Distributed Systems - ENGR-4790U

Distributed Secure Channel Project Report

REVISION 3.4

CREATED BY

DANIEL SMULLEN

JONATHAN GILLETT

JOSEPH HERON

12/03/2013

This page intentionally left blank.

Table of Contents

Tables and Figures

1 INTRODUCTION

1.1 CURRENT APPROACHES AND THEIR PROBLEMS

Nearly all widely adopted cryptographic systems are based on a centralized model. One of the most widespread forms of cryptography, Public Key Infrastructure (PKI), is used for TLS - Transport Layer Security (used in HTTPS). It depends on a direct client-server interaction in addition to a centralized Certificate Authority (CA). Using a centralized key infrastructure means that when a network is compromised by an attacker, the attacker gains full access to all communications throughout the network.

Current common methods of distributed cryptography rely on the use of a symmetric key or a key management system that uses public key cryptography. JGroups provides support for distributed cryptography using a symmetric key. Symmetric keys can potentially be insecure if they are generated as an insecure password, or kept inside the code. This is visible in Figure 1. More secure methods are available, such as generating the key using a secure random number generator. These do not surmount the main problem which precludes securely changing the symmetric key once the network is in operation. Further, since existing distributed mechanisms do not provide any means for changing the symmetric key or providing message authentication, attackers have the ability to masquerade as any member of a compromised network. The only way to remove the attacker is to destroy and recreate the network. Similar to centralized key infrastructure, if the symmetric key is compromised, the attacker has full access to the entire network.

```
34  # This is used for encrypting packets every network
35  # should have its own unique key.
36  network_key = 'password987access!'
37
38  # This is the name of the "hub" which is seen by the user
```

Figure 1 - Visible Symmetric Key in Program Code

Other approaches to distributed cryptography use public key cryptography in order to both encrypt messages and support authentication. They have numerous attacks, such as man in the middle attacks. In the event of a man in the middle attack, the transmitted public key may be intercepted by the attacker and replaced with the attacker's public key. To the recipient, the key will appear as if it is from the original sender, although the attacker will now be able to intercept all messages intended for the original recipient.

Purely distributed public key cryptographic systems also suffer from performance issues. If a network contains N nodes, the number of key exchanges that are required must form a complete graph. This is visible in Figure 2. The process of encrypting the message and distributing it to each individual client must be done $N - 1$ times for each client.

For the network at large, this means the overhead is $N^2 - N$ exchanges. This creates a huge network bandwidth and computational overhead requirement that grows with the number of nodes that are part of the network.

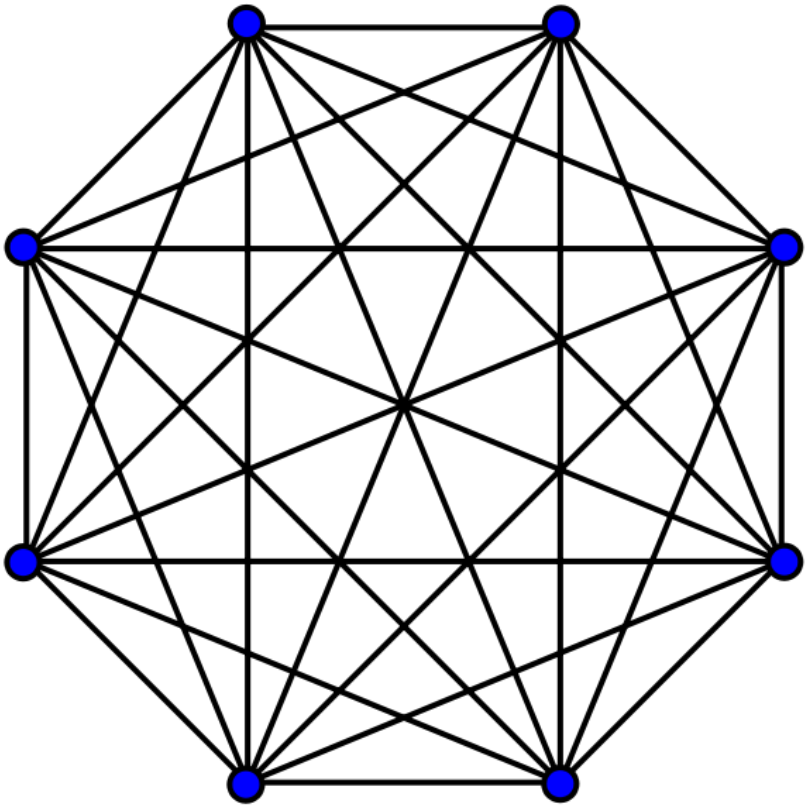

Figure 2 - Public Key Encryption Complete Graph Problem

While ad-hoc networks using WPA can be effective, they have many inherent shortfalls in functionality when scaled to larger distributed networks. These flaws revolve around providing authentication, issuing new keys, and the signing of messages to ensure integrity - no distributed facility is available for another node on the network to perform these functions. Furthermore, WPA traditionally requires a fixed password used in the generation of a symmetric key. This complicates the ability to change the key in the event that a network has been compromised. With no standardized method to propagate the new password to nodes which use the old password, the network must be destroyed and recreated to change the key.

1.2 Our Solution

Our project addresses the challenges of distributed cryptography by creating an innovative new Distributed Secure Channel (DSC) that can operate on top of an existing insecure network infrastructure. The DSC project provides a means for a distributed, fault-tolerant, secure layer for communicating. It makes use of multiple cryptographic techniques to address the inherent weaknesses in existing solutions when they are applied to distributed networks.

In order to resolve the inherent performance issues associated with purely public key systems, a symmetric key is used for the sole purpose of encryption and decryption. However, the symmetric keys are securely generated using a CSPRNG at runtime, and are not specified in the source code. They are also generated internally and are never known or revealed to users of the system. The symmetric key is used to securely transmit traffic amongst trusted nodes without the overhead

requirement of having to re-encrypt messages using the public key of each trusted channel member.

To address the issue of authentication and man in the middle attacks, our solution makes use of an authentication passphrase. The secure channel authentication is not used to perform any encryption, it is only used in authenticating the identity of a node requesting access to a channel. The usage of an authentication passphrase makes it extremely difficult for an attacker to perform a man in the middle attack. The passphrase is used in creating a signed authentication request. It is also used to sign the node's public key when requesting a key exchange.

Our solution prevents the entire network from being compromised if an attacker knows the authentication password through the use of diminishing levels of access. In the event that the attacker learns of the authentication passphrase, the attacker must still be authenticated by each node in order to have complete access to the secure channel. In the event that the attacker is trusted by a subset of the nodes of the secure channel, the attacker will only pose a threat to the nodes that trusted it, rather than compromising the security of the entire secure channel. In the event that an already trusted node has accidentally authenticated an attacker, the trusted node can simply destroy their connection and re-authenticate themselves with the secure channel through a trusted channel member. They will be treated as new users connecting to the network for the first time from an untrusted node.

2 NOMENCLATURE

The following table includes common nomenclature which will be used throughout this report.

Table 1 - Secure Channel Nomenclature

ECC	Elliptic Curve Cryptography	A very hard computational problem that allows cryptographers to generate keys that cannot be easily solved.
ECDSA	Elliptic Curve Digital Signature Algorithm	Used to digitally sign an entity.
Stream Cipher	Cryptography Term	Generates a stream of random data used to encrypt a message.
Grain128	Cryptographic Algorithm	A new, high performance and secure stream cipher.
IV	Initialization Vector	Used to ensure a message is never encrypted the same way twice.
CSPRNG	Cryptographically Secure Pseudorandom Number Generator	A secure random number generator for cryptographic applications.
ISAAC	Cryptographic Algorithm	A stream cipher used to generate a highly secure random number stream. Finding a pattern is almost impossible.
SHA-256	Hashing Algorithm	A secure 256-bit cryptographic hashing algorithm.
MD5	Hashing Algorithm	A secure 128-bit cryptographic hashing algorithm.
MAC	Message Authentication Code	Used to validate the integrity of a message to ensure it hasn't been altered.

3 CORE CONCEPTS

3.1 SECURE CHANNEL

The concept of the secure channel is an essential concept of the project. The secure channel refers to a secure line of communication along an insecure medium. No outside party is able to eavesdrop or tamper with messages sent along the secure line. A secure channel can be created using existing insecure infrastructure. For example, a secure channel can leverage an existing network provided by an insecure public Wi-Fi hotspot.

3.1.1 Use Cases

The following use cases were identified for the DSS. There were also three actors identified.

All users can change their nicknames at any time.

Alice represents the initial channel creator. She has the capability of creating channels. While anyone can do this, in this model she is the sole creator of a test channel. She can also do any of the things that an ordinary trusted user can do - validate access for new users, and can send encrypted messages. She can also ignore malicious users.

Bob is a new, untrusted user. He can request access to existing channels, but he can't send messages or validate access. If he gets authenticated, he has the same role as Jim. At present, he cannot ignore anyone as he doesn't have access to the channel.

Jim is a friend of Alice; he is a trusted user, who can validate access for new users, and send encrypted messages. He can ignore malicious users.

Mallory is an attacker. She can do the same things as Bob, but she is a malicious user. Alice and Jim are likely to ignore her - at that point, she has no ability to request access to existing channels, because other users ignore her attempts.

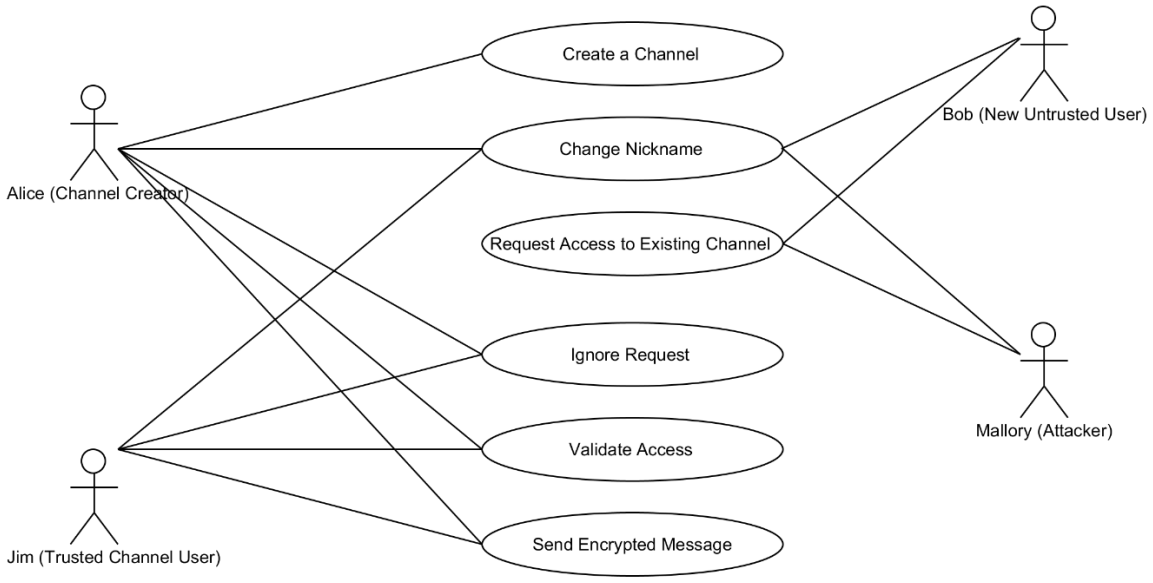

Figure 3 - UML Use Case Diagram

3.2 Public Key Verification

The project uses public key verification as a means of verifying public keys and authenticating a node requesting access to the secure channel. The channel authentication passphrase is used to verify the public key of nodes requesting access against the stored authentication passphrase on trusted nodes. This greatly reduces the risk of a man in the middle attack for nodes that are requesting authentication as the public key cannot be signed without using the correct channel authentication passphrase.

3.3 Public Key Signing

Public key signing using ECSDA is essential to provide a way of guaranteeing the authenticity of a node's public key. Similarly to public key verification, the channel authentication passphrase is used to sign the node's public key, greatly reducing the risk of a man in the middle attack during the key exchange process.

3.4 Authenticated Node Announcement

The authenticated node announcement is performed each time a trusted client announces a newly authenticated node. The announcement is performed to notify all trusted nodes when a new node's identity has been verified. This does not imply that the other nodes must authenticate the node, but makes it possible to track all nodes that are authenticated, even if they are trusted by only a limited number of nodes on the network.

