Integrated MSc. Course on Informatics Engineering, DI@FCT NOVA
Computer Networks and Systems Security - Semester 1, 2019/2020
WORK-ASSIGNMENT #1 REPORT for Evaluation

A Peer-Group Oriented Chat using Secure Multicast Communication Channels

Authors:
Eduardo Brás Silva (emf.silva@campus.fct.unl.pt),
Rúben André Barreiro (r.barreiro@campus.fct.unl.pt)

Summary/Abstract

Abstract. As we see the recent ***increase in the number of users of social networks and texting/chat applications*** in recent years, the protection of data and messages exchanged between these users is becoming increasingly essential. ***New varieties of attacks and new system failures are discovered daily***, so ***it's crucial design and improve new security protocols***, in order to ***protect and mitigate the attack surfaces of our adversaries and threat agents*** (i.e., ***hackers*** or ***malicious agents***). Our proposal, the ***Secure Multicast Peer Group-Oriented Chat***, offers a proposal of a secure protocol, initially implemented in ***Java*** programming language, in order to protect this kind of applications for adversary models following the typology of communication attacks against ***IP Multicast channels*** and ***TCP/IP channels***, as defined in ***OSI X.800***. It was made a ***design analysis***, ***development*** and the ***preparation*** of an ***initial version of a deployment package for demonstration***, to serve this purpose.

Summary table of the TP1 implementation submitted for evaluation (fill with X), according to the Google Submission Form

Coverage of the performed work TP1	YES	NO	Tested it works well	Tested, doesn't work well	Doesn't work
The work only addressed the implementation of the PHASE 1	X		X		
The work addressed the implementation of the PHASE 1 and in this report we present the design and specification of the SAAHP protocol for the Phase 2	X		N/A	N/A	N/A
The work addressed the implementation of Phase 1 and Phase 2, bit the phases were not integrated		X		X	
The work involved the development and the integration of Phase 1 and Phase 2		X			X

URL of the GitHub Repo Project shared with **henriquejoaolopesdomingos**:	**https://github.com/rubenandrebarreiro/secure-multicast-peer-group-oriented-chat**

1. Introduction

It was ***implemented completely the phase 1***, addressing the requested requirements in specification of the project. Our proposal addresses the needed countermeasures for the adversary model defined in the specification of the project, which will be discussed in detail later.

In the specification, initially will exist only a core element, which will be the client. Obviously, as it's being addressed a system of a ***Multicast Peer-Group Chat*** service, it means that more than a ***Client*** can use this service simultaneously.

The ***Secure Multicast Peer-Group Chat*** service offers a set of ***chat sessions*** (i.e., ***groups*** or ***"rooms" of conversations*** between ***Clients*** or ***Users*** of this service, ***exchanging messages between them***). As obvious, it's pretended that the ***Clients*** pretend that ***these conversations are kept in privately*** in the ***chat session***, i.e., only the ***Clients*** or ***Users*** present and currently active (***Online Clients/Users***) in this ***chat session***, can have access to these conversations.

These conversations are made performing an one-to-many communication, where a sender sends a message to all the other active ***Clients/Users*** in the Chat Session, which will be ***continually waiting to listen to new messages*** from any ***Client/User***, also present in the ***Chat Session***. It's pretended to be kept in any ***Chat Session***, the ***log of the exchanged messages*** of it. Any ***Client/User*** will be able to access this log, but only for the exchanged messages after it entered the ***Chat Session***, i.e., joined the ***Multicast IP Address*** corresponding to that ***Chat Session*** and providing the necessary ***parameters*** to be able to use that ***Chat Session***'s ***configurations***. This ***configurations*** and ***parameters*** are kept by the ***Clients/Users***, statically, in a local storage.

Each ***Chat Session*** can be addressed by an ***IP Multicast Address*** which will have some security configurations to be accessed and to guarantee that the messages are exchanged assuming some ***security and privacy principles***, using ***Symmetric Encryption*** each ***Client/User*** that pretend to participate in this ***Chat Session*** will be restricted to use this ***parameters of security***, if want to use this ***Chat Session***.

