Stefan J. More

Java Privacy Guard

The OpenPGP Message Format
and an Implementation in Java

Bachelor's Thesis

Graz University of Technology

Institute for Applied Information Processing and Communication
Head: O.Univ.-Prof. Dipl.-Ing. Dr.techn. Reinhard Posch

Advisers:
Dipl.-Ing. Dieter Bratko,
Ass.Prof. Dipl.-Ing. Dr.techn. Peter Lipp

Graz, August 2015

Abstract

OpenPGP is an Internet standard for securely sending messages over insecure networks like the Internet. It provides end-to-end encryption by combining asymmetric and symmetric cryptography. Trust in any network component except the sender's and receiver's computer is not needed. Furthermore, it guarantees for the integrity of messages using digital signatures. OpenPGP also provides a system for verification of the identity of participants of an communication using a trust model called the Web of Trust.

In this thesis we give an overview of the principles of OpenPGP and its underlying Internet standard, the *OpenPGP Message Format*. Additionally, we explain in which ways OpenPGP implementations are used, its functionalities and inner workings.

Furthermore, we show an example on the basis of our implementation of the OpenPGP standard in Java using the IAIK-JCE cryptographic library. We also discuss considerations regarding security and usability.

Keywords: OpenPGP, PGP, Java, End-to-End Encryption

Contents

1 Introduction

Insecure networks, like the Internet, do not prevent eavesdropping, modifying the contents of messages, or impersonating other entities on the network. As such, secure message exchange over insecure networks requires protecting the transmitted messages. The favored properties of such a protection are confidentiality, integrity and authenticity.

The *OpenPGP Message Protocol* is an Internet standard providing exactly those properties. This is done by combining several algorithms to encrypt messages ahead of transmitting them. In addition, digital signatures are used to provide integrity. OpenPGP provides an end-to-end protection of messages. It is therefore possible to use untrusted communication components for transmission. In chapter 3 of this thesis we give an overview of the OpenPGP Message Protocol and its ecosystem.

OpenPGP achieves the properties mentioned by defining a structure for data and various procedures. The details of OpenPGP's data structures and how they are used by the algorithms are discussed in chapter 4.

We used the IAIK-JCE toolkit for the Java programming language in our own implementation of OpenPGP. Doing so we implemented the data structures defined in the OpenPGP standard. Furthermore, we used the IAIK-JCE to provide the required procedures specified by the OpenPGP standard. We implemented encryption, signing and key management. Our implementation is based on the Java stream architecture. We present this implementation, the *Java Privacy Guard*, in chapter 5.

One concern of the OpenPGP standard affects the used algorithms. Since various algorithms are used in OpenPGP its security depends on the security of those algorithms. In addition, it is required to carefully pick values for the parameters of those algorithms. Another concern applies to the usability of OpenPGP implementations. Since OpenPGP is used in a non-transparent

way, it requires a lot of user interaction. Chapter 6 covers general security and usability concerns.

2 Preliminaries

In this chapter we explain the basic concepts required to understand this thesis. We explain the difference between symmetric and asymmetric cryptography. In addition, we describe the meaning of key ID, fingerprint and keyring. Furthermore, we show the principles of cryptography in Java and the library we used for our implementation.

2.1 Symmetric Cryptography

An encryption function (cipher) is a transformation which transfers a given *plaintext* so that its meaning cannot be understood anymore [HAC, section 1.4]. In the context of *symmetric cryptography* this transfer is parametrized by a so called *encryption key* [HAC, section 1.5]. Equations 2.1 and 2.2 illustrate this process.

$$\text{cipherText} = \text{encryptCipher}(\text{plainText}, \text{key}) \tag{2.1}$$

$$\text{plainText} = \text{decryptCipher}(\text{cipherText}, \text{key}) \tag{2.2}$$

2.2 Public-Key Cryptography

In contrast to symmetric cryptography, *public-key cryptography* references a class of cryptographic algorithms which require two different keys [HAC, section 1.8]. The so called public key is used to encrypt a message. On the other hand, a private key is needed to decrypt the message. Therefore, public-key cryptography is also called *asymmetric cryptography*. The process principle is illustrated in equations 2.3 and 2.4.

$$\text{cipherText} = \text{encryptCipher}(\text{plainText}, \text{publicKey}) \tag{2.3}$$

$$\text{plainText} = \text{decryptCipher}(\text{cipherText}, \text{privateKey}) \tag{2.4}$$

2.3 Key ID & Fingerprint

A fingerprint is a character or byte sequence which is used to identify a key. A *key ID* is the short form of a fingerprint. For example the fingerprint can be the 160-bit SHA-1 hash of a key, while the key id are the lower 64 bits of the fingerprint. Due to the nature of a hash function it is possible that collisions exist. This is further explained in [RFC4880, section 12.2].

