World Academy of Science, Engineering and Technology
International Journal of Computer and Systems Engineering
Vol:16, No:12, 2022

Design and Implementation of Security Middleware for Data Warehouse Signature Framework

Mayada AlMeghari

***Abstract*—**Recently, grid middlewares have provided large integrated use of network resources as the shared data and the CPU to become a virtual supercomputer. In this work, we present the design and implementation of the middleware for Data Warehouse Signature (DWS) Framework. The aim of using the middleware in the proposed DWS framework is to achieve the high performance by the parallel computing. This middleware is developed on Alchemi.Net framework to increase the security among the network nodes through the authentication and group-key distribution model. This model achieves the key security and prevents any intermediate attacks in the middleware. This paper presents the flow process structures of the middleware design. In addition, the paper ensures the implementation of security for DWS middleware enhancement with the authentication and group-key distribution model. Finally, from the analysis of other middleware approaches, the developed middleware of DWS framework is the optimal solution of a complete covering of security issues.

***Keywords*—**Middleware, parallel computing, data warehouse, security, group-key, high performance.

I. Introduction

DWS is a framework for implementing the three security issues: Confidentiality, Integrity, and Availability (CIA) in the data warehouses. The DWS framework is designed by using the Client-Server model [1]. For the data flow between the client and the server, there are two process models in the DWS framework. These models are DWSend model and DWReceive model joined in the middleware.

The middleware named View Manager Layer (VML) is used to achieve high performance for data availability by eliminating the query response timed-out. In the two DWS models (DWSend and DWReceive), the security processes (encryption/decryption and hash computing) are executed in the parallel computing using the VML middleware to reach the high performance [2]. This middleware is supported by a.NET-based grid computing framework called Alchemi to use the computational power of networked nodes [3]-[5]. In addition, the VML middleware of the DWS framework has been enhanced by proposing 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 this middleware [6]. The group-key is employed to transmit the shared key of encryption algorithm in a secure way. This model thwarts any attacks, such as denial of service, replay, and man-in-the middle attacks.

The rest of this paper is organized as follows. Section II reviews the related work. Section III presents VML middleware of DWS framework. The middleware design is described in Section IV. In Section V, VML middleware implementation is presented. Section VI offers a comparison of VML middleware and existing middleware approaches. Finally, the conclusion and future work are given in Section VII.

II. Literature Survey

This section primarily focuses on many related works available in the same category. Ahamed et al. [7] present the design and implementation of secure middleware for pervasive computing applications, S-MARKS. This middleware provides security solutions from the perspective of device validation, resource discovery providing trust, handling malicious recommendations, and avoiding privacy violation. The S-MARKS includes core components such as Impregnable Lightweight Device Discovery, ILDD, Simple and Secure Resource Discovery, SSRD with Trust Management and Security Management, Privacy module and Object Request Broker, ORB. The evaluation results show that the parameters such as length of secret x, η and Ω can be tuned to achieve this optimized performance of the network.

In [8], they propose an enhanced mobile agent middleware, Agilla (see Fig. 1) for Wireless Sensor Networks (WSN) to overcome the queuing delay. The approach enhanced the mobile agents for supporting multiple concurrent applications to support dynamic changes and large scale networks for increasing scalability and heterogeneity. This organized the WSN into multiple clusters to promote reusability of available resources. The analysis results showed that this middleware consumed less amount of energy and produced good throughput for a large amount of data streams.

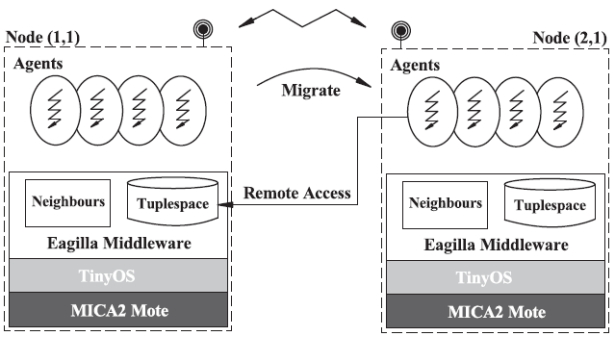

Fig. 1 The architecture of Agilla middleware [8]

Mayada AlMeghari is with Computer Department, Palestine Technical College-Deir ElBalah, Gaza, Palestine (e-mail: malmeghari@ptcdb.edu.ps).

Zheng [9] proposes a real-time middleware based on the publish and subscribe mechanism. In this work, there are two aspects as: First, Real Time eXtention (RTX) is to guarantee the real time performance of data processing and transmission of the middleware. Second, publish/subscribe mechanism is to decouple data producers from data consumers through the communication nodes. The middleware is designed and implemented to meet the real-time requirements of data distribution in the distributed systems. The author adopts a hierarchical design to build his middleware from a model layer, a communication layer and a support layer. The model layer implements the data synchronization of distributed objects. The communication layer provides the publish/subscribe mechanism based on Ethernet. The support layer guarantees the real-time performance of the middleware. This middleware can transmit data correctly in a timely manner and reduce the cost of system development.

Finally, Sallow [10] uses Java-RMI (Remote Method Invocation) middleware to build a distributed system for scheduling the threads. This system includes two separate programs: the server, and the client as shown in Fig. 2. The server program creates some remote objects, makes references to these objects accessible, and waits for clients to invoke methods on these objects. The client program obtains a remote reference to one or more remote objects on a server and then invokes methods on them. This system minimized the complexity of distributed programming and introduced a high degree of transparency.