As in this environment it's assumed a communication channel not secure, the messages exchanged in a ***Chat Session*** will be needed to be layered, using some kind of ***security constructions*** and/or ***combinations of many security constructions***. In other words, each message exchanged in a ***Chat Session***, through the communication channel, will be secured from the moment that a ***Client/User*** sends it to the moment that the other ***Clients/Users*** received it.

It was ***addressed a second designing and initial implementation***, addressing the phase 2, as specified in the project's description. This second implementation can be extended from the first one and only differs in the addition of a second element, which will be a ***Server***. This ***Server*** will be responsible to keep the ***Peer Authentication*** and ***Access Control Configurations*** for each ***Client/User*** which can use this ***Secure Multicast Peer Group-Oriented Chat*** service.

In order to address this new setup, it will be also needed communication between a ***Client/User*** that pretend to join a conversation in a ***Chat's Session*** and the ***Server***, in order to obtain the necessary configurations to access a ***Chat's Session***. This communication assumes the same principle of a communication channel not secure, which also must be needed to be layered, using some kind of ***security constructions*** and ***combinations of many security constructions***, similarly to proposed to in the first implementation. But this secure communications will be made using ***Asymmetric Encryption***, supported also by ***Digital Signatures***.

2. System Model, Architecture and Components

As mentioned previously, the system model considered to phase 1 of the project's specification, ***assumes only communication interactions between the Clients/User***. This ***Clients/Users*** must keep the ***parameters*** and the ***Chat's Sessions*** configurations, locally, in a static fashion. The parameters kept by the ***Users/Clients*** are the ***Cryptographic Parameterizations*** needed to be able to access the ***Chat's Sessions*** and also the ***configurations*** will be the necessary setup to access the ***Chat's Sessions*** which it have rights to do it. Thus, the components addressed to an initial setup of this implementation are the following:

- ***Client/User***, which will keep the following components, locally and in a static fashion:
    - A data structure (i.e., a ***configuration file***) to keep the ***Local Peer Parameters/Configurations of the Chat's Sessions***, which this ***Client/User*** have rights to access, indexed by its ***Multicast IP Address***.
    - A data structure to keep the ***Cryptographic Parameters*** to be able to perform secure messages' exchange with the other ***Users/Clients***, which are currently using the ***Chat's Session*** pretended to join and participate. This parameters must always be validated in each ***Secure Message*** sent by a ***Client/User*** through a ***Chat's Session*** in the ***Secure Multicast Peer-Group Chat*** service, by all the other ***Users/Clients*** participating in it, in order to all the ***Cryptographic Parameters*** be in agreement between all the other ***Users/Clients*** participating in it and also in agreement with ***Cryptographic Parameters***, initially specified for that ***Chat's Session***.

For the phase 2 of the project's specification, as it assumes also communication interactions between the ***Clients/Users*** and a ***Server***, it's strictly necessary consider some assumptions about this second introduced element, performing a small and "cirurgical" redesigning of the setup of the phase 1.

The function of this second introduced element, i.e., the ***Server***, it's to manage the ***Configurations*** for both ***Peer Authentication*** and ***Access Control***, in order to guarantee a secure ***Authentication*** of the ***Users/Clients*** of the ***Secure Multicast Peer-Group Chat*** service, as also, ***control*** and ***grant*** the ***Access*** to these ***Authenticated Users/Clients*** to the respective ***Chat's Sessions*** which have access to.

The ***Server*** will keep also the ***Cryptographic Parameters*** for the ***Chat's Sessions*** for each user, in opposite fashion to what as designed and implemented in the first approach of the project, where these components are managed by each respectively ***Client/User***. In this second approach, the ***Clients/Users*** are only responsible for keeping its ***Local Peer Parameters***, i.e., the information about the ***Chat's Sessions*** which have rights to access, similarly to what was designed and implemented in the phase 1 of the project.

So, the components addressed to an initial setup of this implementation are the following:

- ***Client/User***, which will keep the following components, locally and in a static fashion:
    - A data structure (i.e., a ***configuration file***) to keep the ***Local Peer Parameters/Configurations of the Chat's Sessions***, which this ***Client/User*** have rights to access, indexed by its ***Multicast IP Address***.