2.4 Keyring

A *keyring* is a persistent collection of keys. In general a keyring is a list of public- and private-keys. It can be saved to a disc or stored otherwise. A keyring stores all the keys used by a cryptographic software. This is further explained in [RFC4880, section 3.6].

2.5 Java Crypto Architecture

The *Java Crypto Architecture* (JCA) provides a basic framework, including an API, for working with cryptographic primitives in Java [JCA].

2.6 IAIK-JCE

IAIK-JCE is the *Java Cryptography Extension* (JCE) implementation of the *Institute of Applied Information Processing and Communications* (IAIK). It is "a set of APIs and implementations of cryptographic functionality, including hash functions, message authentication codes, symmetric, asymmetric, stream, and block encryption, key, and certificate management" [IAIKJCE].

3 OpenPGP

In this chapter we give an introduction to OpenPGP. We given an overview of its history and the provided functionality. Furthermore, we explain how the provided functionality is realized by OpenPGP. In addition, we list some common use-cases of OpenPGP implementations and compare OpenPGP with the related S/MIME protocol.

3.1 History

The Pretty Good Privacy (PGP) software was developed by Philip Zimmermann in 1991 [Zimmermann] as a tool for human rights activists [PGP10]. PGP was intended to be used to securely communicate via bulletin board systems, Usenet and email, but it can be used to encrypt and sign all kinds of data.

To allow interoperability with other programs the format used by PGP was standardized in 1997, forming the *OpenPGP Message Format* [RFC4880].

The OpenPGP standard is actively developed by the *OpenPGP Working Group (WG)* of the *Internet Engineering Task Force* (IETF). The WG was reopened in 2015 as a reaction to an emerging discussion about secure (and usable) digital messaging.

3.2 Functionality

In this section we give an overview of the functionality provided by OpenPGP. We explain how OpenPGP provides confidentiality, integrity and authentication. Furthermore, we introduce OpenPGP's trust model, the *Web of Trust*.

3 OpenPGP

Software based on the *OpenPGP Message Format* standard can be used to encrypt and decrypt data as described in [RFC4880, section 2]. Furthermore, it is possible to sign data and verify those signatures. In addition, OpenPGP allows sender (and therefore message) authentication by defining its own trust model, called the Web of Trust.

To do so OpenPGP combines asymmetric cryptosystems with symmetric cryptosystems. Asymmetric cryptosystems are used to provide message integrity by digitally signing data. Additionally, they are used to exchange the keys used by symmetric cryptosystems. Symmetric cryptosystems are used to provide message confidentiality by encrypting data. OpenPGP allows the usage of various symmetric and asymmetric algorithms.

To provide the mentioned functionality every entity participating in a communication secured by OpenPGP needs (at least) one OpenPGP keypair. This keypair consists of a public and a private key to be used by the asymmetric cryptosystem. An OpenPGP key is the composition of several cryptographic keys and meta-data. This composition is explained in detail in section 4.3.

OpenPGP uses different keys for different functionality. For example it uses a different keypair for digital signatures and for encryption. The signing keypair is used to provide integrity-protection and authentication of the key's meta-data. Furthermore it is the trust anchor in the Web of Trust. Therefore it is desired not to replace this keypair. This is enabled by allowing replacement of the encryption keypair while keeping the signing keypair (and thus the trust on it). The structure of an OpenPGP key is explained in more detail in section 4.3.

