

COMPUTER NETWORK SECURITY PROJECT

By: G. Mounica Reddy

Swetha Pasham

Prakruthi Shivram

PROJECT OBJECTIVE

The goal of this project is to design a point-to-point (P2P) Messaging application or tool. This application will support a client or user in sending messages to another user. This application ensures that this communication is secure and that the message is received only by the designated user (receiver). This application must be setup using a password. The application must use a key to encrypt and decrypt the message being communicated, and this key must 56-bits of length. This application uses an Active MQ broker to achieve asynchronous communication. This enable the sender to send the message even if the receiver is not online. The message sent is held in the queue in encrypted form and is sent to the receiver when he is available online.

BACKGROUND INFORMATION

We use an Apache Active MQ message broker to act as a connection between client and server, it provides reliable and asynchronous communication. We implement the code in eclipse oxygen IDE and use Apache Tomcat server to set up a web application. We used a message broker because using a database would create more overhead when it comes to storing and retrieving data.

Apache Active MQ:

In computer programming, a message broker is an intermediary program module that translates a message from the formal messaging protocol of the sender to the formal messaging protocol of the receiver. Message brokers are elements in telecommunication or computer networks where software applications communicate by exchanging formally-defined messages.

The following are examples of actions that might be taken in by the broker:

- Route messages to one or more destinations
- Transform messages to an alternative representation

- Perform message aggregation, decomposing messages into multiple messages and sending them to their destination, then recomposing the responses into one message to return to the user
- Interact with an external repository to augment a message or store it
- Invoke Web services to retrieve data
- Respond to events or errors
- Provide content and topic-based message routing using the publish–subscribe pattern

Apache ActiveMQ is an open source message broker written in Java together with a full Java Message Service client. It provides "Enterprise Features" which in this case means fostering the communication from more than one client or server.

Active MQ broker is used to reliably communicate between two distributed processes. We could also store messages in a database to communicate between two processes, but, as soon as the message is received you would have to delete the message. This means that a row needs to be inserted and deleted for each message. When you try large scale communication, about thousands of messages per second, databases may break down due to large overhead.

Message oriented middle-ware like ActiveMQ on the other hand, are built to handle large scale communication. They assume that messages in a healthy system will be deleted very quickly and, can do optimizations to avoid the overhead. It can also automatically push the messages to the consumer instead of a consumer having to poll for new message by doing a SQL query. This further reduces the latency involved in processing new messages being sent into the system. Hence using Active MQ is advantageous as it reduces cost by reducing overhead to maintain message data.

JMS API

The Java Message Service (JMS) API is a Java Message Oriented Middleware (MOM) API for sending messages between two or more clients. It is implemented in order to handle the Producer-consumer problem. It is a messaging standard that allows application components based on the Java Enterprise Edition (Java EE) to create, send, receive, and read messages.

JMS PROGRAMMING MODEL

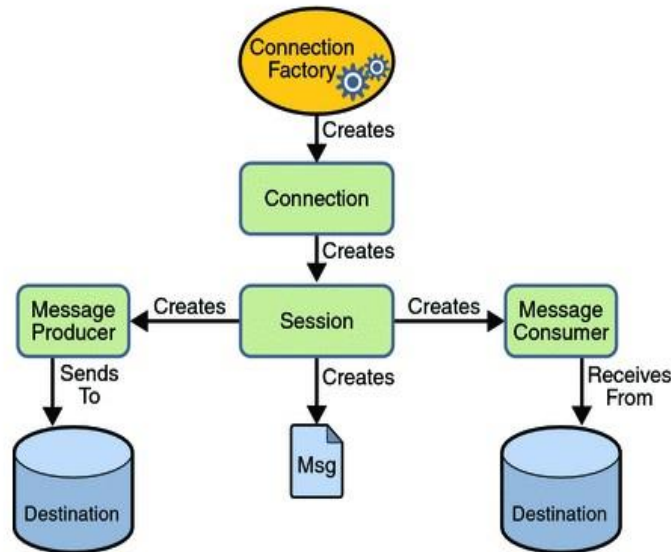


Fig 1. JMS programming model

The basic building blocks of JMS application consists of administered objects – connection factories and destinations, connections, sessions, message producers, message consumers and messages.

- Client uses a **connection factory** object to create a connection to a provider.
- Client uses the **destination** object to indicate the target of messages that it produces and the source of the messages that it consumes.
- A JMS provider is used by the **connection** to encapsulate a virtual connection.
- A **session** is a single-threaded context for producing and consuming messages.
- A **message producer** is an object that is created by a session and used for sending messages to a destination.
- A **message consumer** is an object that is created by a session and used for receiving messages sent to a destination.
- The main purpose of JMS application is to produce and to consume **messages** that can then be used by other software applications.

SCHEME DESIGN

We make use of javax.crypto library to use functions to generate a DES key in order to encrypt and decrypt the message being sent. We use a 8 byte salt and “MD5WithDES” algorithm to achieve the encryption and decryption.