- ***Server***, which will keep the following components, also locally and in a static fashion:
    - A data structure to keep the ***Cryptographic Parameters*** corresponding to each ***Chat's Session*** and which be kept in agreement between all the ***Users/Clients*** participating in it, exchanging ***Secure Messages*** between them, as also, which must be used by an ***User/Client*** which pretends to join and participate in it. This parameters must always be validated in each message pretended to send by a ***Client/User*** through a ***Chat's Session*** in the ***Secure Multicast Peer-Group Chat*** service, by all the other ***Users/Clients*** participating in it, in order to all the ***Cryptographic Parameters*** be in agreement between all the other ***Users/Clients*** participating in it and also in agreement with ***Cryptographic Parameters***, initially specified for that ***Chat's Session,*** in a similar fashion to what occurred in phase 1 of the project.

    - A data structure to manage and treat all the ***accesses*** of the ***Clients/Users***, which use the ***Secure Multicast Peer-Group Chat*** service, i.e., the ***Access Control Configurations***. This data structure it's indexed by the identification of the ***Clients/Users*** (i.e., the ***Peer's Identifier***), the ***hashed password*** of each one of those ***Clients/Users***, in order to guarantee that its access to the ***Secure Multicast Peer-Group Chat*** service it's made in a ***secure and simple fashion***, and an ordered list of ***timestamps*** of the accesses made by the ***User/Client***, in order to keep some kind of log record of the access to each ***User/Client***. The data structure it's designed to keep the ***hash value of the password***, instead of keeping the ***password*** itself, for privacy reasons, because it's a good practice to the ***Server*** not keep the plaintext of the ***passwords*** of the ***Users/Clients***.

    - A data structure to keep the information of ***authentication*** to the ***KeyStore*** of each ***Client/User***, which use the ***Secure Multicast Peer-Group Chat*** service, as also, the ***Identifier of the Chat's Session***, which the ***User/Client*** pretends to join and the necessary ***Salt*** and ***Iteration Counter***, in order to the ***Server*** generate the necessary ***Cryptographic Parameters*** obtained from the ***KeyStore***, i.e., a ***Secret Key*** and ***Initialization Vector***, to be used by the ***Client/User***, performing the ***Password-Based Encryption*** during the exchange of the ***Secure Messages*** with the other ***Clients/Users*** participating in the ***Chat's Session***. This data structure will be named the ***Peer Authentication Configurations***. The ***KeyStore*** of each ***Client/User*** it's protected by its ***hashed password***. The entries of the ***KeyStore*** of each ***Client/User*** are indexed by an ***entry alias*** (i.e., an ***identifier***), which must be given by the ***User/Client*** initially, during the access to its ***KeyStore***. Each entry of this ***KeyStore*** will keep the as mentioned before, that ***Key Store*** will keep the ***Symmetric Encryption Algorithm***, the ***Size of the Secret Key*** to be used and the respectively ***Secret Key***, as obviously. Once the Client/User obtain the previously the previously mentioned ***Secret Key*** of the ***KeyStore*** and the ***Initialization Vector***, from this ***Peer Authentication Configurations***., are able to finally join a ***Chat's Session*** and start to exchange ***Secure Messages*** with other ***Clients/Users*** participating in it.

3. Adversary model

3.3.1 Phase 1. The ***Adversary Model*** for phase 1 of the project is defined as requiring the following ***protection*** and ***security specifications***:

1. ***Message Confidentiality avoiding release of message contents***;
2. ***Message Integrity, protecting from Message Tampering, as well as, from traffic-flow tampering, including disordering attacks on the respective message flows;***
3. ***Authentication Control Services for Message Authentication Controls;***
4. ***Message Replaying Protection***;

As for how each of the previously mentioned ***Protection*** and ***Security Specifications***, was implemented the following ***Countermeasures***, regarding each of the following ***Security Aspects***:

- ***Message Confidentiality:***
  - The ***Secure Protocol*** gets the initial message exchanged through the ***Chat's Session*** and other information, ***Cryptographic Cypher*** using ***Symmetric Cryptography***. All the ***Clients/Users*** need to have the same ***Cryptographic Configuration Parameters***, as mentioned before and the ***Secret Keys*** on their respective ***KeyStores*** to be able to cypher and decypher the contents of the initial message.

