U N I V E R S I T Y OF T A R T U
Faculty of Mathematics and Computer Science
Institute of Computer Science

Ilja Livenson

VirtualLife Security Infrastructure

Master's Thesis

Supervisor: Dan Bogdanov, MSc

Author: ..... "....." June 2009

Supervisor: ..... "....." June 2009

Accepted for defence
Professor: ..... "....." June 2009

TARTU 2009

Contents

List of acronyms

2D	2 Dimensional
3D	3 Dimensional
CA	Certification Authority
GUID	Global Unique Identifier
IM	Instant Messaging
MAC	Message Authentication Code
MMORPG	Massively Multiplayer Online Role Playing Game
NAT	Network Address Translation
P2P	Peer-to-Peer
PKI	Public Key Infrastructure
RA	Registration Authority
SSL	Secure Socket Layer
TLS	Transport Layer Security
VL	VirtualLife
VN	Virtual Nation
VoIP	Voice over Internet Protocol
QoS	Quality of Service
VZ	Virtual Zone
VC	Virtual Client

Chapter 1

Introduction

In the recent years advances in the quality and speed of network connections have made online collaborative environments extremely popular. The most typical examples include massively multiplayer online role-playing games (MMORPGs) like World of Warcraft (over 11 million paying subscribers as of 2009) and more general 3D worlds like Second Life that aim at being not only game environments but also act as e-commerce and educational platforms. The world itself is not always 3D - for example DOFUS, the leading MMORPG in France counting over 10 million users, is 2D based. Essentially, a virtual world is a simulated environment that allows its users to interact via avatars – user's computer-based representation of herself or himself.

The year 2008 saw a significant increase in the popularity of 3D worlds and the interest towards these technologies seems to be constantly growing. Massive media coverage of the events that took place in the Second Life world – public lectures and art exhibits to name only a few – also spawn more and more interest from the industry and research institutions towards the phenomenon as a whole.

Specific "events" – conferences, seminars or business focused meetings – are exclusively dedicated to virtual worlds, for example *Metaverse U Conference* in Stanford University, *State of the play* conference series dealing with the intersection of virtual worlds, games and the law and *Engage! Expo* (previously "Virtual Worlds Conference"). This trend seems to confirm the prediction recently done by the market research group Gartner [Gar], according to which, "by the end of 2011, 80 percent of active Internet users (and Fortune 500 enterprises) will have a "second life", but not necessarily in Second Life". A similar trend is identified in the Metaverse roadmap [SSea09], a study developed by a group of researchers and industry players with the aim of identifying the future of the 3D web, in ten years well have approximately 1,5 billions users of "various forms of 3D-enabled Web".

For companies, creating a virtual presence in terms of becoming part of existing virtual world or even creating their own is an interesting use case. It can result in improved marketing and visibility as such cases are still not too common. This

can also result in new economic and business models: for example, by using a 3D world as a new sales channel or providing premium services to the inhabitants of the virtual world. There's a risk involved, of course, as the development and running of such services is often an expensive undertaking that may or may not pay off.

For more information about the current state of the market and foreseen trends, visit the Metaverse Roadmap group project site [SSea09] or K-Zero (a UK company specialising in virtual worlds) Radar [com09].

1.1 Motivation

The sizes of user bases and revenues of existing virtual worlds are increasing (consider $1.4 billion World of Warcraft earned in 2008 [HR]) and this makes the search for improvements in this area economically motivated. Like all large and complex systems, virtual worlds have a lot of difficulties, both technical and legal. Some of the most common are:

- **High running costs.** All of the popular virtual worlds are based on the client-server architecture where the client is typically a lightweight program, providing a rich user interface as its main functionality. The server on the other hand is responsible for the core functionality, i.e. providing a consistent world to multiple users, handling payments, communication (chat and VoIP) among avatars, object persistency, script calculations and so on. In practice this means that the load on the server is usually very high and handling multiple users is expensive as it requires computational clusters. Network bandwidth is another issue for such centralised worlds as 3D worlds usually require a lot of communication between server and clients.

- **Lack of strong security solution.** Very often the security infrastructure was not designed from the very start but rather added on the "need-to-secure" basis, which might lead to non-secure solutions. Additionally, if the virtual world is to provide also legally binding services – for example e-commerce, own currency, support for signing contracts and so on – it must comply with the laws that contain certain requirements to the security of the system.

- **Lack of support by the legislation system.** It is relatively easy to create a system that can collect user data. What is often missed, however, is the fact that already the fact of such collection can lead to certain legal consequences. Another problem is that supplying medium, for example peer-to-peer based distribution network, for efficient data sharing can lead to the misuse of this medium by the end-user that can affect also the provider of the service. Handling such situations is an expensive task and it makes sense to centralise it and provide already tested and legally accredited solutions. The support for virtual contracts holding in the real life is another concern.

- **Deployment issues.** Though more technical issue and seemingly not too relevant, this is one of the most irritating problems of the virtual world platforms, as it increases the time to market for the products based on the virtual world platforms. For example, consider an e-learning world based on the platform that needs around 100 ports to be open on server - this will pose considerable problems with firewalls and network administrators if the system should be deployed at public schools or universities.

The EU FP7 research project "Secure, Trusted and Legally Ruled Collaboration Environment in Virtual Life" (further referenced as VirtualLife) aims at creating a production quality platform that improves on several aspects of existing 3D world frameworks. To validate the solution, it will be initially used in the Virtual Campus scenario.

VirtualLife project includes nine partners from Estonia, Lithuania, Rumania, Germany, Italy and France, both academical institutions and commercial companies. More details about the organisational aspects of VirtualLife can be found in Appendix A.

The most characteristic features of the VirtualLife platform are:

- Strong level of security. The solution is based on the X.509 PKI security infrastructure standard and provides identity management, support for multiple certificates, single sign-on, generic authorisation framework, signing of multi-party contracts and other functionality.
- Peer-to-peer architecture. It is one of the most widely used approaches for lowering costs for running collaborative software systems that allow real time interaction. However, employing such an architecture adds problems with security and integrity of data. More information about what peer-to-peer architecture actually means in the context of the 3D world is given in Chapter 4.
- Enforceable laws and regulations. It will be possible to describe rules that govern certain parts of the virtual world, or the world as the whole. The rules can be applied to almost all aspects of the virtual world interactions, for example restrictions on the age of the participants or automatic checks for the success criteria of a digital agreement. Typical solutions for this is to have these rules written in the end-user license agreement (EULA), VirtualLife takes a step further and makes a selection them actually enforceable by implementing Virtual Nation Laws.
- Lightweight deployment. By employing service-level routing of network messages and transparant NAT-traversal, VirtualLife poses only minimal deployment requirements.

This thesis focuses on the security aspects of the VirtualLife project. The author was part of the design and implementation team working on security and network infrastructure as well as contributing to the design of the whole system. Results of the work were also described in the project deliverables, that received good evaluation from the reviewers. Although the project is still far

from completion, the main ideas of the security infrastructure are already in place and implemented.

Another aspect of VirtualLife – the peer-to-peer architecture – was an important motivator when designing the security infrastructure. The possibility to lose the single point of failure in the world is a very attractive idea. However, it introduces several complications with identity management, trust models and providing high level of quality of service. VirtualLife employs a hybrid peer-to-peer topology to better address these issues.

1.2 Structure of the thesis

This thesis contains two parts: theoretical and practical. The theoretical part has five main chapters:

- Chapter 2 gives the necessary background knowledge about the information security area;
- Chapter 3 contains a review of the existing solutions and describes the current situation in the area of virtual worlds;
- Chapter 4 gives a high-level description of the VirtualLife architecture;
- Chapter 5 contains a detailed description of the security infrastructure of the VirtualLife platform, which is the focal point of this work;
- Chapter 6 provides an analysis and comparison of VirtualLife solution with the solutions from Chapter 3.

Complimentary material about the VirtualLife project, including relevant deliverables, is attached as appendices at the end of this thesis.

The practical part is the actual implementation of the security infrastructure. Appendix C describes attached code in more details.

Chapter 2

Background Knowledge

A reader knowledgeable of basic networking and cryptography can skip forward to the next chapter.

2.1 Network architectures

2.1.1 The client-server architecture

Client-server is one of the most popular network application architectures where one single dedicated computer called *server* accepts connections from multiple *clients*. The advantage of this solution is in the simplicity of the programming model and a somewhat simple security model – typically, only the server needs to be trusted. Disadvantages include high running costs, having a single point of failure and problems with scalability. Popular examples are traditional web servers, the "World of Warcraft" MMORPG and Second Life.

2.1.2 Peer-to-peer architecture

In peer-to-peer (P2P) architecture applications, every participant can act both as a server and a client. There are several types of peer-to-peer application classes. We list the more relevant ones:

Pure P2P. In this case all of the nodes or peers are completely equal. No single point of failure exists, every peer is completely autonomous. Examples include Gnutella and Freenet.

Hybrid P2P. In case of the hybrid P2P systems, some nodes are chosen to have additional functionality (most often because their are more powerful than

the other peers in the system or have higher bandwidth). These nodes are called *supernodes*. This design is mostly used in instant messengers, for example in Skype.

Structured and unstructured P2P. In P2P systems, all nodes are usually connected into an overlay network. Based on the algorithm used for forming this network, a P2P system can be either unstructured (overlay links are established arbitrarily) or structured (establishing links follows a certain algorithm). Both P2P types have advantages and disadvantages, for example, structured networks provide very efficient "key-value" lookups; however, if the "value" is large in size or the popularity of the "key" is not uniformly distributed, unstructured networks might provide better QoS. For example, storing 3D world geometry data in a distributed hash table (DHT) is not generally feasible because that data follows power law probability distribution.

2.2 The cryptographic primitives

2.2.1 One-way functions

A ne-way function is a function that can be easily computed on every valid input, but given the image of the random input it is hard (hereinafter *hard* is used in the sense of computational complexity theory) to find the original. Though their existence is still not strictly proven, they lie at the core of the many cryptographic primitives and operations.

