DIOMEDES

SEVENTH FRAMEWORK PROGRAMME

DIstribution Of Multi-view Entertainment using content aware DElivery Systems

DIOMEDES

Grant Agreement Number: 247996

D5.4

Report on Security Prototype Components

Document description	
Name of document	Report on Security Prototype Components
Abstract	This document describes the functionality and the architecture of the security components developed in Task 5.6 (Security Server, Security Client). The description also includes security-related workflows, the evaluation of algorithms and performance evaluations of the developed modules.
Document identifier	D5.4
Document class	Deliverable
Version	1.2
Author(s)	J. Hasselbach (IDMT)
QAT team	N. Just (IRT), E. Ekmekcioglu (UNIS)
Date of creation	05 September 2011
Date of last modification	31 January 2012
Status	Final
Destination	EC
WP number	WP5

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

1.1 Purpose of the document

This document provides a description of the modules developed in task 5.6. In particular, it describes the functionalities, the APIs, and the internal structure of the “Security Server” and the “Security Client”. Moreover, the document describes all relevant workflows related to the mentioned modules and includes a summary of the algorithm evaluation process.

1.2 Scope of the work

The workflows described in this document are related to the following scenarios: secure distribution of 3D A/V content via P2P network (including access control and content authentication), upload of user generated content and discovery of content available for P2P download.

1.3 Objectives

The main objective of this document is to describe the modules developed in task 5.6 and the P2P content protection approach in general.

1.4 Structure of the document

Chapter 1 provides the introduction for this document. Chapter 2 contains descriptions of the security modules as well as the specification of the P2P chunk header format. Chapter 3 depicts the workflows related to the security modules. Chapter 4 provides a summary of the algorithm evaluation process. Chapter 5 contains the conclusion.

2 MODULES AND FORMATS

2.1 Security Server

The main functionalities of the Security Server are:

- access control - in order to avoid unauthorized access to content
- content registration and authentication - in order to prevent sharing of malware and unauthorized content
- content discovery – in order to allow users to select content to be retrieved via P2P

The Security Server encrypts and digitally signs the transport stream chunks provided by the 3D Content Server. Moreover, this component is responsible for the management of keys and receivers - including key generation, storage and efficient key distribution using the available broadcast channels. The Security Server generates so called P2P chunks consisting of a P2P chunk header (see Table 1 for details) and the actual payload. The payload can contain encrypted and signed A/V data, signed content metadata, or key information generated by the Security Server (the “broadcast header”, see Chapter 3.1 for details).

2.1.1 API

The Security Server provides an API for registering content (including “user generated content”), i.e. it serves as central entry point for new content. Figure 1 provides an overview of the functionalities, and input and output channels of the Security Server.

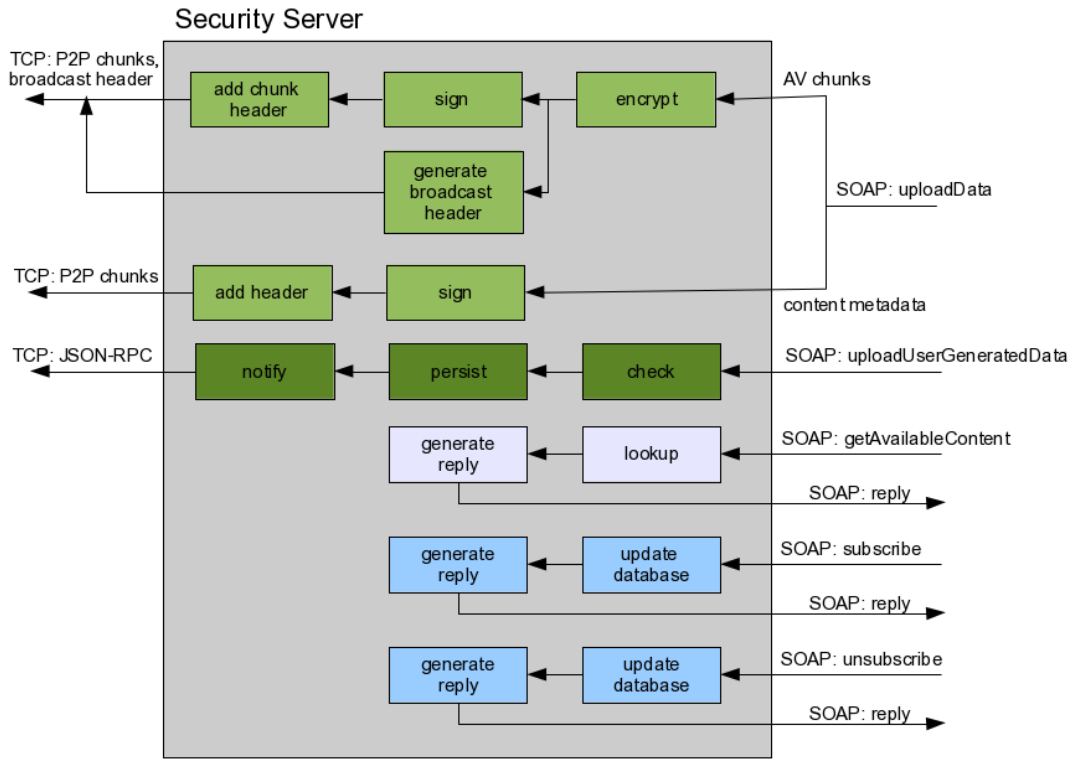

Figure 1 – Security Server (functionalities, input and output channels)

uploadData

```
public void uploadData(java.lang.String contentReference,
                       ContentDescription contentDescription,
                       javax.activation.DataHandler handler)
```

This method triggers the preparation of content coming from a trustworthy source (e.g. the 3D Content Server). One chunk per method call is uploaded.

Parameters:

`contentReference` - unique identifier for a content

`contentDescription` - description of the content, the description object contains the following parameters: metadata flag, b/e-layer flag, a/v flag, PID, view ID, PCR, channel ID, signature over the provided content, certificate of the content source

`handler` - the actual content in binary format (streamed via MTOM attachment). For each chunk generated by the TsChunker, there is one method call

uploadUserGeneratedData

```
public void uploadUserGeneratedData(long receiverId,
                                    byte[] dataSignature,
                                    byte[] receiverCertificate,
                                    java.lang.String description,

javax.activation.DataHandler handler)
```

This method triggers the preparation of user generated content. This request must be signed by the requester in order to be authenticated. One file per method call is uploaded.

Parameters:

`receiverId` - the unique receiver id

`dataSignature` - the user signature over the provided content

`receiverCertificate` - the X.509 certificate of the user

`description` - a short textual description for the provided content

`handler` - the actual content in binary format (streamed via MTOM attachment)

subscribe

```
public boolean subscribe(int channelId,
                         long receiverId)
```

This method can be used in order to subscribe receivers to a “channel”. When this method is called, the binary tree storing the receiver information of the given channel is updated. The given receiver will be able to access future

content of the certain channel. (Remark: There is a 1:N relationship between content provider and channel, i.e. there can be different channels from the same provider. Regarding the prototype implementation, only 1 provider and 1 channel is used.)