Message Confidentiality is ensured by using encryption. This is done by first encrypting data using a symmetric cipher using a session key generated randomly for every encryption, as described in [RFC4880, section 2.1]. The (symmetric) session key is then encrypted using the (asymmetric) public key of the receiver. The resulting encrypted session key is then prepended to the encrypted data and sent to the receiver. The sender uses their (asymmetric) private key to decrypt the encrypted (symmetric) session key. The decrypted session key is then used to decrypt the actual data. This process is visualized in section 4.4.

If data is sent to more than one receiver the (symmetric) session key is

encrypted multiple times using the public key of each receiver. The receiver then picks the encrypted session key encrypted using their key and decrypts it using their private key. Therefore it is not required to encrypt the data for each receiver separately.

In addition to the asymmetric key exchange it is possible to derive a symmetric key from a passphrase. In that case no asymmetric cryptosystem and therefore no keypair is needed. This is useful if the data is stored and not transmited.

Message Integrity is provided by forming a digital signature of the (plaintext) data, as described in [RFC4880, section 2.2]. This is done by first calculating the hash of the data. Then the hash is signed using the private key of the sender. Afterward the calculated signature together with the encrypted data is sent to the receiver. The receiver then uses their private key to decrypt the data. After that they are able to verify the integrity using the signature and the public key of the sender.

This system also allows non-repudiation, since signatures can only be issued by the holder of the private key.

To encrypt a message the OpenPGP public key of the designated receiver is needed. To verify the signature of a received message, the OpenPGP public key of the sender is needed, respectively. OpenPGP users therefore have to distribute their OpenPGP public keys. The easiest way to perform this key-exchange is by sending the OpenPGP public key along with every message or by publishing it on a website. Another way to distribute OpenPGP public keys is by publishing them to a designated key directory, a so called key-server.

However, one cannot assume that the channel over which the key is transmitted is secure: indeed, the goal of OpenPGP is to establish such a secure channel. Therefore it is also necessary to validate the integrity and authenticity of a public key before being able to use the key to validate the signature on a received message. It is necessary to validate this properties before encrypting using a public-key, respectively. In OpenPGP there are multiple ways to ensure **Authentication**.

In the simplest case it is sufficient to hash the sensitive key-parts, forming the fingerprint of the key. Afterwards it is required to compare the fingerprint with the help of an independent channel (for example by phone or by

printing it on a business card). In OpenPGP this hash over the public key is called a fingerprint [RFC4880, section 12].

In some cases it is not possible to compare this fingerprint, for example when the communication parties do not know each other in advance. This is why OpenPGP provides a more sophisticated trust model, called the **Web of Trust (WoT)** [PGPMAN]. The WoT is a trust-model without any central authorities. Binding between the identity of an entity and a keypair is done via cryptographic signatures. A user certifies (confirms) the authenticity of a keypair (its public key) by signing the identifying meta-data of the key with their own key. The resulting relation between the two keys is called certification. Since there exists no central certificate authority all OpenPGP users can issue certifications for all other OpenPGP keys. The result of this process is not a linear trust chain, as in *X.509*, but a directed graph. This principle is shown in figure 3.1.

Sender authentication is created by searching for at least one (directed) patch from the sender's keypair to the receiver's keypair. A directed edge in the trust graph is represented by a digital signature. To establish a cryptographic trust path it is therefore needed to sign other OpenPGP keys. Since the owners of those keys signed other keys too it is possible to reach those keys by traversing on the edges. Thus it is possible to establish a trust path to before-unknown keys.

A detailed explanation of the Web of Trust approach and trust models in general was out of scope of this thesis. An analysis of the OpenPGP Web of Trust can be found in [Ulrich2011].

Because of the longevity of a keypair and the sensitivity of the private key, it can be necessary to be able to replace a keypair or parts of it. Thus, it is necessary to be able to invalidate the old keypair. For this reasons OpenPGP provides the ability to set an **expiration date** for all public parts of an OpenPGP keypair. In addition, it is possible to explicitely **revoke** the keypair or parts of it [RFC4880, section 5.2]. This is possible by adding a self-signature of a certain type to the OpenPGP key. This signals the invalidity to the OpenPGP implementation and requests not to use the specific key part.

In addition, OpenPGP provides funcitonality to protect private key mate-

rial by encrypting it symmetrically using a passphrase [RFC4880, section 3.7]. Therefore this passphrase is required in addition to the private key file to decrypt or sign data.