A special case of a one-way function is a trapdoor function. This is a function that is easy to compute in one direction, but is hard to compute in the opposite direction without special information, called the "trapdoor". Asymmetric cryptosystems are based on using such functions. Examples are the RSA cryptosystem and Rabin cryptosystem.

2.2.2 Cryptographic hash function

A cryptographic hash function is a function h that has the following properties:

1. *compression* – h maps an input x of an arbitrary length to an output $h(x)$ with a fixed length n.
2. *ease of computation* – for any input x calculating $h(x)$ is easy.
3. *preimage resistance* – for essentially all pre-specified outputs, it is hard to find any input, which evaluates to the given output. In other words, it is unfeasible to find a message that has a given hash.
4. *second preimage resistance* – it is hard to find any second input, which has the same output as any specified input, i.e. it is unfeasible to modify a message without changing its hash.

5. *collision resistance* – it is hard to find two distinct inputs of h that would produce the same output, i.e. it is unfeasible to find two different messages with the same hash.

Examples of such function include SHA-1 and MD-5. These function are used extensively in information security application, they are typical in digital signatures, message authentication codes and authentication protocols.

2.2.3 Digital signature

The purpose of the digital signature is to provide a mean for an entity to bind its identity to a piece of information. It is most commonly used for two major cases:

1. a way for the receiver of the message, sent over insecure channel, to assure that the origin of the message is what is claimed;
2. as a mean for implementation of the electronic signature that have the same legal significance as a traditional signature in many countries of Europa and US.

2.2.4 Message authentication code

Message authentication codes, or MACs, are functions used for message authentication. A MAC function accepts as input a secret key and an arbitrary-length message, and outputs a message authentication code. Its value can be used by verifiers in possession of the secret key to detect any changes to the message content. MAC, unlike digital signature, is based on a symmetric cryptosystem (see 2.3.1). As it is much faster than a digital signature, it is often used for verification of the messages once the identity of the parties have been established and a secret key (also called "session key") is shared among participants.

2.2.5 Authentication protocol

An authentication protocol is a set of steps that allows to identify one or more participating parties. It is one of the first steps in many protocols and is sometimes referenced as a "handshake".

2.3 Cryptosystems

A cryptosystem is a particular way for implementing encryption and decryption. Typically, there are at least three algorithms involved: key generation, encryption and decryption. Depending on whether encryption and decryption

keys are equal, cryptosystems are divided into symmetric and asymmetric. Both of them have merits and drawbacks, so in practice a third, hybrid, type of cryptosystems is used more often.

2.3.1 Symmetric cryptosystem

Symmetric cryptosystems are based on symmetric encryption and decryption algorithms, where encryption and decryption keys are the same or one can be transformed into another with a simple operation. Among more popular symmetric algorithms are Blowfish (used in many applications, now superseded by Twofish), RC4 (one of the most widely used stream ciphers, e.g. in Secure Sockets Layer (SSL) and Wireless Encryption Privacy (WEP)) and Rijndael (or AES - Advanced Encryption Standard). Older but widely known algorithms include also DES (Data Encryption Standard, now deprecated as it is vulnerable to brute-force attack because of the small 56-bit key) and Triple DES or 3-DES – updated version of DES, still often used in practice.

Main problems of such systems include the problem with the key distribution among involved parties. The encryption/decryption itself is however reasonably efficient and can be used to efficiently realise secure stream or block based communication channels.

2.3.2 Asymmetric cryptosystem

In an asymmetric or public key cryptosystem each party has a pair of keys - a private and a public. The first one should be kept secret while the latter should be distributed freely. The public key is used for encrypting messages that can after be decrypted only with the corresponding private key. Additionally, the private and public keys can be used for signing and verification respectively. This provides an additional security property - non-repudiation, as only the owner of the private key could have produced a certain signature.

With asymmetric cryptography one must verify whether the distributed public key indeed belongs to a certain party. This problem can tackled by building a hierarchical public key. One of such approaches is described below and is called the X.509 public key infrastructure[1]. Another major problem is the speed of encryption and decryption that is, on average, hundreds to thousands times slower than when using symmetric ciphers.

2.3.3 Hybrid cryptosystem

By combining symmetric and asymmetric cryptosystems one can get a system that incorporates benefits from both of them. A typical hybrid cryptosystem contains following two parts:

[1] X.509 is a standard currently supported by Internet Engineering Task Force (IETF). The name itself dates back to the time when it was part of the larger family of the X.500 electronic directory services.

- a key encapsulation scheme, based on an asymmetric or public-key cryptosystem;
- data encapsulation, based on a symmetric cryptosystem.

The first part is generally used to negotiate a symmetric key that is used afterwards in protocols for efficient data encryption and decryption. The key negotiation is sometimes called the "handshake".

2.3.4 Identity certificate

An identity certificate, or often simply a certificate, is a an electronic document that incorporates a digital signature to bind together a public key with identity information such as the name of a person or an organisation, their address, and so forth. The certificate can be used to verify that a public key belongs to an individual.

In its essence, a certificate is a minimalistic mapping of a public key to an identity, certified by some certificate authority (CA). This certification can be made hierarchical, thus making the problem of public key distribution much easier - only public keys of the CAs have to be distributed to provide verification possibility for all certificates trusted by those CAs. Most of the browsers nowadays have several trusted certificates already built in.

2.4 The X.509 security infrastructure

One of the most popular standards in security is the X.509 PKI standard [KSea]. It specifies standards for public key certificates, certificate revocation lists, attribute certificates and a certification path validation algorithm.

An important aspect is that unlike other industry level public key infrastructures (for example, Kerberos), X.509 is very well suited for less structured networks, for example peer-to-peer ones.

2.4.1 Public key certificates

Public key certificate is an electronic document where identity information is bound together with the public key of the identity. To assure that this binding holds, digital signature is used for signing the certificate data. Digital signature can be given either by the identity itself (a self-signed certificate) or by a trusted third party called Certification Authority (CA).

X.509 version 3 certificate includes the following information:

- version,
- serial number,

- issuer,
- validity period,
- subject,
- subject public key info,
- optional extensions.

Example of the real certificate:

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 857 (0x359)
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: DC=org, DC=balticgrid, CN=Baltic Grid Certification Authority
        Validity
            Not Before: Jul  1 12:04:50 2008 GMT
            Not After : Jul  1 12:04:50 2009 GMT
        Subject: DC=org, DC=balticgrid, OU=ut.ee, CN=Ilja Livenson
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:da:50:c7:a8:ac:df:51:21:35:dc:1a:cc:6a:da:
                    5b:64:91:b7:ab:0f:b6:aa:10:eb:3b:4f:ab:1c:2c:
                    0f:30:b4:bc:4e:66:5f:9a:53:dc:c1:f2:20:39:18:
                    88:00:3f:2c:97:2f:1e:16:70:df:b8:79:c3:f7:30:
                    b3:25:1a:41:9d:45:39:2e:02:a0:87:1f:8a:57:8c:
                    fc:99:2b:d7:c6:de:bb:42:54:f0:64:88:ca:cd:c0:
                    76:7b:55:c8:02:92:51:46:e4:46:d3:84:64:b0:a8:
                    64:3d:54:68:68:a3:50:09:32:9a:d6:91:a1:0d:83:
                    c8:41:d5:2f:76:ad:53:66:5f
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints: critical
                CA:FALSE
            X509v3 Key Usage: critical
                Digital Signature, Non Repudiation, Key Encipherment, Data Encipherment
            X509v3 Subject Key Identifier:
                F1:9C:1F:BE:22:1B:2C:26:87:77:14:71:A2:3F:CE:26:3C:50:D8:E0
            X509v3 Authority Key Identifier:
                keyid:41:4E:DE:40:E9:3E:AE:FC:69:43:FB:38:7C:A0:39:43:F1:48:1E:AC
                DirName:/DC=org/DC=balticgrid/CN=Baltic Grid Certification Authority
                serial:00

            X509v3 Certificate Policies:
                Policy: 1.3.6.1.4.1.19974.11.1.1.1
```

```
            X509v3 Issuer Alternative Name:
                URI:http://ca.balticgrid.org/
            X509v3 CRL Distribution Points:
                URI:http://ca.balticgrid.org/bgca-crl.pem