3.5 Joining the Network (Authentication)

Any node wishing to join the network announces an authentication request to the secure channel, broadcast to all currently trusted nodes on the network. Each individual trusted node can then verify the untrusted client by verifying the signed request against the secure channel authentication passphrase.

3.6 Stream Cipher

A stream cipher is used to securely encrypt the data that is transmitted by the nodes of the secure channel. The Grain128 stream cipher is used as it supersedes other more widely used, and now deprecated stream ciphers, such as RC4.

3.7 Initialization Vector (Nonce)

An initialization vector is used to prevent attacks on the network. These include known plaintext attacks and message retransmission attacks. This is done by ensuring that each encrypted message is encrypted uniquely and sent using a unique numeric identifier.

3.8 Relay Chat Application

As a proof-of-concept application of our distributed secure channel, we implemented a relay chat (IRC) client. The chat client makes it possible to demonstrate sending secure messages over the distributed secure channel to trusted nodes on the network.

4 HIGH LEVEL SYSTEM REQUIREMENTS

The requirements for the system are broken down into functional and non-functional requirements as specified in the following table. In order to focus the efforts of the project on creating a working DSC, rather than focusing on the underlying peer-to-peer networking, the JGroups library was used to satisfy the networking requirements of the project. The project requirements were further revised from the initial proposal in order to reflect the use of JGroups for the underlying peer-to-peer networking and to facilitate blocking an attacker from joining the network.

Table 2 - High Level System Requirements

Requirement Name	Type	Description
1. Distributed Architecture	Non-functional	The system must be distributed without any central node that all other nodes are dependent on.
2. Secure Channel	Non-functional	The system must have the ability to create a secure communication channel that cannot be overheard or tampered with without the knowledge that messages have been tampered with.
2.1. Public Key Verification	Functional	The system must have a means of verifying the public key of new nodes attempting to join the network.
2.2. Public Key Signing	Functional	The system must have a means of signing the public key of a new node attempting to join the network.
2.3. Authenticated Node Announcement	Functional	The system must have a means of announcing each new authenticated node that has joined the network to all other nodes.
2.4 Node Rejection	Functional	The system must have support for permanently ignoring a node requesting access, such as an attacker.
3. Stream Cipher	Functional	The system must have a stream cipher to facilitate the encryption and decryption of information exchanged by nodes.
4. Relay Chat	Functional	One application of the network (for demonstration purposes) must show a simple Internet Relay Chat style client, which allows users to send messages to all trusted users subscribed to the network.

5 ARCHITECTURAL DESIGN

5.1 HIGH LEVEL SYSTEMS ARCHITECTURE

The following diagram illustrates the high level systems architecture, by breaking down the program components into logical units. These loosely reflect the internal mid-level architecture of the DSC program in terms of packages and classes. Actual classes and their internal members here are abstracted as descriptive entities. This is elaborated upon further in the mid-level design section.

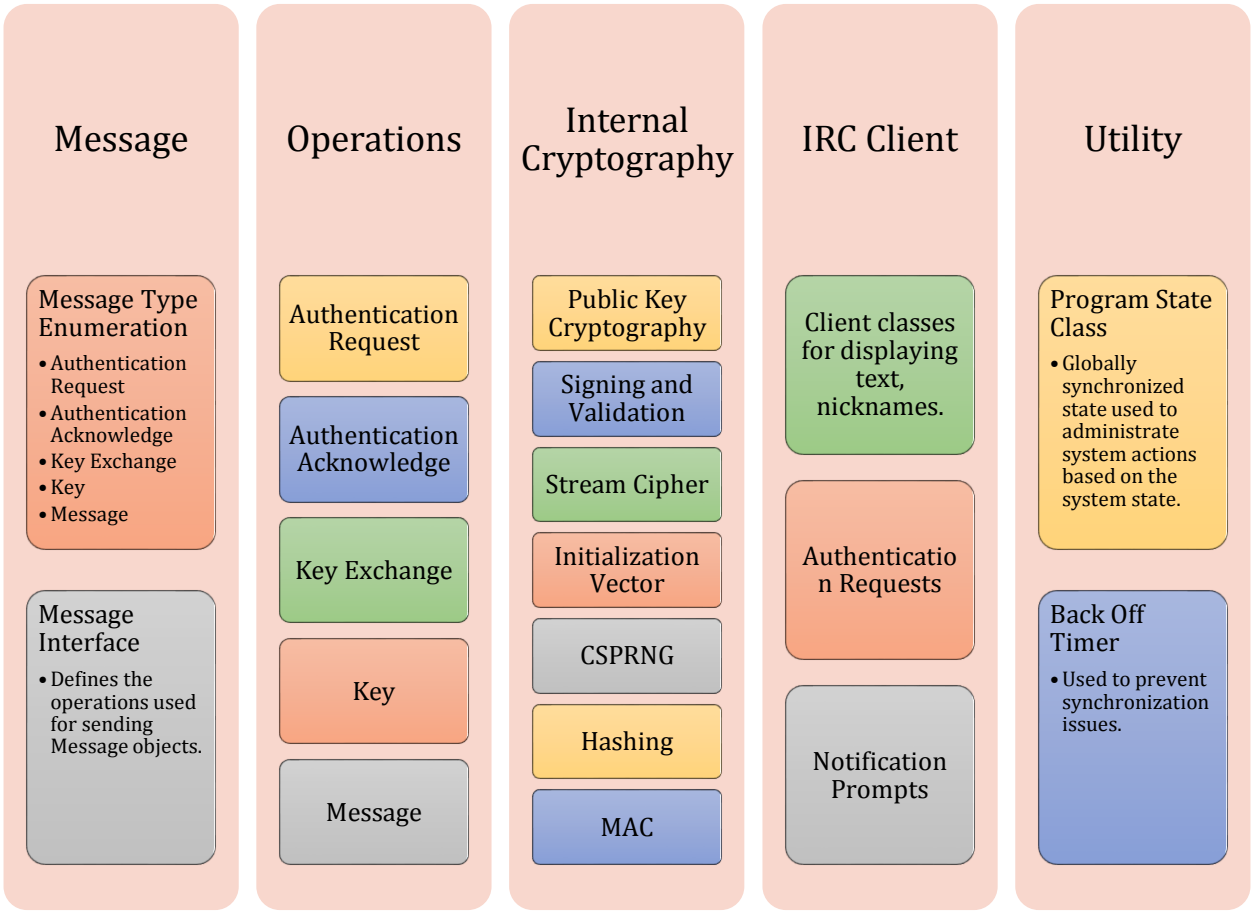

Figure 4 - High Level Systems Architecture Summary Diagram

5.2 MIDDLEWARE AND NETWORKING PLATFORM

JGroups was a central component of our design, providing event driven handling of network messages. It was used to simplify the process of serialization and reconstitution of network objects internal to the DSC program. It was also used to marshal, send, and de-marshal secure message objects. In general, it acts as an intermediary between the network stack and the application layer, handling all communications. The underlying encryption scheme protects the message contents from inspection and tampering from untrusted clients.

5.3 Cryptographic Components

All cryptographic components are listed in a hierarchical diagram which follows. This diagram illustrates the components which make up the public and symmetric key components of the DSC at the top level. This is broken down into types of cryptographic components, and at the bottom level, cryptographic primitives.

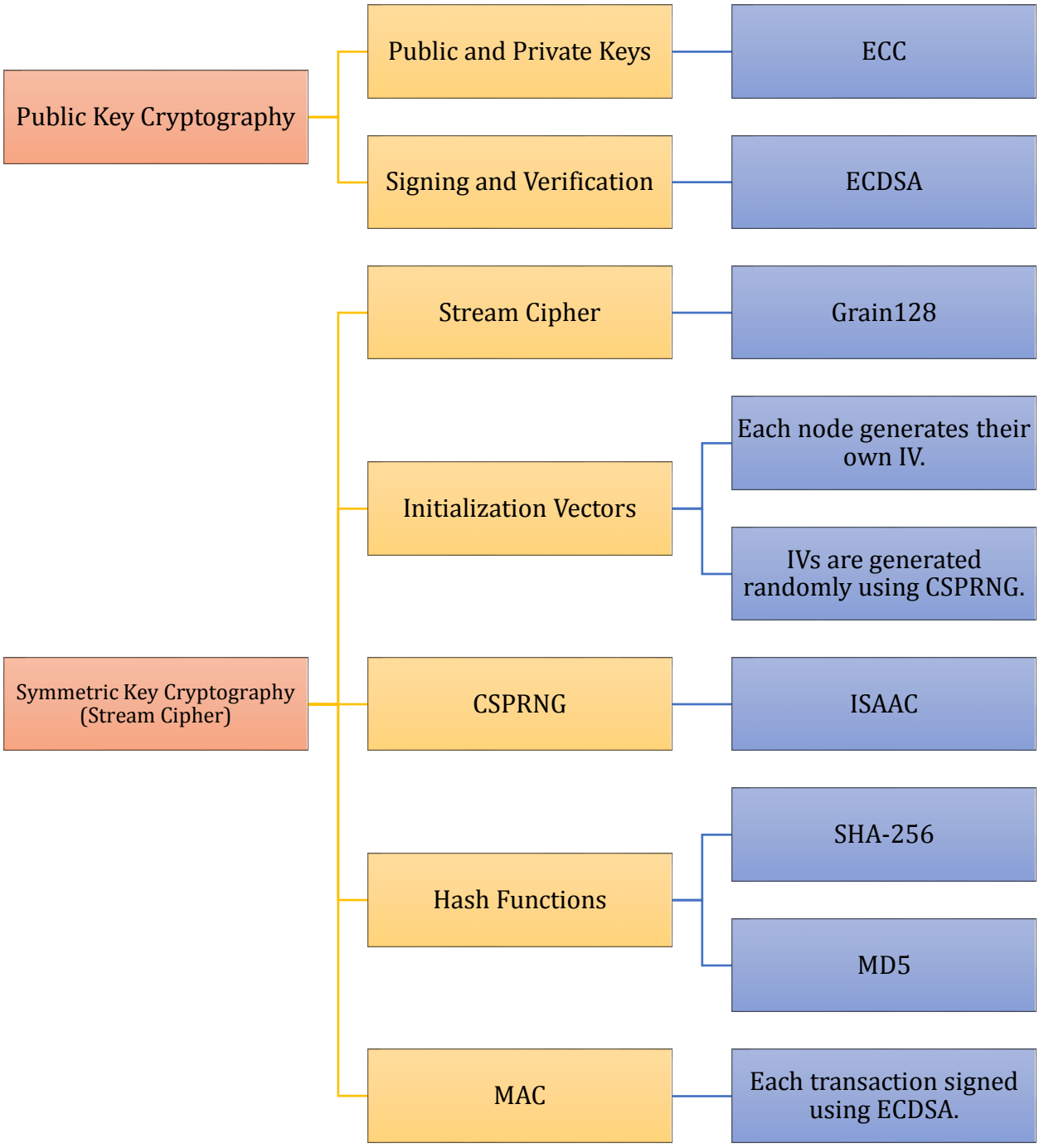

Figure 5 - Cryptographic Components Hierarchy

5.4 Software Component Mid-Level Design

This section details the mid-level design components used in the project architecture.

5.4.1 Messages

See Figure 34 - com.DSC.message; Messages Package UML Class Diagram.

The message package consists of the software components related to the generation of messages, transmitted as serializable objects by JGroups. An abstract factory design pattern is used to simplify the creation of the various secure message objects. All of the secure messages that can be transmitted (AuthAcknowledge, AuthRequest, EncryptedMessage, Key, and KeyExchange), implement the SecureMessage interface which specifies that they must return an object conforming to their message type. The list of available message types is specified by the MessageType enumeration. Each of the messages has a static constant member which is an instance of one of the available message types defined in the enumeration.

