocks or the shared key K_{SK} of AES algorithm which are sent from the manager to the executors and can replay them later. This leads to misdirect the manager and fool it into believing the attacker is in fact an executor.

The DWS framework consists of two trusted sets distributed in its data flow to ensure the security issues in DW systems [7]. The first trusted set is the main element of the web-based system called DWServer, in which it is trusted to provide the query request services to the clients. While the second trusted set is the centralized element of the VML middleware called manager, which schedules the jobs sent by the owner to execute them in a parallel computing.

4.3. Enhancing of DWS framework

Ensuring the security issues, CIA was the first consideration in the DWS framework at the two models (DWSend and DWReceive). The assumption of the VML middleware is worked in isolated and trusted network has not been supported by some DW organizations. Enhancing the DWS framework is a must in order to protect the transmitted data of the QRT. In the DWS, the data of QRT was sent as a clear text to the executors in the VML middleware. To increase the security in the DWS approach, the data will be sent as ambiguous data to the executors. Therefore, in the DWSend model, the owner (DWServer) will compress the QRT before

dividing it into blocks to be Compressed QRT (CQRT). The DWServer sends the Compressed Block Objects (CBO_1, CBO_2, CBO_{n-1}) to the manager with keeping the last block CBO_n, as shown in Fig. 2. In the other model, DWReceive, the owner (Client) will decompress the CQRT after the VML middleware finishes the collection of CBOs (CBO_1, CBO_2, CBO_{n-1}) with its block (CBO_n). In the two models, the owner is working as one of the executors. The manager transmits the compressed blocks to some randomly chosen executors, rather than all executors, each of which performs two processes on its block: encryption/decryption process and computing hash process. In this case, the compression process solves the problem of the insider attack, which will get a block of unclear data as the compressed block. The compression process also reduces the size of the huge QRT, saving time during sending it. In addition, DW organizations may use different nodes in their work environments. This state requires checking the identity of any node connected with DW organizations. In non-trusted network, many attacks can occur from the failure state of the authentication process. So, the VML middleware elements (owner, manger and executor) need to ensure the authentication between them. The shared key K_{SK} was distributed securely using (D-H) algorithm between the DWServer and the Client, but this key, K_{SK} was sent among the middleware nodes from the owner to the manager then to the executors as a clear. Depending on the threat analysis model, there is a need to propose an Authentication and Group-Key Distribution model in the middleware of DWS framewok, as shown in Fig. 2. This model will ensure the authentication between the VML middleware nodes to prevent outsider attacks, such as replay attack, and man-in-the middle attack. Also, The proposed model will generate a group key employed to encrypt the key (K_{SK}) of the AES encryption algorithm. Thus, the K_{SK} is not sent as clear to the executors. This enhancement in the DWS also prevents any attacker to get the complete compressed blocks by keeping one block at the owner without sending it. Finally, in DWS implementation, we have enhanced the VML middleware supported by Alchemi.Net Desktop Enterprise [22] in order to implement the proposed authentication and group-key distribution model.

5. Authentication and group-key distribution model

In the digital computing and the internet communication, the authentication process is defined as "it is the assurance that the communicating entity is the one that it claims to be" [26]. So, the authentication protocol is simulating with a zero-knowledge proof algorithm to check the identity of each communication entity. This means that the one party (Prover) can prove to contact the other party (Verifier) that a certain argument is true without disclosing any information [27]. Ensuring the authentication in the DW systems is utmost important during the data flow over the internet network. The proposed authentication and group-key distribution model in the DWS includes three phases described in Fig. 3: initialization, authentication, and group-key generation phases.

5.1. Initialization phase

In the DWS framework, the manager periodically transmits its certificate to all executors in the system. The certificate is generated using X.509 certificate authentication [28], and it contains this manager's public key, PK_m. Additionally, the manager chooses a symmetric polynomial of degree t, F(x,y), and keeps it secret. As an initialization, every executor, which likes to work with the manager, sends a connect-request message, as depicted in (2) to the manager encrypted with the PK_m by using strong public key encryption algorithm such as Elgamal. The connect-request message includes the executor identity, ID, a random number, $nonce_1$, and a time stamp, ts_1.

$$Conn - req : E_{PK_m}(executor_{ID}||nonce_1||ts_1||executor_{Cert}) \quad (2)$$

where $E_{PK_m}()$ is the encrypted message using the key, PK_m, $nonce_1$ is a random number generated by the executor, and ts_1 is a time stamp at the executor.

In this stage, the reason of encrypting the message in (2) is to hide the executor identity. However, there is no explicit signature is done, because no executor has created a signature key yet. Instead, we add the nonce and ts to prevent replay attack. Additionally, adding the executor's certificate to the connect-request message, before encrypting it, leads to thwarting DoS attack.