    Signature Algorithm: sha1WithRSAEncryption
        41:02:7d: ...<the rest of the signature>
-----BEGIN CERTIFICATE-----
MIID9z.. <the rest of the certificate>
-----END CERTIFICATE-----
```

2.4.2 Certificate revocation lists

If a certificate gets compromised, for example the private key was stolen, or should be disabled for any other reason, it should be added to a certificate revocation list (CRL). The users of the X.509 security infrastructure update their copy of the list periodically and use it during the certificate verification, thus keeping the system relatively safe. One of the main problems with such an approach is the inability to do online verification, hence making the attack window rather wide. A more modern certificate validation protocol exists, which is known as Online Certificate Status Protocol (OCSP). It enables near real-time checks about certificate revocation status.

2.4.3 Examples of X.509 usage

The X.509 infrastructure is used or supported by many popular protocols and applications. To name some of the most popular examples:

- SSL/TLS and SSH – it is possible to establish application layer security channels using keypairs from the X.509 credentials;
- S-MIME – a standard for public key encryption and signing of e-mail messages encapsulated in MIME.
- LDAP – Lightweight Directory Access Protocol is often used for storing user accounts. The use of X.509 allows for secure and fine-grained access to the directory;
- WS-Security – a communication protocol that allows adding security to web services. It describes how to attach signatures and encryption headers to SOAP messages, including security tokens and X.509 certificates among others.
- IPsec – Internet layer protocol for securing IP communication by securing each IP packet of data. It is based on hybrid encryption and supports X.509 certificates.
- HTTPS – Secure version of the HTTP protocol. It is basically HTTP over SSL/TLS. HTTPS has become a standard any secure web applications.

- XMPP – Extensible Messaging and Presence Protocol is a set of XML standards used primarily for instant messaging. It has a number of useful extensions, for example VoIP or support for user avatars. X.509 can be used for authentication of the users.

Chapter 3

Review of the Existing Solutions

There are a number of virtual world solutions out there that can be seen as direct competitors of the VirtualLife platform. To give the reader an overview of the current status in the state-of-the-art development in the area, we review several popular projects that share goals of the VirtualLife. As the total review would consume too much time and not be directly connected to the topic of this thesis, we limit review to the relevant aspects:

- topological structure (client-server/peer-to-peer),
- identity management,
- security infrastructure,
- communication channels (VoIP, chat, video).

The comparison of the reviewed solutions with the VirtualLife platform is done in Chapter 6.

3.1 Method of selection

In our selection of the virtual worlds the main criteria was either similar to VirtualLife area of usage or a certain design aspect that was planned also in VirtualLife – for example, peer-to-peer architecture – as it may lead to serious consequences for security infrastructure. Materials from public sources were used for this review. Additionally, most of the described platforms were installed and tested in practise.

3.2 Second Life

The closest to VirtualLife 3D world in terms of usage idea is Second Life [**?**]. It is a proprietary virtual world developed by Linden Lab, first launched on June 23, 2003. It allows its users, called Residents, to interact with each other through avatars. Residents can explore, meet other residents, socialise, participate in individual and group activities, create and trade virtual property and services with one another, or travel throughout the world, which residents refer to as the grid.

Topological structure. Second Life is based on a client-server architecture. Each region in the Second Life "grid" runs on a single core of a multi-core server. The functionality of these servers includes running scripts in the region and providing communication between avatars and objects located there. The client software in the Second Life is called the *Viewer*. The official Viewer is open source and its main task is to render 3D graphics using OpenGL technology. A number of other Viewers exist, including text-based and browser-based ones.

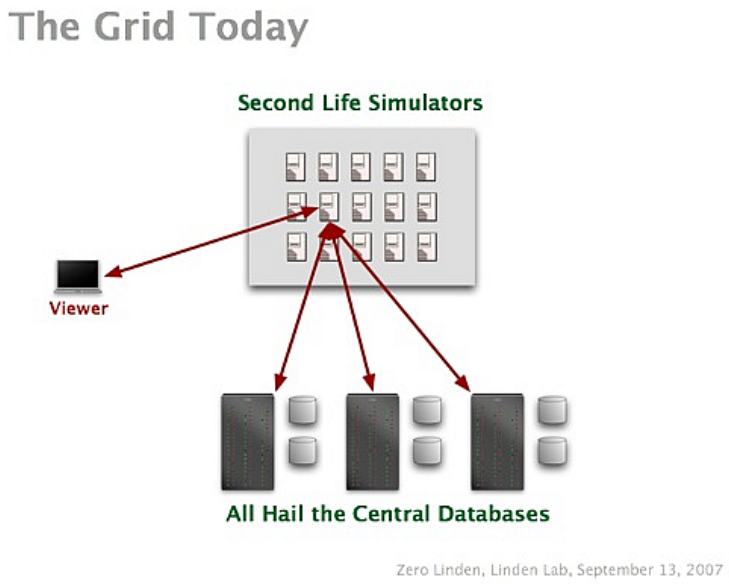

Figure 3.1: With the current Second Life solution the Viewer connects only to the region and the region keeps track of everything and acts as a proxy to the central databases.

Identity management Second Life stores all identity-related data in a central database in a custom format.

Security infrastructure Authentication is based on the username/password pair. Authentication is done over secure HTTP. Access to all assets requires a

login token for establishing identity and permissions. Multiple simultaneous connections from the same account are restricted.

Text chat and voice chat are not currently encrypted, neither is the official viewer's cache.

More information about the Second Life security can be found from the Second Life knowledge base [Lab].

Communication channels Avatars can communicate via local chat or global instant messaging (IM). Chatting is used for localised public conversations between two or more avatars, and is visible to any avatar within a given distance. IMs are used for private conversations, either between two avatars, or among the members of a group, or even between objects and avatars. Unlike chatting, IM communication does not depend on the participants being within a certain distance of each other.

Embedded voice, web, audio, and video streams do not pass through the Second Life servers; they are rather accessed directly by the Second Life viewer.

3.3 OpenSimulator

OpenSimulator [com], often referred to as OpenSim, is an open source server platform for hosting virtual worlds. It is compatible with the Second Life client and supports additional protocols for hosting virtual worlds with different feature sets. It is modular and can be extended by plugins. It is used with some modification by the realXtend platform described below in Section 3.4. Multiple servers can be integrated into a "grid" which allows larger and more complex areas to be simulated.

Topological structure. OpenSimulator has two modes of operations: standalone or grid mode. In standalone mode, everything runs in a single process. In grid mode, modules are separated into multiple processes, which can be run on different machines. There are six modules: the user server, the grid server, the asset server, the inventory server, the messaging server, and the region server. The general structure is similar to the Second Life one – a lightweight client and a server.

Identity management. Though the OpenID solution [Fou] is planned for the OpenSimulator, at the moment all identity related data is stored in a central database in a custom format.

Security infrastructure. OpenSimulator supports groups and permissions. Authentication is done using a username/password pair.

Communication channels. The main protocol supported by OpenSimulator is the Second Life protocol for client to server communication. This protocol utilises both UDP and XML-RPC. Internally OpenSimulator server components communicate with XML-RPC and REST (JSON/HTTP and XML/HTTP). It is unclear whether communications is done in an encrypted manner.

3.4 RealXtend

RealXtend [cona] offers a free open source virtual world platform with which you can use to create your own applications.

Topological structure. The RealXtend is a classical client-server 3D world. Its viewer is an open source browser based on technologies very similar to those used by VirtualLife GUI, for example using OGRE [KSL] as a 3D graphics engine.

The RealXtend server is an open source server based on the OpenSimulator platform (see section 3.3 for details) with several modifications.

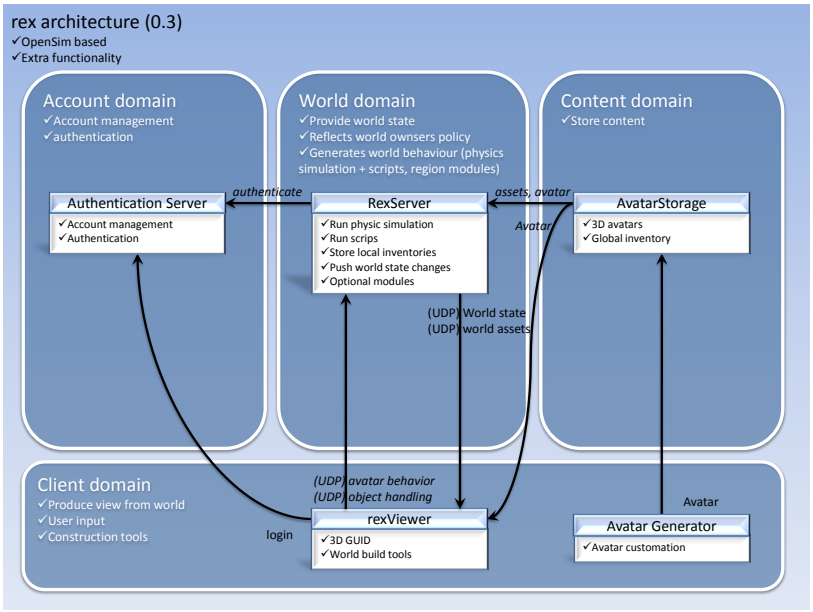

Figure 3.2: Rex architecture. Server-side has 3 modules: an account domain that handles authentication and account management, a world domain reponsible for the main actions in the world and a content domain used for storing world objects. Figure is a courtesy of RealXtend.

Other aspects. The identity management, security infrastructure and communication channels in realXtend are the same as in openSimulator.

3.5 Project Wonderland

Project Wonderland [Mic] is an open source Java programming toolkit for creating collaborative 3D virtual worlds. Apart from the typical functionality of collaborative 3D worlds, Wonderland includes a VoIP solution with the possibility to call land lines and sharing of the live desktop applications and documents. It is sponsored and developed by community developers and Sun Microsystems Laboratories. Project Wonderland, along with realXtend, are two main platforms used by the ImmersiveEducation project [pro] for creating and delivering game-based learning.

Topological structure. Project Wonderland is a client-server software platform. The server software is based upon Project Darkstar, a multi-user technology for the Java SE platform.

Project Darkstar manages a collection of objects (called ManagedObjects) and provides APIs for the synchronized update of the state of these objects in response to events from multiple clients. Updates to ManagedObjects are themselves transactional – either a coherent set of state updates happen across a set of objects or they do not happen at all. All items within a game or virtual world are represented by ManagedObjects on the server. For example, rooms, players, tools, weapons, or any other virtualized real-world objects are represented that way.

The client is a desktop Java application and is based upon two technologies to render the 3D world: Java 3D and Looking Glass 3D (LG3D). Java 3D, a standard extension to Java SE, renders sets of 3D objects to the screen, making use of hardware acceleration if possible. Project Looking Glass provides a 3D desktop environment. It provides a 3D windowing environment and a set of APIs to build 3D applications.

Identity management. Wonderland includes only a very basic profile kept at the server.

Security infrastructure. By default, Project Wonderland does not require users to authenticate to the server, i.e. they do not need to enter a password to get access to the world. An username/password authentication scheme can, however, be configured.

Communication channels. Though Project Wonderland is a client-server application, there's a possibility to create also direct channels between peers, for example for VoIP or chat communication. It should be noted that Wonderland requires a lot of open ports on the server for the normal functioning (suggested number is above 200).

3.6 Croquet and Open Cobalt

3.6.1 Croquet

The Croquet [Conb] is an open source platform on which to build virtual world applications. It features a peer-based messaging protocol. Croquet is built using the Squeak programming language, which is a dialect of Smalltalk.

Architecture Croquet is built as a peer-to-peer system. It is based upon the concept of replicated computation - rather than replicated data – and a synchronised message passing model, where the messages themselves ensure that the replicated systems remain consistent between machines.