3.3 Applications

Although OpenPGP was designed to secure communication on bulletin boards, it can be used in several different ways. In this section we give an overview of some of the possible applications. We explain how OpenPGP can be used to secure email communication, and what its limits are.

In general, terms software implementing the OpenPGP standard can be used to encrypt and sign data [RFC4880, section 2]. It does not limit the means of which the data has to be stored or transmitted. This allows to process generic files, for example to be securely stored on disk, transferred over the Internet or printed on paper. In addition, it is possible to use the *Web of Trust* to authenticate entities.

The main application of OpenPGP is to **secure email communication**. This can be achieved in two different ways. The more intuitive approach is to use a software implementing the OpenPGP standard to encrypt and sign the body and attachments of the message. After doing so the ciphertexts are put in place of the original data. The receiver has to reverse that process by using a OpenPGP implementation to decrypt the message parts. This is possible without any changes to the email standard. The second approach uses an extension of the email standard called PGP/MIME [RFC3156]. It utilizes special headers to structure the email data secured by OpenPGP.

Furthermore, OpenPGP allows to only sign email messages without encrypting it [RFC4880, section 7]. This is also possible if the receiver is not using OpenPGP, resulting in the signature being displayed next to the message. This is an usability issue. To counteract this issue PGP/MIME hides the signature in its own email header.

The main limitation of both approaches is the fact that meta data is not signed [Gre14]. In the email system it is not possible to protect the receiver of a message, since the email headers are not encrypted. In addi-

tion, the email standard does not allow encryption, or signing, of the subject.

OpenPGP is also used to digitally **sign software**, a very common practice in the free/libre/open-source software (FLOSS) community. This is done to ensure integrity and authenticate the source of downloaded software.
Because the community providing the software already built a Web of Trust it is easy to use OpenPGP. The used approach provides better security compared to unauthorized hashes. Furthermore, it allows automated processing by package managers.

In addition, it is possible to use OpenPGP keys for **Transport Layer Security** (TLS) authentication [RFC6091]. This allows using the decentralized *Web of Trust* model instead of the hierarchical *Public Key Infrastructure* [RFC5280].

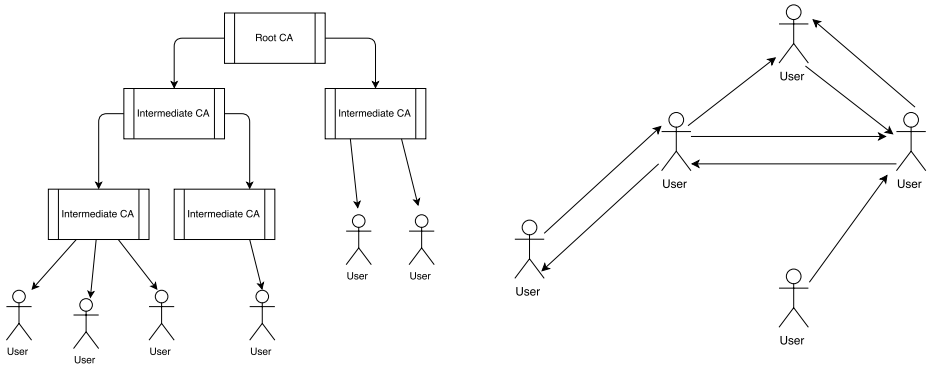
3.4 Related Protocol

In this section we give an overview of another protocol which provides the properties provided by OpenPGP: the Secure/Multipurpose Internet Mail Extensions (S/MIME). Additionally, the main differences between S/MIME and OpenPGP are highlighted.

The Secure/Multipurpose Internet Mail Extensions (S/MIME) [RFC5751] is a standard for secure communication. It uses techniques similar to those used by OpenPGP to provide confidentiality, message integrity and sender authentication. S/MIME was designed to secure email communication.

The key difference between OpenPGP and S/MIME is the structure of the trust model.