Secret Key Factory:

SecretKeyFactory is a class from the javax.crypto library. This class is used for generating secret symmetric keys. In other words, it generates a SecretKey object from the input key specification (e.g. PBEMD5WithDES). The given SecretKey instance inherits from java.security.Key which is Serializable and uses a default RAW encoding format.

We create a PBEKeySpec object that encapsulates the password, then a PBEPParameterSpec object that encapsulates the salt and iteration count (the number of times the hash function will be applied to derive the symmetric key from the salt and password: typical values would be between 1000 and 2000)

Password-based encryption


To perform password-based encryption, we've seen that we need a random salt sequence to prevent dictionary attacks, and that we need to use a deliberately slow function to generate a key from a given password and salt. Since the project requires a 56-bit key we will use DES.

The Key generation process is as follows:

- we append the password to the salt and, also append a counter, which will start at 1;
- we calculate a secure hash of the sequence we just created;
- we then repeat the process for some number of iterations, each time forming the new sequence to be hashed from the output of the previous hash and, appending the salt and incremented counter.

PROJECT OUTCOME

Our project achieves all the goals stated in the project objective. Our point-to-point messaging web application provides a user login page, which ensures authentication of clients/ users accessing the application. We also show in the figure 2 below, the Active MQ queue which holds the message until the receiver is available online for communication. Figure 3 shows the pending messages which are yet to be delivered to the receiver. Figure 4 shows the active clients which are holding a conversation.




Home | Queues | Topics | Subscribers | Connections | Network | Scheduled | Send

Queue Name Create

Queues

Name	Number Of Pending Messages	Number Of Consumers	Messages Enqueued	Messages Dequeued	Views	Operations
myqueue	1	1	11	10	Browse Active Consumers atom rss	Send To Purge Delete

Fig 2. Apache Active MQ broker showing the pending messages to be received by the receiver.




Home | Queues | Topics | Subscribers | Connections | Network | Scheduled | Send

Browse myqueue

Message ID	Correlation ID	Persistence	Priority	Redelivered	Reply To	Timestamp	Type	Operations
ID:DESKTOP-4N3FV5E-59514-1524598391133-1:605:1:1:1	swetha	Persistent	4	false		2018-04-24 16:00:24:354 EDT		Delete

[View Consumers](#)

Fig 3: Pending messages



Home | Queues | Topics | Subscribers | Connections | Network | Scheduled | Send

Active Consumers for myqueue

Client ID Connection ID ↑	SessionId	Selector	Enqueues	Dequeues	Dispatched	Dispatched Queue	Prefetch Max pending	Exclusive Retroactive
ID:DESKTOP-4N3FV5E-59514-1524598391133-2:641 ID:DESKTOP-4N3FV5E-59514-1524598391133-1:641	1	JMSCorrelationID='prakruthi'	0	0	0	0	1000 0	false false
ID:DESKTOP-4N3FV5E-59514-1524598391133-2:642 ID:DESKTOP-4N3FV5E-59514-1524598391133-1:642	1	JMSCorrelationID='mounica'	0	0	0	0	1000 0	false false

Fig 4: Active clients

LOGIN PAGE

Username

Password

Fig 5. Log-in page for user authentication

After the user is validated, he can access our application to send secure messages to the receiver. The images below show the communication between two users. This application provides an option to select the receiver to whom the message is to be sent. The Chat history maintains the chat log of currently communicating users, this is achieved by maintaining a list of all the messages sent in a session. This chat disappears once the session ends. The message which was passed is shown in its encrypted form in red under the label “**Message**”. This is shown only for the sake of demonstrating the encryption part during this project, and hence it can be removed during actual use of the application.

Figure 6.1 and figure 6.2 shows the conversation between the receiver and the sender.

Secure Instant Point to Point Messaging

User Name - swetha - [\(Logout\)](#)

Message - X2lk: d0nkWZ9ozjynZHYuv==

Receiver : mounica

Message:

Chat History

mounica : Hi Swetha

swetha : Hi Mounica

mounica : wassup

swetha : ntng much

swetha : what r u dng

mounica : testing the project

Secure Instant Point to Point Messaging

User Name - mounica - [\(Logout\)](#)

Message - AXeO2FXRxEGDB7eH1cwhCzF66MYOCh++

Receiver : swetha

Message:

Chat History

mounica : Hi Swetha

swetha : Hi Mounica

mounica : wassup

swetha : ntng much

swetha : what r u dng

mounica : testing the project

prakruthi : hey

Fig 6.1 Sender side interface for point-to-point Message

Fig 6.2 Receiver side interface for point-to-point Message

References:

1. https://en.wikipedia.org/wiki/Apache_ActiveMQ
2. <https://docs.oracle.com/javaee/5/tutorial/doc/bnceh.html>
3. https://en.wikipedia.org/wiki/Java_Message_Service
4. https://www.javamex.com/tutorials/cryptography/pbe_key_derivation.shtml
5. https://en.wikipedia.org/wiki/Message_broker