Croquet is based on the *TeaTime* architecture – real-time multi-user architecture used for replicated computation and synchronisation. The core class Croquet's architecture is the *TObject* class, which acts as a superclass for *Tea* objects. A *Tea* object has a property that messages sent to it are redirected to replicated copies of itself located on other users' machines in the peer-to-peer network. All of the more complicated objects inside of Croquet are constructed as subclasses of *TObject*.

Though it is necessary to synchronise the world state of a new user by transferring the current contents of the world, after that, the worlds stay consistent only through the creation and processing of time based messages. Due to the high network requirements, a single peer in Croquet can handle on average up to 7-8 concurrent clients.

Security. Apart from basic username/password authentication system, Croquet does not include any significant security mechanisms.

3.6.2 Open Cobalt

Open Cobalt [ea], built on top of the Croquet platorm, is an open source virtual world browser. The Open Cobalt application is a type of 3D browser that can be used to define and access a network of interlinked 3D virtual environments, similar to the way web browsers are used to access different sites. Transportation is visually implemented by portals.

3.7 LifeSocial

LifeSocial [GPM+08] is a P2P based online multimedia platform developed at TU Darmstadt's Multimedia Communications Lab. It is not a 3D world, but shares many of the use cases with the VirtualLife platform. Apart from the basic functionality of the collaborative environments, such as creating and sharing profiles or searching for other people profiles, it offers tools for the following:

Figure 3.3: Example screenshot of the Open Cobalt application. It shows embedded web browser alongside with the 3D model of the Erechtheum temple.

- a distributed storage with access control,
- voice and text chat,
- distributed storage of personal data,
- sharing of files,
- streaming of music to your friends.

Topological structure. LifeSocial is a peer-to-peer network based on a structured overlay network, i.e. a distributed hash table (DHT). To make the programming model easier, LifeSocial also introduces the distributed data structures. These structures contain several storable objects, identified by a key, that store besides usage specific content (e.g. meta data to images) pointers to other data objects.

Identity management and security. LifeSocial's identity system is based on the usage of pseudonyms – each new node gets an unique identifier. In case of LifeSocial, they are using asymmetrical cryptographic public keys as node IDs. To attach more personal data, a self-signed document containing the username, the node identifier and the IP address of the node is stored in the LifeSocial peer-to-peer network.

Additional personal profile data and information about the groups is stored using a combination of an asymmetric cryptosystem, that is signing data with

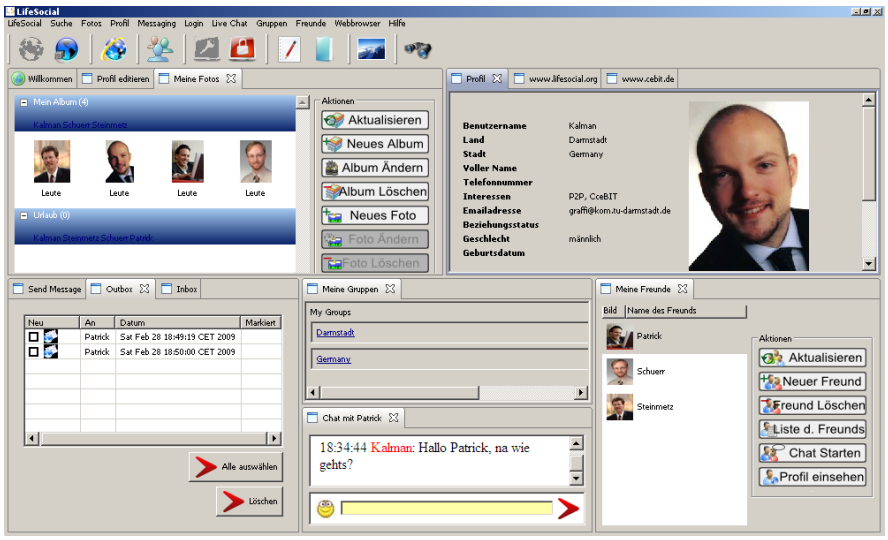

Figure 3.4: Although the graphical user interface of the LifeSocial application is not 3D, it still shares a lot of concepts similar to those of the VirtualLife.

the public key, and a symmetric one – in case the stored data is confidential, it can be encrypted with symmetric key that is shared among authorised parties.

Communication channels. LifeSocial supports chat and sharing of photos and music by using a system of plugins. Plugins can be implemented by either storing objects in the LifeSocial DHT network or by using information about the IP of the peer from the identity information.

Chapter 4

The Architecture of VirtualLife

4.1 Architectural overview

The architecture of VirtualLife is based on the idea of a connected network of peers. Though potentially, every peer can act both as a server and a client, in practice it makes sense to distinguish some more powerful nodes. So topologically, VirtulLife is a hybrid peer-to-peer network.

Each node can run various services. Examples include, but are not limited to:

- graphical user interface – built with a state-of-the-art 3D rendering engine;
- certificate authority (CA) service – for issuing lightweight certificates;
- group, account and reputation management – for storing corresponding information;
- persistence database – holds information about objects and their metadata (ownership, location, etc);
- search engine – built using the information retrieval engine Xapian that allows, among other things, full text search, approximate matching, synonyms and multilingual search.

As it doesn't make sense to run all services on one node and also taking into consideration such aspects as user configuration and machine capabilities (CPU and network bandwidth), we divide peers according to their roles into Virtual Clients (VC), Virtual Zones (VZ) and Virtual Nations (VN). The prefix "virtual" is often skipped in the following text when Client, Zone and Nation do not have other context.

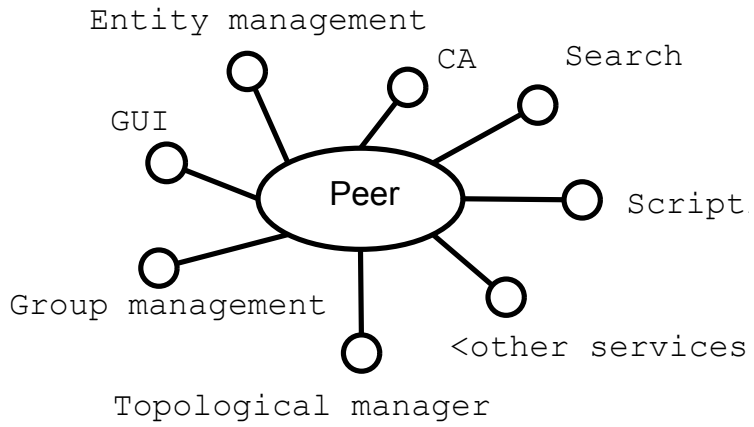

Figure 4.1: "Building block" of the VirtualLife platform. A peer that depending on the active services can act as a Client, Zone or Nation.

4.1.1 Virtual Client

The Client is typically an end-user running state-of-the-art 3D graphical user interface, though a command line client interface is also available. It connects to the Nation or the Zone (in case of a standalone Zone) and interacts with the VirtualLife.

4.1.2 Virtual Zone

A Zone is a node in the VirtualLife network. It represents a part of the world, taking care of the world geometry and assets, management of groups and permissions. The Zone gives the users possibility to create and share content and participate in other transactions.

The Zone software can be combined with the Client if, for example, the user wants to host his or her own part of the world.

Zones can be standalone, acting as a server in the typical client-server 3D world, but what makes the VirtualLife solution more interesting, is the possibility for Zones to join into Nations.

4.1.3 Virtual Nation

The Nation is an agglomeration of Zones. When a Zone decides to become a part of the Nation, it accepts the regulations of the Nation (for example, laws of simulated physics or limitations on the age of the users).

Nation also provides additional services. They are mostly limited to the ones that are needed for bootstrapping the new user, for example providing a list of

available Zones, or that are Nation specific – enabling common currency (will not be supported in the first version of the VirtualLife platform) or common Constitution a.k.a. Virtual Law.

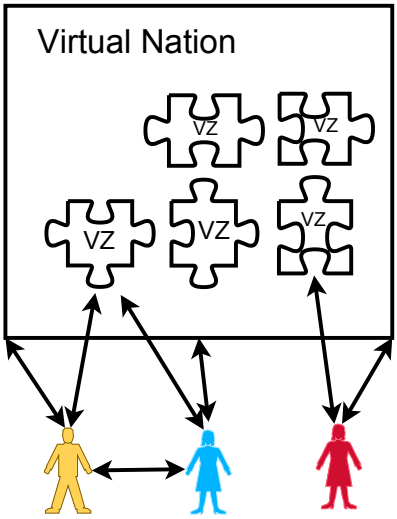

Figure 4.2: A schematic structure of the VL components in case of one Nation and five Zones. For the Clients the world entry point is the Nation server, though communication itself takes place mostly between Clients and Zones. Although the Zones do have a topological location, they are not forming a continuous 3D world. Transportation between the Zones is done via teleportation.

4.1.4 Constitution

One of the major goals of the VirtualLife project is to try and implement enforceable laws and regulations. The Constitution, or Virtual Law, is a set of directives that govern processes within VirtualLife. When a Zone joins the Nation, it starts following the Nation's laws. Hence, the user travelling within the Zones, that belong to the same Nation, can count on a certain homogeneity of the environment and provided services.

Among possible regulations are:

- limitations on public chats and client-to-client chats, for example when taking a test in the virtual university;
- location-based restrictions;
- management of contracts between users and between user and Nation;
- handling of disputes among users and between a user and Nation;
- handling of amendments to Constitution by user vote.

4.2 Interactions among peers

The communication model is important for the design and implementation of services provided by VirtualLife. The model includes not only creation and management of network connections, but also the level of abstraction of the communication channel and its representation at the network level.

We have analysed several models while choosing the communication model. For example, allowing only one Client-to-Zone connection simplifies logic of the program, but also makes it inefficient:

- the Zone node must be powerful enough to manage many connections;
- if the Zone node disappears, then the entire Zone disappears;
- additional network overhead is imposed on the Zone node, for example for routing VoIP or chat messages.

Another approach would be to establish separate connections for each new logical connection, for example new chat window or new agreement signing. Though it is a more "pure" approach and also more stable. For example, the loss of one communication partner does not influence channels with other participants. It also has a number of disadvantages:

- more complicated management of connections – one can more easily "forget" about a connection and thus cause memory leaks or a security risk;
- having a number of open ports is considered a bad behaviour by many administrators and anti-virus software tools, so it is bad for software acceptance;
- a missing single point of control over connections makes the implementation of application-wise security policies complicated.

As always, choosing something in-between is more optimal, so the VirtualLife communication model is based on the idea that at most one connection is established with every possible party. So, for example, if peer A is connected to nation B that runs 15 different services, it will only establish one network connection. All other messages will be routed within that connection. In that sense it is very similar to the point-to-point virtual private network (VPN). More details about the message routing model are given in Section 5.4.

To illustrate what kind of communication occurs between peers in the VirtualLife system, we bring some typical examples. They are not describing the system at full, but rather give an overview of what happens in the system and where security infrastructure might be applicable.

4.2.1 Client-to-Client communication

All protocols that do not depend explicitly on a third party, run in a peer-to-peer fashion – directly from client to client. It must be noted that in some cases the actual stream of bits could be routed through other peers, for example for NAT traversal or higher bandwidth, but it doesn't change the logical end-points.

Some of the more typical interaction between clients are:

- chat or multichat (in the latter case more than one client can be involved),
- VoIP communication,
- file transfers,
- signing a two-party contract.

4.2.2 Client-to-Zone communication

The Zone is a core building block of VirtualLife. It is generally a more powerful node as it must handle a higher computational and network load due to several semi-central services being run there. This includes physics engine, entity manager, geometry distribution, etc.

The Client typical communicates with a Zone in the following cases:

- logging in into the Zone, which can be transparent for the user, if he or she have logged in into the Nation,
- getting new geometry data and location of avatars and other entities,
- modifying the world, creation, moving, deleting of objects,
- managing group memberships,
- queries to the search engine,
- queries to the authorisation service.

4.2.3 Client-to-Nation communication

A Nation hosts several central services that are reasonable to have in a single instance:

- registration,
- logging in to the Nation in a single-sign-on manner,
- certification authority service,
- zone topology,

- nation-wide groups,
- status of other avatars,
- timestamping.

4.2.4 Zone-to-Zone communication