5.2. Authentication phase

In this phase, the manager authenticates the executor that transmitted a connect-request as the follow steps:

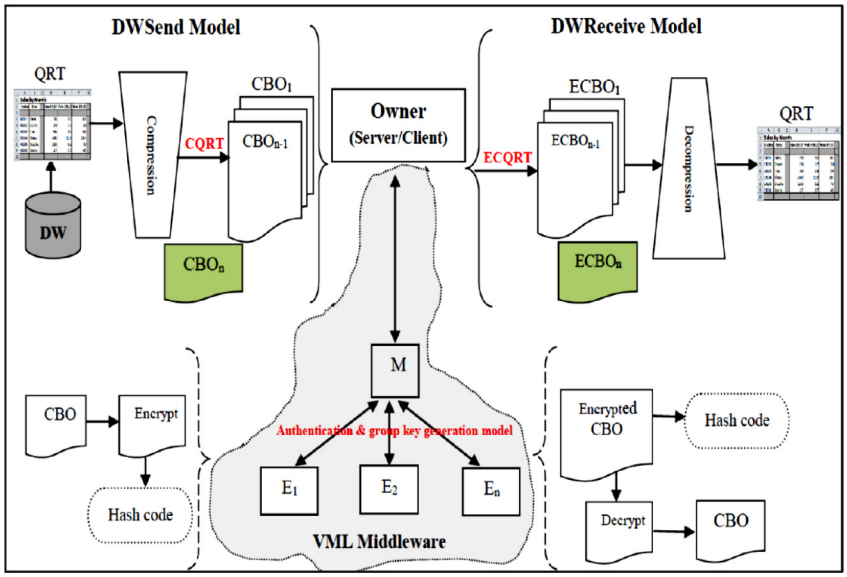

Fig. 2. The architecture of DWS framework after enhancing.

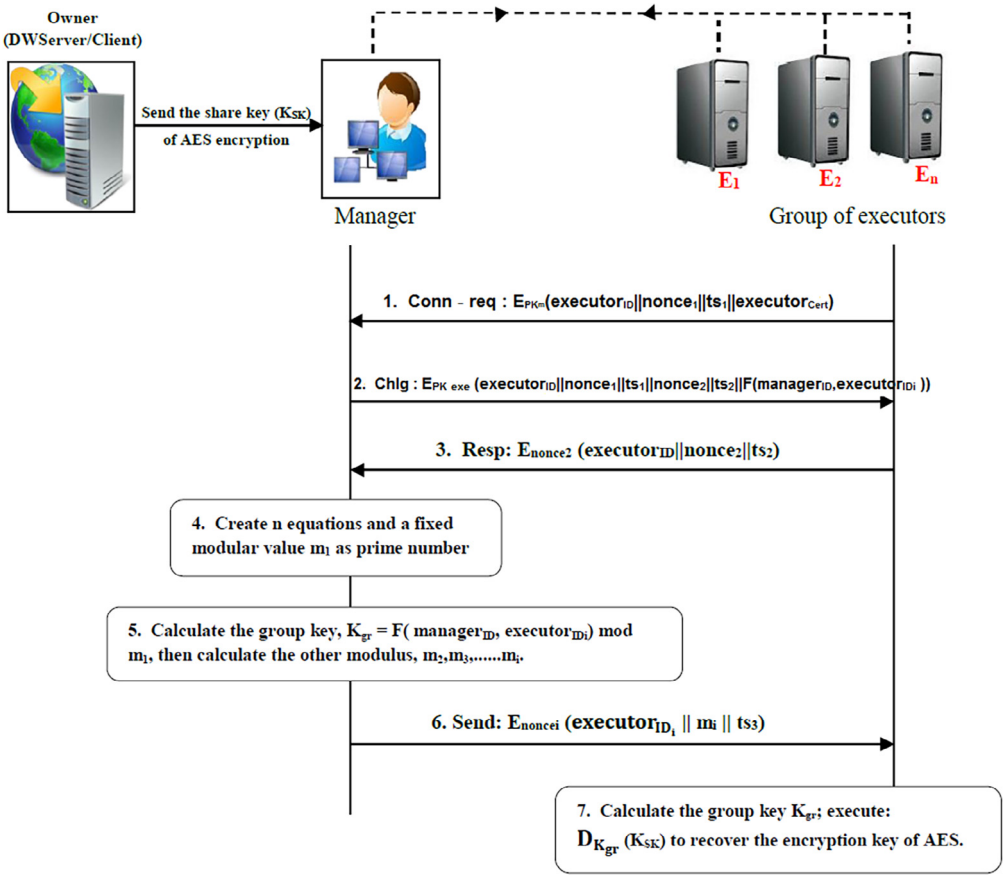

Fig. 3. The phases of proposed model.