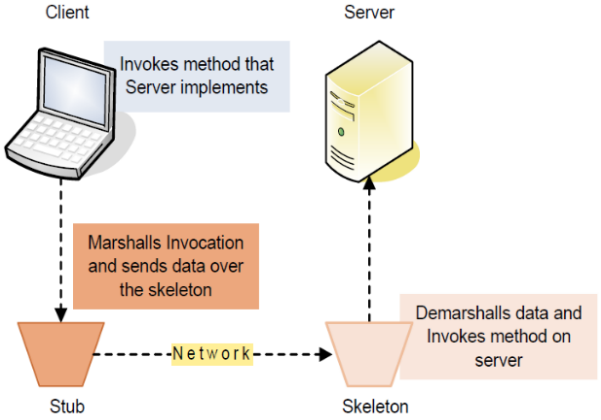

Fig. 2 Distributed RMI middleware [10]

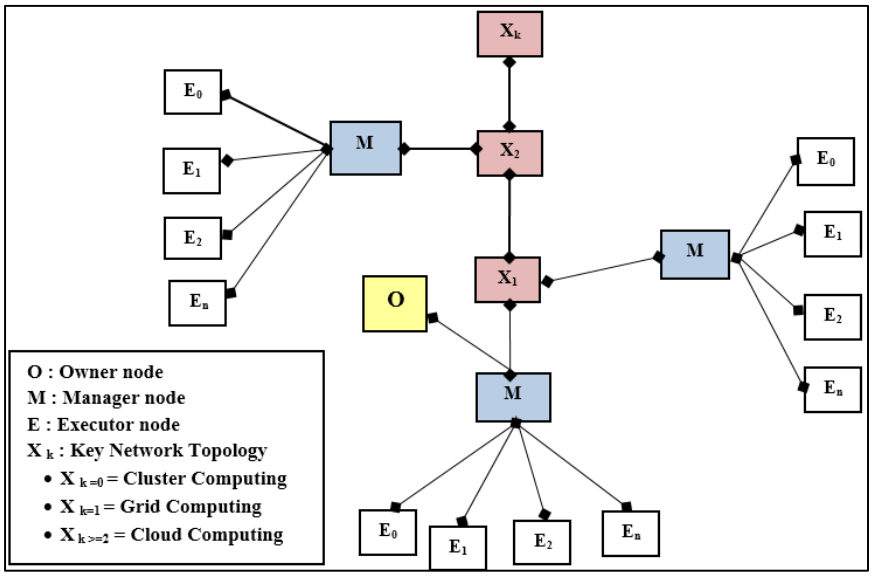

Fig. 3 Types of VML middleware

The previous middleware approaches indicate the absence of a middleware achieving of all security issues in their systems. This motivates us to go more deeply trying to develop a middleware for DWS framework to reach the high performance as well as to increase the security among the network nodes.

III. VML Middleware of DWS Framework

There are many middlewares used to execute the huge amount of data in the parallel computing. The common large distributed middleware is called Alchemi. Alchemi is an open-source .Net based Enterprise Grid computing framework. It allows to aggregate the computing power of networked machines into a virtual supercomputer [11], [12]. In the DWS implementation, we develop the VML middleware supported by Alchemi.Net Desktop Enterprise to achieve the security issues, CIA in the data warehouse systems. The VML middleware supports many types of computing: cluster computing for private network organization; grid computing for public network organization; cloud computing for enterprise desktop network organization [1] (see Fig. 3). This middleware is applied on two sides, the DWServer side (DWSend model) and the Client side (DWReceive model). It consists of three nodes described as:

1. Owner: The owner, O is the DWServer or the Client at the two model sides. This owner is responsible for separating

the Query Result Table (QRT)/Encrypted QRT (EQRT) into blocks with the same size of the number of records to achieve confidentiality. In addition, the owner computes the current hash code value with the previous hash code value for blocks to get the final hash code to achieve integrity.

2. Manager: The manager, M is the main node in the VML middleware, which distributes the blocks/encrypted blocks to a set of chosen executors. Also, this manager returns the computed hash value of each block/encrypted block to the owner to get the final hash code value. Then, the manager recollects all encrypted/decrypted blocks to become EQRT/QRT to be sent into the owner.
3. Executor: The executor, E is the worker that performs two security processes on its block: the encryption/decryption process using Advanced Encryption Standard (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. Executing the security processes on a query result of a huge number of records, QRT as blocks using parallel computing through the VML middleware saves more time than serial computing.

Regarding the DWS framework [1], [2], all transmitted blocks of QRT along with the key, K_{SK}, were sent in a clear-text to the executors, which in turn is a chance for any intermediate attacker to intercept the data before encrypting it. For enhancing DWS framework, the problem of sending the data blocks in a clear-text is solved by compressing the QRT and dividing it to blocks, then keeping one block 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 whole blocks [6].

The other problem of sending the shared key (K_{SK}) of encryption algorithm in a clear-text is solved by proposing an authentication and group-key distribution model as shown in Fig. 4. The goal of the model is to mutually authenticate the manager and the executors to prevent any intermediate attacks, such as denial of service, replay, and man-in-the middle attacks. Also, this model generates a group key among the manager and the chosen executors employed to encrypt the K_{SK} of the AES encryption algorithm. It is based on the idea of modular symmetric polynomial and we are the first to use the modular symmetric polynomial in generating and distributing a group key. Finally, the proposed model also prevents any attacker to get the complete compressed blocks by keeping one block at the owner without sending it.