5.4.2 Internal Message Handling Operations

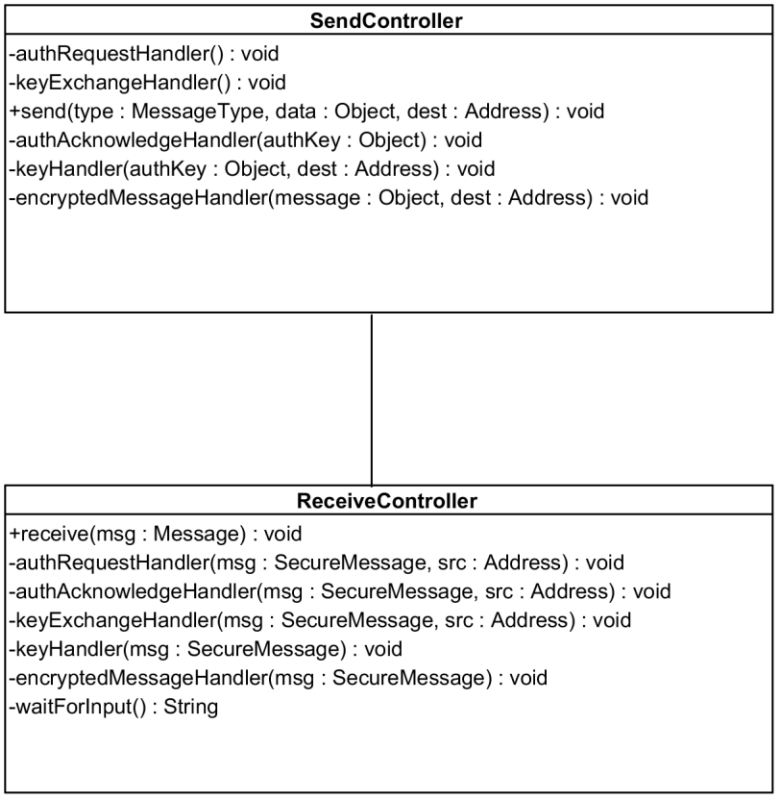

Figure 6 - com.DSC.controller; Controller Package UML Class Diagram

The controller package, seen in Figure 6, handles all internal operations. It consists of the Send and Receive controllers, whose logic and control flow extends to the other packages internally. Leveraging the event driven nature of JGroups, these controllers are used by the underlying middleware for callback operations in processing messages that are sent and received. The Send and Receive controllers have corresponding public send and receive methods, which are invoked based on the type of message the client is handling. These controllers are responsible for handling

all of the messages that are sent and received by the client. They initiate all of the necessary cryptographic operations related to signing, verification, encryption, and decryption.

5.4.3 Cryptography

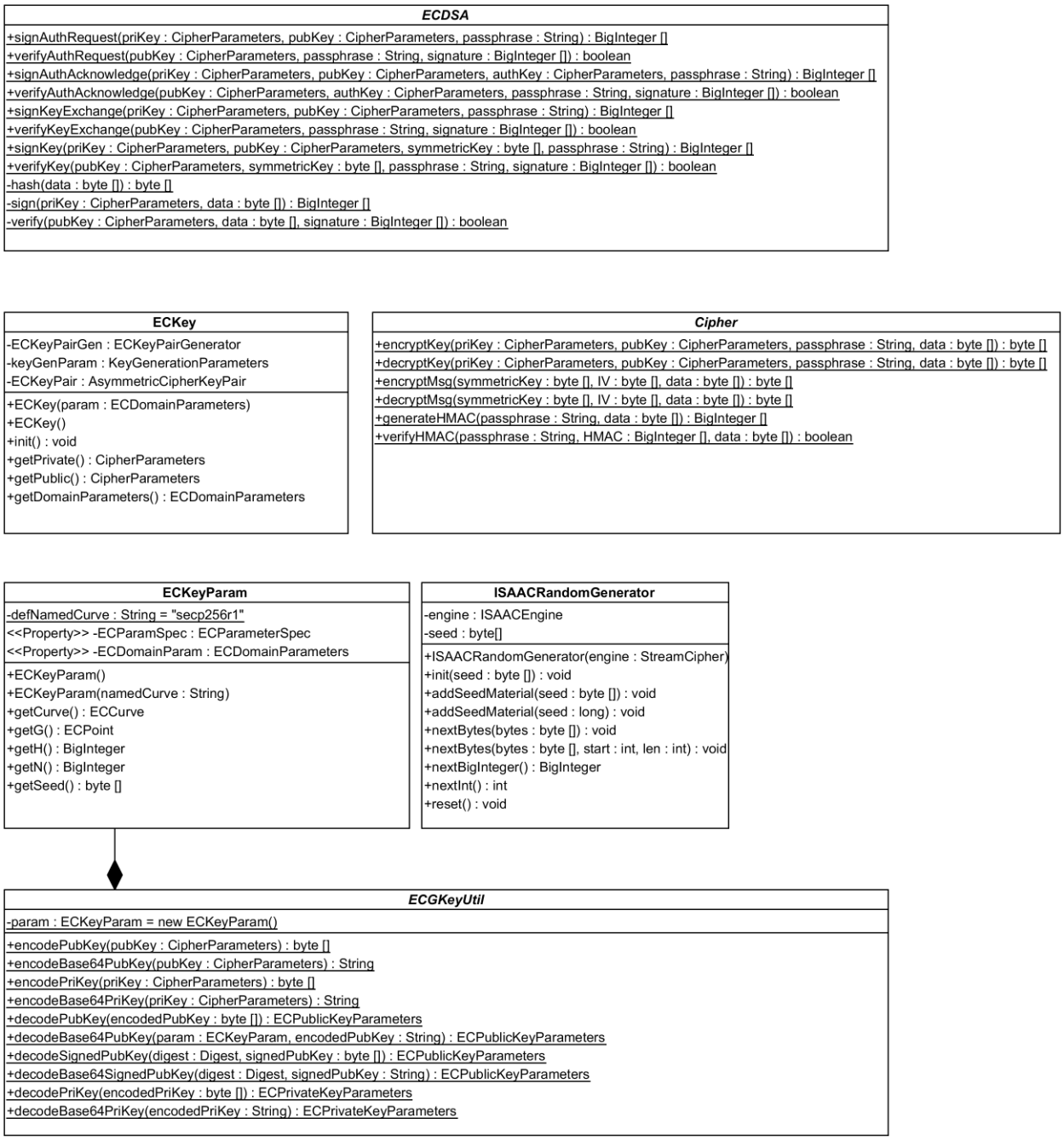

Figure 7 - com.DSC.crypto; Cryptography Package UML Class Diagram

The cryptography package (Figure 7) consists of software components related to the cryptography used in the system. Each type of cryptographic operation is contained within its own class. There are also abstract classes which contain static methods used as facades for simplifying certain repetitive operations. The generation of public keys is handled by the ECKey and ECKeyParam

classes and common operations are provided by the abstract ECGKeyUtil helper class. The signing and verification operations using ECDSA are implemented using the abstract class ECDSA. It provides helper methods for the various operations. Lastly, the IV (nonce) generation and stream cipher operations are provided by the ISAACRandomGenerator, which generates unique IVs. Stream cipher encryption and decryption is handled by the abstract Cipher class, which provides helper methods for encryption and decryption operations.

5.4.4 Relay Chat Client Application

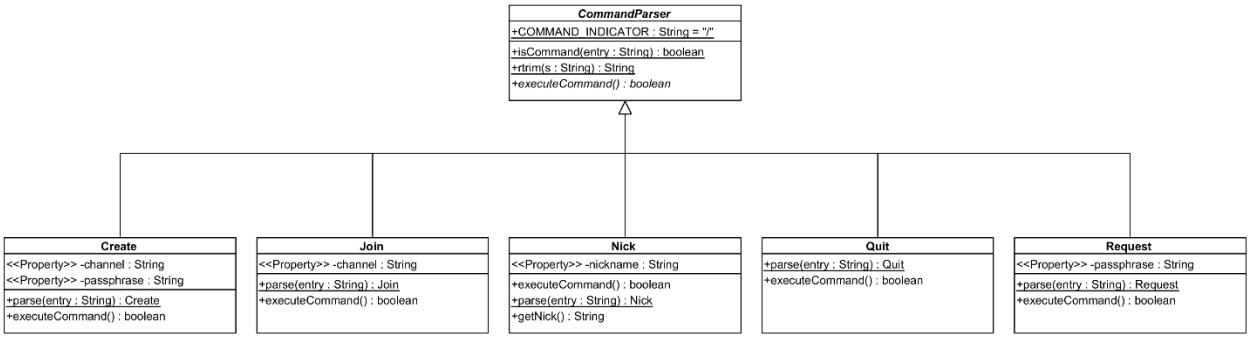

Figure 8 - com.DSC.chat; Chat Package UML Class Diagram

The chat package, seen in Figure 8, contains the software components used for parsing and executing the various IRC commands. All of the available commands are represented by a class which inherit from the command parser the logic for parsing the input for the command from the terminal.

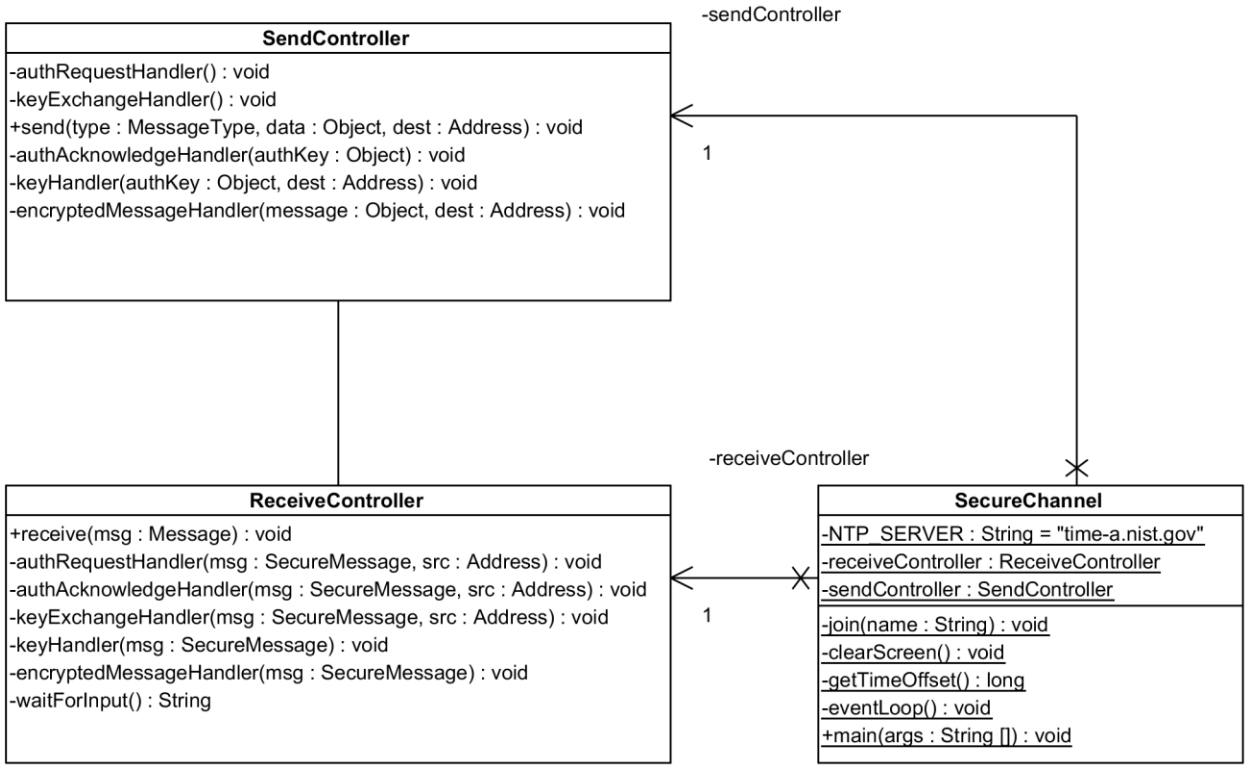

Figure 9 - com.DSC.client; Client Package UML Class Diagram

The client package, seen in Figure 34, contains the SecureChannel class, which implements the main driver method for the application. The SecureChannel class is responsible for synchronizing the clocks of all nodes using Network Time Protocol, generating the public and private key pairs, starting the interactive terminal client, and initializing JGroups with the send and receive controllers for handling messages.