1. Upon receiving the connect request, the manager decrypts the request using its private key. Then, the manager sends a challenge message to the executor. This message includes the received connect-request and another random number, $nonce_2$, and a new time stamp, ts_2, as follows:

$$Chlg : E_{PK_{exe}}(executor_{ID}||nonce_1||ts_1|| nonce_2||ts_2||F(manager_{ID}, executor_{ID_i})) \quad (3)$$

where, $F(manager_{ID}, executor_{ID_i})$ is an evaluation of the symmetric polynomial that the manager creates to be used in creating a group key, K_{gr}, as depicted in the next phase. This message is encrypted using the executor public key, PK_{exe} to be sent to the executor.
2. The executor decrypts the challenge message using its created $nonce_1$ and replies with the response message, Resp, as follows:

$$Resp : E_{nonce_2}(executor_{ID}||nonce_2||ts_2) \quad (4)$$

3. Upon receiving the response message, the manager decrypts the message using $nonce_2$, then compares the values of $nonce_2$ and ts_2 at the response and challenge messages and finally authenticate the executor if the values are identical.
4. If the manager authenticates the executor, it creates a row for this executor in its database including its ID, $executor_{ID}$, and a shared key between the manager and the executer, which is $nonce_1$. Therefore, for subsequent communications between the manager and this executor, the manager uses $nonce_1$ as a shared key between them.

5.3. Group key generation phase

First, the DWServer compresses the QRT to a number of records and then divides them into compressed block objects, $CBO_1, CBO_2, CBO_3, \ldots\ldots CBO_n$. All blocks except the last one are transmitted to the manager. After authenticating the executors, the manager randomly chooses a group of executors to be used in encrypting the data of the CQRT. For the transmitted blocks, the manager needs to send the shared key, K_{SK} ,of AES algorithm to those chosen authenticated executors. However, the key should not be transmitted as a clear text because it would be an easy job for the eavesdropper to get the encryption key. Therefore, the chosen executors along with the manager can generate a group key, K_{gr} by applying the following steps:

1. After authenticating the executor using the previous phase, the manager chooses a secure symmetric polynomial, $F(x, y)$, and evaluates it for its ID and the executor's ID, $F(manager_{ID}, executor_{ID})$.
2. If the manager already has a defined symmetric polynomial for another authenticated executor, the manager evaluates this defined polynomial for this new executor.
3. To create the group key, K_{gr} for n executors, the manager creates the n equations, as depicted in (1), assumes a fixed prime value for the first modulus, m_1, then calculates the other modulus, $m_2, m_3, \ldots\ldots m_n$, and the group-key, K_{gr}.
4. The manager then transmits the modular values, $m_1, m_2, \ldots\ldots m_n$ each of which to the intended executor after encrypting it using individual executor's key, $nonce_i$.

Table 1
The experiments average execution time before and after DWS enhancing for DWSend model.

Number of chosen executors	Average execution time (*sec*) before enhancing						Average execution time (*sec*) after enhancing					
	5KR	10KR	50KR	100KR	500KR	1MR	5KR	10KR	50KR	100KR	500KR	1MR
1	2.130	4.504	11.741	25.989	87.314	138.797	1.564	2.510	4.306	8.627	22.309	45.573
2	1.835	3.782	10.566	20.990	74.782	116.023	1.355	2.044	3.894	7.764	19.240	38.322
3	1.847	3.793	8.750	17.802	66.963	105.424	1.360	2.078	3.908	5.660	16.096	32.839
4	1.840	3.813	8.765	17.948	57.163	96.604	1.357	2.079	3.888	5.689	14.266	28.438
5	1.848	3.813	8.768	17.995	57.481	88.208	1.366	2.082	3.914	5.898	14.365	28.620
6	1.841	3.808	8.770	17.989	57.874	88.313	1.365	2.071	3.916	5.758	15.123	29.122
7	1.828	3.813	8.794	17.963	58.003	88.289	1.369	2.084	3.892	5.701	14.914	29.046

Table 2
The experiments average execution time before and after DWS enhancing for DWReceive model.

Number of chosen executors	Average execution time (*sec*) before enhancing						Average execution time (*sec*) after enhancing					
	5KR	10KR	50KR	100KR	500KR	1MR	5KR	10KR	50KR	100KR	500KR	1MR
1	3.212	5.305	17.118	30.236	107.792	214.465	2.390	2.783	4.620	9.033	26.998	51.958
2	2.552	4.236	15.209	25.340	92.129	180.239	2.006	2.216	4.180	7.539	23.946	46.148
3	2.592	4.303	10.167	18.797	88.617	167.917	2.013	2.219	4.188	5.693	18.333	42.484
4	2.578	4.318	10.178	18.830	77.053	161.198	2.026	2.227	4.186	5.701	18.346	35.758
5	2.557	4.294	10.192	18.951	77.529	145.449	2.025	2.210	4.208	5.696	18.377	35.838
6	2.559	4.279	10.386	18.652	77.580	145.659	2.020	2.221	4.190	5.714	18.401	35.810
7	2.599	4.280	10.142	18.906	77.720	146.089	2.013	2.225	4.184	5.702	18.352	35.771

5. Using its modular value m_i, each executor can recalculate the group key by applying Eq. (5) with the received evaluated symmetric polynomial, $F(manager_{ID}, executor_{ID_i})$, and the modular value, m_i.

$$K_{gr} \equiv F(manager_{ID}, executor_{ID_i}) \bmod m_i, \quad for\ i = 1, \ldots\ldots\ldots n \tag{5}$$

Note that only the manager knows the secret symmetric polynomial, $F(manager_{ID}, executor_{ID_i})$. Each of the chosen executors then decrypts the encrypted shared key, K_{SK}, using the calculated group key to recover the shared key of AES algorithm. Then, these executors can encrypt/decrypt the data of the CQRT by K_{SK}. Thus, the authentication and group-key distribution algorithm solves the problem of sending the shared key (K_{SK}) in a clear-text.