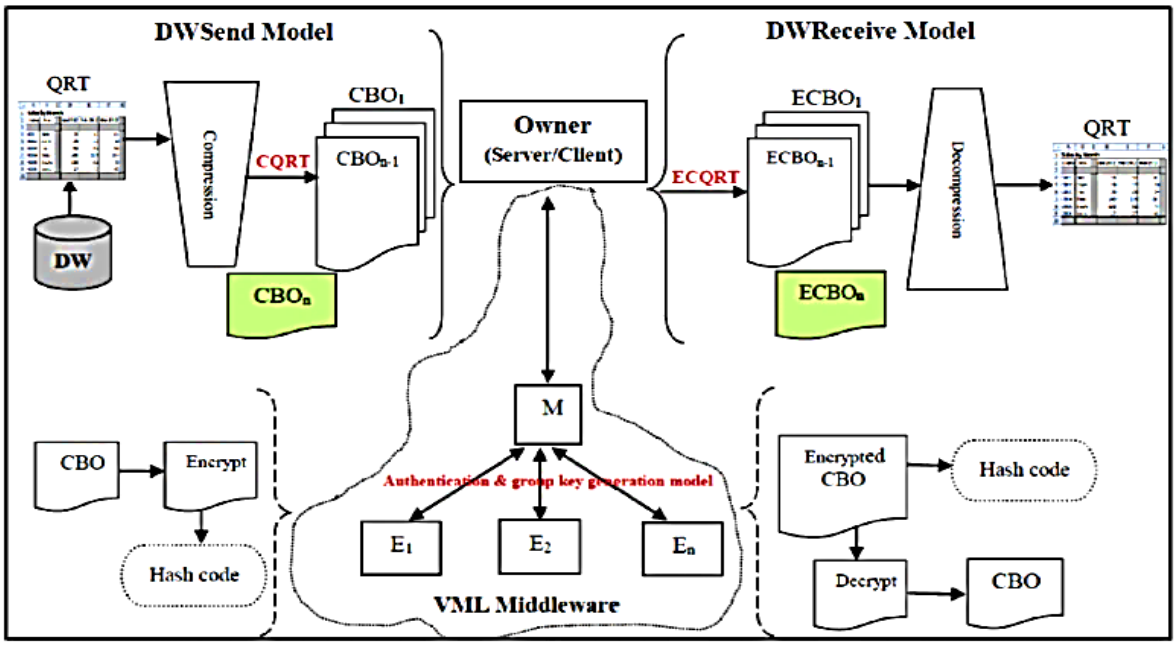

Fig. 4 The architecture of enhancing of DWS framework

The VML middleware of enhancing DWS framework achieves the high performance during executing the security processes by using the parallel computing. Additionally, this middleware increases the security level among its nodes using the authentication and group-key distribution model. It achieves the key freshness, confidentiality, and authentication. It also prevents 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. The use case diagram shown in Fig. 5 details how this model is executed.

IV. VML Middleware Design

The VML middleware is applied in the two models of DWS framework as Server VML for DWSend model and Client VML for DWReceive model. Therefore, there are two flow process structures for the middleware described below:

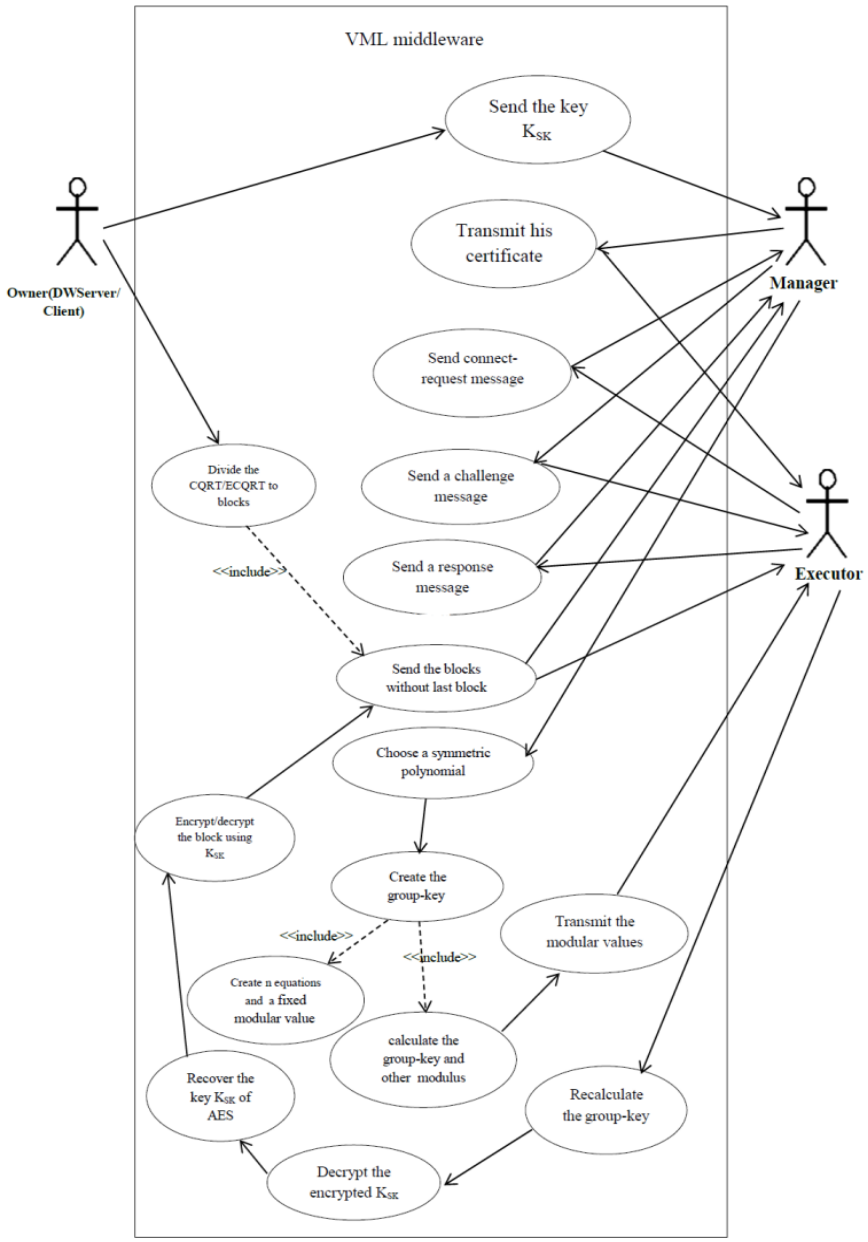

Fig. 5 Use case diagram for authentication and group-key distribution in VML middleware

A. DWSend Middleware

There are three main types of flow process for DWSend structure depending on the VML middleware nodes (Owner, Manager, Executor) as shown in Fig. 6.

• Owner (DWServer)

The owner executes a set of processes in the VML middleware of DWSend ordered as follows:

1. Get the QRT from the DW.
2. Compress the QRT.
3. Read CQRT as a file.
4. Divide the CQRT into blocks objects. The last block is kept in the owner without sending it to the manager.
5. Assign a thread for every block object.
6. Start the thread.
7. Wait for the thread to finish.
8. Make XOR result to other to generate the computed hash for all encrypted block objects to get the Hash of Encrypted Compressed Block Objects (HECBOs).
9. Encrypt the HECBOs by using RSA algorithm with the private key (K_{Pr}) of DWServer to get the Encrypted Hash of Encrypted Block Objects (EHECBOs).
10. Write ECQRT || EHECBOs as a view table.
11. Calculate execution time.