The OpenPGP protocol provides sender and key authentication via a decentralized trust mode. It uses a *Web of Trust* to bind the identity of an entity to a key. This allows a trust model without any central authority. Every user of OpenPGP is able to issue a trust certification for every other OpenPGP key. This model is explained in section 3.2. The downside of this approach is a lack of usability. To securely authenticate a key it is required to find a cryptographic trust path to the receiver's key. Another issue is that



(a)Public Key Infrastructure (PKI) (b)Web of Trust (WOT)

Figure 3.1: Comparison of PKI and WoT

it is not defined what the meaning of an OpenPGP certification is. Every user decides for themselves about the requirements for a certification. The topic of usability is further discussed in chapter 6.3.

In contrast to this, the S/MIME standard [RFC5751] provides sender authentication by setting up a tree-like trust hierarchy. S/MIME uses the *X.509 public key infrastructure (*PKI*)* [RFC5280] as trust model. Trust is issued by central Certificate Authorities (CAs). A CA uses their private key to digitally sign the receiver's key. The resulting signature plus the certified identity information is called a certificate. To authenticate a key, it is required to find a cryptographic trust path from a trusted CA to the receiver's key. There are multiple ways to ensure if a CA is trusted. One was is to install the CA's certificate (containing its public key) on the receiver's system. This can be done by operating-system or browser vendors, IT departments, or manually by the user. Such a CA is called *root CA*. Another way to ensure trust in the CA is to sign the CA's certificate itself. This creates a trust hierarchy.

Figure 3.1 shows the difference in terms of structure of the two mentioned trust models.

4 The OpenPGP Message Format

In this chapter we illustrate some important data structures used by OpenPGP. Furthermore, we explain the principles behind the inner workings.

In chapter 4.1 we introduce the low-level data structures called *packets*. In chapter 4.2 we show how packets are composed to form the high-level data structures called *transferables*.
In chapters 4.3 and 4.4 we show how OpenPGP messages and keys are organized internally.

We show how OpenPGP carries out the principles described in chapter 3. In chapter 4.4 we discuss how encryption and decryption works. In chapter 4.5 we explain how key certification works. In chapter 4.6 we show how OpenPGP data can be represented.

4.1 Packets

In this section we give an overview of the basic building blocks of OpenPGP called *packets* [RFC4880, section 5]. We describe how OpenPGP organizes data on its lowest level. We list and describe some important packets. Furthermore, we show the structure of a packet by example.

Packets are a structured and defined sequence of bytes. Each packet consists of a header and a body. The header describes the type of the packet and its length. The body contains the actual data.
The Packet header is divided into a packet tag and the Packet's length. To allow large packets the packet tag also contains information about how the packet length is stored.
The structure of the packet body is depended on the type of the packet.

OpenPGP defines two different versions of packets (version 3 and 4). The main difference between the version is the structure of the packet header. The new version allows 64 different packet types. The old version reserves only 4 bit for the packet type, and therefore allows a maximum of 16 different packet types.

OpenPGP currently defines 17 different packet types. The following paragraph gives an overview of the most important packet types. A detailed description can be found in [RFC4880, section 5].

Key related packets contain the cryptographic primitives used by OpenPGP. They hold and structure the actual key-bytes used by the different cryptographic algorithms. In addition, they hold metadata such as the identifying User-ID and information about the used algorithms.

- Key Data:
  - *Public-Key* packet (Tag 6)
  - *Public-Subkey* packet (Tag 14)
  - *Secret-Key* packet (Tag 5)
  - *Secret-Subkey* packet (Tag 7)
- Identification and Metadata:
  - *User ID* packet (Tag 13)
  - *User Attribute* packet (Tag 17)
  - *Trust* packet (Tag 12)

Message related packets hold the protected data. In case of integrity protection and authentication they contain the signature of the data. In case of confidentiality they contain the encrypted data. Furthermore they hold the encrypted session key used to encrypt the data, used for the key exchange.

- Key Exchange:
  - *Public-Key Encrypted Session Key* packet (Tag 1)
  - *Symmetric-Key Encrypted Session Key* packet (Tag 3)
- Message:
  - *Symmetrically Encrypted Data* packet (Tag 9)

  - *Symmetrically Encrypted Integrity Protected Data* packet (Tag 18)
  - *Modification Detection Code* packet (Tag 19)
  - *Literal Data* packet (Tag 11)
- Signature:
  - *Signature* packet (Tag 2)
  - *One-Pass Signature* packet (Tag 4)

To encode large numbers (e.g. key material) OpenPGP defines a structure called *Multi Precision Integer* (MPI). A MPI consists of 2 octets holding the length of the Integer (in bits) followed by the actual integer.