6. Performance evaluation

The components in the DWS are connected with the VML middleware that is developed on Alchemi 1.0.4.Net Framework 2 [22] to reduce the potential attacks. So, the environment needs to divide the network nodes into three types: the owner (DWServer or Client), the manger, and the executors. In the DWS approach, the experiments are conducted using Windows 7(SP1)-64bit, CPU Intel core (3.30 GHz) with RAM (16 GB) for all VML nodes. The experimental network data flow is in a star network topology with a cluster structure. The programming language used in developing the DWS approach is C# with Microsoft Visual Studio 2013. The data set used is Sales DW as the QRT required from Microsoft SQL Server Management Studio 2008. We used Crypto++ 5.6.5 Benchmark[1] to measure the computation time for the proposed model in DWS framework. The experimental environment has a regular equipment available in DW organizations.

6.1. High performance for parallel computing

The DWS framework included two models, which are DWSend and DWReceive, in which two execution processes are performed in parallel: encryption/decryption process and hash computing process. These processes can be executed on the executors that are connected in the VML middleware. So, the execution of a huge amount of QRT blocks in parallel-computing saves more time than that in serial-computing DWS[23]. In DWS experiments, the QRT sizes are distributed as ordered 5KR (1.037 MB), 10KR (2.043 MB), 50KR (10.672 MB), 100KR (20.387 MB), 500KR (102.336 MB), 1MR (204.406 MB). Here, the 1KR is equal to 1000 records. Also, the number of chosen executors is distributed from one to seven executors in the VML middleware.

This section presents the evaluation of DWS results in two period states. First, the results of DWS with increasing the executor's resource powers. Then, it is compared to the results presented in [23] that used a different environment with lower executor's power resources. Second, the results of DWS between before enhancing and after the proposed enhancing.

With increased the executor's power resources (CPU speed and RAM), the Average Execution Time-Before Enhancing (AET-BE) for executing the QRT sizes has decreased. For DWSend model, Table 1 shows that the result of AET-BE reaches to a higher performance than the result presented in [23]. For example, when executing the QRT of 1MR, the Percentage of High Performance (%HP) for the AET was 26% [23] to increase into 36% for AET-BE. Also, for DWReceive model, the %HP of AET-BE presented in Table 2 has increased comparing to the result of [23]. For example, when executing the EQRT of 1MR, the %HP of AET was 29% [23] to increase into 32% for AET-BE. This increase in the %HP of AET-BE is from the increase of the executor's power resources.

For enhancing DWS, the compression process is applied for the QRT in the DWSend model. The compression process is performed using GZIP algorithm to generate the CQRT as shown in Fig. 2. In DWSend model, the owner (DWServer) divides the CQRT to a number of compressed blocks. Then, these blocks are sent to the manager without sending the last block, which is processed in the owner machine. The manager schedules the blocks to be sent to the chosen executors as a load balance for each one. All executors perform two processes, encryption process and hash computing process on their compressed blocks in parallel computing.

The Average Execution Time-After Enhancing (AET-AE) for executing the compressed QRT has decreased because the compression process reduces the QRT size. For instance, when executing

[1] https://www.cryptopp.com/benchmarks.html.

the QRT size of 1MR, the AET is equal to 116.023 s at 2 executors before enhancing to decrease it into 38.322 s after enhancing, as shown in Table 1. For parallel computing, the %HP of AET-AE is estimated from 15% to 37%. By analyzing the AET's values with increasing the number of executors for 1MR size, we find that the best number of executors before enhancing is 5 executors. But, after enhancing, the best number of executors is 4 executors, as shown in Fig. 4. Thus, this enhancing reduces the execution time and the enough number of chosen executors to reach high performance.

In DWReceive model, the owner (Client) gets the encrypted compressed QRT with its hash code from the DWServer. Then,this owner divides it into a number of encrypted compressed blocks. These blocks are sent to the VML middleware while keeping the last block. In this middleware, the manager schedules and sends those blocks to the chosen executors in a load balance way. Each of the chosen executors performs two processes, hash computing process and decryption process on its block. Then, the owner compares between the two hash codes (received and computed). If the two hash codes are the same, the owner performs a decompression process to obtain the QRT.

The result of enhancing the DWReceive model shows that the AET has decreased comparing to the AET before enhancing, as shown in Table 2. For instance, the EQRT size of 500KR, the AET is equal to 88.617 s. at 3 executors before enhancing to decrease into 18.333 s. after enhancing. For parallel computing, the %HP of AET-AE is estimated from 11% to 32%. As in the DWsend model, we find that the best number of chosen executors for executing the EQRT of 500KR before enhancing is 4 executors. But, after enhancing, the best number of chosen executors is 3 executors for executing the EQRT of the same size, as shown in Fig. 5.