5.4.5 Utility

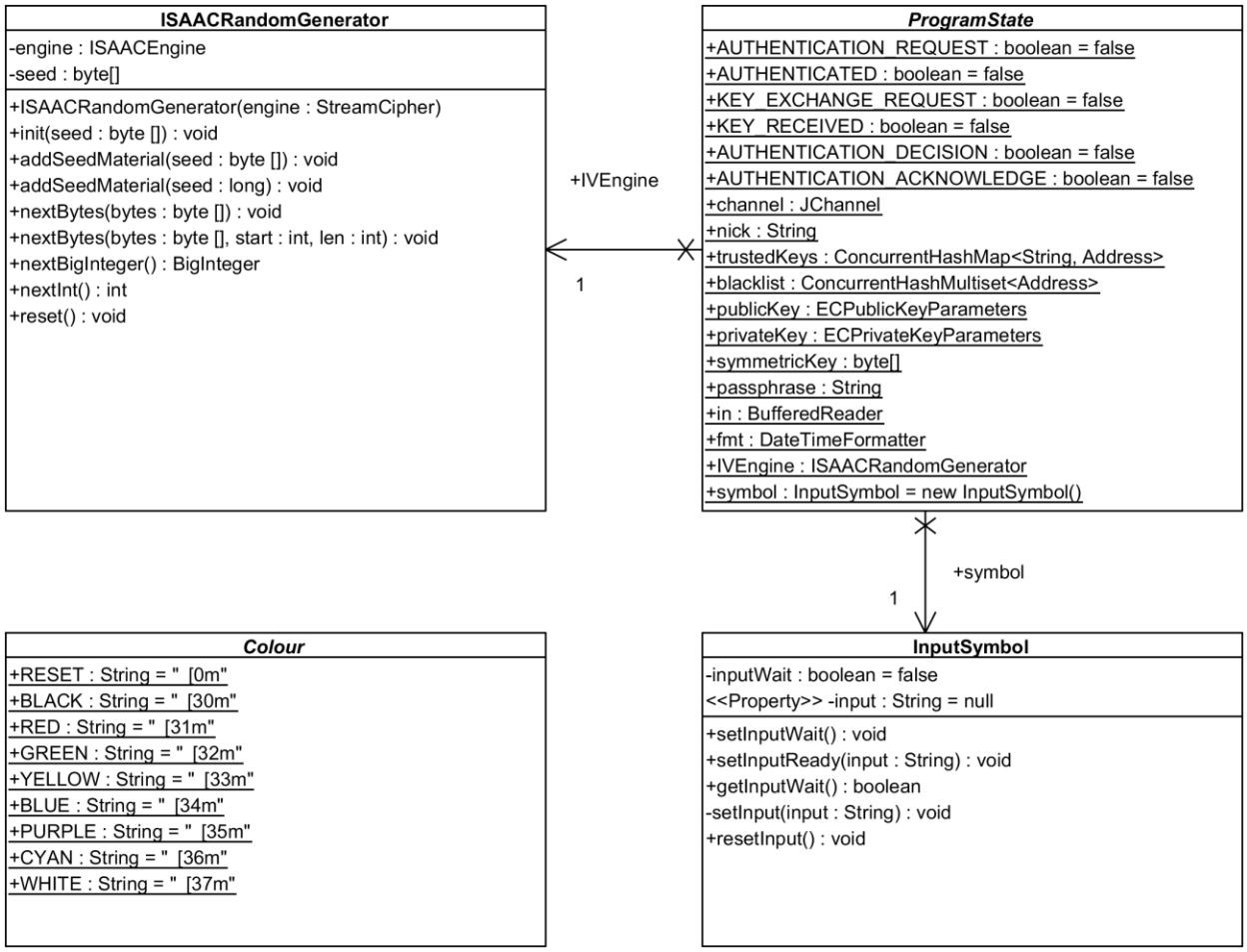

Figure 10 - com.DSC.utility; Utility Package UML Class Diagram

The Utility package (Figure 10) consists of utility software components. It contains the ProgramState abstract class. This operates as a statically accessible global state object that is used by the program to check the current state of the application when performing operations. This was an important design choice used to facilitate the appropriate handling of send and receive operations in a concurrent, event driven manner that is compatible with JGroups.

5.5 Communication and Application Protocols

DSC communication protocol relies on JGroups "reliable UDP". JGroups internal methods are used to facilitate the process of serializing objects for transmission. All messages implement the SecureMessage interface, which requires that they provide a getter method to determine the message type. Each message object contains a static constant identifier, which is one of the

enumerated items defined in the MessageType identifier. These are used in order to appropriately typecast the received secure messages into the appropriate type of message object upon being received.

The protocol has two types of communication methods: multicast and unicast. Each is used depending on the type of message that is sent. Any request or acknowledgement message is multicast. They are made available by broadcasts to the entire secure channel. The authentication request, key exchange request, and authentication acknowledge are all messages that are multicast to the entire secure channel.

Encrypted and symmetric key messages are unicast private network communications that are only sent to authenticated, trusted nodes. They are sent directly using unicast rather than being broadcast to the entire secure channel due to their private nature.

The following tables describe each of the individual message protocols required by the DSC communication protocol. The specific message identifier for each type of message are specified in parentheses. Any message that does not exactly match one of these following messages is automatically discarded. Any message that has an invalid signature is likewise discarded.

Table 3 - Authentication Request Message

Message Type Identifier (AUTH_REQUEST)
Client's Public Key
Message Signature

Table 4 - Authentication Acknowledge Message

Message Type Identifier (AUTH_ACKNOWLEDGE)
Authenticator's Public Key
Authorized Client's Public Key
Message Signature

Table 5 - Key Exchange Message

Message Type Identifier (KEY_EXCHANGE)
Client's Public Key
Message Signature

Table 6 - Symmetric Key Message

Message Type Identifier (KEY)
Sender's Public Key
Encrypted Symmetric Key
Message Signature

Table 7 - Encrypted Message

Message Type Identifier (ENCRYPTED_MESSAGE)
Initialization Vector
Encrypted Message Content

Daniel Smullen
Jonathan Gillett
Joseph Heron

6 BEHAVIORAL DESIGN

This section describes the behavior of the system, including system states, state transitions, and the various communications processes required for joining and participating in the network.

6.1 SYSTEM STATES

The behaviour of the system is defined based on one of the following states as shown in the following table.

Table 8 - System States

AUTHENTICATED
AUTHENTICATION_REQUEST
AUTHENTICATION_DECISION
AUTHENTICATION_ACKNOWLEDGE
KEY_EXCHANGE_REQUEST
KEY_RECEIVED
NULL

The state diagram that captures the full system behavior is visible in Figure 11.

These state definitions are used extensively by the send and receive controllers in processing and handling messages that are sent and received. The system can be in more than one state simultaneously, however the AUTHENTICATED and AUTHENTICATION_DECISIONS states can never be entered unless a client is authenticated.

Any client that is trusted remains in the AUTHENTICATED state. Any private message that is received when in the AUTHENTICATED state is processed. A client that has not been authenticated can never be in the AUTHENTICATED state unless it is becomes authenticated. Any client requesting authentication remains in the AUTHENTICATION_REQUEST state until the request times out (and returns to the NULL state), or is acknowledged. A trusted client that is validating a request is in the AUTHENTICATION_DECISION state. While in this state, subsequent requests are ignored. If the node requesting authentication is valid and acknowledged the trusted client then leaves the AUTHENTICATION_DECISION state and transitions into the AUTHENTICATED state.

If the client requesting authentication sees an acknowledgement broadcast it then enters the AUTHENTICATION_ACKNOWLEDGE state then enters the KEY_EXCHANGE_REQUEST state and broadcasts a request for a key exchange until the request times out or a key is received. While in this state, if the client receives a key it then enters the KEY_RECEIVED state, if the client successfully can join the secure channel it then finally enters the AUTHENTICATED state.

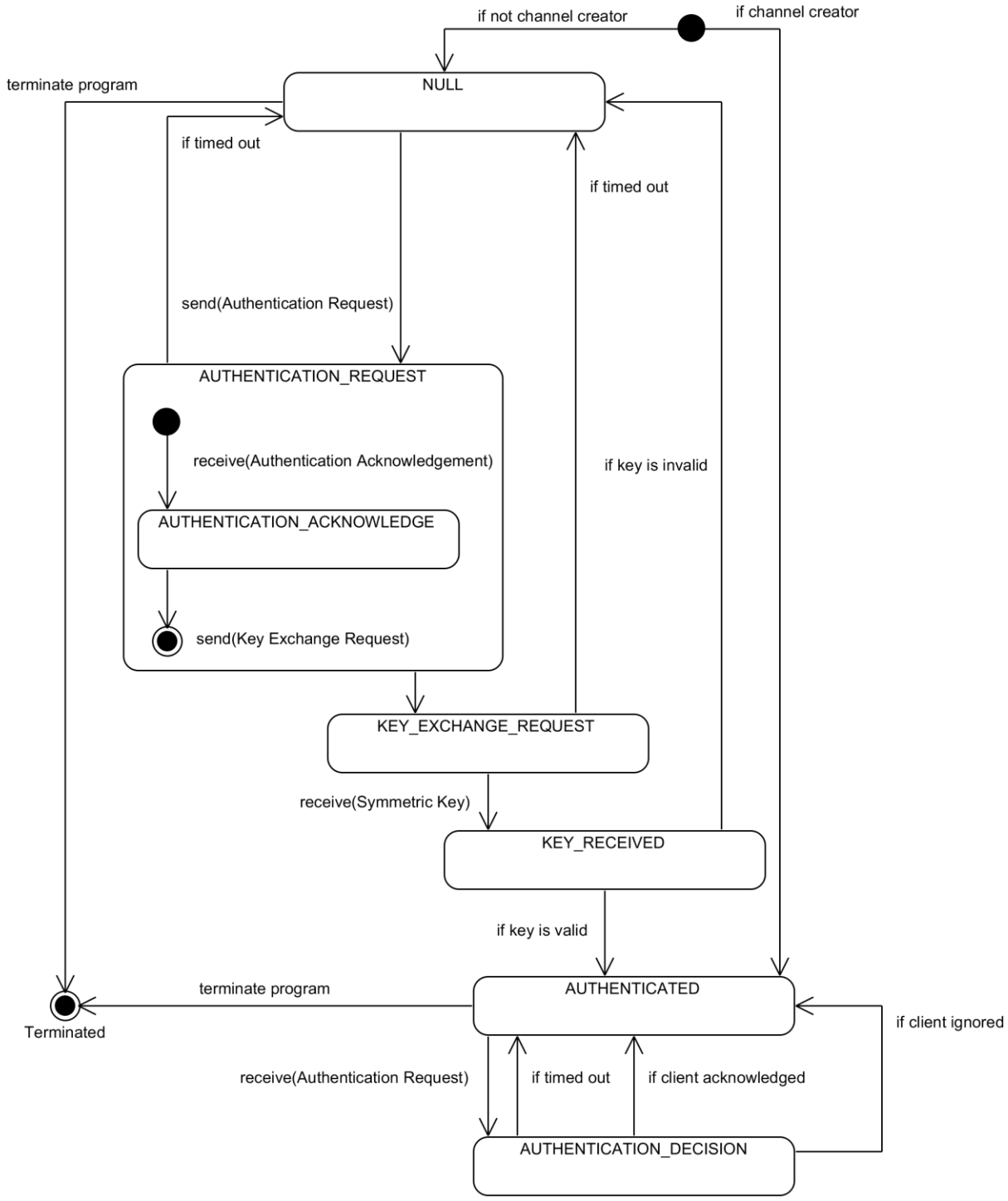

Figure 11 - Global System States UML State Diagram

6.2 AUTHENTICATION REQUEST

The following diagram demonstrates the initial authentication process. Bob broadcasts an authentication request to the secure channel (Figure 12). Both Alice (the channel creator) and Jim (an already authenticated client trusted by Alice) receive the request, seen in Figure 13. Only Alice responds to Bob's request. Jim has not responded to the request and ignores Bob.