VirtualLife is not continuous at the moment – it is not possible to walk from one Zone to another. The decision was made because of the complexity of such solutions. This is also not the goal of the VirtualLife project. However, Zones do interact with each other, mostly transparent for the user, in the following cases:

- storing and retrieving objects (master copies might reside in different Zones),
- world update propagation,
- connection routing,
- asset replication.

4.2.5 Zone-to-Nation communication

It provides rules and general services for all the Clients that inhabit its Zones. As it is a single point-of-failure, it was designed to be very light-weight, so only minimal communication between Zone and Nation (or Nation and Client) is envisaged.

Some interactions that need to take place anyway, are:

- providing information about logged-in Clients,
- providing a master replica of the Virtual Law,
- providing policies, including permissions,
- topology updates from Zone to Nation.

Chapter 5

VirtualLife Security Infrastructure

One of the main aspects of the VirtualLife platform is security. As security is a very broad term and as such not a very useful one, we need to refine that. In VirtualLife security is a cross-cutting concern, which means that it affects all parts of the system. It covers all transactions allowed within the system as well as more specific aspects of the application. For example, the storage of sensitive data (private keys), trust towards Certification Authority (CA) or the influence of the reputation of the participant on his or her capabilities. The security layer of the VirtualLife infrastructure assures that the allowed operations are executed safely and only by authorised identities.

Building a secure system often means that certain sacrifices in efficiency or usability are inevitable. One of the design goals of the VirtualLife security infrastructure was to provide a possibility to adjust the level of security needed for operations. For example, if speed of data transfer is more important than its security – consider streaming world geometry data – then it could be done with a minimal overhead. However, if security is of a higher concern – think of a point-to-point chat or VoIP conversation where new deals or business ideas are being discussed – it is possible to turn on all security mechanisms, including encryption, display of partner's reputation and the reputation of the CA, that signed certificate used for establishing VoIP channel.

5.1 High-level description

The top level abstraction in the VirtualLife security infrastructure is the notion of a transaction. A transaction is an operation on some object that follows a certain strict protocol and involves one or more actors. Due to the diverse nature of transactions it makes sense to define allowed transactions in the system and analyse them in more detail.

In VirtualLife, the security infrastructure is implemented in three libraries – **vlsec**, **vlnet** and **vlprotocol** that share a common identity management

framework. Figure 5.1 describes the dependencies of the parts, higher blocks depend on the underlying ones. There are additional dependencies in the system. For example, most of the components depend on the **vlcommon** library providing basic services, but for the sake of simplicity they are not described in this work.

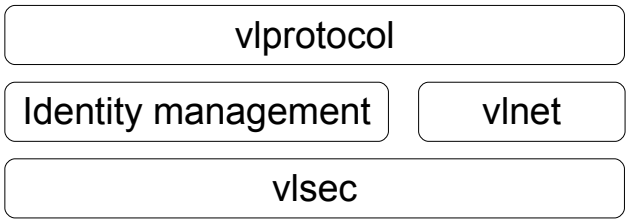

Figure 5.1: The VL security infrastructure consists of 4 main components: **vlsec**, **vlnet**, **vlprotocol** and identity management framework.

Briefly, the functionality of these parts is:

- **vlsec** – security primitives and low and mid level credential management;
- **vlnet** – implementation of the networking layer and tightly connected core like transaction authentication;
- **identity management** – a module of the program that spans several libraries and provides abstractions for managing avatar specific information. In terms of source code, **identity management** module is comprised of the classes in **vlsec** and **vlcommon** and network protocols;
- **vlprotocol** – implementation of the state machines for supported transactions.

Each component is described in more detail below. A number of third party libraries were used in the implementation of the VirtualLife security layer, the key ones are described in appendix E.

5.1.1 Security challenges

Every security aspect or transaction has attacks associated with them. Handling these possible attacks is necessary in any production quality system. More specific attacks and possible solutions are discussed below in the corresponding sections, but there are also more general and technical issues. We base our security heavily on the X.509 security infrastructure. Therefore, any attack that can compromise the security level provided by the infrastructure essentially concerns every action of an actor and system as a whole. Problems and possible solutions specific to X.509 infrastructure are described below.

Breaking private key protection. The private key could be encrypted with a very weak password. In VirtualLife we will enforce a password strength check for storing private keys.

Server spoofing. We will verify the identity of both parties in the beginning of every connection to disallow this attack.

The lunchtime attack. This attack refers to the idea that a user's computer, with the ability to decrypt, is available to an attacker while the user is away. To defeat this attack the parties will close connections that have been inactive for a certain period of time.

Abusing the key revocation delay. Key revocation is not immediate and that may cause problems. In VirtualLife we have a network built around the Virtual Nations, so it is suitable to use them for propagating key revocation information.

Blind trust towards Certification Authorities. An actor will have to trust the other actors CA to be able to perform transactions. It is known that users are sometimes too trusting so the VirtualLife software will include verbose explanations of the provided security guarantees.

5.2 The vlsec library

The **vlsec** library forms the core of the VL security infrastructure. It provides basic cryptographic operations used by higher level libraries, utilities for managing security credentials and all associated data structures. The **vlsec** library is described in more details in VL deliverable D3.1 and in addendum to D3.2 [BLL08, BLL09].

The implementation of **vlsec** is based on the OpenSSL toolkit – an open source library for cryptographic operations, brief description of OpenSSL is given in Appendix E.

5.2.1 Securable objects

Defining data objects, that could be protected with the security infrastructure is an important design decision that affects architecture. These objects are neither objects in a programming sense nor simple byte arrays, but rather entities that have some contextual meaning and that we want to e.g. encrypt, sign or validate.

Objects include, but are not limited to the following:

1. items that might have copyright;
2. digital signatures;
3. reputation information;
4. citizenship, belonging to some community or group;

5. secure communication channels.

To accommodate for these use cases, cryptographic operations in VirtualLife work on standard strings presented in the C++ string datatype. This way we do not restrict security to specific object types. There is however a potential increase of complexity of operations for the developer using **vlsec** library. If we want to use the security primitives on an object, we first need to serialise it into a string form, which might not always be a trivial task. It must be noted, however, that the same requirement of being able transform object into a byte array is in place also for transporting objects using **vlnet** networking library, so it is a general requirement for all the objects in question.

The next important cryptographic objects are the keys. Both asymmetric keys and symmetric keys are supported by the **vlsec** library. Asymmetric keys are used in public key cryptography where keys are strongly bound to identity. In public key cryptography, every identity has a keypair that contains a public key and a private key. The private key must be kept secret at all times as possession of this key effectively implies being its owner in several cases, for example when giving digital signatures. The public key is used for encryption and signature verification, whereas the private key is used for decryption and giving signatures. In VirtualLife we use the RSA cipher for public key cryptography. Symmetric keys are used for encrypting large or streaming data. Symmetric keys are stored as standard strings, similarly to asymmetric cipher initialisation vectors and random nonces. VirtualLife uses AES as the symmetric cipher of choice.

The **vlsec** library in VirtualLife has the classes *KeyPair* and *PrivateKey* for representing keypairs and private keys, respectively. We do not have a separate public key object as we use an X.509 certificate in its place. Since the certificate contains a public key and also identity information, we use it to simplify the system. All primitives requiring a public key as input take a certificate instead.

VirtualLife is extensively using the X.509 certificate infrastructure in transactions. For this reason there is a Certificate class in **vlsec** that represents an instance of a certificate. The certificate contains information about its owner and the public key. If the certificate is not yet signed, the certificate request object is used. The X.509 standard is very common and much used in applications, so supporting this standard allows the users of VirtualLife to also use credentials issued by other parties to identify themselves.

5.2.2 Key management

The keys and certificates of VirtualLife will have to be securely stored and handled to maintain security guarantees. Also, there needs to be a mechanism for creating new keys. These tasks are all provided by the **vlsec** library in our implementation of the security architecture.

Firstly we discuss key generation. Good randomness generation is important when generating both symmetric and asymmetric keys. For this reason, the *CryptoService* class in the **vlsec** library provides a randomness generator for both numbers and strings. The *KeyManager* class generates both symmetric

and asymmetric keys. The class is also capable of deriving a symmetric session key and an initialisation vector from a random seed. This capability is important in key agreement protocols.

The keypairs also need to be stored permanently. For this reason we created the *KeyStore* class. This class can store certificates, private keys and keypairs. The certificates are stored in the PEM format and private keys in the PKCS#12 format. Private keys are encrypted and protected by a password. The certificates are identified by a hash of the certificate's issuer, serial number and subject name. Private key identifiers are provided by the developer.

There is additional convenience functionality in the **vlsec** library, for instance, getting a list of trusted Certificate Authorities, importing and exporting users' keys etc. This functionality is mostly needed by the identity management module and is described in Section 5.3.

To minimise the risk of unauthorised access to the private key, for example, by extracting it from the hibernate file or the virtual machine memory, the private key object is stored in memory only when it is needed for the operations.

5.2.3 Cryptographic primitives and algorithms

