Computer Networks and Systems Security

Project Assignment #1

A secure “pay-per-view” real-time media streaming system

Abstract

The objective of this project assignment is the design, implementation, and experimental evaluation of a secure “pay-per-view” real-time media streaming system. The system allows users to request, pay and receive real-time encoded FFMEPG-encoded movies sent from a streaming server. The dissemination of such movies is supported by a secure connectionless datagram secure channel with cryptographic protection that can use IP multicast or IP unicast. To access the available movies, users must be initially pre-registered in the system. To request and to see the movies, users must pay-per-view and payments are made once per movie. Valid payments together with users’ authentication proofs also include payments using a form of valid crypto-currency coins or e-vouchers (that can be pre-obtained orthogonally to be used in the system). If the user authentication and the payment is valid, the requested movies are then sent as a sequence of cipher-media MP4 frames as payloads in encrypted datagrams (with the payloads following an FFMEPG framing encoding) received and processed by a user-trusted proxy. This proxy must be able to decrypt and process the protected frames, in real-time conditions. This proxy will decrypt the received frames in real-time, delivering the plain-movie segments to be played by a standard media real-time player tool. Users must play the movies with the required visualization quality and a good real-time playing experience: without noticing frame drops, real-time playback delays or frame-clippings during the playback time.

Keywords: secure real-time streaming; symmetric and asymmetric cryptography, secure hashing and message authentication codes; password-based encryption; secure datagram-based connectionless communication channel; real-time FFMEG media playing

Development and delivery process:
Ref delivery date: 24/Nov/2021
- **Mandatory delivery: Delivery 1 (ref., until 12 points)**
- **Complete delivery: Delivery 2 (ref., until 20 points)**

Delivery process
- **Via a Form-Quiz (Google Form open after 20/Nov and closed on 30/Nov)**
- **The form includes a characterization of the implementation and concluded achievements**
- **URL of shard GitHub repo with the implementation included in the Form**
- **Penalizations considered for source code after the due dates**

In the next sections of the document, you can find the reference and guidelines for the project implementation

1. Introduction

The goal of this project is to develop a solution for a secure streaming protocol implemented with cryptographically protected connectionless-oriented UDP unicasting or multicasting channels.
The protocol will be used by a demo distributed application composed by a ***Signaling Server***, a ***Streaming Server***, a ***ProxyBox*** and an ***MPEG Player tool***.

Signaling server: implements a service supporting user authentication, and for authenticated/registered users provides the establishment of security association parameters – including cryptographic keys or other secrecy parameters), to allow the proxy to receive real-time protected streams and to decode them to be sent for playing with the media player tool;

Streaming Server: a component that can disseminate pay-per-view movies, encoded in sequences of protected FFMPEG media frames, sent to the remote proxies

ProxyBox: a component to receive the protected streams (in the communication channel used by the SreamingServer for the media dissemination). At the proxy level, the media frames must be processed to be decrypted and to control the required integrity , and then the streams are sent (decrypted – or in clear-format) to be played by the media player tool.

MPEG Player Tool: a "standard" media-player tool. For the project we suggest the use of the VLC tool (https://www.videolan.org/index.pt.html) or any other free or open-source tool (ex., https://mpv.io/)[1]

2. Security Protocols

Beyond the components described above, that are the relevant components of the System Model and Architecture for development, the system will be supported by two relevant protocols: **SRTSP - Secure RealTime Streaming Protocol** and **SAPKDP - Secure Authentication, Payment and Key-Distribution Protocol**. The reference specifications for protocol will be provided as initial reference specifications considering the base or mandatory requirements. However once the base specifications are complied, you can consider your own features or extensions. You can discuss your SRTSP and SAPKDP proposals in Lab classes.

SRTSP. The media streams sent by the Streaming Server must and received by the ProxyBox are protected by SRTSP The ProxyBox must be able to decrypt and process the protected frames (encapsulated in SRTSP/UDP), delivering the received plain movie MPEG frame segments traveling in the SRTSP payload after decryption, in clear format (plainframes), to be played by media player applications that can be used by users playing the movie.