• Manager

1. Initialize the authentication process with the chosen executors joint in the VML middleware of DWSend model.
2. Get the thread from the owner.
3. Generate the group key in order to encrypt the K_{SK} and

then send it in a secure way to the chosen executors.

4. Assign the thread to an executor.
5. Pool finish thread.
6. Send the results to the owner. These results are:

- The hash value for each Encrypted Compressed Block Object (ECBO).
- The ECBOs are recollected to become one block object as ECQRT.

•Executor

1. Set the authentication with the manager to verify the authorized executors.
2. Get a thread from the manager.
3. Calculate the group key to be used for getting the sent K_{SK}.
4. Encrypt it CBO by using AES algorithm with the share key (K_{SK}).
5. Execute the hash code to compute the hash function for each ECBO using SHA-1 algorithm.
6. Return the results to the manager.

B. DWReceive Middleware

The structure of DWReceive model is also separated into three types of flow process based on its VML middleware nodes (see Fig. 7).

•Owner (Client)

1. Separate the ECQRT from the EHECBOs, which are received from the DWServer.
2. Decrypt the EHECBOs by using RSA algorithm with the public key (K_{Pu}) of DWServer.
3. Read the number of blocks chosen in the DWSend model.
4. Read the ECQRT as data block objects.
5. Assign a thread for each block object of number of records.
6. Start the thread.
7. Wait for the thread to finish.
8. Make XOR result to other to generate the computed hash for all encrypted block objects to get the Hash of Encrypted Block Objects (HECBOs).
9. Decompress the CQRT after collecting CBOs blocks. With condition, its block is added with other blocks, which are sent from manager to be become QRT.

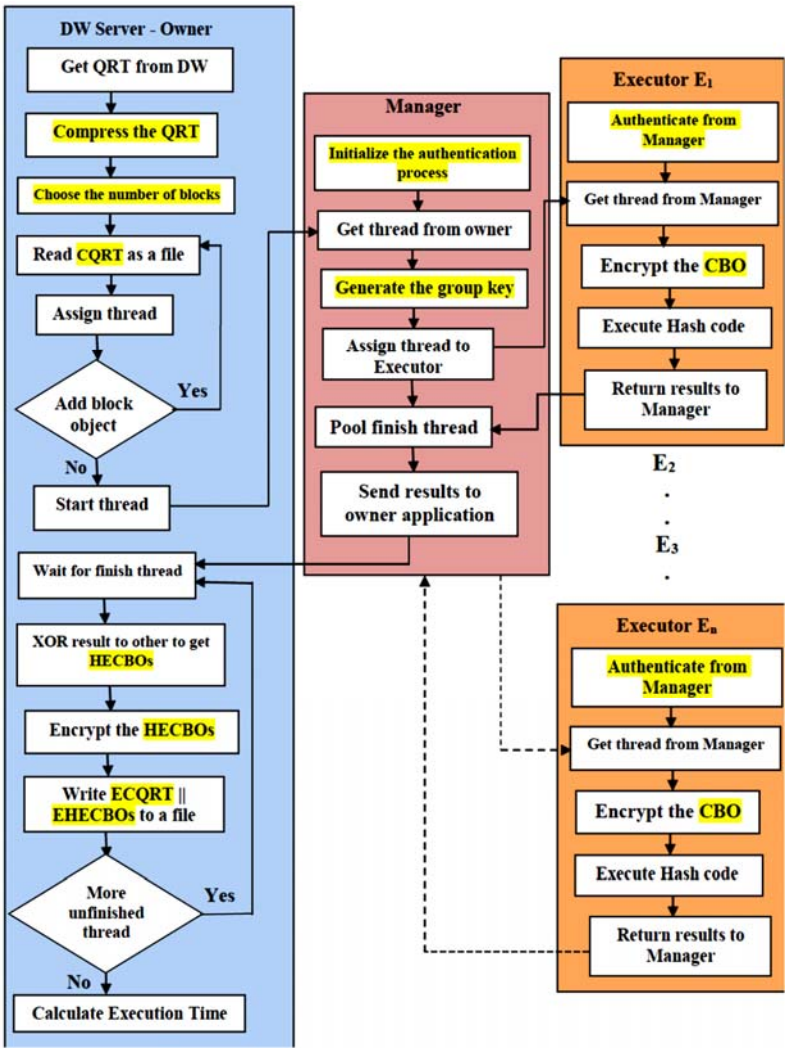

Fig. 6 The steps of DWSend middleware design

10. Compare the calculated HECBOs with the received HECBOs.
11. Calculate the execution time.

•Manger

1. Initialize the authentication process with the chosen executors joint in the VML middleware of DWReceive model.
2. Get the thread from application.
3. Generate the group key to encrypt the K_{SK} and then send it in a secure way to the executors.
4. Assign the thread to an executor.
5. Pool finish thread.
6. Send the results to the owner. These results are:

- The hash value for each ECBO.
- The decrypted block objects are recollected to get CQRT.

• Executor

1. Set the authentication with the manager to verify the authorized executors.
2. Get the thread from the manager.
3. Execute the hash code to compute the hash function for its encrypted compressed block object, ECBO using SHA-1 algorithm.
4. Calculate the group key to be used for getting the sent K_{SK}.
5. Decrypt each ECBO by using AES algorithm with the K_{SK}.
6. Return the results to the manager.