Once the keys and certificates are loaded and ready, we can start using them in cryptographic operations. The **vlsec** library can perform encryption and decryption with both asymmetric and symmetric keys, it can sign and verify strings using asymmetric keys, compute hash functions and message authentication codes. These services are provided in the *CryptoService* class of **vlsec** library.

Input texts are given as strings, keys are given using their respective types. All operations return a boolean value to signal the success status of the operation. The results are written to output parameters given as C++ variable references.

The number of primitives is not large, but it is sufficient for implementing a wide range of secure transactions. These transactions can be run by a single machine (algorithms for complex cryptographic operations) or multiple machines (protocols for running secure operations between many parties). The actual implementation of the algorithms is in the OpenSSL's **crypto** library that **vlsec** depends upon.

Operations typical for X.509 certificates are also located in **vlsec** library, both for the normal usage, for example, verifying certificate signatures, and for implementing custom Certification Authority – signing certificate requests and managing CA signing policies.

5.2.4 Authorisation

Authorisation is required to verify the user's permission and acceptability for using a service provided by the Virtual Zone or a Virtual Nation. It is implemented as a separate module in the **vlsec** library. It allows the developer

to define own policies for various object types and also support linking of policies together, thus enabling creation of complex policies, e.g. inheriting policies from the parent objects. It is also pretty lightweight and can be easily integrated into higher level protocols from **vlprotocol** library.

Authorisation is tightly related to the *Identity* structure, described in more details in section 5.3. At the moment, three types of authorisation policies are supported: whitelist, blacklist and token-based policies.

Whitelist policy

In the case of a whitelist, the service provider has a list of identities or certificates who are allowed to use the service. To use the service, the user must have proved the identity to the provider. If the provider finds the identity in its whitelist, the service will be provided. Otherwise, the user will be denied the use of the service.

To defeat this scheme, the attacker must either fake an identity (fake authentication) or modify the service providers database. The first case is covered in authentication. The service providers database can be modified only by avatars with the necessary authorisation. This kind of authorisation is suitable in scenarios where the number of authorised actors is small.

Blacklist policy

The blacklist has a similar concept to the one of the whitelist, but keeps a database for different purposes. The database of the service provider will now contain identities who are not allowed to use the services. All other VirtualLife users can use the service. If the service provider receives a request to use a service, but finds the identity of the requesting actor in its database, it will deny the use of the service.

We assume, that a client (the attacker) has been banned from using the service by inclusion in the blacklist. To defeat this scheme, the attacker must now fake identity or remove its identity from the database. The security is again guaranteed by the security of authorisation and access to the service providers database.

This kind of authorisation is suitable for use in scenarios where normally all users are allowed to use the service, but some are banned for misconduct.

Token-based policy

This approach is different from the previous two as it does not require the service provider to have a database of accepted identities. Instead, the service provider may issue “tickets” that allow the client to use a particular service. The ticket can be any structure as long as it is uniquely serializable into C++ strings.

These tokens are given to the client who can later claim rights to use the service by presenting them to the service provider. The service provider will then check its database to see if such a token is indeed authorised for the operation. If it is, the service provider allows the client to use the service. Otherwise, access is denied.

This scheme is suitable when the service provider does not wish to keep large databases. It is also good when the description of permissions is complex or the service provider wishes to give the client a feeling of a "real-life permit".

The attacks and countermeasures are the same as in the case of two previous policy types.

Implementation

The implementation of the authorisation is based on two main classes: *AuthorizationPolicy* and *Validator*.

AuthorizationPolicy stands for a set of rules that define whether a certain *Identity* is allowed to perform certain action on the *Object*. Each *AuthorizationPolicy* object has a type to help *Validator* to choose the correct authorization mechanism. Basically, *AuthorizationPolicy* serves as a container that maps object actions to sets of identities/certificate owners/token owners.

Validator's main function validate checks whether given *Identity* can perform a certain action on an object. The whole chain of policies linked to the object is taken into account. Based on the check *Validator* returns an exit code, some of the more relevant exit codes are:

1. `AUTHZ_VALIDATION_SUCCESS` – validation succeeded;
2. `AUTHZ_VALIDATION_MISSING_POLICY` – object doesn't have any policy linked to it. It's up for the application to interpret this exit code;
3. `AUTHZ_VALIDATION_FAILED` – validation failed.

If an object has more than one policy linked to it, then *Validator* handles them in a one-by-one fashion. If at least one policy fails, then the whole validation fails as well.

Authorisation policies allow providing blacklists and whitelists for both identities and certificates. The motivation behind it as follows: sometimes you might want simply to block a user (for example, you don't like him) from doing something. Specifying all possible user certificates would then be annoying, so you can simple add the unique identifier of the user to the list. On the other hand, sometimes it is required to be more fine-grained: for example, we allow only users with certificates issued by highly trusted CAs to sign an important document.

Examples of usage of authorisation framework are given in deliverable [BLL09].

5.3 Identity management

The essential component of every stateful secure system is identity management – control of information about its users, providing information for authentication and authorisation protocols. This is not a technically complicated problem in the case of client-server architecture or in presence of a single trusted party. However, in distributed systems, it is not so easy any more. A number of solutions exist that tackle this problem. It is also one of the popular research topics.

Some of the more widely known solutions include:

- **Kerberos** – an industry standard system offering account management together with authentication and authorisation. It is a good solution for a more controlled system, but using it in dynamic system with distributed trust model is not so easy. It also has a single point-of-failure as it relies on a central trusted server.
- **OpenID** – open and decentralized identity system that allows to use accounts in one system for authentication in other systems that support OpenID. For example, using a Google account for logging in to other sites (Wordpress or Blogger) is technically implemented using OpenID standard.
- **Shibboleth** – federated identity-based authentication and authorization infrastructure that is based on Security Assertion Markup Language (XML-based standard for exchanging authentication and authorization data between security domains). Intended primarily for web applications, though it could be adapted for other applications as well.
- **X.509 security infrastructure** – mentioned many times in this thesis, is a PKI infrastructure suited very nicely for distributed trust model with multiple trusted parties.

5.3.1 Motivation for another solution

In the digital world, people tend to have more than one digital identity. Aggregating them together is a nice functionality to have, as essentially they all belong to the same person.

Another problem is that many of the solutions listed above rely on a standard socket programming interface, hence adapting them to the **vlnet** overlay network would be a very time-consuming task.

Last but not least, non-standard planned functionality, for example, possibility to rate other identities or assign trust levels to different trusted parties, was not trivial in cases of the existing identity management solutions.

5.3.2 Overview of the VirtualLife solution

The identity management system of VirtualLife is built on top of the X.509 security infrastructure. Therefore, in order to qualify for being an actor in the system, an entity must have valid X.509 credentials: signed public key and a private key. This is required as almost all of the operations in VirtualLife assume that parties can present their credentials.

The VL identity system follows the single identity - multiple credentials model. It is possible to import credentials acquired from different sources. Figure 5.2 describes the high level idea of VL identity.

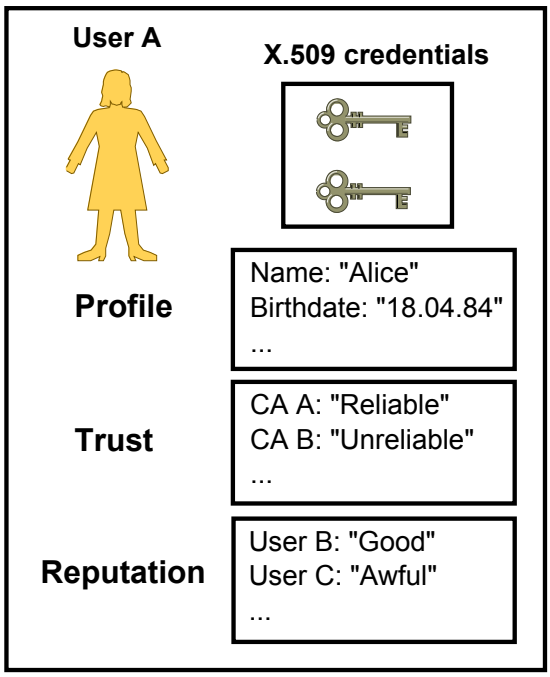

Figure 5.2: Schema of VL digital identity. It includes several parts: a profile of the user, X.509 security credentials, trust and reputation database.

5.3.3 Implementation

The main data structure defined in the *Identity* class is located in the **vlcommon** library and contains general information about the user (name, contact details, etc), while cryptography-related abstractions are in the **vlsec** library: *TrustManager* and *AccountManager* classes. The first one is responsible for managing the user's trust towards other identities/credentials/CAs, as well as reputation values. The latter one, *AccountManager* provides more general functionality like creation of new accounts, import and export of accounts, for example, if account information should be stored on a trusted flash disk.

5.4 The vlnet library

Though the design of the networking layer was not the keypoint of this thesis nor can be attributed to the author, we think that for better understanding a short explanation is needed. Certain **vlnet** properties and interfaces have outreaching consequences also for protocols. A more precise description of **vlnet** along with usage examples is given in the deliverable D3.2 [BLL09].

The **vlnet** library is used to create communication layer for all multiparty operations. It is designed to create and support a hybrid peer-to-peer network topology that will is used by VirtualLife nodes. The library is built on top of RakNet cross-platform high performance networking library and provides some high level abstractions to facilitate protocol implementations. See Appendix E for details about the RakNet library.

The core of the **vlnet** library is the *NetworkPeer*-*NetworkProtocol* tandem of connection and message handling classes. *NetworkPeer* operates on the message and IP level while *NetworkProtocol* operates with persistent connections and data streams. Every node in the VirtualLife network will run one instance of *NetworkPeer* and one instance of *NetworkProtocol*.

The *NetworkPeer* wraps the functionality of the networking library and provides a message and event stream to the *NetworkProtocol*. *NetworkProtocol* in turn uses the *NetworkPeer* to send messages to other peers.