SAPKDP. The authentication process of ProxyBoxes and their Users, as well as the movie requests and payments, with the subsequent reception of all the required credentials to receive the required real-time media frames (movies) in case of successful authentication and payments are protected by the SRTSP protocol. The protocol is the used as a Secure Handshake for the Authentication and Payments, between each ProxyBox and the Signalling Server

To run the SAPKDP protocol between a ProxyBox and the Signalling Server, ProxyBoxes and ProxyBox Users (that we can regard as ProxyBox Owners) must be pre-registered in the system. The specific pre-registration of a ProxyBox involves the registration of the ProxyBox Unique ID (that can be implemented as a Public-Key). For ProxyBox owner, the registration will involve a username/password pair, as commonly used. Tis pre-registratio process is orthogonal to the system operation and is out of-scope of the required implementation[2]. The registered information will be used by the Signaling server during the authentication procedure suppored by the SAPKDP protocol.

3. SRTSP and SAPKDP protocol encapsulation in the TCP/IP Stack

You must notice that for the SRTSP encapsulation is mandatory to use the UDP transport protocol (with the SRTSP payloads supported as payloads of UDP Datagrams. However, for the SAPKDP protocol you will be free to decide your encapsulation. You can support SAPKDP as an overlayed protocol, where the SAPKDP payloads can be supported on top of HTTP data-messages, TCP packets or UDP datagrams. For the case of HTTP, it is also possible to address SAPKDP as encapsulated data on REST/HTTP request/reply messages, if you decide to implement the SignallingServer as a WEB REST-enabled Service. Anyway, the implementation of the SAKDP and SRTSP communication endpoints, as well as the SteramingServer and ProxyBox, can be developed using TCP/IP or Data-Stream Sockets or UDP/IP Datagram Sockets.

4. System model, architecture, and application scenario

The components and protocols in the system model (introduced before in sections 2 and 3) are represented in the following figure, showing the System Model and Architecture for the required solution.

The System model assumptions and an illustration of a possible approach for a related application scenario was discussed in LABs (see in the LAB materials, PA1 elements, particularly PA1-Approach Slides).

[1] For other tools, search for: Mplayer, Haruna, SMPlayer, Dragon, GNOME Videos (alias Totem), Deepin, etc.
[2] The registration process for ProxyBoxes (ProxyBox IDs) and their Users-Owners (Usernames/Passwords) will be addressed by a manual setup for the system, where the necessary information for the registration is provided manually, written in configuration files accessible to the Signalling Server.

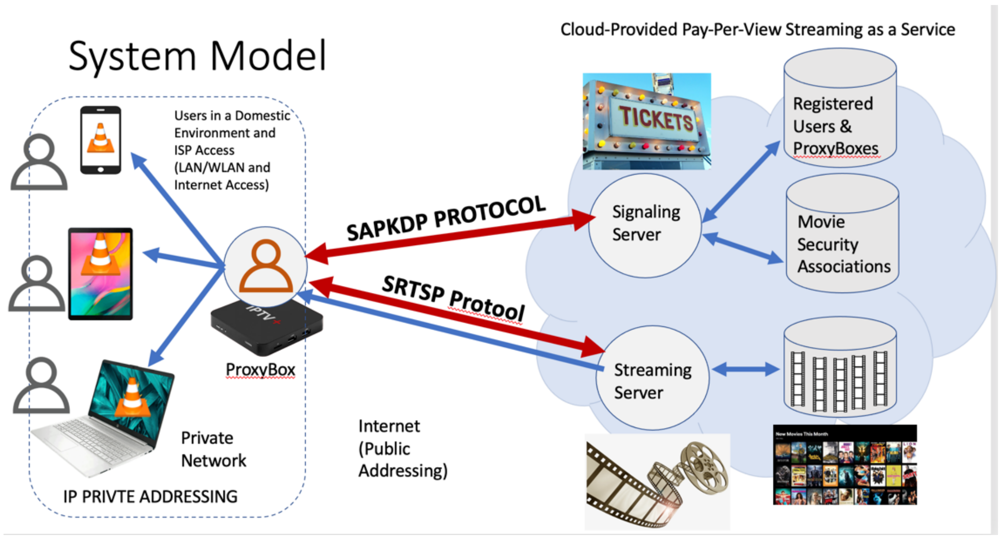
System Model
Cloud-Provided Pay-Per-View Streaming as a Service
Users in a Domestic Environment and ISP Access (LAN/WLAN and Internet Access)
SAPKDP PROTOCOL
SRTSP Protool
Signaling Server
Streaming Server
Registered Users & ProxyBoxes
Movie Security Associations
ProxyBox
Private Network
IP PRIVTE ADDRESSING
Internet (Public Addressing)
TICKETS

Fig.1 Application Scenario and Environment for the Pay-Per View Streaming Service

A concretization of the Application Scenario for implementation purposes is more essentially represented in the following figure. As represented, the figure show the necessary configuration setups for operation.