Parameters:
`channelId` - determines the channel
`receiverId` - determines the receiver to be revoked

Returns:
boolean - true in case of success, false otherwise

unsubscribe

```
public boolean unsubscribe(int channelId,
                           long receiverId)
```

This method can be used in order to revoke receivers from a "channel". When this method is called, the binary tree storing the receiver information of the given channel is updated. The given receiver will be excluded from access of future content.

Parameters:
`channelId` - determines the channel
`receiverId` - determines the receiver to be revoked

Returns:
boolean - true in case of success, false otherwise

getAvailableContent

```
public byte[] getAvailableContent()
```

This method can be used in order to request a list of available content. Each content item, which has been provided by the 3D Content Server using the uploadData method, will be part of this list.

Returns:
byte [] - a serialized HashMap containing pairs of content reference and content description

2.1.2 Internal Structure

In contrast to chapter 2.1.1., which depicts the API, figure 2 shows the internal structure of the Security Server, i.e. the most important classes.

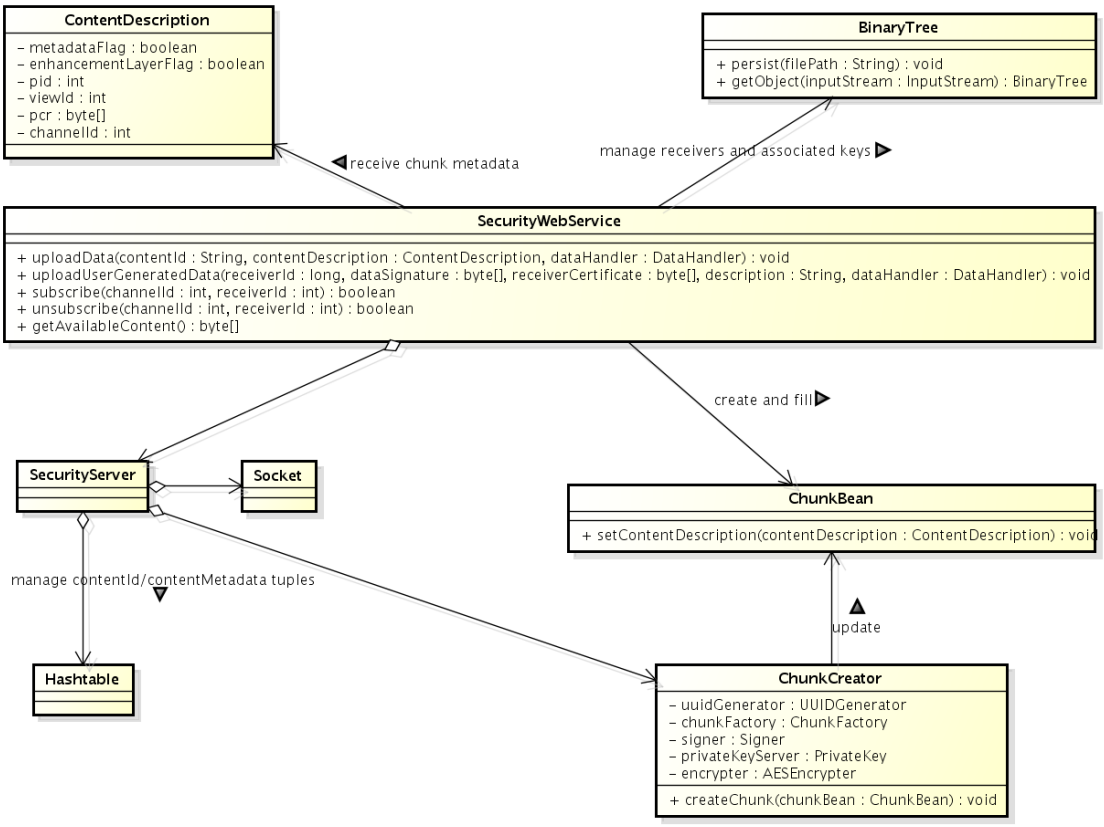

Figure 2 – Security Server (internal structure)

2.1.3 Performance

The following section provides the details of the test setup, including the used hardware, software and test content parameters.

- Intel Core 2 Duo CPU (T7500@2,2 GHz), 2 GB Ram, Windows 7 (64 bit)
- Content: ~200 MB size, ~6500 chunks having sizes between 20 kb and 70 kb per chunk, real time data rate: ~1,1 MB/s
- 10 test runs
- Max. data rate of the Security Server: ~1,7 MB/s
- Peak CPU usage: 40%
- Peak Heap Memory Usage: 100 MB

2.2 Security Client

The Security Client decrypts P2P chunks received from the P2P Client and validates content integrity and authenticity. In case of validation failure, e.g. due to transmission error or manipulation, it notifies the P2P Client to recover data and/or to prevent further sharing of corrupted data. Finally it sends decrypted and validated MPEG2-TS packets to the AV-Sync Module.

The main functionalities of this module are:

- to authenticate data coming from the P2P Software Module and to provide feedback in case of failed authentication
- to decrypt data coming from the P2P Client
- to manage related key information (for both content decryption and authentication)

2.2.1 API

Figure 3 provides an overview of the functionalities and input and output channels of the Security Client.

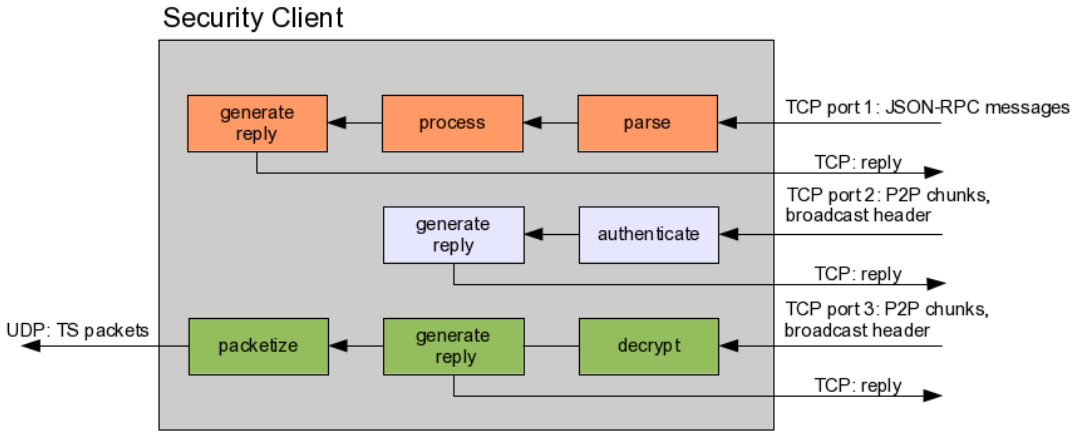

Figure 3 – Security Client (functionalities, input and output channels)

TCP port 1 – message port

The following messages are supported:

UserLogin:

- required for uploading user generated content, in order to sign the upload request
- the password unlocks the private key of the user