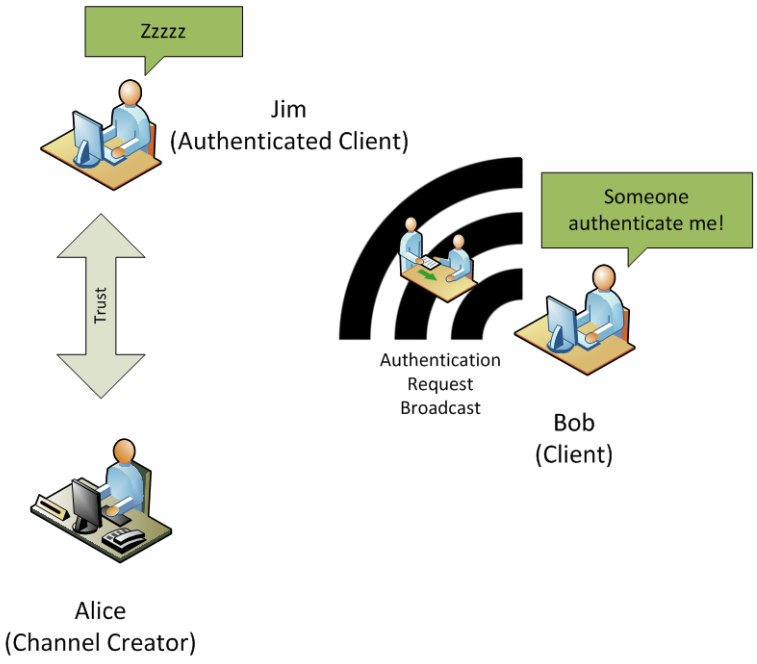

Figure 12 - Authentication Request Broadcast

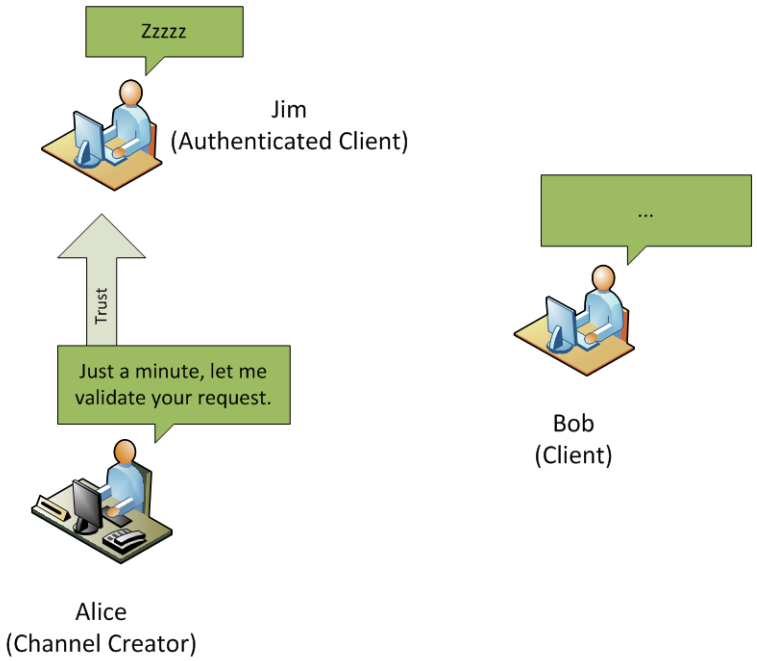

Figure 13 - Authentication Broadcast Received

The full message sequence can be seen in the following sequence diagram (Figure 14).

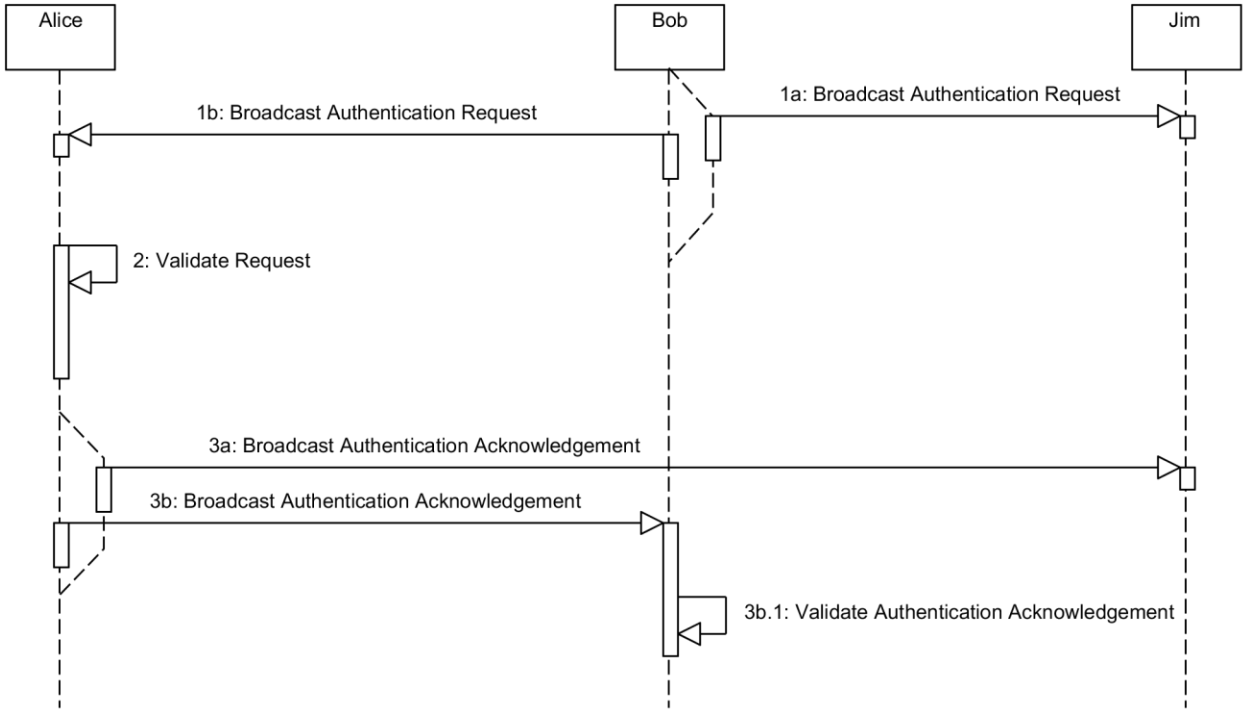

Figure 14 - Authentication Request UML Sequence Diagram

6.2.1 Authentication Announcement

Figure 15 demonstrates the authentication announcement process. Alice has received Bob's request and validated the authenticity of Bob. Alice then broadcasts her acknowledgement of Bob as trusted. Jim, who is already trusted by Alice, continues to ignore Bob.

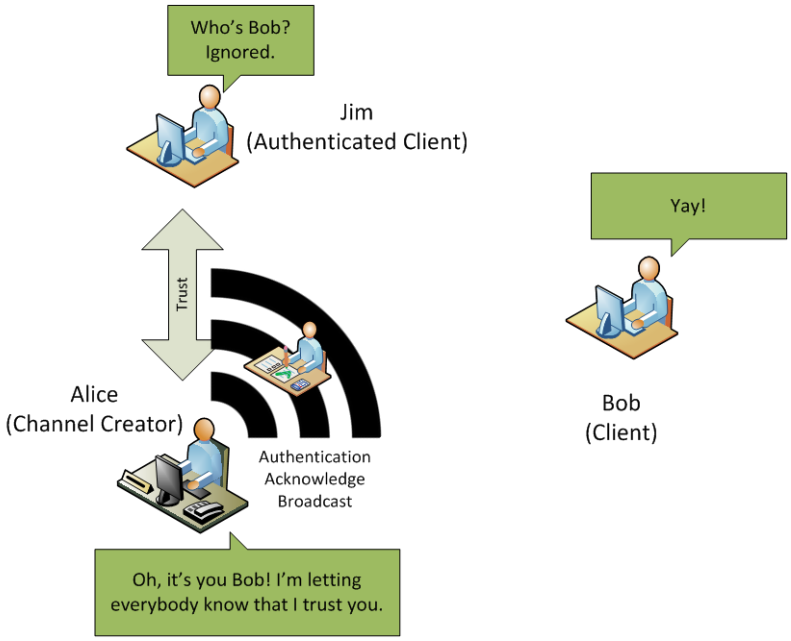

Figure 15 - Authentication Announcement Broadcast

6.3 KEY EXCHANGE

Figure 16, Figure 17 and Figure 18 demonstrate the key exchange process. Now that Bob has been authenticated by Alice, he sends a broadcast request to the secure channel requesting a key exchange.

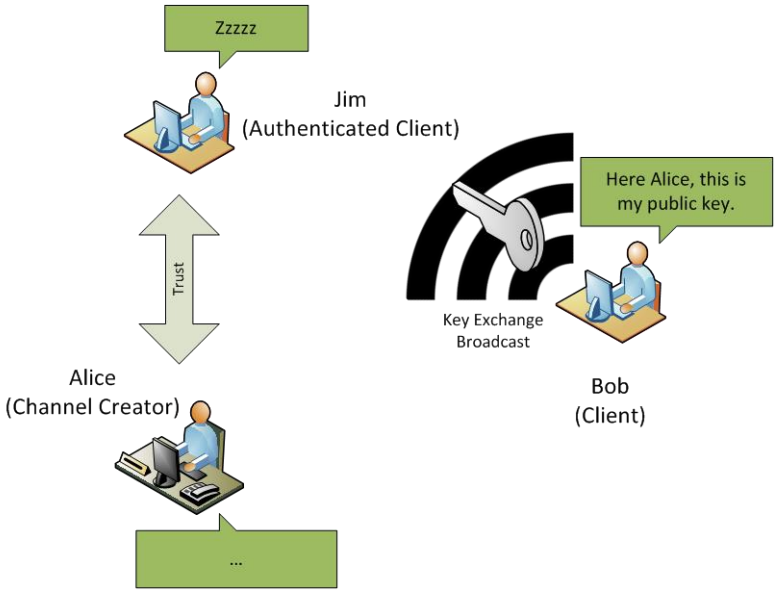

Figure 16 - Key Exchange Request Broadcast

In Figure 17, Alice (who trusts Bob) accepts the key exchange request and validates Bob's key. The key is signed using the channel authentication passphrase to prevent a potential man in the middle attack. Jim, who never trusted Bob, continues to ignore any broadcast requests made by Bob.

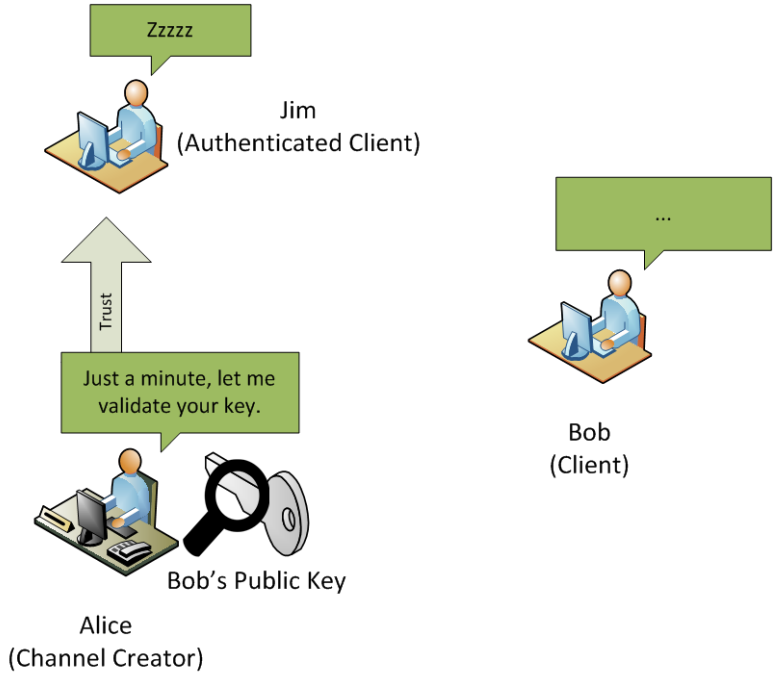

Figure 17 - Public Key Validation

Daniel Smullen
Jonathan Gillett
Joseph Heron

Finally, in Figure 18 Alice responds to the key exchange request by encrypting the network key using Bob's public key. This ensures that the network key is securely transmitted and can only be decrypted by Bob. Alice sends the key directly to Bob using unicast. The key is not made public to the network.