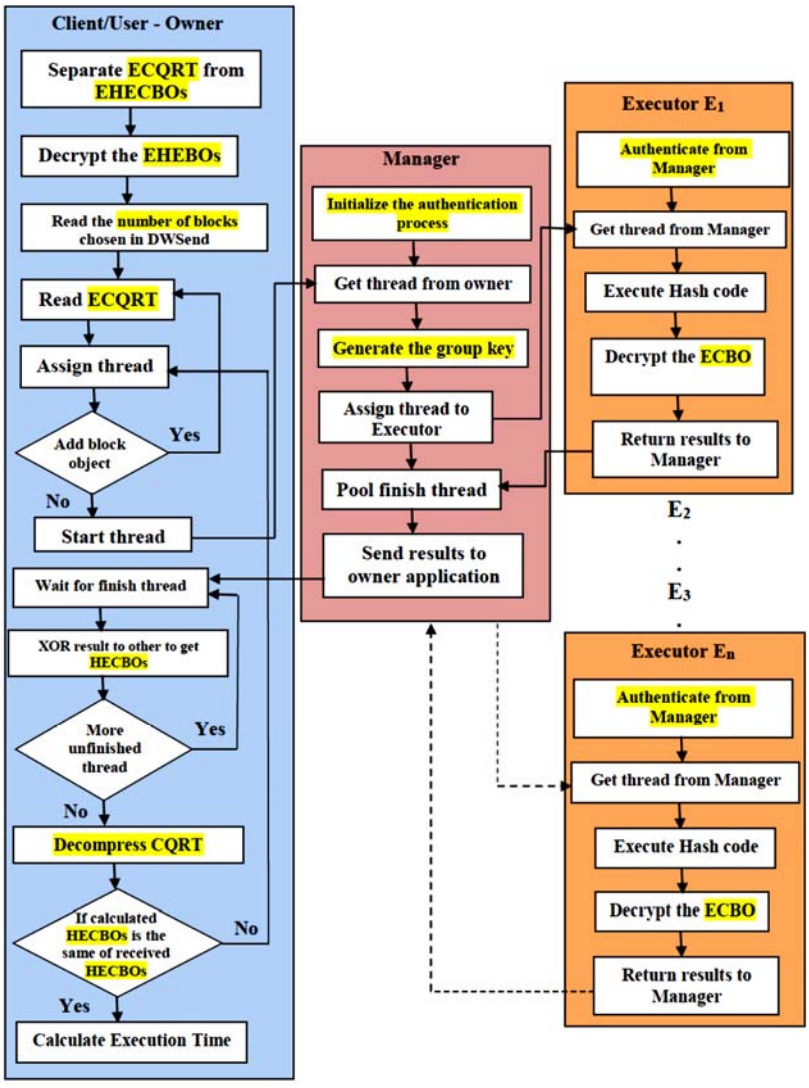

Fig. 7 The steps of DWReceive middleware design

V. VML Middleware Implementation

The VML middleware is developed on Alchemi 1.0.4.Net Framework 2 [3] to resist the potential attacks through the proposed authentication and group-key distribution model. For managing a set of schedule threads, the VML used a manager node. For executing a set of these threads, the VML used the executor nodes. The result of execution of schedule jobs is returned to the owner. The proposed model for VML provides a secure authentication between the manager and the executors to prevent any intermediate attacks. Also, this model generates a group-key that is distributed between the manager and the chosen executors employed to transmit the key, K_{SK} in a secure

way. The experimental network data flow is in a star network topology with a cluster structure. The programming language used in developing the VML middleware is C# with Microsoft Visual Studio 2013.

A. VML Database

The VML middleware needs to store the data for each of the executors, the threads, the applications, the groups, and the users. So, the VML middleware has a database stored at the manager node, which is the main component in this middleware. The database that manages the data flow in the VML middleware is Microsoft SQL Server Management Studio 2008. In the executor table, there is a set of data fields added for authentication and group-key generation processes, such as mangerID, executorID, nonce, and modular. In addition, in the application table, two data fields are added to be used in the group-key generation process, such as AES_Key and group key (see Fig. 8).