Request example:

```
{
	"jsonrpc":"2.0",
	"method":"UserLogin",
	"params":
		{
			"ConnNumber":301,
			"login":"name",
			"password":"changeme"
		},
	"id":1
}
```

Response example:

```
{
	"jsonrpc":"2.0",
	"result":true,
	"id":1
}
```

JSON schema for the request:

```
{
	"properties":{
		"method":{
			"type":"string"
		},
		"params":{
			"type":"object",
			"properties":{
				"ConnNumber":{"type":"number"},
				"login":{"type":"string"},
				"password":{"type":"string"}
			}
		},
		"id":{
			"type": "number"
		}
	}
}
```

UploadContent:

- triggers the upload of user generated content to the server (chapter 3.4 depicts the related work flow, including the server side processing)

Request example:

```
{
	"jsonrpc":"2.0",
	"method":"UploadContent",
	"params":
		{
			"ConnNumber":302,
			"path":"mymovie.avi",
			"description":"some info about the content"
		},
	"id":2
}
```

Response example:

```
{
	"jsonrpc":"2.0",
	"result":true,
	"id":2
}
```

JSON schema for the request:

```
{
	"properties":{
		"method":{
			"type":"string"
		},
		"params":{
			"type":"object",
			"properties":{
				"ConnNumber":{"type":"number"},
				"path":{"type":"string"},
				"description":{"type":"string"}
			}
		},
		"id":{
			"type": "number"
		}
	}
}
```

GetAvailableContent:

- requests a list of available P2P content to be shown to the user for selection

Request example:

```
{
	"jsonrpc":"2.0",
	"method":"GetAvailableContent",
	"params":
		{
			"ConnNumber":303
		},
	"id":2
}
```

Response example:

```
{
	"jsonrpc":"2.0",
	"result":
		[1,
		[["2F2CCCEB-62DA-451F-9CAF-495DE9429F54",
		"Fancy Band in Surrey"]]],
	"id":2
}
```

JSON Schema for the request:

```
{
	"properties":{
		"method":{
			"type":"string"
		},
		"params":{
			"type":"object",
			"properties":{
				"ConnNumber":{"type":"number"}
			}
		},
		"id":{
			"type": "number"
		}
	}
}
```

TCP port 2 – chunk authentication port
This port accepts binary data in the format specified in Chapter 2.3. For each received binary chunk, there is a response message indicating the status of the binary chunk.

Possible return values are:
OK – chunk has been successfully validated
INVALID – signature is missing or corrupt
ERROR – the chunk structure is corrupt

TCP port 3 – chunk decryption port
This port accepts binary data in the format specified in Chapter 2.3. For each received binary chunk there is a response message indicating the status of the binary chunk.

Possible return values are:
OK – chunk has been successfully decrypted
ERROR – the chunk structure is corrupt, i.e. the chunk does not comply with the format specified in Chapter 2.3

2.2.2 Internal Structure

In contrast to chapter 2.2.1., which depicts the API, figure 4 shows the internal structure of the Security Client, i.e. the most important classes.

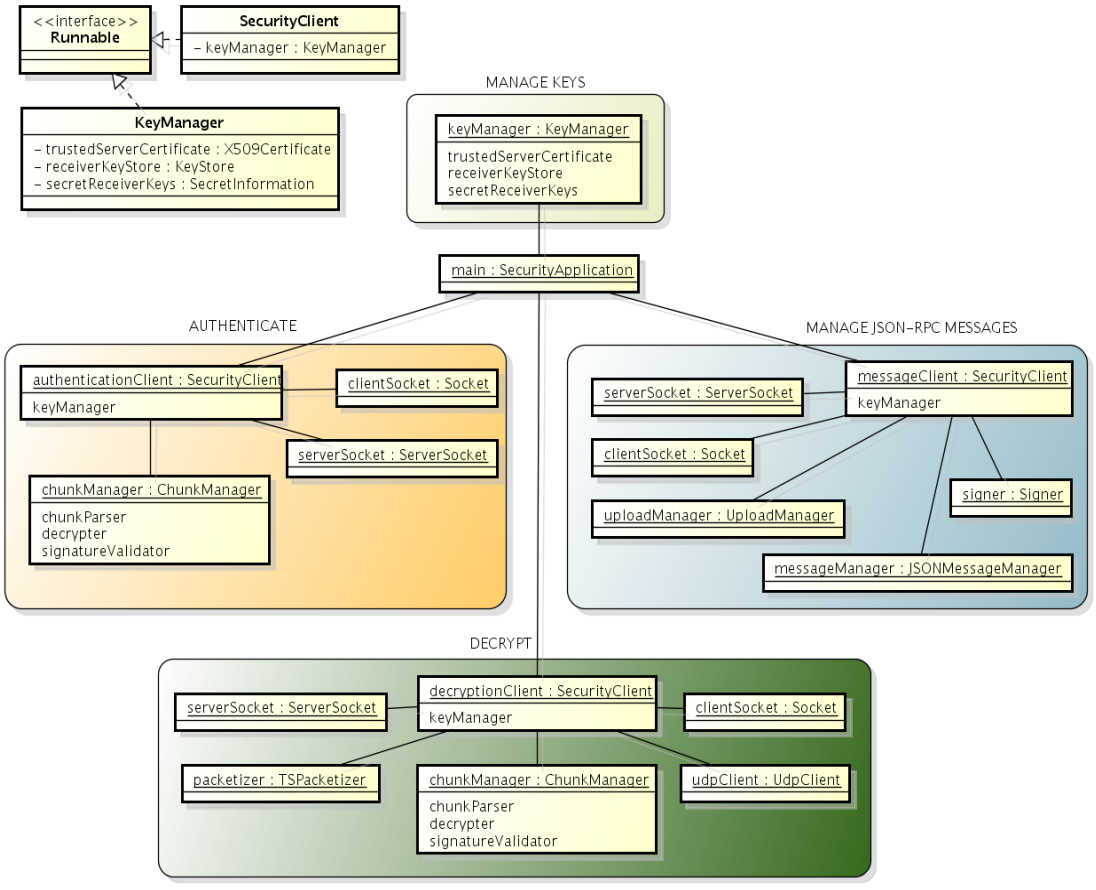

Figure 4 – Security Client (internal structure)

2.2.3 Performance

The following section provides the details of the test setup, including the used hardware, software and test content parameters.

- Intel Core 2 Duo CPU (T7500@2,2 GHz), 2 GB Ram, Windows 7 (64 bit)
- Content: ~200 MB size, ~6500 chunks with a size between 20 kb and 70 kb per chunk, real time data rate: ~1,1 MB/s
- 10 test runs
- Decryption:
    - Max. data rate: ~5,9 MB/s
    - Peak CPU usage: 14%
    - Peak heap memory usage: 18 MB
- Authentication:
    - Max. data rate: ~1,9 MB/s,
    - Peak CPU usage: 40%
    - Peak heap memory usage: 40 MB

2.3 P2P Chunk Header Format

For each chunk received from the 3D Content Server, the Security Server creates a “P2P chunk” consisting of a header (see Table 1) and the actual payload, which could be AV-data, metadata or key information.