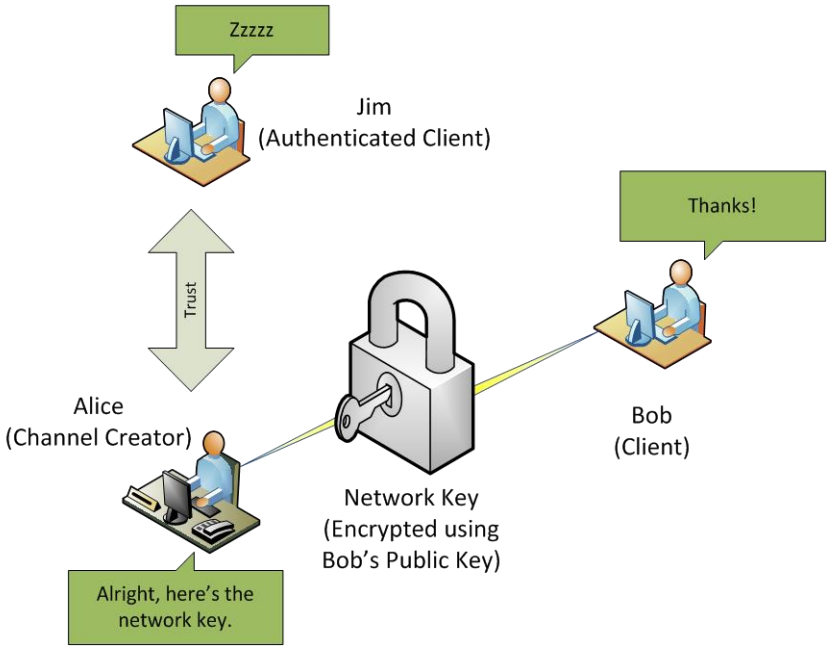

Figure 18 - Sending an Encrypted Symmetric Key

The entire process is illustrated in the following sequence diagram (Figure 19).

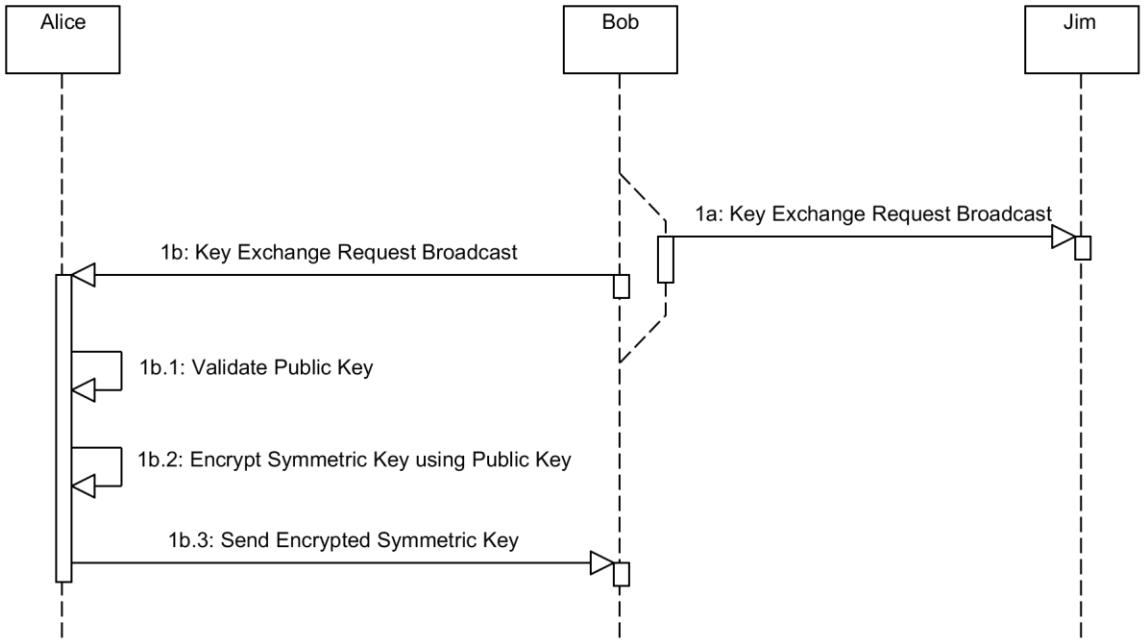

Figure 19 - Key Exchange Procedure UML Sequence Diagram

Daniel Smullen
Jonathan Gillett
Joseph Heron

6.4 AUTHENTICATED STATE

Figure 20 demonstrates secure channel communication. Bob is now a member of the secure channel. However, Bob only has limited access. Bob is only able to communicate with Alice. Jim never trusted Bob - as a result there is no communication possible between Bob or Jim.

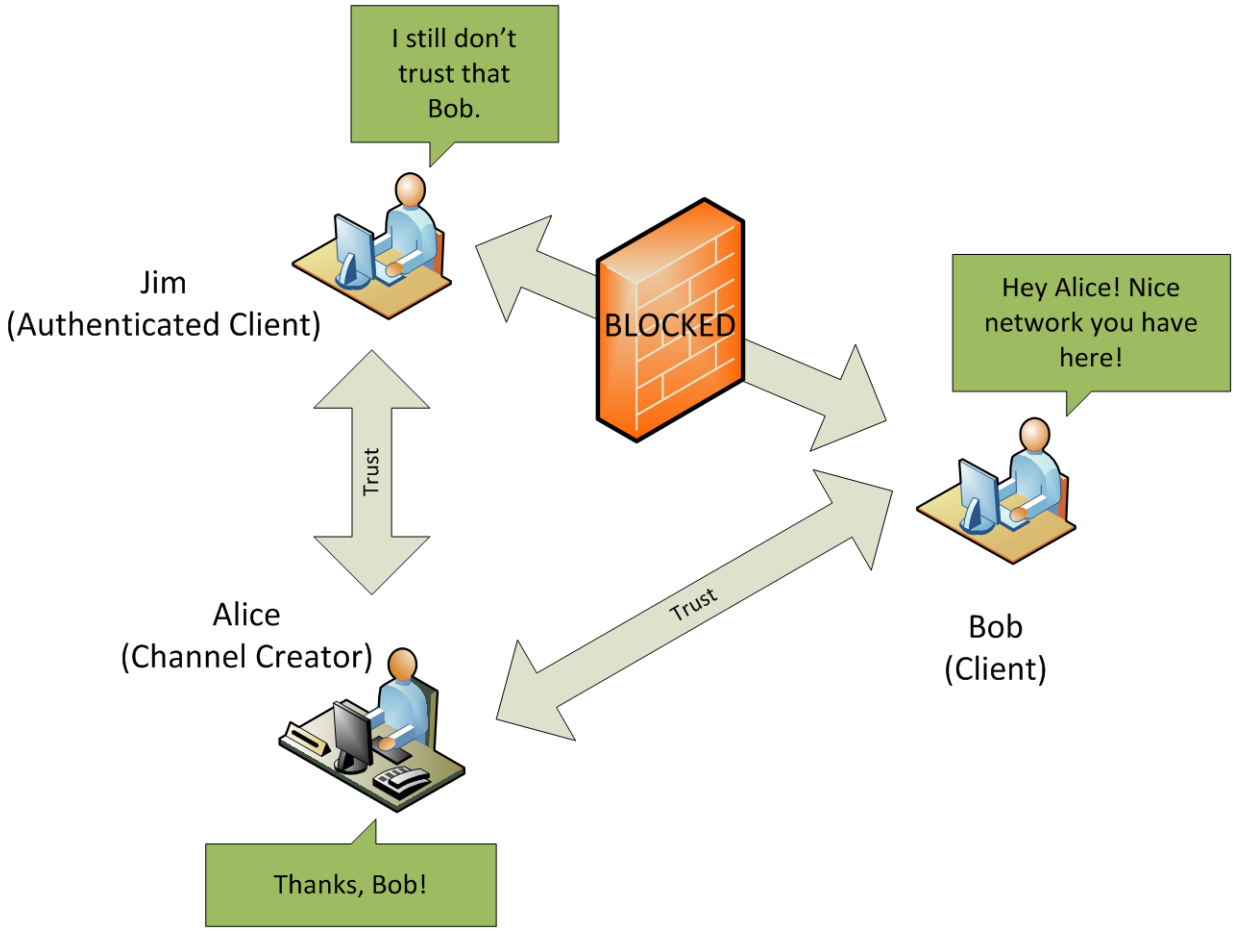

Figure 20 - Authenticated Communication

Daniel Smullen
Jonathan Gillett
Joseph Heron

The activity diagram in Figure 21 illustrates the entire process flow, from receiving a message to the actual outcome. This is required in order to process any message received.

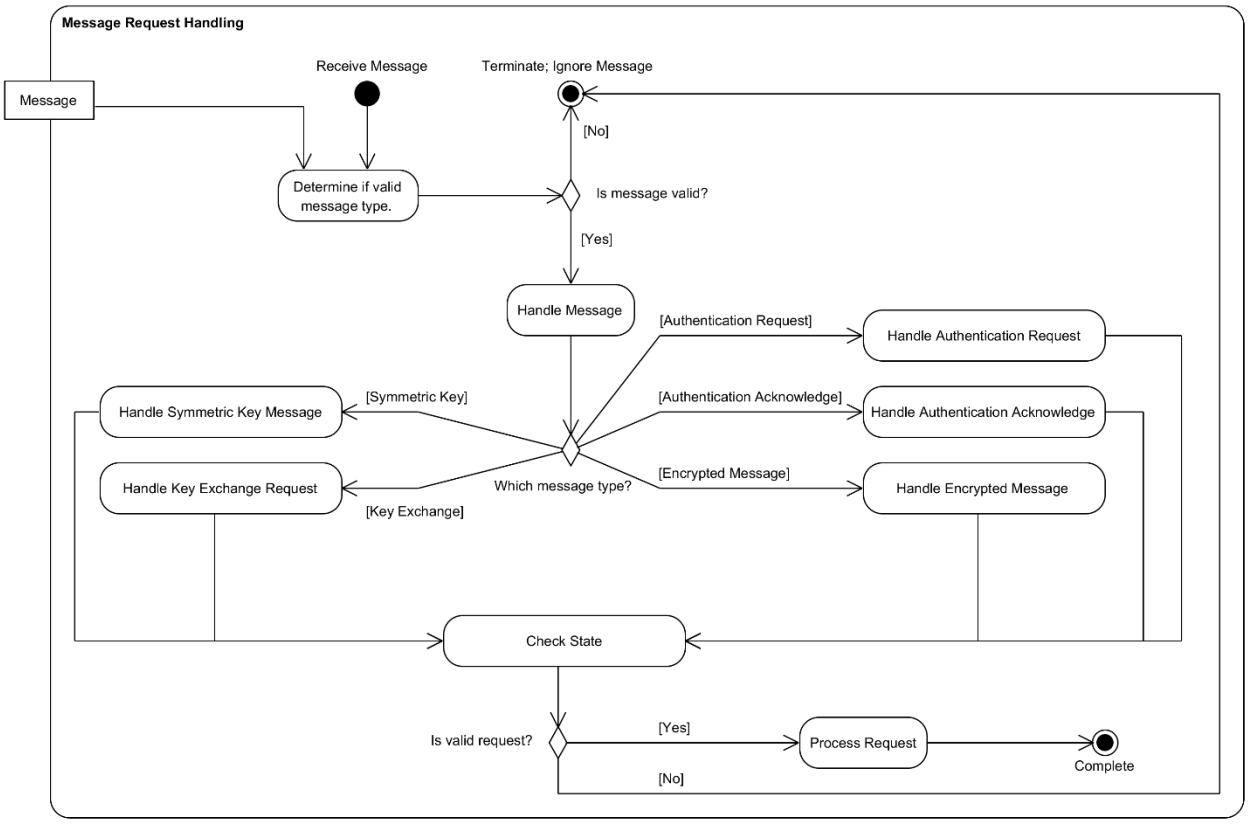

Figure 21 - Request Handling UML Activity Diagram

7 APPLICATION DEMONSTRATION

Included with the submission is a README file with instructions on how to execute the application for demonstration purposes. For ease of use, it is recommended to use the included JAR file. The JAR file has been extensively tested on the latest version of Ubuntu Linux, but should also work under Windows. Due to the firewall restrictions on Windows and Linux, it is easiest to test the use of the application on the same machine by simply running two separate instances of the application on the local host.

7.1 NORMAL OPERATION DEMONSTRATION

For a video recording demonstrating the application when used under normal operations, please see the included video main_demo.mp4 in the Videos directory.

Figure 22, Figure 23, Figure 24, and Figure 25 are a selection of important instances in the videos followed by a detailed explanation of what is happening at that stage in the operation of the application.

Figure 22 - Requesting Access, Invalid Passphrase

In Figure 23, Alice has created a secure channel using the authentication "test". Bob has joined the channel and is attempting to be authenticated. Bob has accidentally entered the incorrect authentication passphrase "test123", signed his key using this authentication, and broadcast his request to the channel.