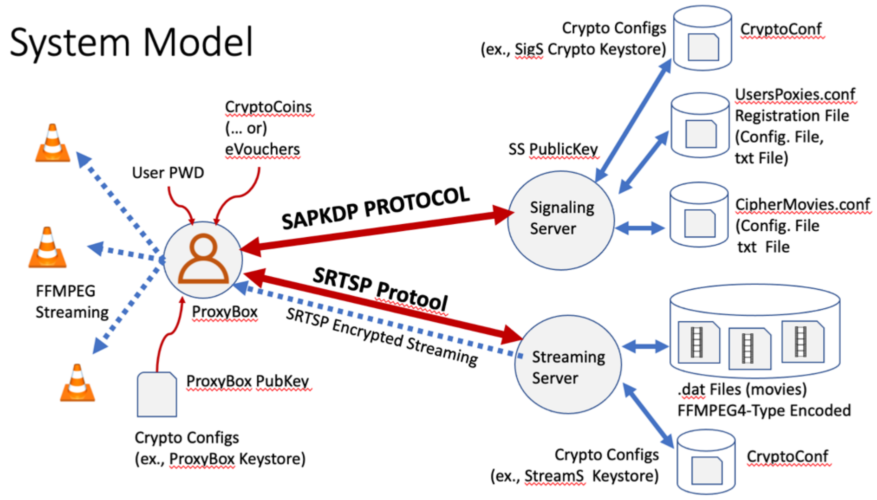
System Model
Crypto Configs (ex., SigS Crypto Keystore)
CryptoConf
UsersPoxies.conf Registration File (Config. File, txt File)
CipherMovies.conf (Config. File txt File)
SS PublicKey
CryptoCoins (... or) eVouchers
User PWD
SAPKDP PROTOCOL
Signaling Server
SRTSP Protool
SRTSP Encrypted Streaming
Streaming Server
FFMPEG Streaming
ProxyBox
ProxyBox PubKey
Crypto Configs (ex., ProxyBox Keystore)
.dat Files (movies) FFMPEG4-Type Encoded
Crypto Configs (ex., StreamS Keystore)
CryptoConf

Fig.2 The system model for the complete implementation

Context and application scenario. As an application scenario inspiring the project assignment, you can imagine that the Streaming Server works as a media-streaming content provider for movies, movie-podcasts, TV shows, etc., (as a Netflix or a Digital TV Operator, for example). The ProxyBox is seen as a software-enabled “Trusted Setup box” (as ISP boxes in our domestic networks), receiving the streams in encrypted closed-channels (supported in our case as SRTSP packets supported the UDP transport protocol) to be decrypted and delivered to the users. These users can use media players (in their laptops, smartphones or tablets) connected to the domestic private addressed network. Moreover, for our demo scenario, note that the ProxyBox also can receive the encrypted streaming feeds from the internet (public addressing environment) and can deliver the decrypted media frames via NAT, to be played in the private addressed (domestic) network (such as the case of the typical infrastructure in our domestic wired or wireless networks interconnected to service providers by using our domestic routers and tv boxes.

Pay-per-View Environment. To access the available movies, the designed system will follow a pay-per-view model. Authentication of ProxyBoxes and their users, as well as Payments are supported by the **SAPKDP protocol** and the Signaling Service. Payments will be supported with a form of crypto-currency coins or (or more precisely eCryptoVouchers (pre-obtained). These “coins” must be used in such a way that the system will control the authenticity and integrity, with no possibility to be used for double or multi-spending purposes. Then, each obtained e-voucher or cryptocoin will be used to see one and just one movie (on demand).