Byte	Bytes	Parameter
1	1	Flags Bit 1 set: payload is encrypted Bit 2 set: payload contains enhancement layer information Bit 3 set: payload has been signed Bit 4 set: payload contains “content metadata” (instead of AV-content) Bit 5 set: payload contains audio data Bit 6/7: reserved for future use Bit 8 set: payload contains “key information”
2 - 17	16	Chunk ID: unique chunk identifier
18 - 33	16	Content ID: unique content identifier
34 - 37	4	PID: packet ID, used for identifying different elementary streams
38 - 43	6	PCR: program clock reference, used for synchronizing P2P and DVB-T streams
44 - 47	4	Chunk count
48 - 303	256	Signature field (containing: length of the signature (4 bytes), the actual signature over the header - excluding the signature field itself, 24 bytes payload hash)
304	1	View ID: identifies different video views
305 - 308	4	Payload size (in bytes)

Table 1 – P2P Chunk Header Format

3 WORKFLOWS

3.1 P2P Chunk Generation (at the server side)

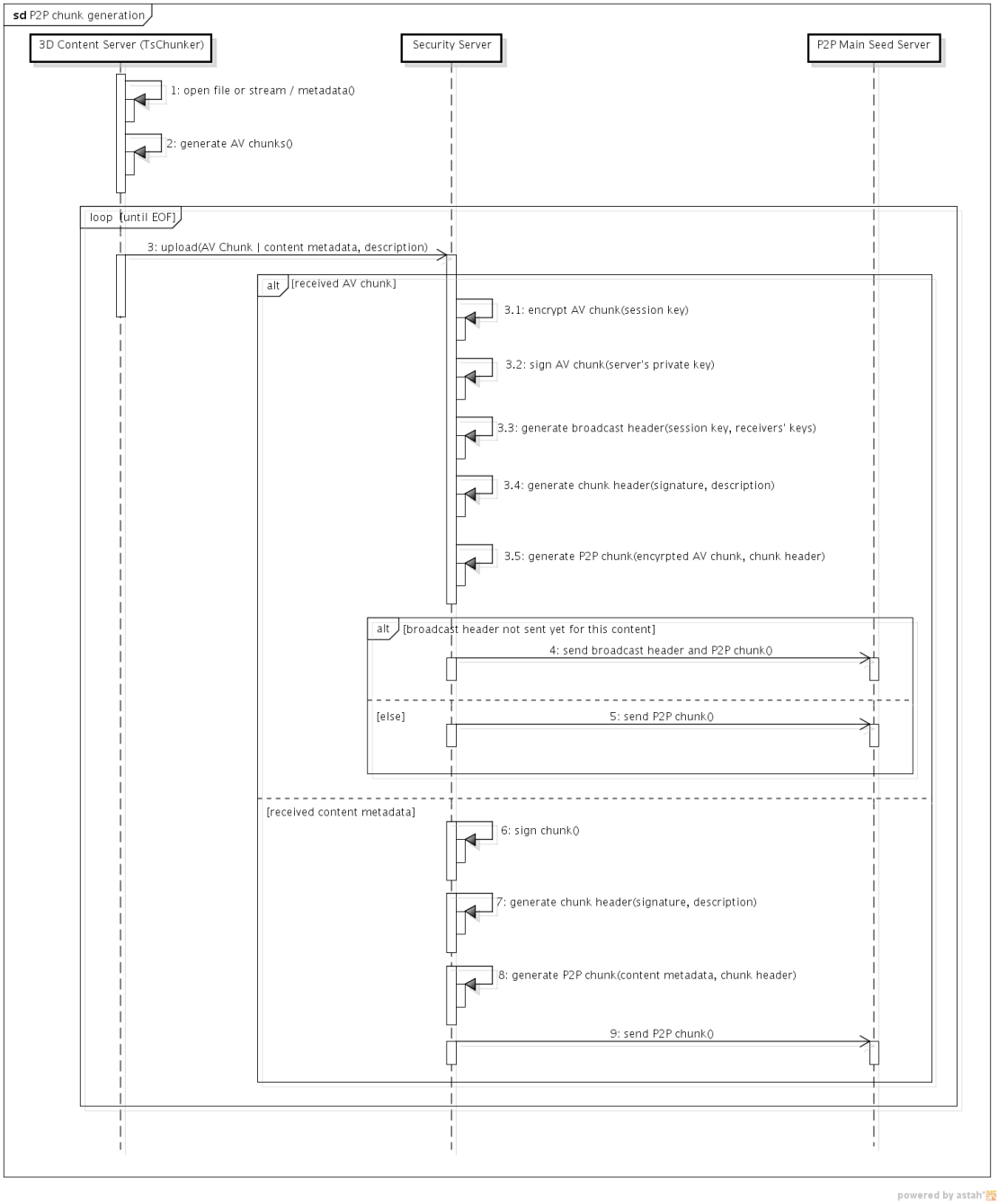

Figure 5 – Work flow for the generation of P2P chunks at the server side

In order to prepare the content for the P2P distribution, video and audio streams (MPEG2-TS) are encapsulated in so called “P2P Chunks”. In the first step, the 3D Content Server (TsChunker module) generates chunks for each stream belonging to certain content (e.g. different views/ quality layers/ audio objects). In order to generate chunks that can be decoded independently, the size of a single chunk is defined by one GOP (Group of Pictures - frames

between two key frames that can be decoded without additional information) for video streams and one second for audio streams. The video streams are split into “base layer chunks” that are mandatory for video decoding, and “enhancement layer chunks” that are available for decoding only if the communication channel capacity is high enough. MPEG2-TS chunks and related information are transferred to the Security Server. The Security Server generates “P2P Chunks” from the MPEG2-TS chunks, which consist of a header (see Table 1) and the payload.

Key Management

Within the DIOMEDES project, an important aspect is to implement access control mechanism for content to be distributed via P2P communication. While all receivers should be able to participate in the distribution process, only authorized users should have access to the actual content. In general, the broadcasting scheme to be used should be able to encrypt a message so that multiple users are able to decrypt it.

A broadcast message is usually divided into 2 parts: a header (“broadcast header”) and a body part. The body contains the protected content and the header contains information needed to access the content (i.e. key information).

Regarding the choice of an appropriate broadcast encryption scheme for DIOMEDES, the following factors have been identified as relevant: the number of sources (multi-source vs. single-source), the availability of additional communication channels besides the broadcasting channel(s), and performance (considering various criteria). Performance criteria include the amount of encrypted session keys (header size) to be broadcast, the amount of keys to be stored by each receiver (storage space), and the computational overhead for receivers (processing time). Depending on the specific setup, the importance of each criterion may differ. Considering the above, the following (non-functional) requirements have been identified for the DIOMEDES project [1]:

1. There is no additional (secure) point-to-point channel besides the broadcast channels DVB-T and P2P available, thus a stateless broadcast encryption scheme is preferred. (stateless schemes: schemes that do not require additional communication channels for distributing key information)
2. A single-source broadcast setup is used, i.e. the application of public key cryptography is not required.
3. The processing delay introduced by the security operations should be as low as possible, i.e. processing time is an important factor.
4. Depending on the availability of tamper resistant memory on the receiver device, the required space for storing keys might become important.
5. Depending on the broadcast channel to be used for the distribution of key information, the size of the broadcast header might be an important factor, e.g. the header size is more relevant for the DVB-T channel than for the P2P channel.