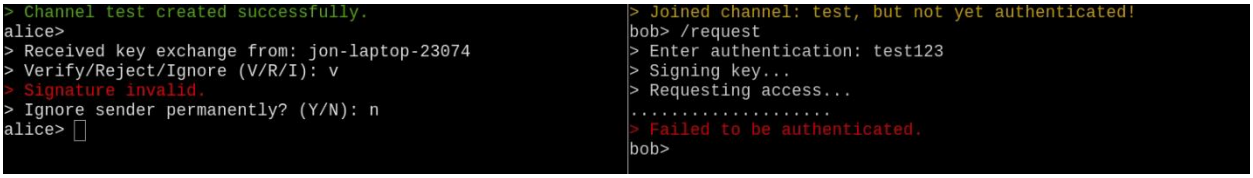

Figure 23 - Requesting Access, Invalid Signature

Alice verifies the signature, it is invalid. Bob did not have the correct authentication. Alice decides not to ignore Bob permanently. Bob's request eventually times out, and he was unable to be authenticated (Figure 23).

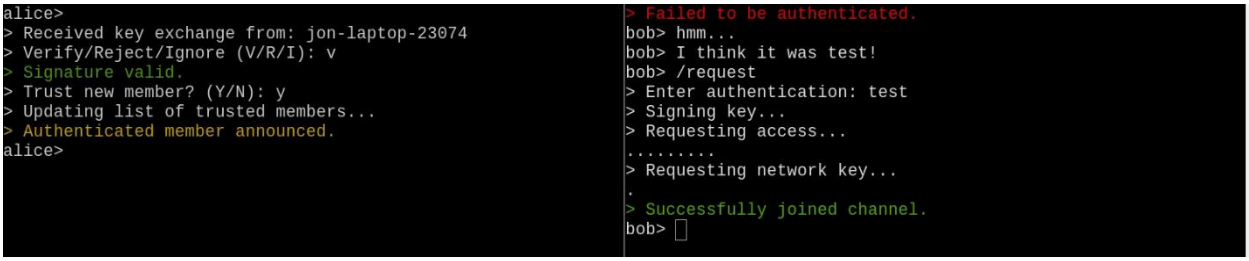

Figure 24 - Successful Authentication

After failing to be authenticated, Bob recalls what the correct authentication was. Bob broadcasts another request using the correct authentication, "test". Alice verifies his signature. This time it is valid as Bob signed it with the correct authentication passphrase. Alice trusts Bob and announces her decision to the network. Bob sees that Alice trusts him and requests the network key from Alice. Alice encrypts the network key using Bob's public key and sends it to Bob. Bob receives the key and successfully joins the network (Figure 24).

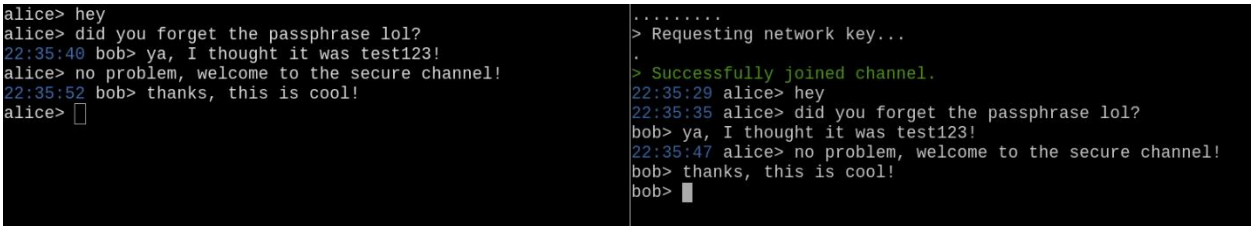

Figure 25 - Encrypted Communication with Bob and Alice

Now that Bob has the network key, he is able to communicate securely with Alice (Figure 25). All conversations they have are now encrypted and signed, preventing eavesdropping and tampering.

7.2 ATTACKS

For a video recording demonstrating the operation of the application when an attacker is trying to gain access, please see the included video attacker.mp4 in the Videos directory.

Figure 26, Figure 27, Figure 28, and Figure 29 are a selection of important instances in the videos followed by a detailed explanation of what is happening at that stage in the operation of the application.

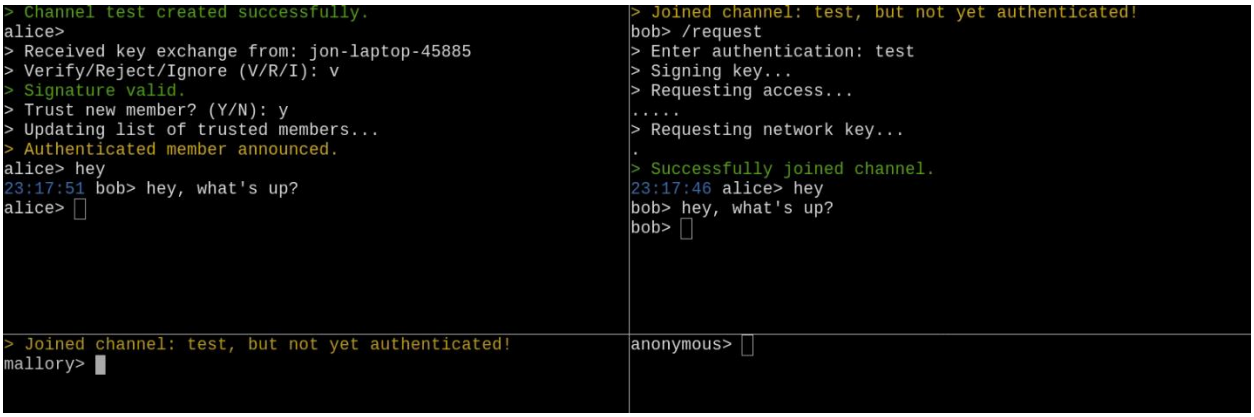

Figure 26 - Attacker Mallory

In Figure 26, Alice and Bob already trust each other and are communicating securely. Attacker Mallory is attempting to attack the network.

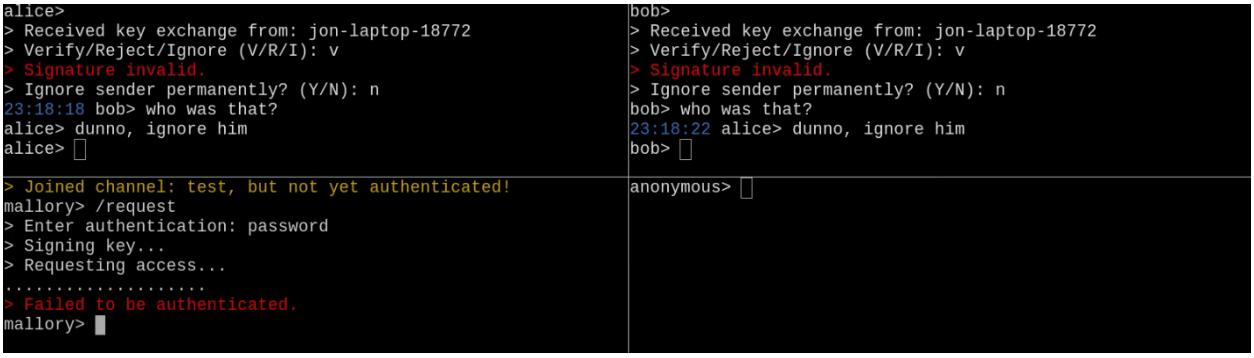

Figure 27 - Attempted Brute Force Authentication

Mallory is attempting to gain access to the secure channel by guessing the authentication (Figure 27). He submits a request using the authentication "password". Alice and Bob both receive the authentication request, but the signature is invalid. They both do not ignore Mallory permanently, and Mallory's request to be authenticated fails.

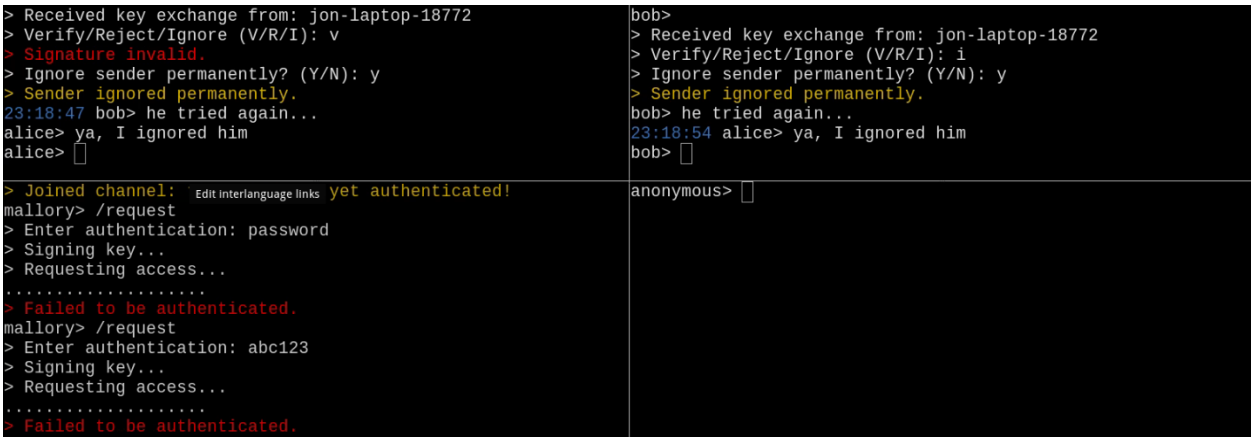

Figure 28 - Attacker is Permanently Ignored

In Figure 28, Mallory attempts to gain access again (through brute force), by guessing the authentication repeatedly. He submits a request using the authentication "abc123". Alice and Bob both receive the authentication request, but the signature is invalid. They both ignore Mallory permanently, and Mallory's request to be authenticated fails.

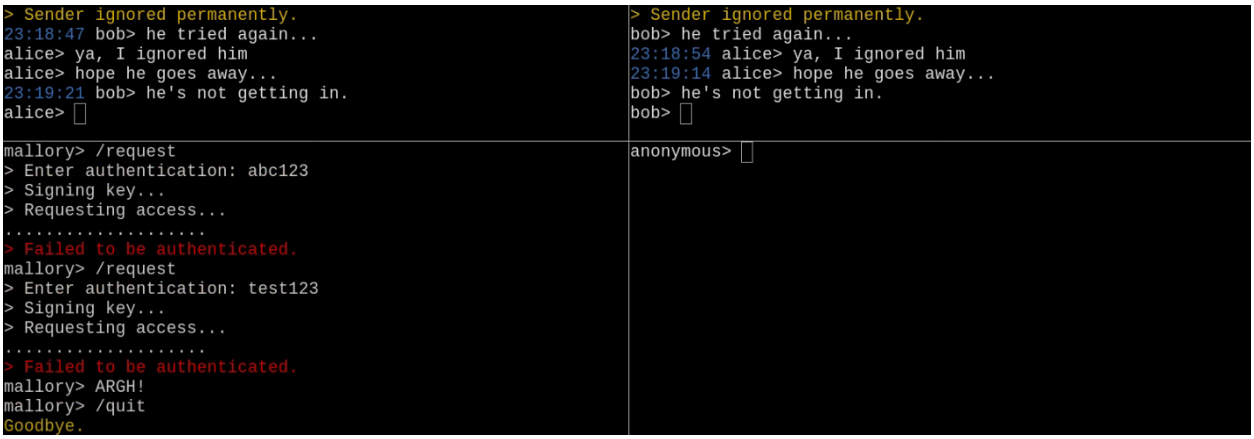

Figure 29 - Attacker Gives Up

Mallory has been permanently ignored by Alice and Bob. All future requests made trying to be authenticated on the network are ignored. Mallory quits in aggravation (Figure 29).

7.3 Diminishing Access

For a video recording demonstrating the diminishing access capabilities of the application, please see the included video diminishing_access.mp4 in the Videos directory.

Figure 30, Figure 31, Figure 32, and Figure 33 are a selection of important instances in the videos followed by a detailed explanation of what is happening at that stage in the operation of the application.