The payment is done under authentication proof of pre-registered user and proxy box (in request/repky transactions supported by the SAPKDP protocol). If a valid and acknowledged proxy box (namely its ProxyBox Unique Identifier) and its valid user (username-UserID / Password and other possible related Attributes) are pre-registered in the system, then the authenticated user can request and pay for each real-time movie-stream. The authentication and payment for those authorized users will be verified by the Signaling Server. If everything is correct the Signaling Server will send to the ProxyBox the necessary information-credentials and cryptographic configurations required to process the streamed frames that will be sent by the StreamingServer. This information includes the necessary cryptographic material and ciphersuites' configuration that the proxy must use to receive the requested movie and to deliver the movie decrypted for user-media players. This information is managed in the CipherMovies.conf file

5. Setup Specifications

5.1 ProxyBox Setup

The setup is provided by the CryptoConfig File. This can be implemented as a Java Keystore where the Private and Public Key of the ProxyBox are stored. For this purpose we will use Elliptic Curve Cryptography (ECC) keypairs that will be used by the ProxyBox I the context of the SAPKDP and SRTSP protocols (where different msg-transactions sent bu the ProxyBox to the Signalling Server and Streaming Server will eb digitally signed by ECCDSA Signatures.

Then, the CryptoConfigFile is nothing more than a keystore, managing the ECC keys for the proxyBox. The recommendation here is to adopt an universal storetype for those keystores, ex: PKCS#12 format will be ok. You can use the keytool to generate the keys.

See. Ex. https://zombiesecured.com/html/tutorials/Keytool/ECC-PK.html or see the Java documentation.

Remember that all the entities in the model must have access to the public keys of the other entities for whish is necessary to validate digital signatures. In particular, the Signaling server and the StreamServer must know the publickey of the ProxyBox to operate the required protocols. Of course that the ProxyBox private key must ne secured and only known by the ProxyBox itself ! See the documentation or discuss in the Lab if you have doubts about this.

The ECC Public Key, together (concatenated) with a Pre-Assigned Randomly Generated as Secure Random Identifier will be used as the ProxyBox Unique ID (and this is the ID that must be pre-registered in the UsersProxies configuration file that must be accessible in the StreamingServer side. Moreover, to be ready for requesting movies and for the authentication during the SAPKDP protocol, a valid user must start the ProxyBox with a correct username and password (or passphrase) pair. The password or passphrase are then used as the User Authentication factor owning the ProxyBox.

Then the ProxyBox will be started in the following way (as a shell command line):

```
ProxyBox -user <username> -password <pwd> -keystore <keystore-file> -proxyinfo <proxyinfo-file>
```

- keystorefile: a java jks storetype keystore file that must have the keypairs (private/public keys) for the ECDSA Cryptographic operations
- proxinfo file: a text file that stores a valid ProxyBox identifier (can be in clear text but if you prefer, you can protect this using the password of the valid user that can extract the ProxyBox ID)

Format for the proxyID: <ECC publickey>||<securerandom generated suffix>
The ECC pubkey can be used with a strog sie for ECDSA Signatures (size >= 256 bits) and the securerandom generated suffix also with 256 bits (which combined will form a robust unique ID with 512 bits).

You must follow the discussion on LABs about the ECC Curves you can use for security, the ECC Keygeneratuion process, as well as for ECCDSA Digital signtures you will require for your implementation. You must notice that for the ProxyBox operation, it is required that a valid user starts correctly the ProxyBox. If the user is not logged on the proxy, there is no access to eth cryptographic materials in the setup configurations.

5.2 Cryptocoins or Cryptovouchers Setup

Additionally you must provide the proxy with "pre-loaded" cryptocoins – or cryptovouchers".
The format for a cryptiocoin or cryptovoucher is a file with the following format:

```
Coin: <Stringified CoinName>                       ex.,    PPVMovieCoin
Coin issuer <Stringifeied Coin Issuer Name>        ex.,   JoseManuelIssuerBank, Inc.
CoinValue: <value>                                 the value of the coin, ex., 0,5
ExpireDate: <date>                  The expiration validity of the coin
CoinPubicKey: <PublicKey>           bytes, PublicKey of the Coin (generated per coin when issued
CoinAuthenticy: <DigitalSignature of the Coin>     bytes
IssueSignature                                     bytes, Digital Signatur of the Issuer
IssuePublicKey                                     bytes, publicKey of the Coin Issuer
IntegrityProof1                     bytes, SHA256 of previous fields of the coin
                                    (Except Integrity Proofs)
IntegrityProof2                     bytes, a complementay 256bit hash of previous fields of the
                                    coin (Except Integrity Proofs)
```

If you want, you can decide to change the format (including the required cryptocoin data in the format above) using a different structure (ex., inspired in a XML or even in a JSON representation).
You must develop a program that you will use to generate the coins (using the format as an initial reference). You will use your program to issue several coins with your decided values to demonstrate your system.
As recommendations: use ECC Curves and ECC Cryptography with recommended sizes above 256 bits. Follow the classes about the discussion on ECC Curves, methods, ECC keypair generation and Secure Digital Signature constructions. As you can imagine, your proxy will use the files with your cryptocoins (or cryptovouchers) as a wallet. Is up to you to consider if you want or not to protect additionally the "wallet".