Finally, the high performance of the DWS enhancing increases to reach the constant values with increased the number of chosen executors. For example, when executing the QRT size of 100KR, the AET-AE is equal to 5.660 s. at 3 executors. After increased the number of executors from 4 to 7, the AET-AE takes the same time(i.e., the AET-AE has not decreased), as shown in Table 1. These results show that the performance doesn't reach to a complete save of time for parallel computing. The reason of that is the cost of using parallel computing is determined by the two time factors(transmission time, separation and collection time), which resist to achieve the high performance completely. The time factors lead to two challenges through separating the QRT to blocks. First, dividing the QRT to small blocks (more executors) takes a lot of time in the collection, although the transmission time for these blocks is a small. Second, dividing the QRT to large blocks (fewer executors) takes a little time in the collection, although the transmission time is a lot. Therefore, the load balance for the number of executors solves these challenges of time factors.

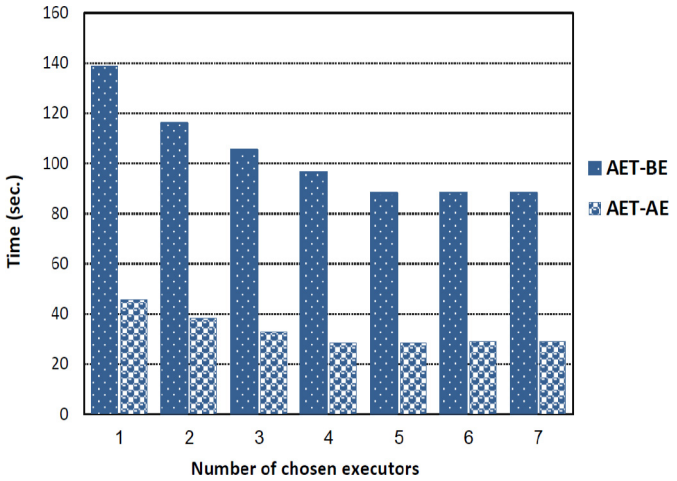

Fig. 4. The average execution time for executing the QRT size of 1MR before and after DWS enhancing for DWSend model.

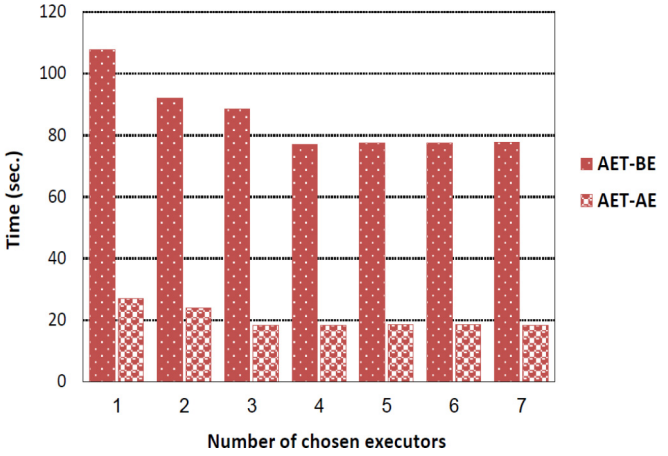

Fig. 5. The average execution time for executing the EQRT size of 500KR before and after DWS enhancing for DWReceive model.

6.2. Computation, communication and complexity overheads

This section presents the performance evaluation of the proposed model comparing to other authentication and group key generation models. Table 3 shows the comparison among these models in terms of computation, communication, and complexity overheads. The proposed model achieves the smallest computation overhead compared to the other models, because it needs 3 symmetric encryption/decryption operations for authentication and group key generation with either the executor public key or $nonce_i$. The model of [17] needs 3 hash code operations which are a one-way hash function as a collision-free for registration of members and two one-way hash functions of broadcasted messages and the group key to ensure group key authentication during group key generation and distribution process. This model also needs 2 ART (Aryabhata Remainder Theorem) operations used to generate a number for each group based on secret pairs. The model of [18] needs 3 hash code operations which are a one-way hash function as a collision-free for authentication message, and two one-way hash functions for group key distribution and mutual key confirmation. This model also needs 2 CRT (Chinese Remainder Theorem) operations to generate an integer based on private shares. Thus, these models [17] [18] achieve computation overhead higher than that achieved in the proposed model. The model of [19] needs 2 hash code operations as one-way hash output with the secret group key and all members' random challenges as input.

Table 3
A comparison of computation, communication and complexity overheads.

Model	Computation Overhead (*msec*)	Communication Overhead (*bit*)	Complexity Overhead
Liu et al. (2015)[17]	0.223028682	3840	$O(nb^2)$
Liu et al. (2014) [18]	0.401028682	5120	$O(n^2b^2)$
Harn and Lin (2010)[19]	0.000585893	3072	$O(n^2b^2)$
Golumbeanu (2017)[16]	0.580964946	1536	$O(n^2b^2)$
Proposed model	0.000349331	1200	$O(nb^2)$

This model also needs one symmetric encryption/decryption operation to ensure the confidentiality of group key, so, it achieves as computation overhead as our proposed model. The model of [16] has the highest computation time, because it requires 4 hash codes of the SHA-1 hash function, 2 asymmetric encryption/decryption operations for the messages with the session key or the public key, and 2 elliptic curve point addition operations.