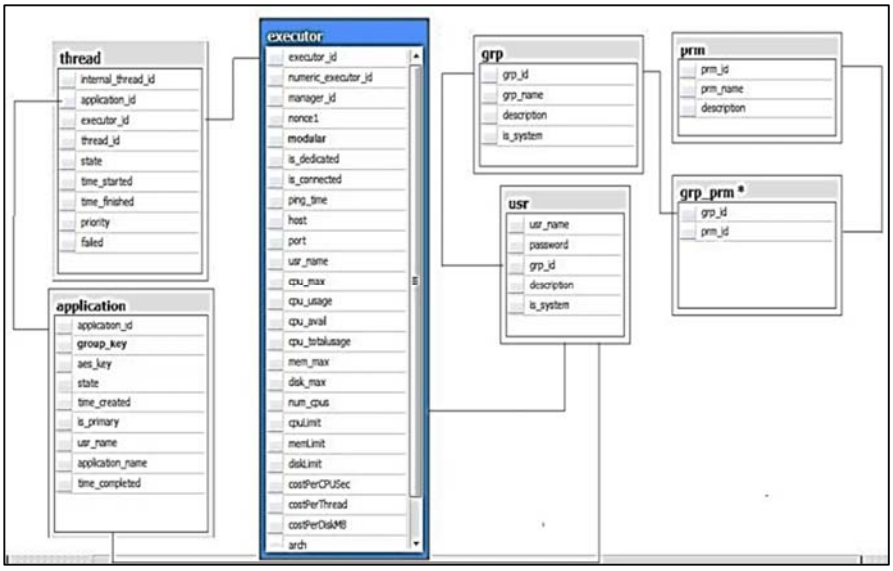

Fig. 8 The VML database diagram

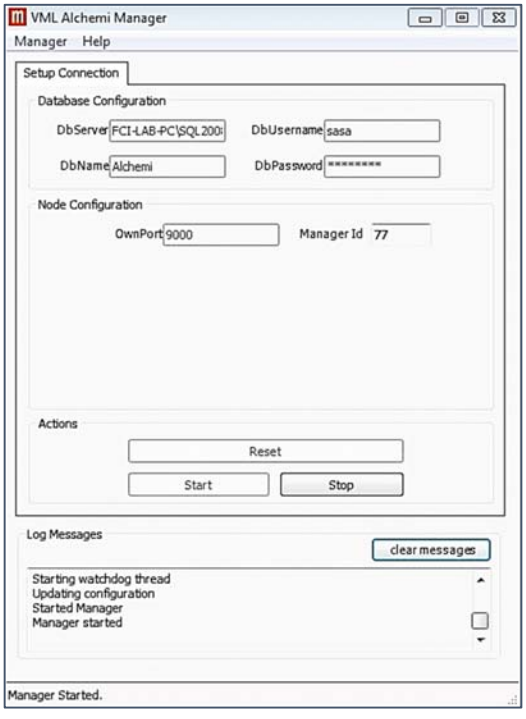

Fig. 9 (a) Start manager

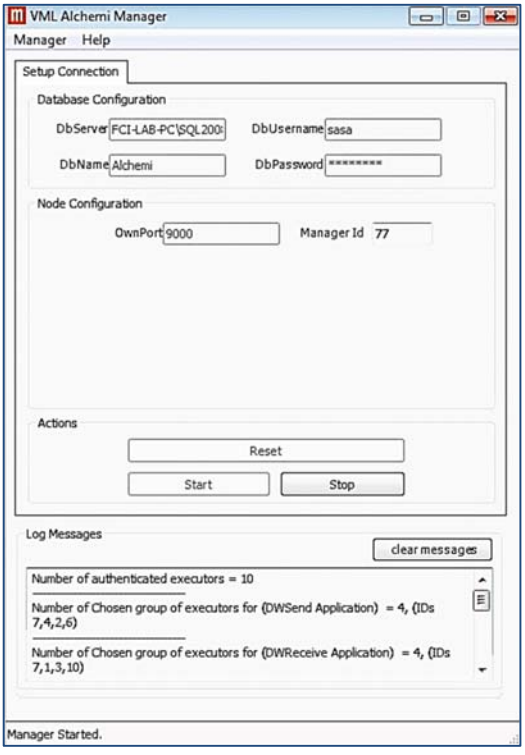

Fig. 9 (b) The authenticated executors and the chosen executors for working

B. VML Manager

The manager schedules a set of threads as jobs and manages the data of threads for sending it to the chosen executors as First Come First Served (FCFS). The database configuration for the VML manager should be entered to start the connection process, then the managerID is generated randomly (see Fig. 9 (a)). For the authentication in the VML middleware, the manager shows the connected executors, which are authenticated with it. Also, after finishing the DWS application, this manager shows the chosen executors, which execute the application process (see Fig. 9 (b)).

C. VML Executors

In VML middleware for DWS framework, the chosen executors used their threads as blocks, which are sent by the manager. In the two models (DWSend and DWReceive), these executors perform two processes, encryption/decryption process and hash code function process in a parallel computing. There are two types of executors as dedicated and non-dedicated. The dedicated executor is always executing the threads, while the non-dedicated executor executes the threads on a voluntary basis. In the VML executors, the IP address of manager is entered to connect it with the executors, and then the executorIDs are generated sequentially. After that, the authentication process (described in [6]) is done between the manager and those connected executors. Fig. 10 shows a snapshot for one of the connected executors (ID:10) which is authenticated with the manager.

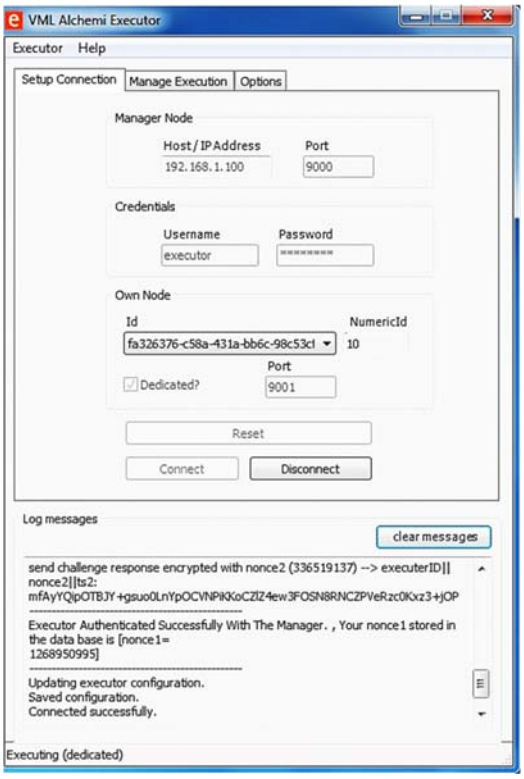

Fig. 10 The authentication process between the manager and the connected executors

For the chosen executors by the manger, the group-key is generated and then distributed among the VML nodes in order to send the AES key, K_{SK} securely [6]. For example, the AES key (K_{SK}) was 215 generated using Diffie-Hellman (D-H) algorithm to be distributed among the two models (see Figs. 11 and 12).

In the VML middleware, the group-key is recalculated and periodically changed for each application run. For example, in the middleware at DWSend, the generated group-key was 16370 at the chosen executors with IDs (7,4,2,6) to execute their threads of blocks. Fig. 11 shows a thread of executor ID 2. While, in the middleware at DWReceive, the group-key was 42848 at the chosen executors with IDs (7,1,3,10) to perform their threads. Fig. 12 shows a thread of executor ID 10.

D. Application of Authentication and Group-key Distribution

The authentication and group-key distribution in the VML middleware is implemented with C# languages to provide a secure way among the middleware nodes. This includes initialization, authentication, and group-key generation phases [6]. In the initialization, each executor sends connect-request message to the manager encrypted with its public key. This manager receives the connect request to be decrypted using its private key (see C# code of SendConnectionRequest method).