Based on the requirements stated above, various broadcast encryption schemes have been evaluated (see Chapter 4 for details).

The selected (and implemented) broadcast encryption scheme (“Complete Subtree” – CS) is based on a balanced binary tree of height *log* where the leaves represent the *N* receivers (Figure 6). During the setup of the system, unique AES keys (128 bit) are generated and assigned to each vertex in the tree. In a next step, each receiver is provided with a secret information (*log N* keys of all its tree ancestors, see Figure 6) to be stored in a tamper resistant area (e.g. on a Smart Card or in the memory of the receiver device). A 1:1 relationship between channel and tree is assumed, i.e. for each channel (which will play out different content), there is exactly one tree. This means the owner of the “secret information” is authorized to access all content related to a specific channel (resp. tree).

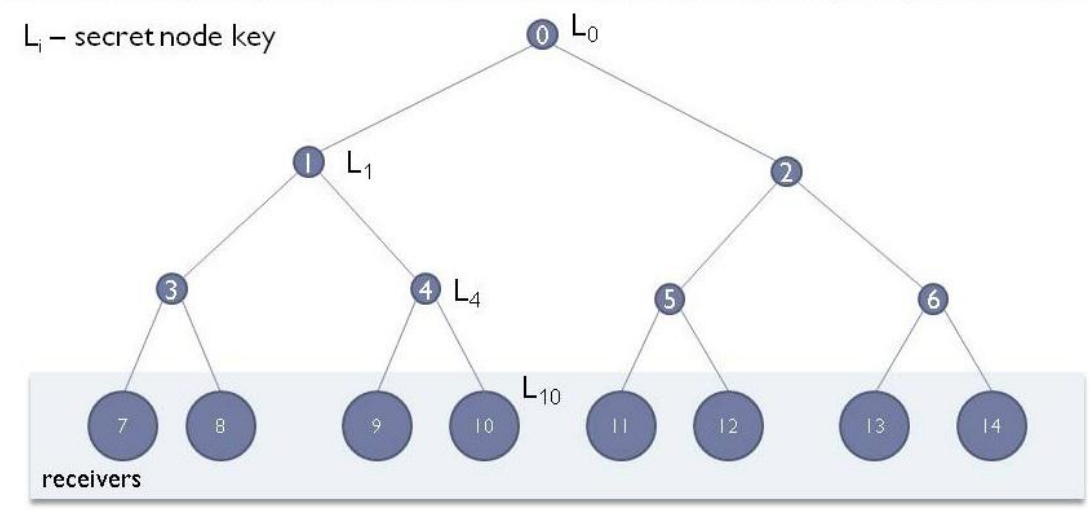

Figure 6 – Example binary tree for storing receiver keys

To find the cover for a set of privileged receivers, the *Steiner tree* for the set of revoked receivers is created by marking the edges between the revoked receivers and the root (Figure 7). Due to the tree representation, a single message can be used to distribute this information (the so called "broadcast header") to all receivers. On the receiving side, a matching key for one of the privileged subsets needs to be found and subsequently, the actual decryption needs to be performed.

The actual content is distributed only once, encrypted with a session key *K* (also 128 bit AES), while the session key *K* is encrypted with the keys of the privileged subsets.

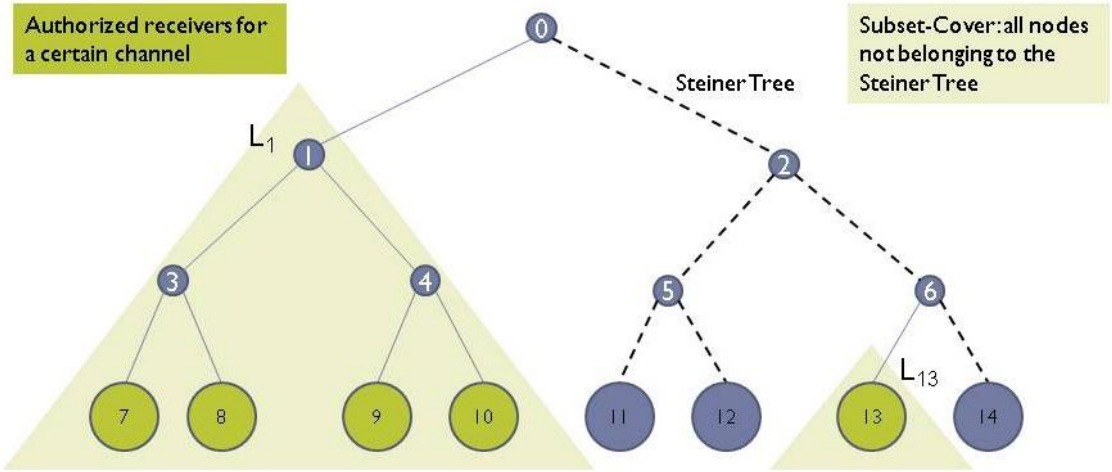

Figure 7 – Example Steiner Tree

Figure 7 shows an example Steiner Tree. Leafs (receivers) L_{11}, L_{12} and L_{14} are marked as revoked. By removing the resulting Steiner Tree from the binary tree, two subsets L_1 and its descendants and L_{13} are remaining.

The encryption process consists of the following steps:

ENCRYPT

K – session key
H – broadcast header
C – plain content
B – broadcast body (the encrypted content)
S_r – receiver's secret key set
ENC - encrypt

1. Generate the receiver's secret information S_r and store it at the receiver (see Figure 6)
2. Generate session key K
3. B = $ENC_K(C)$
4. Generate broadcast header H = $ENC_{Sr}(K)$