To illustrate the described structures, figure 4.1 shows the structure of a version 4 RSA key packet as an example. In addition to the packet header the packet holds metadata and the actual public-key material. The structure of the key material is depended on the given Algorithm ID.

Tag (1 octet)	**Packet Length** (up to 5 octets, depending on tag)	
Version (1 octet)	**Creation time** (4 octets)	**Asymmetric Algorithm ID** (1 octet)
MPI length (2 octets)	**RSA public modulus n** (length depending on MPI header)	
MPI length (2 octets)	**RSA public encryption exponent e** (length depending on MPI header)	

Figure 4.1: The structure of a key packet

Packets are composed to form the OpenPGP data structures, called transferables. This is described in section 4.2.

4.2 Transferables

In this section we give an overview of the higher-level data structures used by OpenPGP, called transferables [RFC4880, section 11]. In addition, we introduce the four transferables used by OpenPGP: public key, private key, message and detached signature.

Packets are representations of very basic data structures. To create OpenPGP messages and keys some of this packets are composed in a meaningful way.

A (defined) composition of packets is called transferable, since its purpose is to be transfered from one OpenPGP implementation to another, or be stored and used later.

OpenPGP defines four different transferable types [RFC4880, section 11]:

First of all it is necessary to export public and secret keys in form of *Transferable Public Keys* and *Transferable Private Keys*. This enables storing the keys in keyrings while the OpenPGP implementation is halted. Additionally, it is necessary to transmit the public key to other parties of the communication, as described in section 3.2. The structure of Transferable keys is explained in section 4.3.

The core functionality of OpenPGP is to securely transmit data. The third type of transferables therefore is a composition of packets representing such a OpenPGP Message. The structure of transferable messages is explained in section 4.4.

Furthermore, it is possible to transmit signatures separated from the actual signed data, so called Detached Signatures.

4.3 Keys

In this section we give an overview of OpenPGP keys. We describe what an OpenPGP keypair is and explain how multiple keys and identifying metadata are stored in one OpenPGP key.

An *OpenPGP key* is a composition of multiple keys, identity information and other metadata, as described in [RFC4880, section 11.1]. OpenPGP organizes key material of two kinds: *primary-key* and *subkeys*. The primary key is a signing key used to issue signatures of data. Furthermore, the primary key is used to sign (certify) all other (critical) key parts. In addition the primary key can be used to issue certifications of other keys, as explained in chapter 4.5. A primary key may also be used for encryption. If the primary

key is only used for signing, but data encryption is still needed, subkeys are required. Subkeys are additional keys inside the OpenPGP key.

In addition to primary key and subkey, identity information is stored in the OpenPGP key in form of simple strings. The identifying information is called *User ID*. Furthermore, it is possible to include images, so called *User Attributes*.

Figure 4.2 shows the structure of a *Transferable Public Key* to illustrate this structure.

Because of the asymmetric nature of OpenPGP at last one keypair is needed for every OpenPGP secured communication (the only exception being password-based encrypting, as briefly described in 3.2). An OpenPGP keypair consists of a *Transferable Public Key* and its corresponding *Transferable Secret Key*. Both transferable keys contain one or more keys.

Since the public key is shared over possibly insecure channels, it needs to be protected. Revocation is done by using the primary key to issue signatures over other key parts and appending the signatures to the key. The parts to protect are subkeys, User IDs and User Attributes. Certification is explained in section 4.5.

In addition it is possible to revoke a key or certain parts of it. This is done by signing the part and appending the signature of a certain type. It is possible to revoke a subkey, the User ID, a User Attribute or the whole OpenPGP key. Revocation is explained in section 4.5.

4.4 Messages

In this section we give an overview of the structure of OpenPGP messages. We explain how various packets are composed together to form an OpenPGP message. Furthermore, we show how an OpenPGP transferable is constructed during encryption and decryption of an OpenPGP message.