The *NetworkProtocol* class provides more high-level services. It operates with streams of data. One stream has two endpoints and these can be managed by any module of the software. The *NetworkProtocol* performs a post-office-like task of identifying instances of a protocol and managing their message flows. The direct benefit of this approach is that a complex software application like VirtualLife can offload some complexity to a central service that is fit to handle it.

5.4.1 Streams and state machines

A stream in the **vlnet** library is basically a pipe. You put data in at one end and it comes out on the other end. Specific types of streams can also take care of encoding and decoding formats used in them. This is achieved by letting the stream know the serialisation methods for the given objects.

To force the stream into behaving in a predetermined way the developer can add a state machine to the stream. This state machine can then be used to discard messages that should not arrive at the given state of the protocol and force the path of the protocol execution.

The state machine may be really strict by going into a new state when every message arrives or is sent. This is useful for cryptographic protocols with a clear construction. If the stream is used for transferring data, for example geometry and textures, the stream may be in a perpetual “connected” state and accept data messages all the time.

When designing a protocol the developer will have to build a stream for transferring its data. Based on the construction of the protocol the developer can use different aspects of stream design outlined previously.

The main business logic of a protocol is not stored in the stream, but rather in a helper or a service. A helper provides a high-level interface to the protocol for starting it and getting the results. A helper might have a command *login* with the required parameters and a command *getLoginResult* for learning how it went. This again simplifies the construction of a graphical user interface that can basically bind user input to the respective protocol helpers and forward protocol results to the user visually, whether it is a success or an error message.

A service in VirtualLife is a type of protocol endpoint that performs a certain task. An example might be logging the user into the system. If the GUI would use a protocol helper then the server will use a service for answering the clients' queries. A helper usually has only a few streams whereas a service might be working with many stream at once. The helpers and services may have their own state machines, if it is required.

5.4.2 Secure streams and the authentication protocol

A stream may ask the *NetworkProtocol* for a secure communication channel. In that case a channel will be opened, if it was not opened before, and on the streams request messages will be encrypted during transport. This process is transparent for the stream, as the *NetworkProtocol* will take care of channel negotiation and encryption automatically, given that the necessary keys are provided. The stream may also then ask the *NetworkProtocol* for the identity of the node on the other end of the stream for verification.

In the beginning of any communication session the parties prove their identities to each other. The authentication mechanism is based on the PKI used in VirtualLife. Intuitively, parties use their certificates to present their identity, and the private key to prove that they possess the private key that goes to together with this identity. Authentication will have to be redone for every connection, because otherwise it will be easier to mount man-in-the-middle attacks.

To provide such functionality a new authentication protocol – VirtualLife Authentication Protocol (VLAP) – was designed and implemented on top of **vlnet** architecture. It is inspired by TLSv1 protocol and is used for establishing one- or two-way authentication between parties and negotiation and distribution of the session key.

To test out the security properties of the designed protocol, it was modelled using *spi*-calculus[1] and verified using the ProVerif automatic cryptographic protocol verifier [Bla]. More information about VLAP and its verification is given in Appendix D.

[1]Spi-calculus, an extension of π-calculus, is as a formal notation for describing and reasoning about cryptographic protocols.

Attacks and countermeasures

There are some specific attacks on the secure channel listed below.

Faking identity. The attacker might want to create a stream while claiming to be someone else. The responsibility of verifying the identity of the other party lies on the user. The client software will provide as much information as possible about the identification of the channel endpoints.

Eavesdropping. The attacker might want to listen in on the communication. The proper application of encryption schemes will solve this problem.

Man-in-the-middle attack. The attacker might want to insert new messages into an already existing channel, or remove existing messages from there. Again, the proper application of encryption schemes together with message authentication codes will make this attack unfeasible.

Denial of Service. In case a channel is routed through another peer, that peer might decide to drop connection forwarding. If there are other possible routing paths, the parties in the stream might route the messages through other nodes in the VirtualLife network.

5.4.3 Provided security properties

As **vlnet** is used as a platform for creating application-specific protocols, we wanted to make sure that at least the basic security properties are met. As such, **vlnet** assures the following properties for the protocols built on top, when using secure channels:

- confidentiality - guaranteeing the secrecy of sensitive information;
- origin integrity - ensuring that you know whom you are talking to;
- message integrity - unauthorised modifications to messages can be detected.

5.5 The vlprotocol library

Most of the application specific transactions are implemented in the **vlprotocol** library. We use terms “protocol” and “transaction” interchangeably, because using state machine model of **vlnet**, their meaning becomes the same. The implementation of most of the protocols is not directly connected to the security infrastructure, therefore only a few are described here.

5.5.1 Login

In VirtualLife we have a single-sign-on solution based on the X.509 infrastructure. Login essentially consists of two steps:

- establishing a secure connection – this is handled by the **vlnet** library's secure channel and includes bilateral certificate validation;
- user authorisation – handled by authorisation framework and is more application/configuration specific, for example users with low reputation could be banned from logging in into some Zones.

Attacks on this protocol are composed from the attacks on its two subparts: creation of user channel and authorisation framework.

5.5.2 Signing a contract

Signing a contract means attaching a digital signature that is generated using the document and the security credentials (private key) of the identity. The signatures are then attached to the document using standard solutions.

VirtualLife support for digital signature is based on the DigiDoc library [Ser] that follows XML-DSIG and XAdES standards for digital signature. The key class is *SignedDocument* that abstracts the idea of any digital document and provides methods for operations related to digital signing. *SignedDocument* can be created from any string, essentially meaning that we can sign not only document files, but also dynamically generated content, for example a chat history.

SignedDocument supports the following types of operations:

- Creating signed document from any string with specified MIME type. The string has to be saved to a temporary file.
- Creating signed document from any file with specified MIME type. Given the path of the source file, the MIME type of the file, and the path of the destination DDOC[2] file, a signed document structure will be created. The source file will be embedded in BASE64 into the resulting signed document structure. No signatures will be added yet.
- Adding one or more signatures to the document using provided private keys and certificates. Signatures are to be added one by one to the signed document structure, which contains the source file and possibly some already present signatures. The signers certificate with the public RSA key will be added to the structure. The signature will be constructed from the hash of the data to be signed.

[2]DDOC is XML-based container format that contains signed file along with the given signatures.

- Verifying the correctness of the signatures to check if they match the content of the document.
- Saving and loading document to/from the DDOC files.

Created signed documents can be held both in memory and stored in a standard form on the persistent storage (e.g. HDD or flash disk). The document can be verified as long as at least one copy of the *SignedDocument* object or the DDOC file exists.

Attacks and countermeasures

Contract modifications. The attacker would like to modify sums or terms in the contact. This problem is solved through the correct application of cryptographic digital signatures. The signatures will not verify for a modified contract.

Contract copy. Attacker creates n copies of document and claims that the contract should be applied n times. Every contract will need to contain a serial number or a random nonce. If two contracts contain the same number, they are considered the same.

Signing on behalf of somebody else. The attacker might want to forge a signature. This problem is solved through the correct application of cryptographic digital signatures. The attack will only be possible if the attacker has the private key of the attack target.

Deniability. Attacker claims that she didn't sign the contract. The used digital signature scheme does not allow deniable signatures. As long as there exists one copy of the signature and the contract, the existence of a signature can not be mathematically denied. To be more specific, there is no infallible protection against this attack. Instead, in most jurisdictions the digital signature scheme will lead to the assumption that the person who has signed the contract is party to the contract since only this person had the private key required for the digital signature.

Role change. Attacker tries to change his role in a contract (e.g. from "buyer" to "seller"). This is covered by protecting the contract from modifications. It is assumed that the contract contains the mapping between signers and roles.

5.5.3 Group management

Dealing with single object instances is often too complicated, hence VirtuaLife provides possibility to group objects together. The only requirement on the

object type is to have a GUID. There are multiple use cases for groups. For example, managing of citizens of the Virtual Nation or support for "user-driven" organisations, e.g. trade unions or guilds.

It is possible to assign permissions based on the membership in a particular group. For instance, only members of the "UT Cryptographers" group can enter a particular Virtual Zone.

There's no authorisation built in into the group management as it might be very use case-specific. Developer should integrate group management with authorisation in most cases manually, using the provided authorisation mechanisms.

In VirtualLife, a peer that provides group management service exposes interface for the following operations:

- `GROUP_OPERATION_CREATE_GROUP` – create new group, the user can specify group name and description