Fig. 6 shows the comparison of computation overhead for each model. We calculate the computation time of those models through Crypto++ 5.6.5 Benchmark, which we recommend AES/RNG for symmetric operation; RSA 1024 for asymmetric operation; and SHA-1 for hash operation. Also, we use the execution time of ART and CRT operations [29], and the execution time of elliptic curve point addition operation[30].

For communication overhead, as shown in Fig. 7, our proposed model has the lowest communication overhead comparing to the previous models. The reason of that is the proposed model requires a small number of parameters to be sent between the manager and the chosen executors. While the model of [18] requires high communication overhead to send a large number of parameters used in the group key generation and distribution.Here, the communication overhead is measured by the number of parameters as bits in each message between two parties.

For complexity overhead, in the DWS approach, the proposed model uses the modular symmetric polynomial for generating the group key, K_{gr} . The modular symmetric polynomial consists of a set of digit bit operations, b, such as multiplication, addition, and modular. In the proposed model, the manager generates the K_{gr} using these operations, then calculates and transmits the modular values, m_i to the number of chosen executors, n. Therefore, the cost of complexity overhead is equal to $O(nb^2)$. This model achieves less complexity overhead than the other models [18,19,16], which require the complexity overhead $O(n^2b^2)$. The model of [17] requires the same complexity overhead as the proposed model.

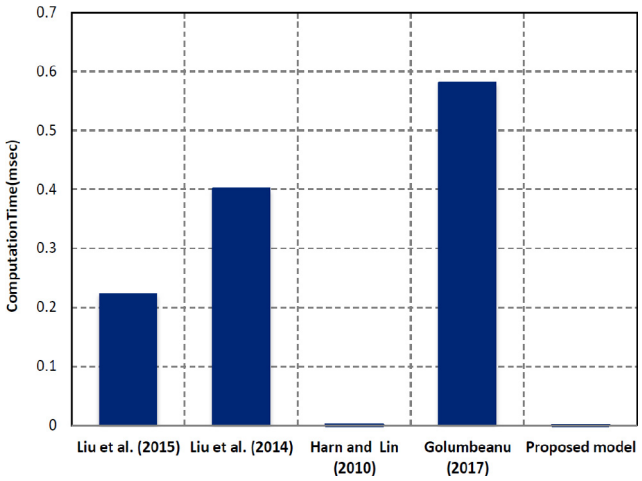

Fig. 6. Computation overhead for proposed model and existing models.

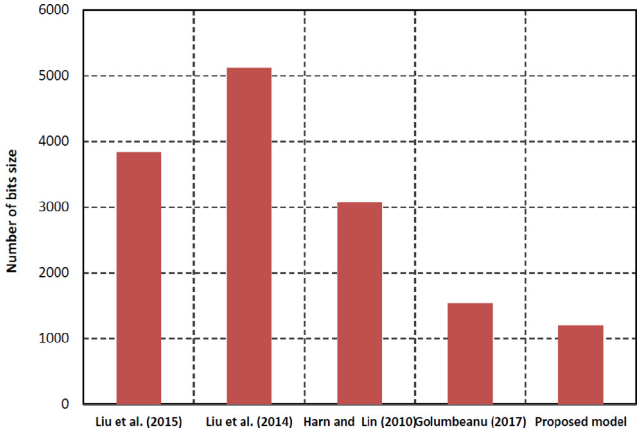

Fig. 7. Communication overhead for proposed model and existing models.

6.3. Security analysis

In this subsection, we prove that the proposed model for the DWS framework achieves the following security requirements:

1. Achieve key freshness
2. Achieve key confidentiality
3. Achieve key authentication
4. Prevent insider attacks
5. Prevent outsider attacks
6. Provide forward and backward secrecy

6.3.1. Key freshness, confidentiality, and authentication

Theorem$_1$: The proposed model achieves key freshness, confidentiality, and authentication.

- **Key freshness**: The group key is generated at the manager, based on the chosen group of executors. Additionally, the group key has periodically changed with every QRT transaction that is needed to be encrypted by a specific group of executors. Note that the chosen executors are selected randomly by the manager. Therefore, the key freshness is achieved.
- **Key confidentiality**: Assume adversary *A*, which aims to break the group key confidentiality (i.e., reveal the group key). The attacker *A* could eavesdrop the communicated messages between the manager and the executors. However, those messages are encrypted using the manager public key, PK_m and the $nonce_1, nonce_2$ as symmetric keys. Additionally, the group key is never sent inside the encrypted messages, instead, the evaluated symmetric polynomial function is sent. However, according to *Theorem*$_2$, even if the attacker knows the polynomial function, they are still could not able to identify the key. Therefore, the confidentiality of the key is achieved.
- **Key authentication**: It is achieved using the challenge-response way, where the manager randomly chooses the executors whom receive the challenge message. If the executor successfully decrypts the challenge message and encrypts the response message using $nonce_2$, which is sent in the challenge, then the executor is authenticated and in turn, the key is authenticated as well.