- ***Message Integrity is preserved using two different mechanisms:***
  - The ***Message Integrity*** of the initial message is checked by applying a **Cryptographic Hash Function** (as defined in the local static ***Cryptographic Configuration Parameters*** file) over this message and appending the result of a ***Cryptographic Hash Function*** on the message, i.e., performing a ***Message Digest***. The ***Client/User*** that receives the ***Secure Message*** will then get the initial message, apply the same ***Cryptographic Hash Function*** over all the message and compare his result of the ***Cryptographic Hash Function*** with the result that came with the received message. If they are both the same, it means the message passed through the ***Communication Channel*** has not suffered a ***Tampering Attack***.
  - However, other parts of the message may also suffer a ***Tampering Attack***. So, to check those, we apply a **Message Authentication Code** (**MAC**) over the whole message. As with the ***Secure Message***, a **CMAC (Cypher based Message Authentication Code)** or an **HMAC (Hash based Message Authentication Code)** is applied over the whole ***Secure Message***, said ***Secure Message*** is sent over to the other user, and that same user must apply the same transformation and compare what he got with what was sent. If the result is different, the message passed through the ***Communication Channel*** has suffered a ***Tampering Attack*** and will be ignored. To note that while ***Cryptographic Hash Functions*** and ***MACs*** both seem similar, ***MACs*** also provide an ***Authentication Control*** system. ***MACs*** need a ***Secret Key*** (provided by the ***KeyStore***) that must have been previously agreed upon for this system to work. If when comparing ***MACs*** the receiver could compare the right value, it means that he used the same ***Secret Key*** as the one that sent him the message, the one that has previously agreed to use the same ***Secret Key***.

- For ***Replay Attack Protection***, two approaches are used:
    - The use of a ***Nonce***, generated from a ***secure seed***, guaranteeing that a ***Replayed Message*** will be ignored as it will contain the same ***Nonce***, a ***random numerical value***, which is considered to be ***statistically and mathematically very improbable to happen twice in a short amount of time***.
    - The use of a ***Sequence Number*** useful, by not accepting any ***Secure Message*** that was previously accepted even if a previous ***Nonce*** that was used before could be used again, by ***Message Replaying Attack***.

3.3.2 Phase 2. The ***Adversary Model*** for phase 2 of the project is defined as requiring the same following ***protection*** and ***security specifications*** of the phase 1.

- In phase 2 of the project, the exchanged messages made using ***Asymmetric Cryptography***, and are following the next defined ***Secure Protocol***, where it's defined a nomenclature for the elements involver during the exchange of this message, for a better presentation of the ***Secret Keys*** exchanged:
    - ***Client - Element (a)***
    - ***Server - Element (b)***

- The models defined for the exchanged ***Secure Messages*** between the ***Client*** and the ***Server***, are the following:
    - ***Proposal #1:***
        - ***1st Message ($Client_{(a)}$ -> $Server_{(b)}$):***
            - $M_1 = [\{timestamp_1, nonce_1\}_{Kab\_1}, \{client\_id, session\_id, salt, iteration\_counter, H(password_{client\_id}), conf\_sym\_alias_{client\_id}\}_{Kab\_2}, \{client\_id, session\_id, H(client\_id, session\_id)\}_{KPrivClient}\}_{KPubServerChain}, \{Kab\_1, Kab\_2\}_{KPubServerChain}]$
            - $FinalMessage_1 = [\ M_1 \mid H(M_1)\ ]$
        - ***2nd Message ($Client_{(a)}$ <- $Server_{(b)}$):***
            - $M_2 = [\{timestamp_2, nonce_2, client\_id\}_{Kba\_1}, \{conf\_sym\_alg_{client\_id}, conf\_sym\_key\_size_{client\_id}, conf\_sym\_key_{client\_id}, initialization\_vector_{client\_id}\}_{Kba\_2}, \{client\_id, session\_id, H(client\_id, session\_id)\}_{KPrivServer}\}_{KPubClientRoot}, \{Kba\_1, Kba\_2\}_{KPubClientRoot}]$
            - $FinalMessage_2 = [\ M_2 \mid H(M_2)\ ]$
        - ***3rd Message ($Client_{(a)}$ -> $Server_{(b)}$):***
            - $M_3 = [\{nonce_3, client\_id\}_{Kab'}, \{client\_id, H(client\_id)\}_{KPrivClient}\}_{KPubServerChain}, \{Kab'\}_{KPubServerChain}]$
            - $FinalMessage_3 = [\ M_3 \mid H(M_3)\ ]$