Fig. 11 The group-key generation in the VML middleware- DWSend

```
public void SendConnectionRequest(byte[] encMsg)
    {
        string appDir = AppDomain.CurrentDomain.BaseDirectory +
"\\private.rsa.cs.key";
        var key = File.ReadAllText(appDir);
        var bytes = NewUtils.RSADecryptMsg(encMsg, key);
        var data = System.Text.Encoding.UTF8.GetString(bytes);
        //receive --> executerID||nonce1||ts1
```

```
        var dataArray = NewUtils.SplitMessage(data);
        var nonce1 = int.Parse(dataArray[1]);
        var ExeNumeric = int.Parse(dataArray[0]);
        logger.Debug("received connection request message from exe
cutor (" + ExeNumeric + ")--
>" + Convert.ToBase64String(encMsg));
    }
```

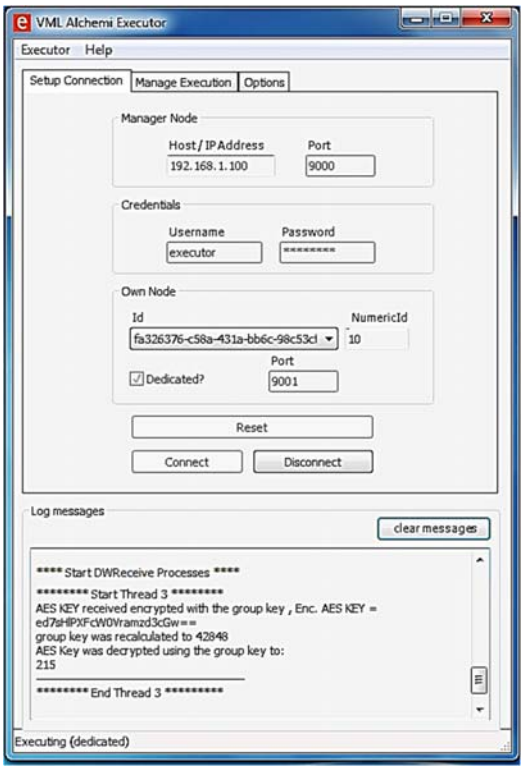

Fig. 12 The group-key generation in the VML middleware-DWReceive

As an authentication phase, the manager sends a challenge message to the executor. This is encrypted using the public key of executor. The definition of a method called SendChallengeMessage is presented as follows:

```
public String SendChallengeMessage (byte[] encMsg)
    {
        var f = NewUtils.F(ExeNumeric, ManagerId);
        var nonce2 = NewUtils.GenerateNonce();
        var ts1 = dataArray[2];
        var ts2 = NewUtils.GetTimestamp(DateTime.Now);
        var chanllengeMsg = NewUtils.JoinMessage(ExeNumeric, no
nce1, ts1, nonce2, ts2, f);
        //send --> executerID||nonce1||ts1||nonce2||ts2, F(x, y)
        var EncChanllengeMsg = RijndaelManagedEncryption.Encry
ptRijndael(chanllengeMsg, nonce1.ToString());
        logger.Debug("send challenge message to executor (" + ExeN
umeric + ")-->" + EncChanllengeMsg);
        NonceDict[ExeNumeric] = new NonceMessageModel { Nonc
e1 = nonce1, Nonce2 = nonce2, Ts2 = ts2 };
        return EncChanllengeMsg;
    }
```

The executor decrypts the received challenge message and replies with the response message to manager shown in the method SendChallengeResponse. The manger decrypts this message and compares the values at the messages and then authenticates the executor if the values are identical.

```
public void SendChallengeResponse(string executorId, string EnctMe
ssage)
    {
        logger.Debug("received challenge response message from exe
cutor (" + EnctMessage + ")-->" + EnctMessage);
        var numId = ManagerStorageFactory.ManagerStorage().GetE
xecutorNumericId(executorId);
        var StoredNonceModel = NonceDict[numId];
        var StoredNonce2 = StoredNonceModel.Nonce2;
        var StoredTs2 = StoredNonceModel.Ts2;
        //msg = executerID||nonce2||ts2
        var DectRespMsg = RijndaelManagedEncryption.DecryptRijn
dael(EnctMessage, StoredNonce2.ToString());
        var dataArray = NewUtils.SplitMessage(DectRespMsg);
        var ReceivedNonce2 = int.Parse(dataArray[1]);
        var ReceivedTs2 = dataArray[2];
        if (ReceivedNonce2 == StoredNonce2 && ReceivedTs2 == S
toredTs2)
        {
          ManagerStorageFactory.ManagerStorage().UpdateExecutor
Nonce1(executorId, ManagerId, StoredNonceModel.Nonce1);
          logger.Debug(string.Format("ReceivedTs2:{0} equals Stor
edTs2:{1} and ReceivedNonce2 == StoredNonce2", ReceivedTs2, St
oredTs2));
        }
        else
        {
          logger.Debug("Invalid Challenge Response");
          throw new InvalidChallengeResponse();
        }
    }
```

After authentication, the manager with the chosen executors can generate a group key, Kgr to send the shared key of AES algorithm among them securely. The manager chooses a secure symmetric polynomial, F(x,y) and transmits the modular values each of which to the intended executor after encrypting it using $nonce_i$. The SetExecutorModular method shows how each executor gets its modular. Using the modular value, each executor can recalculate the group key as defined in the method CalculateGroupKey. Thus, the authentication and group-key distribution algorithm solves the problem of sending the shared key (K_{SK}) in a clear-text.

```
public void SetExecutorModular(string encryptedModular)
    {
        string nonce1 = Configuration.GetConfiguration().Nonce1.To
String();
        var dec_m = RijndaelManagedEncryption.DecryptRijndael(en
cryptedModular, nonce1);
        this.m = int.Parse(dec_m);
        logger.Debug(string.Format("UpdateExecutorModular: execu
torId {0}, m value:{1}", Id, m));
    }

    public double CalculateGroupKey()
    {
```

```
        var f = NewUtils.F(this.NumericId, this.Manager.GetManager
Id());
        var L = f % m;
        if (L == 0)
        {
           var g = L;
        }
        return L;
    }
```

VI. Comparison of VML Middleware and Existing Middleware Approaches

The VML middleware is applied to obtain the high performance using parallel computing and to achieve a high level of security among its nodes. This section presents the evaluation of the VML middleware comparing to other existing middleware approaches as shown in Table I. All of them achieved the performance according to their evaluation criteria. The middleware of [9] failed to provide security solutions. Most of these middleware achieved the authentication; the middleware of [7] achieved the authentication using ILDD challenge – response method; the middleware of [8] achieved the authentication using profiling for sensor nodes; and, the middleware of [10] achieved the authentication using Remote Method Invocation, RMI Security Manager. While our middleware achieved the authentication using secure challenge-response method to be prevented any intermediate attacks, such as denial of service, replay, and man-in-the middle attacks, only one middleware [7] achieved the confidentiality using Learning Parity in the presence of Noise, LPN secret algorithm with the problem of sending the key in a clear-text among their middleware nodes. The VML middleware achieved the confidentiality using two encryption algorithms, symmetric AES and asymmetric public-key with distributing the key in a secure way among its nodes based on the modular symmetric polynomial. Finally, most of the middleware shown in Table I have not addressed all security issues in terms of authentication, confidentially and key distribution. The VML middleware addresses these issues and resists the potential attacks as well as increases the performance.