6.3.2. Insider attacks

Considering colluders as insider attackers, the proposed model thwarts insider attacks because it increases the security level of the authentication from t to $\sum_{i=1}^{c}\binom{c}{i} \times t$ according to the following theorem. Therefore, the number of colluders who are able to disclose the symmetric polynomial has also increased accordingly.

Theorem$_2$: Using the idea of modular symmetric polynomial, the security level of the proposed model has increased from t to $\sum_{i=1}^{c}\binom{c}{i} \times t$. Where t is the degree of the polynomial and c is the number of colluders (executors). In the proposed model, we modify the idea of symmetric polynomial to modular symmetric polynomial. Additionally, the manager randomly chooses the executors, who will perform the encryption process and the hash computing process for their CBOs. Therefore, if a group of executors colludes to reveal the secret symmetric polynomial, they need first to know

the executors who are chosen from the manager. However, the shared executors would never know each other, because every executor receives different modular value, m_i , where i is the number of executors. Therefore, the colluded executors need to try all possible combination of the shared executors together. So, the number of combination is $\sum_{i=1}^{c} \binom{c}{i}$. Returning backs to the original symmetric polynomial idea, the achieved security level is t with a symmetric polynomial of degree t. Therefore, in our case, the security level will be increased to be:

$$S = \sum_{i=1}^{c} \binom{c}{i} \times t \tag{6}$$

6.3.3. Outsider attacks

Theorem$_3$: The proposed authentication and group-key distribution model thwarts outsider attacks, include DoS, MITM, and Replay attacks.

- For Denial of Service (DoS) attack, the attacker needs to include the valid executor ID in the connect-request message. However, this attacker is not an authorized executor. Therefore, when an attacker sends a fake connect-request message to the manager, the manager easily identifies this attacker when decrypts the message and checks the included executor ID. We consider that the manager processing resources is huge, so it could afford decrypting DoS attacks' messages.
- For a Man-in-the Middle (MITM) attack, the attacker is located between the manager and the executors and tries to successfully authenticate himself. This kind of attacker cannot change the content of the messages, because they are encrypted using the manager public key $PK_m, nonce_1$, and $nonce_2$. Therefore, the man-in-the middle attacks is prevented.
- For replay attack, this attack is also prevented, because the proposed model uses the random numbers, nonce, and times-tamp.

6.3.4. Forward and backward secrecy

Any secure group key distribution model should have forward secrecy and backward secrecy. Forward secrecy ensures that it cannot discover any previous group key by exposing the groups current key. Backward secrecy ensures that the current group key cannot be determined from a previous group key that has been disclosed. According to the following theorem, both forward and backward secrecy are achieved since the attacker cannot disclose any group-keys in any previous or subsequent sessions even if the current session's group-key is disclosed.

Theorem$_4$: The group-key value of the current request is independent of previous and next requests.

Every request sent from the owner application to the VML middleware, the manager randomly chooses the group of executors to share the group key with them. The chosen group of executors only calculates the group-key. Therefore, for request r, the manager may select the executors {1, 2, 3}, however, in request r + 1, the chosen group of executors will differ to {5, 9, 12} and so on. Therefore, the generated group-key is different in every request sent to the VML middleware.

7. Conclusions and future work

In this paper, based on an enhancement of the DWS approach and the modular symmetric polynomial, we have proposed an authentication and group-key distribution model to resist the security attack as well as to increase the performance. The proposed model aims to generate a group-key to be distributed between the manger and the chosen group of executors in the VML middleware. These executors have performed two processes, the encryption process and the hash computing process on their CBOs in a parallel computing. Therefore, the enhancement of DWS has achieved the high performance as presented in the experimental results. Generally, the high performance reaches limited values when the number of executors increase, as a result of using the time factors with the parallel computing. Additionally, the proposed model achieves the key freshness, confidentiality, and authentication. It also thwarts the insider and outsider attackers by increasing the security level from t to $\sum_{i=1}^{c} \binom{c}{i} \times t$, where t is the symmetric polynomial degree, and c is the number of chosen executors.

In the future, the proposed model is aimed to be extended to create an owner certificate. This certificate is needed to get the authorization services through the VML middleware. In addition, we will protect the data stored in the manager, including the group key, from any possible attack on that manger. Moreover, a migration from the parallel computing to the cloud computing will be applied to our secure DWS framework along with the proposed authentication and group-key distribution model. With a cloud computing, an increase is expected in the performance of the DWS, comparing with that implemented in the parallel computing.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Chowdhury R, Chatterjee P, Mitra P, Roy O. Design and implementation of security mechanism for data warehouse performance enhancement using two tier user authentication techniques. Int J Innov Res Sci Eng Technol 2014;3(6):165–72.

[2] Sophia Tutorial, Trends and Challenges with Data Warehousing. The American Council on Education's College Credit Recommendation Service (ACE Credit) 2019.https://www.sophia.org/tutorials/trends-and-challenges-with-data-warehousing..