An OpenPGP Message is a composition of different packets, as described in [RFC4880, section 11.3]. The actual data is contained in a *Literal Data Packet*. It is possible to compress and sign the data. In addition, OpenPGP provides the functionality to encrypt the data and to ensure the integrity of

One **Public-Key** packet	
Zero or more **revocation** signature packets (for the public key)	
One or more **User ID** packets	
	After each User ID packet: zero or more **Signature** packets (certifications)
Zero or more **User Attribute** packets	
	After each User Attribute packet: zero or more **Signature** packets (certifications)
Zero or more public **Subkey** packets	
	After each Subkey packet: one **Signature** packet (certifications)
	After each Subkey packet: optionally a **revocation** signature packet (for the subkey)

Figure 4.2: The structure of a Transferable Public Key

the (encrypted) data.

OpenPGP does not specify a precise structure of an *OpenPGP Message*, allowing various combinations of the involved packets. For example it is not defined whether to sign the literal data and then compress, or first compress the data. In addition, it is not defined whether sign-then-encrypt or encrypt-then-sign has to be performed. Thus, the example shown in figure 4.3 shows only one possibility.

*Public-Key **Encrypted Session Key** Packet 1*		
*Public-Key **Encrypted Session Key** Packet 2*		
*Symmetrically **Encrypted Data** Packet*		
	One-Pass Signature Packet	
	***Compressed Data** Packet*	
		***Literal Data** Packet*
	***Signature** Packet*	

Figure 4.3: The structure of a transferable OpenPGP Message

The process of encryption and decryption is shown in figure 4.4. In figure 4.3 the data is encapsulated in a *Literal Data Packet*. This *Literal Data Packet* is compressed, resulting in a *Compressed Data Packet*. The *Compressed Data*

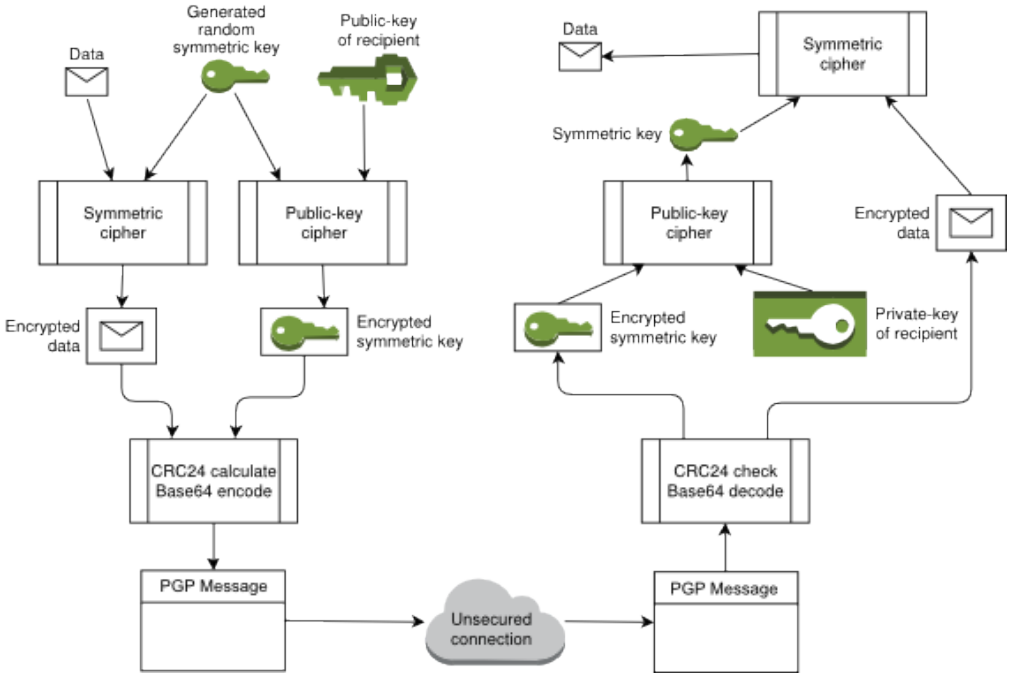


Figure 4.4: Visualization of process of encrypting and decrypting data

Packet is signed using the *One-Pass* mechanism. The three resulting packets are encrypted using a symmetric cipher and a randomly generated session-key, resulting in a *Symmetrically Encrypted Data Packet*. In this example the session-key is encrypted twice using two different public-keys, resulting in two *Public-Key Encrypted Session Key Packets*.