- ***Proposal #2:***
  - **1st Message (*Client*$_{(a)}$ -> *Server*$_{(b)}$):**
    - $M_1 = [\{timestamp_1, nonce_1\}_{Kab\_1}, \{client\_id, session\_id, salt, iteration\_counter, H(password_{client\_id}), conf\_sym\_alias_{client\_id}\}_{Kab\_2}, \{Kab\_1, Kab\_2, H(Kab\_1, Kab\_2)\}_{KPrivClient}\}_{KPubServerChain}]$
    - $FinalMessage_1 = [\ M_1 \mid H(M_1)\ ]$
  - **2nd Message (*Client*$_{(a)}$ <- *Server*$_{(b)}$):**
    - $M_2 = [\{timestamp_2, nonce_2, client\_id\}_{Kba\_1}, \{conf\_sym\_alg_{client\_id}, conf\_sym\_key\_size_{client\_id}, conf\_sym\_key_{client\_id}, initialization\_vector_{client\_id}\}_{Kba\_2}, \{\{Kba\_1, Kba\_2, H(Kba\_1, Kba\_2)\}_{KPrivServer}\}_{KPubClientRoot}]$
    - $FinalMessage_2 = [\ M_2 \mid H(M_2)\ ]$
  - **3rd Message (*Client*$_{(a)}$ -> *Server*$_{(b)}$):**
    - $M_3 = [\{nonce_3, client\_id\}_{Kab'}, \{\{Kab', H(Kab')\}_{KPrivClient}\}_{KPubServerChain}]$
    - $FinalMessage_3 = [\ M_3 \mid H(M_3)\ ]$

If your implementation addresses the phase 2, describe what you consider is different from the adversary model for phase 2. You must only consider this section of you implemented phase 2 specifications or specification and implementation. If you consider that the adversary model conditions are exactly the same, you must only say it.

4. Phase 1 – Secure Multicast Communication Protocol (SMCP)

Report here the specific format of SMCP messages, as you have in your implementation. Don't repeat things that are already in the initial specifications. You must only consider to report what is different and why you decide to make it different, as well as, you argumentation on the correctness and completeness of your specific implementation, compared with the initial specification and requirements.

4.1 Configuration file for MchatClient applications protected by SCMP

- The ***Configuration File*** format being used is just like the one specified in the work assignment sheet. It's inside of the "***res***" folder and is named as **SMCP.conf**.

- It requires a blank line to exist between the configuration of each ***IP:PORT*** combination, as the following examples:

    - ***Example #1:***

```
<224.1.1.1:9876>
  <SID>FCTSpeak</SID>
  <SEA>DES</SEA>
  <SEAKS>64</SEAKS>
  <MODE>CBC</MODE>
  <PADDING>PKCS7Padding</PADDING>
  <INTHASH>SHA-256</INTHASH>
  <MAC>HMacSHA256</MAC>
  <MACKS>256</MACKS>
</224.1.1.1>
```

    - ***Example #2:***

```
<224.1.1.2:1111>
  <SID>SRCS</SID>
  <SEA>AES</SEA>
  <SEAKS>128</SEAKS>
  <MODE>ECB</MODE>
  <PADDING>PKCS5Padding</PADDING>
  <INTHASH>SHA1</INTHASH>
  <MAC>HmacMD5</MAC>
  <MACKS>128</MACKS>
</224.1.1.2>
```