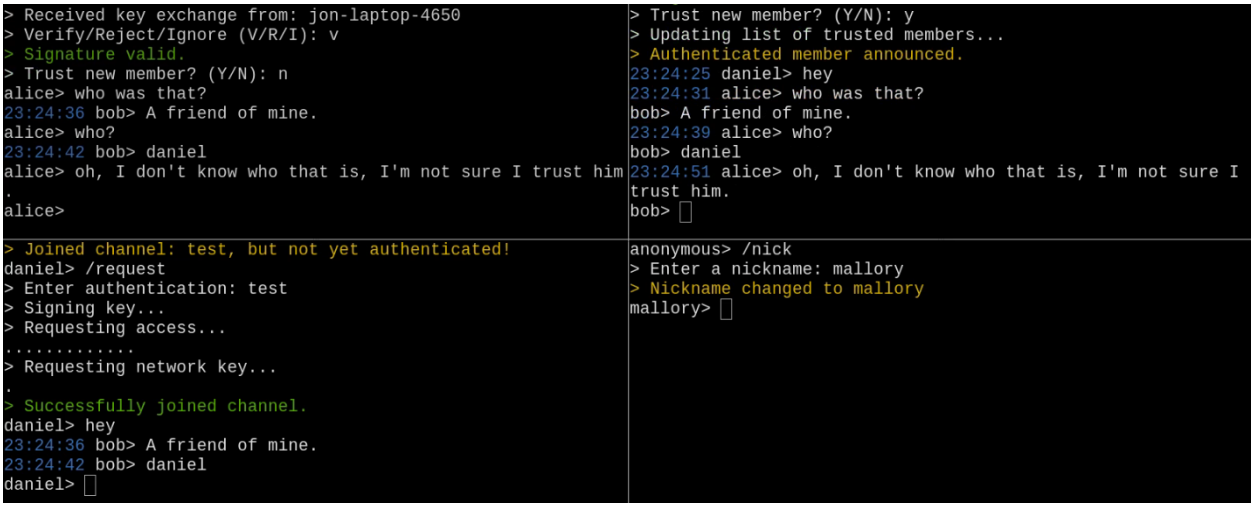

Figure 30 - Introducing Alice, Bob, and Daniel (Limited Trust)

In Figure 30, Alice and Bob already trust each other and are communicating securely. A friend of Bob's, Daniel (whom Alice does not trust), requests access. Bob authenticates and trusts Daniel, but

Alice does not. Daniel joins with limited access. He is trusted only by Bob, but not the whole network. No communication takes place between Alice and Daniel.

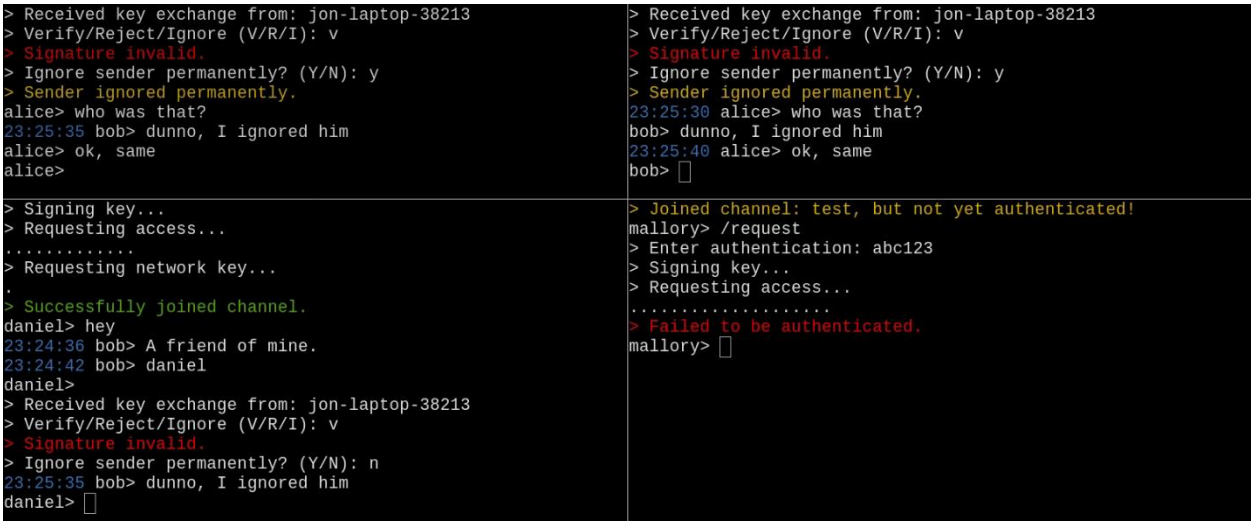

Figure 31 - Attacker Mallory Fails to Authenticate

An attacker, Mallory, attempts to gain access by guessing the authentication passphrase (Figure 31). He submits a request using the passphrase "abc123". All authenticated members of the network receive a request. Alice and Bob permanently ignore the attacker, but Daniel does not permanently ignore him.

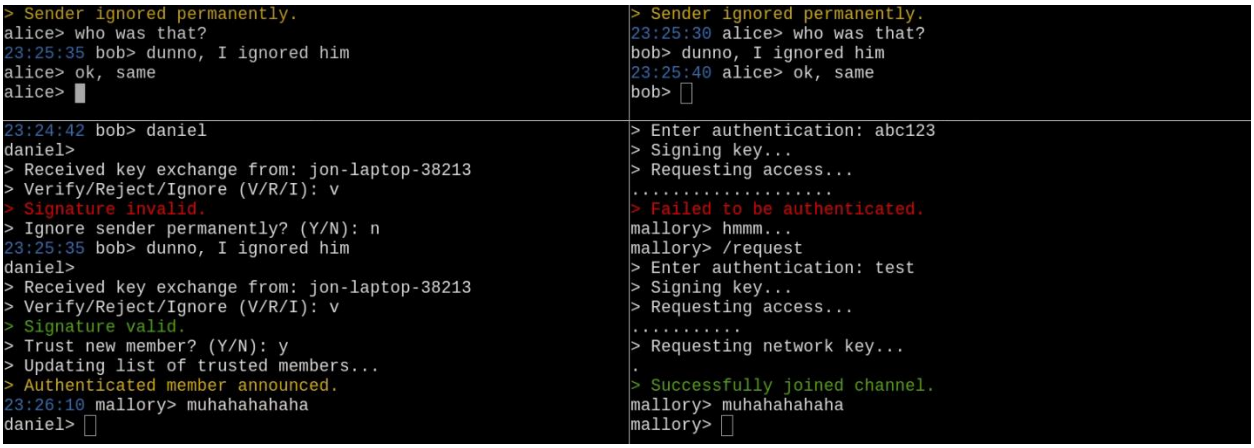

Figure 32 - Mallory Gains Brute-Force Access, Subverting Daniel

The attacker, Mallory, attempts to gain authentication by guessing the authentication repeatedly through brute force (Figure 32). This time he guesses the correct authentication "test". Mallory's request is permanently ignored by Alice and Bob, however the limited trust user Daniel makes the mistake of authenticating the attacker on the network.

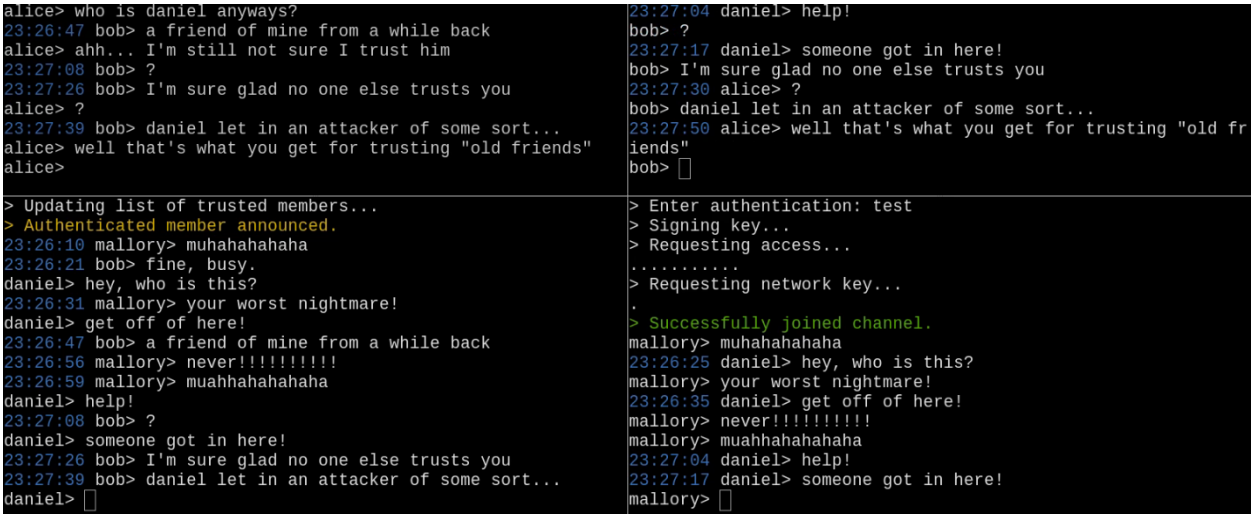

Figure 33 - Isolated Attacker through Diminished Access

In Figure 33, the attacker has now gained only limited access to the secure channel, as he was only authenticated by Daniel, who already had limited access. Mo communication occurs between Mallory and Alice or Bob. Daniel was able to accidentally authenticate Mallory without compromising the entire secure channel, only affecting himself.

Alice remarks, “That's what you get for trusting ‘old friends’”. Bob's mistake of allowing his friend to join the network which indirectly allowed an attacker to gain access has not compromised the integrity of the network.

8 CONCLUSIONS

The project demonstrates the ability to create a secure communication channel anywhere amongst trusted nodes. While there are existing methods for secure communication (such as TLS, which is widely used for HTTPS), they still require a centralized certificate authority in order to authenticate the identity of a node. While there are existing distributed cryptographic methods, such as those provided by JGroups or the WPA protocol, they still rely on the use of a symmetric key which can be difficult or impossible to change and redistribute. Existing distributed cryptography mechanisms lack a means of providing message and sender authentication.

Our project demonstrates a novel distributed system that also has support for truly decentralized authentication. It does not rely on any central certificate authority, and does not rely on using an immutable symmetric key. In addition, our project demonstrates the ability within a secure channel to have diminishing access for certain nodes as a result of limited trust by the network. For example, in the case that only a subset of all the nodes authenticate and trust the client, untrusted nodes do not gain access to communications throughout the network at large. These features are essential to the distributed nature of the project and give it the ability to create a secure channel anywhere - even in insecure networks, which can be used by a group of trusted nodes without the threat of eavesdropping or message tampering.

9 APPENDIX

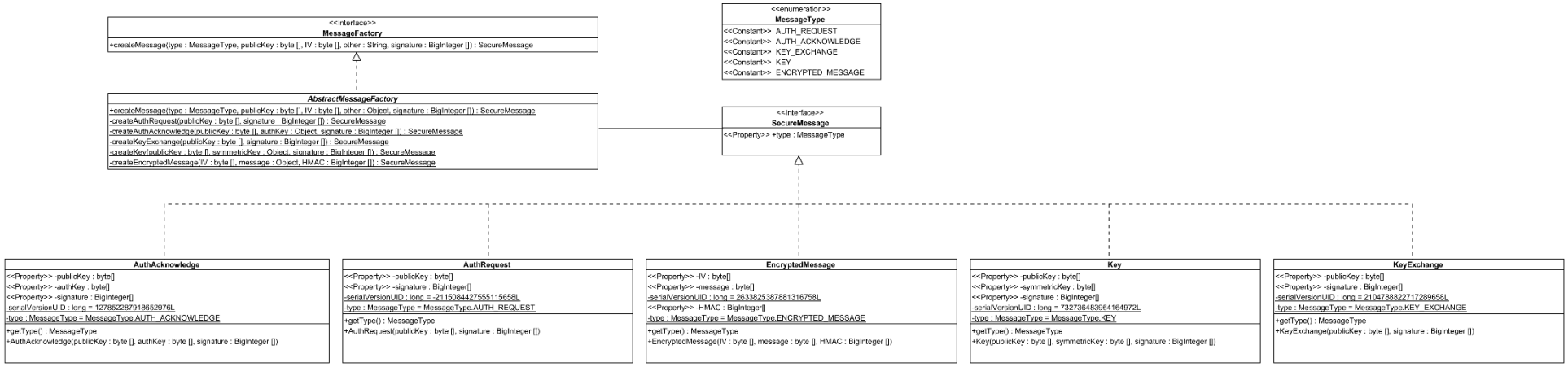

Figure 34 - com.DSC.message; Messages Package UML Class Diagram