3.2 P2P Chunk Processing (at the client side)

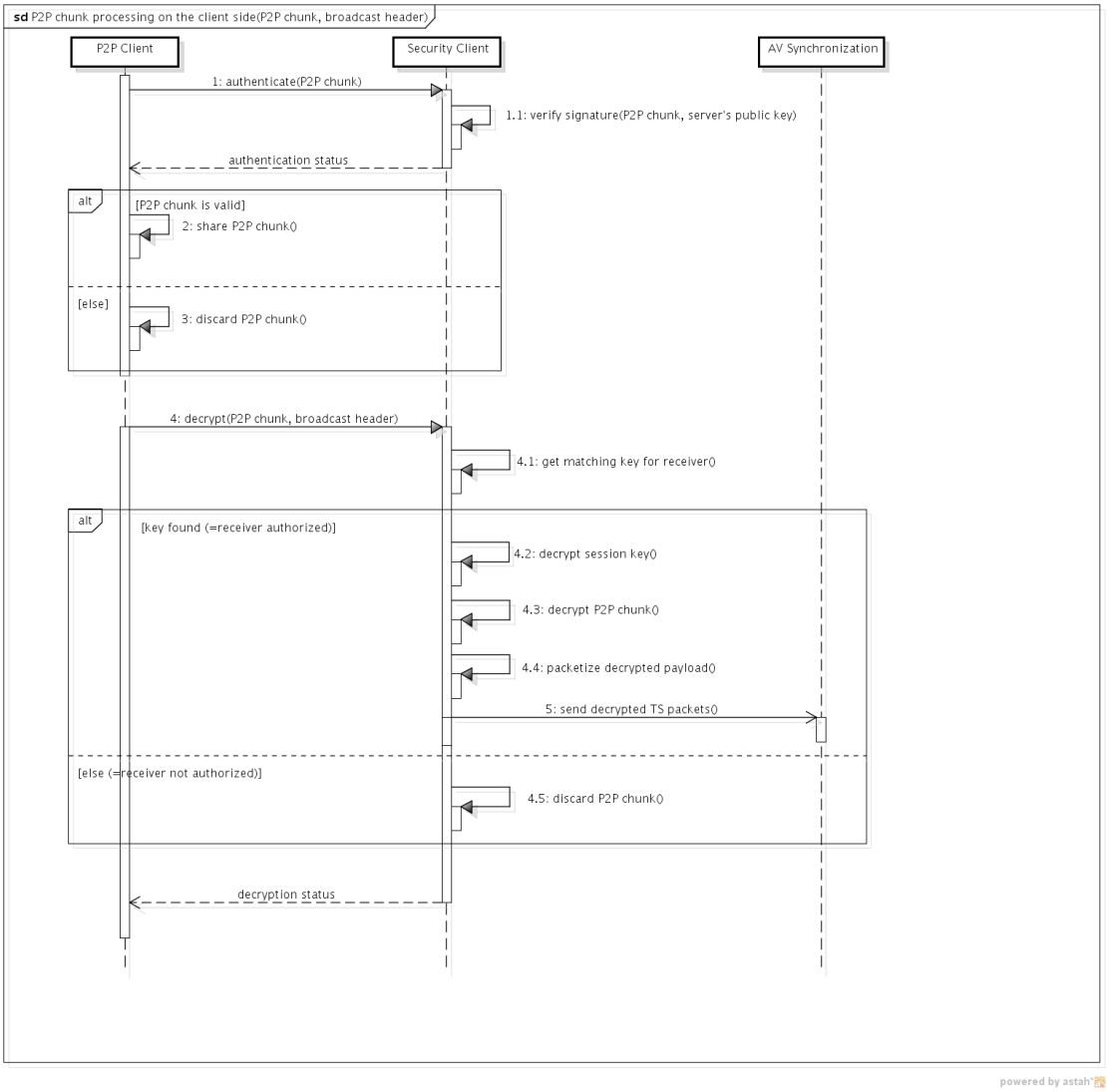

Figure 8 – Workflow for the processing of P2P chunks at the client side

Figure 8 shows the data flow at the client side. Once the P2P chunks are received by the P2P Client, the Security Client analyzes the chunk header and verifies the signature. In case of success, the P2P client immediately shares the chunks with other peers. Chunks, which have been successfully verified, are ready for decryption by the Security Client. The Security Client extracts and decrypts the payload. The "broadcast header" containing the key information required for decrypting the AV content, is also distributed via the P2P network using special P2P chunks ("key info chunks"). After decrypting and de-packetising, the payload is sent to the AV-Sync Module.

The decryption process can be defined as follows:

DECRYPT

K – session key
H – broadcast header
C – plain content
B – broadcast body (the encrypted content)
S_r – receiver's secret key set
DEC - decrypt

1. $K = DEC_{Sr}(H)$
2. $C = DEC_K(B)$

3.3 Discovery of available P2P content

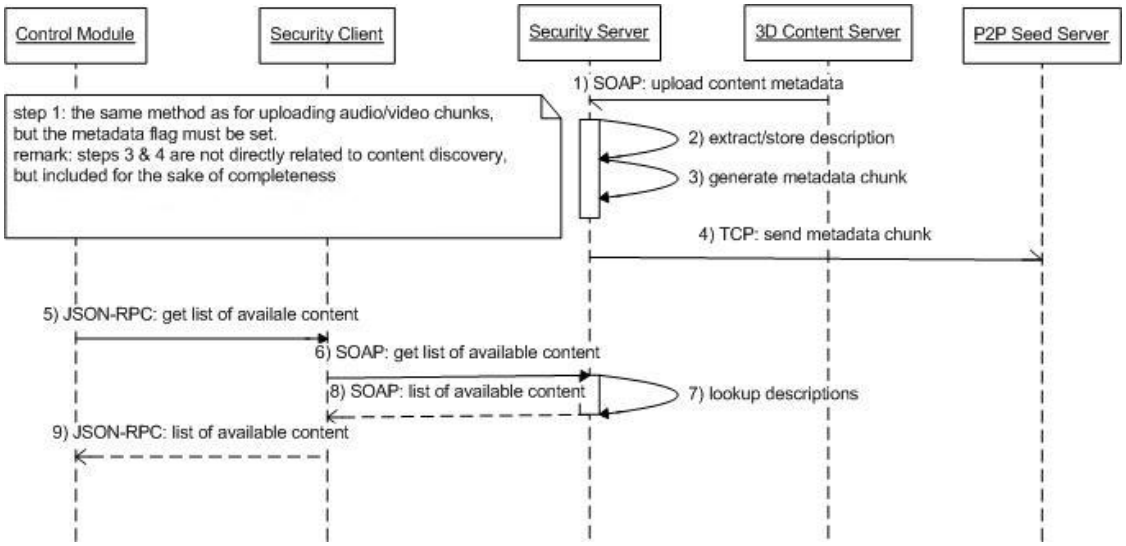

Figure 9 – Workflow for discovering available P2P content

Figure 9 depicts the P2P content discovery workflow. For each content item, there is so called "content metadata". A "content metadata" file is a text file in JSON format providing information about the related content. The Security Server stores the metadata files and provides an interface for accessing the contained information.