- `GROUP_OPERATION_DELETE_GROUP` – delete specified group
- `GROUP_OPERATION_ADD_MEMBER` – add member to the group
- `GROUP_OPERATION_REMOVE_MEMBER` – remove member from the group
- `GROUP_OPERATION_IS_MEMBER` – query if the specified member belongs to the group
- `GROUP_OPERATION_GET_MEMBERS` – get all members of the group
- `GROUP_OPERATION_GET_GROUPS` – get all groups stored at the server
- `GROUP_OPERATION_GROUP_INFO` – get group metadata: name and description

Attacks and countermeasures

Claiming membership or non-membership. Membership can easily be verified with an online query to the peer providing group management service.

Chapter 6

Analysis of Solution

The VirtualLife project is still far from completion. Therefore, it is complicated to say whether it will succeed in creating the innovative platform for secure collaboration. However, the major architectural solution have already been established and partly implemented so we can compare it with solutions from another vendors or projects.

Below is a feature comparison of the VirtualLife platform with solutions reviewed in Chapter 3.

Authentication	Security	ID Management	Architecture	Status
Second Life	UP	SP	CS	PR
OpenSimulator	UP	SP	CS	DV
Wonderland	UP	SP	CS	DV
Croquet	UP	PP	PP	DV
LifeSocial	PK	PP	PP	DV
VirtualLife	PK	PP	PP	DV

Legend

UP	username-password authentication	PK	PKI based authentication
SP	profile stored at server	PP	profile is distributed
PR	production system	DV	in development

The VirtualLife platform was designed from the very beginning as a platform that provides secure collaborative environments. The accent on security and connection to the legal system is something not seen in the other solutions. VirtualLife provides a strong identity management solution based on X.509 standard that binds with the external authorities. Extensive usage of X.509 security infrastructure makes future integration of the system supporting the same standard easier, without breaking the security properties within the VirtualLife. It is also easier to bind operations in the VirtualLife with the

legal system – not something that can be seen in the other solutions. Having a strong identity management also allows users of the VirtualLife system to have a much more reliable understanding of whom the are communicating with.

Support for digital signatures makes it possible to sign legally binding contracts possible without the need for additional end-user agreements from the virtual world provider.

VirtualLife implements several ideas from the social networks, like rating of the identities and trust towards trusted third parties. Using these ideas in conjunction with operational and behavioural patterns from the 3D world, it is possible to implement new types of authorisation and authentication algorithms.

Another important aspect is the usage of the service-level routing and protocols implemented as state machines. Service-level routing allows to minimise the number of required ports for operation, making deployment and also NAT traversal easier. State-machines for protocols do make implementation of the protocols more complicated, but this also raises the reliability of the protocols and makes the formal protocol analysis easier.

Using state-of-the-art libraries for visual appearance makes VirtualLife comparable with with the production solutions.

6.1 Future work

Although the core security infrastructure of the VirtualLife is in place, there is a number of things that need to be done in this area. This includes, for example, addition of X.509 enabled VoIP channel, more fine-grained access control for scripts, implementation of event recording and logging system.

7

Resümee

VirtualLife'i turvainfrastruktuur

Ilja Livenson

Magistritöö (20 AP)

Käesolevas magistritöös kirjeldame VirtualLife projekti raames tehtud turvainfrastruktuuri. VirtualLife on Euroopa Liidu 7. raamprogrammi teadusprojekt, mille sihiks on luua turvaline ning kindel võrgupõhine 3D koostöö keskkond.

Tänapäeva virtuaalmaailmad hõlmavad suurt valdkonda, kuhu kuuluvad näiteks võrgu mängud, õpperakendused ning firmade esindused Internetis. " Tavaliselt on need maailmad ehitatud klient-server arhitektuuriga. Sellega kaasnevad teatud probleemid: kõrged kulud serverite ülevalpidamiseks, vajadus usaldada serverit kõikide transaktsioonide puhul. Samuti on seni vähe tähelepanu pööratud virtuaalmaailmade seosele seadusandlusega. VirtualLife projekti idee on proovida neid probleeme lahendada, pakkudes turvalist hajutatud 3D virtuaalmaailma raamistikku.

Töös esitame projekti käigus disainitud ning realiseeritud turvaraamistiku. Meie infrastruktuur põhineb X.509 turvastandardil ning koosneb neljast osast:

- **vlsec** – krüptoteek, mis pakkub nii krüptoprimitiive ning operatsioone nendega, kui ka kõrgema taseme andmestruktuure ning operatsioone, nt. sertifikaatide signeerimine ning valideerimine.
- **vlnet** – võrguteek, mis peale kiire andmeedastuse pakub olekumasinatel põhinevat mudelit protokollide realiseerimiseks ning turvalisi ühendusi, mis võimaldavad tuvastada otspunktide identiteeti.

- **vlprotocol** – teek, mis koosneb rakendusspetsiifilistest protokollidest, realiseeritud on sellised baasprotokollid ja toimingud nagu registreerumine, sisselogimine ning dokumentide allkirjastamine.
- identiteedi haldussüsteem, mis on integreeritud kirjeldatud teekidesse ning pakkub võimalus siduda kasutajaprofiiliga ka tema krüptograafilised võtmed ning hinnang teistele identiteeditele (reputatsioon) ning usaldatud kolmandatele osapooltele, nagu näiteks sertifitseerimiskeskused.

Autor soovib tänada oma juhendajat, kelle pühendumus ja põhjalikkus olid hindamatuks panuseks käesoleva magistritöö valmimisele, ning VirtualLife projekti liikmeid.

Bibliography

[Bla] Bruno Blanchet. Automatic cryptographic protocol verifier, in the formal model, http://www.proverif.ens.fr/.

[BLL08] Dan Bogdanov, Peeter Laud, and Ilja Livenson. D3.1. VirtualLife Security Infrastructure System. Confidential, Cybernetica AS, November 2008.

[BLL09] Dan Bogdanov, Peeter Laud, and Ilja Livenson. D3.2.VirtualLife Client Server Message Exchange System. Confidential, Cybernetica AS, March 2009.

[com] OpenSimulator community. Opensimulator, http://opensimulator.org/.

[com09] KZero company. Kzero Radar, 2009.

[cona] RealXtend consortium. Realxtend platform, http://www.realxtend.org/.

[Conb] The Croquet Consortium. thttp://www.opencroquet.org.

[ea] Julian Lombardi et al. Open Cobalt, http://www.duke.edu/~julian/Cobalt/Home.html.

[Fou] OpenID Foundation. http://openid.net/.

[Gar] Inc Gartner. Report on the virtual worlds' development trends, http://www.gartner.com/it/page.jsp?id=503861.

[GPM+08] Kalman Graffi, Sergey Podrajanski, Patrick Mukherjee, Aleksandra Kovacevic, and Ralf Steinmetz. A distributed platform for multimedia communities. In *IEEE International Symposium on Multimedia (ISM '08)*, Berkley, USA, Dec 2008. IEEE, IEEE Computer Society Press.

[HR] Piers Harding-Rolls. Subscription MMOGs: Life beyond World of Warcraft.

[KSea] Stephen Kent, Stefan Santesson, and et al. Public-Key Infrastructure (X.509).

[KSL] Torus Knot Software Ltd. Ogre rendering engine, `http://www.ogre3d.org/`.

[Lab] Linden Lab. An overview of Second Life security, `https://support.secondlife.com/ics/support/KBAnswer.asp?questionID=5566`.

[Mic] Sun Microsystem. Project wonderland, `https://lg3d-wonderland.dev.java.net/`.

[pro] MediaGrid project. The Immersive Education Initiative, `http://immersiveeducation.org/`.

[Ser] Sertifitseerimiskeskus AS, `http://www.sk.ee/files/soft/DigiDocLib-2.2.5-eng.pdf`. *DigiDoc library.*

[SSea09] John Smart, John Smart, and Jerry Paffendorf et al. Metaverse Roadmap Overview. Technical report, Virtual Worlds Roadmap group, http://www.metaverseroadmap.org/overview/, 2009.

Appendix A

VirtualLife Project Information

Project name	"Secure, Trusted and Legally Ruled Collaboration Environment in Virtual Life"
Project page	`http://www.ict-virtuallife.eu`
Participants	Nergal S.r.l. (Italy), SME Cybernetica AS (Estonia), SME Digital Video S.p.A. (Italy), SME Geumacs (Romania), SME MIF VU (Lithuania), University Panebarco S.a.s. (Italy), SME Tavae (France), SME Universität Göttingen (Germany), University Virtual Italian Parks (Italy), SME
Start of the project	January 2008
Duration	36 Month
Budget	3.3 million euro

Appendix B

VL Project Deliverables

A better picture about the project, including some more specific details of design and implementation, can be acquired from the project deliverables. Though some of them, especially early ones, are not describing the system exactly due to the nature of the software development projects, they might be helpful for the reader.

Due to the publication policy of the project, most of the deliverables are confidential. To get access to them, please, contact author or supervisor of this thesis directly.

Appendix C

VL Source Code

There was no stable release of the VL code up to now. The release is planned later this year, approximately in September 2009. Although the security and networking part of the application are not in their final form as well, their are relatively stable and are mostly filled with additional functionality – no major changes in design are undertaken. The license of the code is still not finally decided, hence we cannot attach full source tree; only the source code of the described libraries is attached.

- `code/lib/vlsec/`
- `code/lib/vlnet/`
- `code/lib/vlprotocol/`

Getting access to the full source code is possible as well, for that, please, contact author or supervisor of this thesis directly.

Appendix D

VL Authentication Protocol

VirtualLife Authentication Protocol (VLAP) is described in more details in seminar paper. Text of the seminar paper along with the VLAP model in spi-format and presentation can be found on the attached CD. Description of relevant files:

`vlap/vlap_sem_2008.pdf`	article describing VLAP
`vlap/VL_ssl.cvpi`	model of the VLAP in the ProVerif-compatible format
`vlap/vlap_presentation.pptx`	presentation about VLAP
`vlap/proverifbsd1.15.tar.gz`	archive containing ProVerif protocol verifier that can be used to reproduce results

Appendix E

Key libraries used

E.1 OpenSSL toolkit

OpenSSL is an open-source cryptography library, one of the most popular nowadays. It consists of two main parts:

1. **crypto** - contains implementation of many cryptographic algorithms, this library is among other things used for implementation of SSH and OpenPGP. The functionality includes symmetric encryption, public key cryptography and key agreement, certificate handling, cryptographic hash functions and a cryptographic pseudo-random number generator.
2. **ssl** - implementation of SSL v2 and v3 and TLS v1 protocols.

E.2 RakNet

RakNet library is a state-of-the-art networking library used in many modern applications, multiplayer games among others. RakNet is based on the concept of a reliable UDP and handles many issues typical for network application automatically:

- orders or sequences packets that arrived out of order, and does so efficiently,
- resends packets that didn't arrive,
- transparently handles network issues such as flow control and aggregation.

It also contains a number of plugins very useful when creating an application. For example, implementation of low-bandwidth VoIP communications and NAT traversal.

A somewhat different from TCP/IP stack model of RakNet makes usage of libraries that rely on the socket interface more difficult.

E.3 DigiDoc

DigiDoc is a C library that allows to create, sign and verify DigiDoc files. These files are compliant with the XML-DSIG and XadES standards for digital signatures. The DigiDoc library also supports the usage of smartcards .