5.3 SignalingServer Setup

UsersProxy config file. As initial reference, you can have the following format for this file. Note that we could have more than one entry if we want to support more than one ProxyBox and or Users in different proxyBoxes. If you want you can also include other attributes (ex., PhoneNumber, age, etc etc etc) in the username element.

```
<entry1>// a valid set for a user in a valid proxyBox
<username>
"username"      // a valid user name
</username>
<password>
H(pwd>          // hash value of the pwd or passphrase: recommendation, use SHA-256
                // or if you want you can use double-hashing construction w/ differeht Hash functions
</password>
<proxybox ID>
proxuBoxID
</proxyBoxID>
</entry 1
….
```

CipherMovies config file. As initial reference you can consider the following structure.

```
<moventry>
<movie>
"the name of the movie"         // can be the name of the erspective .dat file
</movie>
<ppvprice>
value                           // integer with the price, ex: 0,5 coins
</ppvprice>
<cryptographicconfig>
<ciphersuite>
<confidentiality>
Ciphersuite spec                // ex., AES/CTR/NoPadding  or whatever.
keysize                         // keysize valid for the defined ciphersuite spec, ex: 256
key                             // generated key bytes
iv                              // Null or bytes used for the IV, depending on the used mode in the ciphersuite

</confidentiality>
<integrity>
Macsuite spec                   //ex., HMAC-SHA256 or whatever
mackeysize                      // mac key sise used in the defined macsuite spec
makkey                          // generated mackey bytes
</integrity>
<ciphersuite>
</moventry>
….
```

Cryptoconf file

These files (in the Siganaling Server and in the SteramServer) are nothing more than keystores for the management of ECCDSA keys.

The recommendation here is to adopt an universal storetype for those keystores, ex: PKCS#12 format will be ok.

You can use the keytool to generate the keys.

See. Ex. https://zombiesecured.com/html/tutorials/Keytool/ECC-PK.html

Remember that all the entities in the model must have access to the public keys of the other entities for whish is necessary to validate digital signatures. In particular, the Signaling server and the StreamServer and the ProxyBox must know the respective publickeys. Of course that private keys must be known and secured, exclusively, by each involved entity in the system model. See the documentation or discuss and learn in the Lab how to do it.

Movies in .dat files

These are the files storing the encoded movies. You can use the files already provided in the Streaming Example in Lab3 Materials and shown/demo in the Lab example.

6. Reference specification for the SAPKDP and SRTSP Protocols

This reference will be presented and discussed initially using a functional description of the multiple rounds (see Lab materials for PA1). In a second phase, an initial reference for the protocols and their message formats will be defined in specific documents, developed in the Labs.

7. How to develop the PA1: implementation stages

The project can be developed in two different stages and goals (as initially presented). Following this suggestion, you cam develop two different delivered implementations in each stage:

Delivery 1: Mandatory for frequency requirements, with a simplified and partial implementation, with a maximum evaluation of 3/5 points (or 12/20 points in the 0-20 scale)

Delivery 2: Complete implementation, with the compete evaluation, with a maximum evaluation of 5/5 points (or 20/20 pointsin the 0-20 scale)

7.1 Stage 1: Delivery 1

In stage 1 you must implement a simplified version of the system, only containing the StreamingServer and the ProxyBox. In this case all the required setups are only two common files shared as setup files commonly used by the Streaming Server and the Proxy Box. The files follow the configurations as previously mentioned for te CipherMovies config file.

The idea is to support the StreamingService in a similar way as in the original provided implementations (Lab3 Streaming example), but in a secure way only using symmetric cryptography and Mac protection. As you will see the price for payment (in the configuration) will not be used in the Stage 1.The simplified system model for the stage 1 is represented in the following figure.