[3] Wentzlaff A. Business Development Analyst at OnApproach.http://www.trellance.com/7-challenges-consider-building-data-warehouse/..

[4] Konda S, More R. Augmenting data warehouse security techniques – a selective survey. Int Res J Eng Technol 2015;2(3):2209–13.

[5] Aleem S, Capretz LF, Ahmed F. Data security approaches and solutions for data warehouse. Int J Comput 2015;9:91–7.

[6] AlMeghari M. Survey on security issues techniques used in data warehouses. Int J Comput Sci Network Secur 2017;17(3):236–43.

[7] AlMeghari M. Data warehouse signature: a framework for implementing security issues in data warehouses. J Comput Sci Appl Sci Educ Publ 2017;5(1):17–24.

[8] Mollin RA. An introduction to cryptography. 2nd ed. Boca Raton, London, New York: Chapman and Hall/CRC, Taylor and Francis Group; 2007.

[9] Velumadhava Rao R, Ma Vikhnesh, Kailash B, Selvamani K, Elakkiya R. A secure key computation protocol for secure group communication with password based authentication. J Comput Sci Inf Technol 2013:175–82. doi: https://doi.org/10.5121/csit.2013.3619.

[10] Saeb M. A one-pass key distribution protocol based on the hamming code. Int J Comput Sci Commun Secur 2016;6:61–4.

[11] Geetha AV, Rajesh Kumar PM. Threshold cryptography-based group authentication scheme for the smart home environments. Int J Adv Res Comput Commun Eng 2016;5(1):141–8.

[12] Banaie F, Seno SAH, Aldmour I, Budiarto R. A polynomial-based pairwise key pre-distribution and node authentication protocol for WSNs. Telecommun Comput Electron Control 2015;13(4):1113–20.

[13] Shi R-H, Zhong H, Zhang S. A novel authenticated group key distribution scheme. KSII Trans Internet Inf Syst 2016;10(2):935–49.

[14] Piaoa Y, Kima J, Tariqb U, Hong M. Polynomial-based key management for secure intra-group and inter-group communication. Comput Math Appl Elsevier 2013;65(9):1300–9.

[15] Prasanna S, Shummuga Priya S, Balaji N. Group key distribution using authentication based broadcast encryption. Asian J Inf Technol Medwell J 2016;15(20):4002–10.

[16] Golumbeanu AI. Application of differential cryptography to a GN authentication hierarchy scheme. Electron J Differ Equ 2017;2017(20):1–8.

[17] Liu Y, Chang C-C, Chang S-C. A group key distribution system based on the generalized Aryabhata remainder theorem for enterprise digital rights management. J Inf Hiding Multimedia Signal Process 2015;6(1):140–53.

[18] Liu Y, Harn L, Chang C-C. An authenticated group key distribution mechanism using theory of numbers. Int J Commun Syst 2014;27(11):3502–12.
[19] Harn L, Lin C. Authenticated group key transfer protocol based on secret sharing. IEEE Trans Comput 2010;59(6):842–6.
[20] Blum-Smith B, Coskey S. The fundamental theorem on symmetric polynomials: history's first whiff of galois theory. College Math J 2017;48 (1):18–29.
[21] Taha S, Cespedes S, Shen X. EM^{3}A: efficient mutual multi-hop mobile authentication scheme for PMIP networks. In: Proceedings of IEEE International Conference on Communications(ICC), Ottawa, Canada; 2012. pp. 873–877..
[22] Luther A, Buyya R, Ranjan and Venugopal S, Alchemi: A.NET-based Enterprise Grid Computing System, in: Proceedings of International Conference on Internet Computing, 2005, pp. 269–278..
[23] AlMeghari M. Data warehouse signature: high performance evaluation for implementing security issues in data warehouses through a new framework. Int J Secur Appl SERSC 2017;11(6):53–68.
[24] Raghavendra M, Nagendra M. Improvement of DiDRiP protocol in security and efficiency for data discovery and dissemination. Int J Scientific Eng Technol Res 2016;5(47):9729–33.
[25] Shostack A. Threat modeling: designing for security. Indianapolis, Indiana: John Wiley and Sons; 2014.
[26] Stallings W. Cryptography and network security principles and practice. Pearson Education: United States of America; 2014.
[27] Kurmi J, Sodhi A. A survey of zero-knowledge proof for authentication. Int J Adv Res Comput Sci Software Eng 2015;5(1):494–501.
[28] Bhatnagar R, Patel J. Security model for scady grid toolkit – analysis and implementation. Int J Innov Res Comput Commun Eng 2015;3(12):12080–6.
[29] Singh A. Improving multi-prime RSA crypto-computations. Master of Science, University of Louisiana at Lafayette; 2006.
[30] Özcan AB. Performance analysis of elliptic curve multiplication algorithms for elliptic curve cryptography. [Master thesis]. Natural and Applied Sciences, Middle East Technical University (Sep 2006). URL: http://etd.lib.metu.edu.tr/upload/12607698/index.pdf..