3.4 Upload of "user generated content"

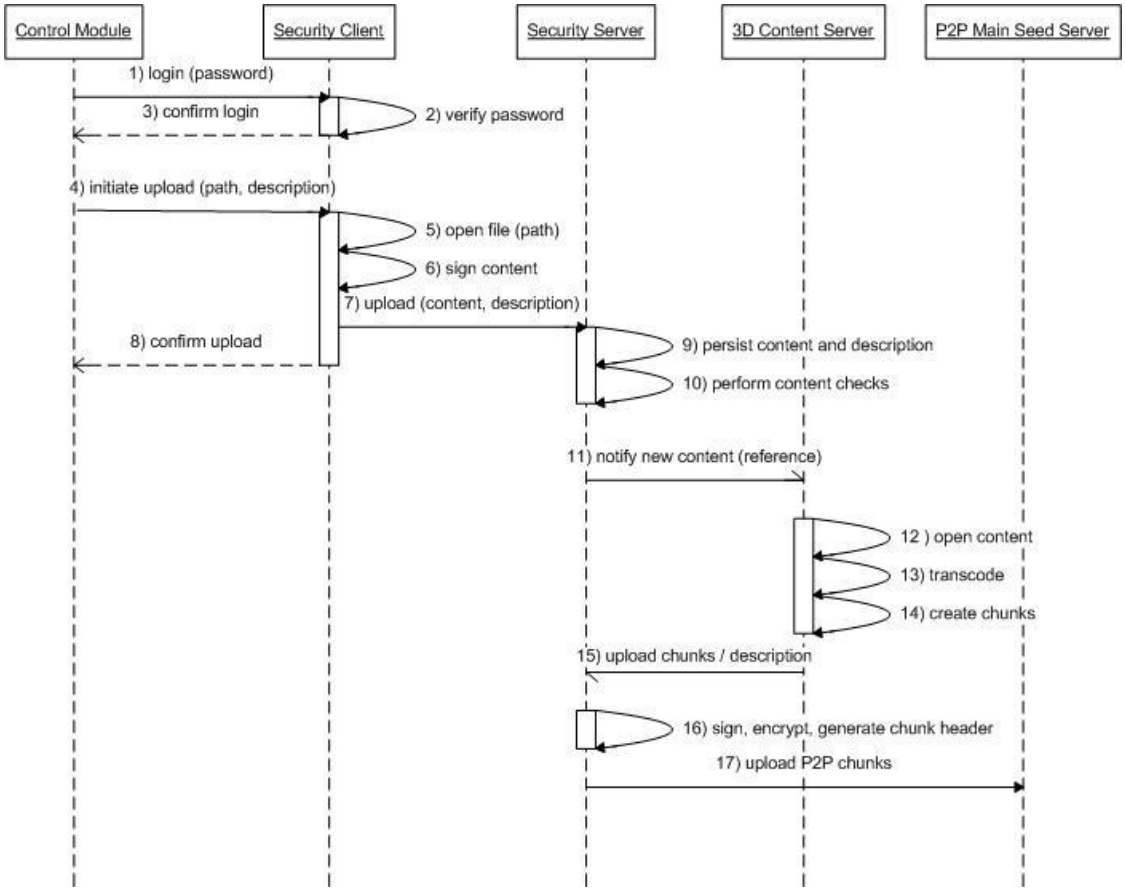

Figure 10 – Workflow for uploading "user generated content"

Figure 10 shows the steps for uploading the "user generated content". In order to establish non-repudiation during the upload of "user generated content", the user needs to digitally sign the upload request using his own private key that is encrypted with the user's secret password.

4 ALGORITHM EVALUATION

This chapter provides an update of the algorithm evaluation based on the requirements of the integrated demo-setup, especially considering the average chunk size as produced in the 3D Content Server.

4.1 Recommendations

In order to select appropriate algorithms and key lengths, the recent recommendations (2011) of the ECRYPT EU-project [3] have been taken into consideration:

"*ECRYPT II - European Network of Excellence for Cryptology II* is [...] funded within the Information & Communication Technologies (ICT) Programme of the European Commission's Seventh Framework Programme (FP7) under contract number ICT-2007-216676."

The following table summarizes the recommendations regarding encryption and signature algorithms [2]. For the DIOMEDES security prototypes, level 5 key lengths have been considered as minimum and level 7 key lengths as maximum values.

Protection Level	**Symmetric**	**Asymmetric**	**Elliptic Curve**	**Hash**
5 Legacy Standard Level (2011 – 2020)	96 bit	1776 bit	192 bit	192 bit
6 Medium-term (2011 – 2030)	112 bit	2432 bit	224 bit	224 bit
7 Long-term (2011 – 2040)	128 bit	3248 bit	256 bit	256 bit

Table 2 – Recommended key lengths [2]

4.2 Symmetric Encryption Algorithms

	Tested Key Length	***Performance***	***Implementations (Java)***	***IP & Costs***
Twofish	128 bit	*44,5 MB/s*	*BouncyCastle*	not patented, free use
Serpent	128 bit	*36,0 MB/s*	*BouncyCastle*	not patented, free use
AES (Rijndael)	**128 bit**	***50,3 MB/s (ECB-mode)***	***BouncyCastle, SunJCE***	**not patented, free use**
RC6	128 bit	*66,1 MB/s*	*BouncyCastle*	patented (RSA Laboratories), may require licensing and royalty payments
MARS	128 bit	27,9 MB/s	*No common provider*	not patented, available worldwide under a royalty-free license (IBM)

Table 3 – Comparison of symmetric encryption algorithms

Table 3 compares common symmetric encryption algorithms regarding their performance, support by software-frameworks, intellectual property rights and costs. The test setup is as follows:

- Intel Core2 Duo E8500 @ 3,16 GHz, 4GB Ram
- Linux (Mandriva 2008.1), kernel version: 2.6.35.13-92.fc14.x86_64
- JRE 1.6.0_21-b06
- Cryptography Frameworks: Bouncy Castle (BC), MARS implementation by Popa Tiberiu (Avalable at: http://n3vrax.wordpress.com/2011/07/28/mars-encryption-algorithm-in-java/)
- 1000 encryption jobs with 50kb input each

For the prototype implementation, the **AES algorithm using 128 bit keys** has been selected (RC6 provides better performance, but it is patented and license payments may be required). According to Table 2, this can be considered as “Level 7 security”. (Remark: “Level 5 security” would have been sufficient for the project’s purposes, but the BC provider does not support key lengths lower than 128 bit for the AES algorithm.)

4.3 Signature Algorithms

	Tested Key Length	***Performance***	***Implementations (Java)***	***IP & Costs***
RSA/SHA-256	1776 bit	*3,3 MB/sec*	*BouncyCastle, SunRsaSign*	not patented, free use
ECDSA/SHA-256	**192 bit**	***4,9 MB/sec***	***BouncyCastle***	**not patented, free use**

Table 4 – Comparison of signature algorithms

Table 4 compares common signature algorithms regarding aspects such as performance, support by software-frameworks, intellectual property rights and costs. The test setup is as follows:

- Intel Core2 Duo E8500 @ 3,16 GHz, 4GB Ram
- Linux (Mandriva 2008.1), kernel version: 2.6.35.13-92.fc14.x86_64
- JRE 1.6.0_21-b06
- Cryptography Framework: Bouncy Castle (BC)
- 1000 signing jobs with 50kb input each

For the prototype implementation, the **ECDSA/SHA-256 algorithm using 192 bit** keys has been selected. According to Table 2, this can be considered as “Level 5 security”.