VII. Conclusion and Future work

In this paper, we have presented the design and the implementation of a secure middleware for DWS framework. This middleware was developed to increase the security level among its nodes using the authentication and group-key distribution model with the increase of high performance. It also achieved the key security and prevented both insider and outsider attackers. This paper has provided a comparative study of some middleware approaches with the proposed VML middleware. In the future, the VML middleware can be extended for applying our secure DWS framework along with the authentication and group-key distribution in cloud computing.

TABLE I
A Comparison of VML Middleware for DWS Framework with Existing Middleware Approaches

Approach	Architecture	Implementation Tools	Thread Support	Computing Environment	Performance	Security		
						Authentication	Confidentiality	Key Distribution
Zheng (2019) [9]	Hierarchical	Aircraft program and Radar program	No	Distributed Computing	Minimize communication delays under CPU and network loads	No	No	No
Lingaraj et al. (2018) [8]	Hybrid Multi-Layered	Simulation using prototypes	No	Cluster Computing	Less amount of energy and good throughput for a large amount of data streams	Profiling	No	No
Sallow (2020) [10]	Client/Server	Java Platform	Yes	Distributed Computing	Minimize the complexity of distributed programming and introduce a high degree of transparency	RMISecurity Manager	No	No
Ahamed et al. (2009) [7]	Hierarchical	Prototypes for Impregnable Lightweight Device Discovery, ILDD and Simple and Secure Resource Discovery, SSRD	Yes	Pervasive Computing	Achieve the optimized performance of the network for parameters as length of secret x, η and Ω	ILDD challenge-response	LPN secret algorithm	No
Our Middleware	Hierarchical with centralized database	C# with Microsoft Visual Studio 2013, Microsoft SQL Server Management Studio 2008 and .Net Framework 2.	Yes	Cluster Computing	Less the cost of computation, communication and complexity overheads	Challenge-response method	AES and public-key encryption algorithms	Modular symmetric polynomial function

References

[1] M. AlMeghari, "Data Warehouse Signature: A Framework for Implementing Security Issues in Data Warehouses", Journal of Computer Sciences and Applications, Science and Education Publishing (SciEP), DOI: 10.12691/jcsa-5-1-3, Vol. 5, No. 1, pp. 17-24, April (2017).

[2] M. AlMeghari, "Data warehouse signature: high performance evaluation for implementing security issues in data warehouses through a new framework", International Journal of Security and Its Applications (IJSIA), Science & Engineering Research Support society (SERSC), Journal ISSN: 1738 -9976, Vol. 11, No. 6, pp. 53-68, June (2017).

[3] A. Luther, R. Buyya, R. Ranjan and S. Venugopal, " Alchemi: A.NET-based Enterprise Grid Computing System", in: Proceedings of International Conference on Internet Computing, Las Vegas, Nevada, USA, pp. 269–278, June 27-30(2005).

[4] A. Poshtkohi, A. H.Abutalebi and S. Hessabi, "DotGrid: a .NET-based cross-platform software for desktop grids", International Journal of Web and Grid Services, Vol. 3, No. 3, pp. 313-332, August (2007).

[5] P. Nagamani, P. Suresh Varma and M. Upendra Kumar, "Security Design Framework for Data Management and Distribution for Middleware in Grid Extended Cloud Computing", Journal of Engineering Research and Application, ISSN: 2248-9622, Vol. 8, Issue 5, pp .57-76, May (2018).

[6] M. AlMeghari, S. Taha, H. Elmahdy, X. Shen, "Proposed Authentication and Group-Key Distribution Model for Data Warehouse Signature, DWS Framework", Egyptian Informatics Journal, DOI: 10.1016/j.eij.2020.09.002, Volume 22, Issue 3, pp. 245-255, Elsevier, September (2021).

[7] S. I. Ahamed, H. Li, N. Talukder, M. Monjur and C.S. Hasan, "Design and implementation of S-MARKS: A secure middleware for pervasive computing applications", Vol. 82, No. 10, pp.1657-1677, Elsevier, October (2009).

[8] K. Lingaraj, R.V. Biradar, V.C. Patil, "Eagilla: An Enhanced Mobile Agent Middleware for Wireless Sensor Networks", Alexandria Engineering Journal, Vol. 57, pp. 1197–1204, Elsevier (2018).

[9] P. Zheng, "Design and Implementation of a Real-time Publish/Subscribe Middleware", Recent Advances in Electrical & Electronic Engineering, Vol. 12, No. 6, pp. 513-518 (2019).

[10] A. B. Sallow, "Design and Implementation Distributed System Using Java-RMI Middleware", Academic Journal of Nawroz University (AJNU), Volume 9, No 1, pp. 113-120, February (2020).

[11] M. Bhardwaj, S. Sarbjeet, M. Singh, "Implementation of Single Sign-On and Delegation Mechanisms in Alchemi.Net Based Grid Computing Framework", International Journal of Information Technology and Knowledge Management, Vol. 4, No. 1, pp. 289-292, January-June (2011).

[12] P. Arora and H. Arora, "A Review of Various Grid Middleware Technologies", International Journal of Advanced Research in Computer Science and Software Engineering, Vol. 2, Issue 6, pp. 6-11, June (2012).