4.2 Keystores

- For this assignment we are using two different ***KeyStores***. Both use the same ***password***:
    - **CSNS1920**

- The one inside the "***res***" folder, called **SMCPKeystore.jecks**, it's the ***KeyStore*** in which are stored all the Secret Keys used for the several ***Cryptographic Constructions***, both for the ***Symmetric Encryption Algorithm***, using an {***IP***}:{***PORT***} ***identifier***, and the for the ***MAC***, using a {***IP***}:{***PORT***}***:mac identifier***, where ***mac*** is literally the string "***mac***".

- The other ***KeyStore*** is on the root of the project called ***signKeystore***. It's used to store the ***certificate*** to sign the generated ***JAR*** file.

4.3 Running the Client as a Standalone Application

- In the SecureMChatClient-Application-Phase1 folder can be found a signed SecureMchatClient.jar File.

- To run the program, go to the root of the project and run the following command in the prompt/terminal:
  - ***java -jar SecureMChatClient-Application-Phase1/SecureMchatClient.jar ReportUser 224.1.1.1 9876***;

- **Or** run the ***runJar.sh*** script found on the root of the folder;

- The generated ***JAR File*** expected to be used as, with the following usage:
  - ***java -jar SecureMchatClient.jar {USERNAME} {MULTICAST_IP} {PORT}***

4.4 Tested Cryptographic Parameters

- The ***Cryptographic Parameters Configurations***, ***which were used and tested with success*** (i.e., it worked fine), are the following:
  - ***DES/CBC/PKCS7Padding***
  - ***AES/ECB/PKCS5Padding***
  - ***AES/CTR/NoPadding***
  - ***Serpent/OFB/PKCS5Padding***
  - ***Twofish/CFB/PKCS5Padding***
  - ***Blowfish/CTR/PKCS7Padding***
  - ***DESede/CBC/PKCS5Padding***

- The ***Cryptographic Parameters Configurations***, ***which were used and tested with no success*** (i.e., it not worked fine), are the following:
  - ***AES/GCM/NoPadding***

5 Phase 2 – Authentication and Dynamic Establishment of Security Association Parameters

If you didn't address Phase 2, you dont need to write this section 5 and your section 5 will be the conclusion section. Otherwise you must start by describing the objectives and results achieved in addressing the Phase 2, according to your submitted work. After the initial summary please include the following sub-sections.

5.1 Generic specification of the SAAHP protocol

Try to present here an initial specification of your designed protocol, namely message requests and responses, according to the required rounds between a client and the SAAHP server (endpoint). A good idea would be the representation of the protocol in a sequence temporal diagram.

5.2 Formal specification of the SAAHP Protocol: SAAHP Message format

Present here a good formalization of the SAAHP message format specification, clarifying each part of the message format, the cryptographic contents and parameterizations, as well as, the security properties satisfied by the cryptographic constructions in your message format.

5.3 Detail of the protocol operation to support client JOINS and client LEAVE

Present here if you have specific support to the JOIN and LEAVE operations from clients in chat sessions

5.4 Rekeying strategies

Discuss here if you considered a rekeying mechanism, in the support of JOINS and LEAVES from clients, in CHAT sessions.

5.5 Configuration files in the SAAHP Server

Present and exemplify the required setup and configuration files in the SAAHP server, namely for: user-authentication, access-control, chat-session parameterizations, keystores and truststores.

5.6 Configuration files in the client side

Present and exemplify the required setup and configuration files required in the client side

5.7 Trust-Management Assumptions

Explain here how you addressed trust-management support, namely, what is required to setup initially, for the trust-ability-assumptions of clients and SAAHP server to manage public-keys and/or verification, assessment and validation conditions of public-keys and public-key certificates.

6 Highlights of the implementation

This optional section can be used if you want to highlight something you want to emphasize from your implementation, or if you want to present arguments for your design-options and implementation models and/or used techniques.

7 Conclusions and Final Remarks

Use this to summarize your main conclusions and to present your final remarks on the implementation of the TP#1.