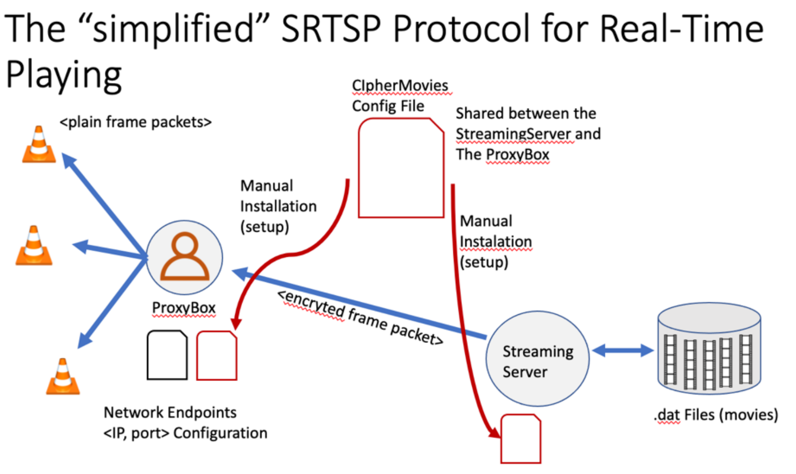
The "simplified" SRTSP Protocol for Real-Time Playing
<plain frame packets>
ClpherMovies Config File
Shared between the StreamingServer and The ProxyBox
Manual Installation (setup)
Manual Instalation (setup)
ProxyBox
<encryted frame packet>
Streaming Server
Network Endpoints <IP, port> Configuration
.dat Files (movies)

Fig.3 Simplified version for the stage-1 development and mandatory delivery

Following the Fig.3, the CipherMovies.config is shared (installed for the setup) of the Streaming Server. The Network endpoints and addresses are configured in the proxyBox side in the same way as in the provided implementations. ProxyBox and StreamingServer operate also as in the provided implementations. The difference is that the datagram payloads carrying the MMPEG frames will be protected by the Simplified version of SRTSP, described below.

Simplified version of SRTSP for Stage 1: Message format and UDP encapsulation

The simplified version of the SRTSP is represented in the following figure

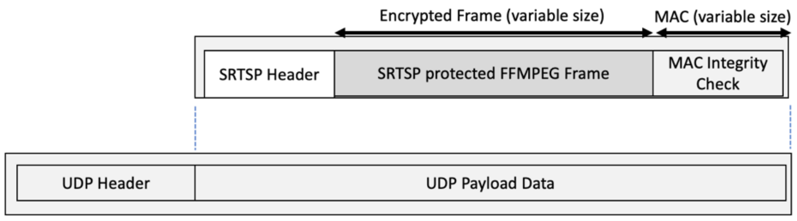

Fig. 4 – Simplified format for simplified SRTSP protected frames and UDP encapsulation

SRTSP Header is composed by the following elements:

- 4 bits: contains the version id of the Simplified SRTSP protocol: 0001
- 4 bits: contains an indication that the setup is manual: 0000 means manual configuration of endpoints
- 16 bits: integer, contains the size in bytes of the encapsulated encrypted frame (considering the SRTSP protected frames and the Mac integrity check)

SRTSP protected FFMEP Frame. Frames in the payload are encrypted according to the cryptographic configuration setup in the Streaming Server and in the ProxyBox endpoints implementing the SRSTP protocol. The variable size depends on the cryptographic parameterizations (including the symmetric cryptographic algorithm used, the cryptographic mode and the padding.

MAC Integrity Check. Integrity check is supported by the HMAC or CMAC parameterization in the cryptographic configurations and depending on the configurations we can have variable sizes.

Evaluation criteria of Stage 1 – Delivery 1 (to obtain 12/20 points)

- Movies played with good Real Time quality criteria: visualization quality and a good real-time playing experience: without noticing frame drops, real-time playback delays or frame-clippings during the playback time (not requiring the help of buffering capacities in the media players and not requiring playout delays for such buffering): 6 points
- Proof that the cryptographic configurations are flexible and we can use any combination of valid cryptographic parameterizations (which must be transparent for the implementation): 4 points
- The implementation of the required StreamingServer and the ProxyBoxes work in the same way as in the original implementations and the difference in the number of LoC (lines of code) compared with original implementations is neglectable (1 to 5 lines affected as maximum – the less the better): 2 points

The last criterion states that the cryptographic support and SRTSP implementation must be hidden and implemented in implemented abstractions (and not in the original application code). For this purpose you can implement abstractions, such as (examples):

- mySimplfiedSRTSPDataGramPackets (extending the Java standard DatagramPacket class) or
- mySimplifiedSRTSPDatagramSockets (extending the Java standard DatagramSocket class)

At the same time all the code to manage the configuarations must be also transparent to the code at the level of the StreaingServer and ProxyBox.