4.4 Hash Algorithms

	Tested Key Length	***Performance***	***Implementations (Java)***	***IP & Costs***
SHA256 (256 bit)	**256 bit**	***61,3 MB/s***	***BouncyCastle, SUN***	**not patented, free use**
Tiger (192 bit)	**192 bit**	***71,9 MB/s***	***BouncyCastle***	**not patented, free use**
WHIRLPOOL	512 bit	*7,7 MB/s*	*BouncyCastle*	not patented, free use
RIPEMD	*160 bit* Comment: no known attacks, can be compared to SHA1 - but due to its lesser popularity it has been not as much scrutinized, thus there is higher risk of undiscovered weaknesses	*33,7 MB/s*	*BouncyCastle*	not patented, free use

Table 5 – Comparison of hash algorithms

Table 5 compares common hashing algorithms regarding aspects such as performance, support by software-frameworks, intellectual property rights and costs. The test setup is as follows:

- Intel Core2 Duo E8500 @ 3,16 GHz, 4GB Ram
- Linux (Mandriva 2008.1), kernel version: 2.6.35.13-92.fc14.x86_64
- JRE 1.6.0_21-b06
- Cryptography Framework: Bouncy Castle (BC)
- 1000 hashing jobs with 50kb input each

For the prototype implementation, either **Tiger or SHA256** are used (depending on the application context, e.g. regarding digital signatures (chapter 4.4), there is no implementation available using the Tiger algorithm).

4.5 Key Management Approaches

From a security perspective, the main challenges in the DIOMEDES broadcasting setup are:

1. Efficient content and key distribution for large user bases
2. Access control
3. Authentication of content

While existing DRMS are able to address point 2, they do not provide solutions for point 1 and point 3. Although it would be technically possible to use existing DRMS “on top” of some broadcasting scheme (addressing point 1), there would be a large overhead when applied to small-sized chunks, such as the ones used by the DIOMEDES-P2P system.

Scheme	Processing Time	Storage Space	Header Size
CS	$\mathcal{O}(log(logN))$	$logN$	$rlog(\frac{N}{r})$
SD	$\mathcal{O}(logN)$	$\frac{log^2 N}{2}$	$2r-1$
LSD (basic)	$\mathcal{O}(logN)$	$log^{\frac{3}{2}} N$	$4r-2$
SSD (zs)	$\mathcal{O}(N^{\frac{1}{k}})$	$klogN$	kr
BGW	$\mathcal{O}(N^{\frac{1}{2}})$	$N^{\frac{1}{2}}$	$N^{\frac{1}{2}} + r\frac{logN}{8}$

Table 6 – Comparison of stateless broadcast encryption schemes [1]

Table 6 compares common stateless broadcast encryption schemes regarding processing time, storage space and broadcast header size. Parameters are: N – the total number of receivers, r – the number of revoked receivers and k – an arbitrary integer value. Overall, the Complete Subtree scheme (CS) seems to be a good choice for the specific DVB-T/P2P broadcast scenario, since this scheme provides the best trade-off:

- CS provides a good performance with respect to the processing time at the receiver
- CS provides a good performance with respect to the required storage space at the receiver
- Although CS cannot compete with other schemes for *r* << *N*, it seems to be a good choice because of its flexibility with respect to changing *r*.
- Another point in favor of CS is the low implementation effort (in comparison to the other schemes).

The detailed evaluation can be found in the recently published paper: „Access Control and Content Authentication for Hybrid DVB-T2/P2P Broadcasting“ [1].

4.6 Conclusion

Task	Selected algorithms	Summary
Symmetric Encryption	**AES (128 bit)**	• Secure, fast, patent-free • Flexible: also streaming modes are supported, which typically execute at a higher speed than block ciphers and which are particularly suited for applications where plaintext comes in quantities of unknown length. • In case OFB streaming mode is used, AES provides good robustness with respect to damaged message parts, i.e. in case bytes are damaged in transmission only those bytes in the decrypted cipher text are affected, thus the error does not propagate to other parts of the message.
Cryptographic Hashes	**SHA-256 (256 bit) or Tiger (192 bit)**	• Secure, fast, patent-free
Signatures	**ECDSA/SHA-256 (192/256 bit)**	• Secure, fast, patent-free
Key Management	**Complete Sub-tree**	• Good performance with respect to the processing time at the receiver • Good performance with respect to the required storage space at the receiver • Header size: good overall performance with respect to a growing number of revoked receivers *r* • Low implementation effort (in comparison to the other schemes)

Table 7 – Selected algorithms

Table 7 gives the summary of the selected algorithms for encryption, signing, hashing and key management.

5 CONCLUSION

This document presented how access control, content registration and content authentication are addressed within the context of a hybrid DVB-T/P2P MPEG2-TS broadcasting scenario. It was shown how the respective functionalities have been implemented within the DIOMEDES architecture. The presented approaches are agnostic to the content format and distribution channel, and thus should be adaptable to other application scenarios. The modules and workflows described in this document have been implemented and successfully tested in an integrated setup. The presented approach has been also described in a conference paper [1] (published in the proceedings of ICITST 2011).

REFERENCES

[1] J. Hasselbach, P. Aichroth. “Access control and content authentication for hybrid DVB-T2/P2P broadcasting” presented at International Conference for Internet Technology and Secured Transactions (ICITST-2011), Abu Dhabi, UAE, Dec. 2011.

[2] ECRYPT II “Yearly Report on Algorithms and Keysizes (2010-2011)”, Internet: http://www.ecrypt.eu.org/documents/D.SPA.17.pdf, 2011, [Dec. 05, 2011]

[3] ECRYPT II Project, Internet: http://www.ecrypt.eu.org, 2011, [Dec. 05, 2011]

APPENDIX A: GLOSSARY OF ABBREVIATIONS

A	
AES	Advanced Encryption Standard
A/V	Audio / Video
API	Application Programming Interface
B	
BGW	Boneh Gentry Waters (Broadcast Encryption Scheme)
C	
CS	Complete Subtree (Broadcast Encryption Scheme)
D	
DES	Data Encryption Standard
DSA	Digital Signature Algorithm
DRMS	Digital Rights Management System
DVB	Digital Video Broadcast
E	
ECDSA	Elliptic Curve DSA
ECB	Electronic Code Book (Block Cipher Mode)
G	
GOP	Group Of Pictures
I	
ICITST	International Conference for Internet Technology and Secured Transactions
IP	Intellectual Property
J	
JSON-RPC	JavaScript Object Notation – Remote Procedure Call
JRE	Java Runtime Environment
L	
LSD	Layered Subset Difference (Broadcast Encryption Scheme)
M	
MTOM	Message Transmission Optimization Mechanism
MPEG-TS	Motion Picture Experts Group - Transport Stream

N	
NIST	National Institute of Standards and Technology (USA)
NSA	National Security Agency (USA)
O	
OFB	Output Feedback Mode (Block Cipher Mode)
P	
P2P	Peer to Peer
PCR	Program Clock Reference
PID	Packet ID
R	
RC6	Rivest Cipher 6 (symmetric encryption scheme)
RSA	Rivest Shamir Adleman (asymmetric encryption scheme)
S	
SOAP	“Simple Object Access Protocol” – not used anymore as acronym
SHA	Secure Hash Algorithm
SSD	Stratified Subset Difference (Broadcast Encryption Scheme)
SD	Subset Difference (Broadcast Encryption Scheme)
T	
TCP/IP	Transmission Control Protocol / Internet Protocol
TS	Transport Stream
U	
UDP	User Datagram Protocol
UUID	Universally Unique Identifier
X	
X.509	Standard for a public key infrastructure