To decrypt the message, the receiver first finds the *Public-Key Encrypted Session Key Packet* encrypted using their public key, and uses their private key to decrypt the session-key. Afterward, the session-key is used to decrypt the three packets. The *One-Pass Signature Packets* holds the information needed to calculate the hash while decompressing the *Compressed Data Packet*. The resulting hash, and the sender's public key, are then used to verify the signature contained in the *Signature Packet*.

Afterward, the *Literal Data Packet* can be used my the application, for example to display it.

This process can be seen on the right side of figure 4.4.

4.5 Key Certification

In this section we given an overview of how OpenPGP solves the problem of key exchange using insecure channels. We explain how the integrity of OpenPGP keys is protected. In addition, we show how the binding between identity and key is ensured. Furthermore, we describe how OpenPGP handles the revocation of keys.

OpenPGP keys are typically exchanged over an insecure channel like the Internet. During transmission all parts of the key are subject to change by an attacker. Furthermore, it is possible that an attacker creates a key on their own claiming a specific identity. Therefore it is necessary to authenticate a key before using it to encrypt data or verify signatures. This means it is required to verify the relationship between the identity information contained in an OpenPGP key and the actual identity. In addition, it is required to confirm the integrity of a key. This means ensuring that no part of the key has been changed during transmission.

To enable this, OpenPGP uses digital signatures in two ways [RFC4880, section 5.2.4]. Self signatures are used to bind the different parts of a key together. Furthermore, signatures by other keys are used to confirm the identity stored in the key, so called certifications. All signatures are formed by computing a hash-digest of the data to be signed, and then signing this hash.

Binding signatures: As shown in figure 4.2 an OpenPGP key consists of different parts. The most important parts are the primary-key, User ID and Subkeys. To cryptographically bind those parts together signatures are used. A binding signature is formed over the packet to bind and the key to bind the packet to. Additionally, some control bytes and a trailer containing metadata are appended.

The following list illustrates the structure of the hashed data for a User ID binding.

- Key to bind data to:
  - $0x99$ (1 octet)
  - Length x of key (2 octets)

  - Key packet (x octets)
- Data to bind:
  - $0xB4$ (1 octet)
  - Length y of User ID (4 octets)
  - User ID packet (y octets)
- Trailer:
  - Signature Version (1 octet)
  - Signature Type (1 octet)
  - Key Algorithm ID (1 octet)
  - Hash-Digest Algorithm ID (1 octet)
  - Protected subpackets (z octets)
  - Signature Version (1 octet)
  - $0xFF$ (1 octet)
  - Length of trailer: $6 + z$ (4 octets)

To ensure the integrity of an OpenPGP key, self signatures are used to bind all key parts to the primary key. This is done by signing each key part together with the primary key using the primary key. Afterward, the resulting signature is appended to the key part as shown in figure 4.2.

Certification signatures are binding signatures issued by other keys. They confirm the authenticity of the signed key part. Specifically, a signature over a User ID and a key means that the issuer of the signature endorses the relation between the key, the User ID and the real identity of the entity.

The meaning of a certification is not defined by OpenPGP. More specifically, it is not defined by what means the verification of the identity has to be performed. Thus, it is possible that different OpenPGP users use certifications in a different way. This is further discussed in section 6.3.

To verify the integrity of an OpenPGP key, an implementation has to verify the respective self-signed binding signatures.

To validate the identity information contained in a OpenPGP key, an implementation has to verify a signature issued by an other key. For this it uses the trust model called *Web of Trust*. To do so, it is required to find a trust path from the own key to one of the binding signatures on the key. In addition, it is necessary to verify all signatures contained in the path. The

Web of Trust is further discussed in chapter 3.2.

Furthermore, OpenPGP provides the functionality to **revoke signatures**. This can be used to revoke parts of an OpenPGP key, for example to replace it. This is done by appending a signature of type "revocation" to the part to be revoked, as shown in figure 4.2. An implementation checks if such a signature exists on the part it wants to process. The existence of such a signature invalidates the revoked part of the key. Thus, the implementation is requested to not using it. A reason for the revocation can be included into the signature's metadata.

In general, only the primary key is allowed to issue revocation signatures for its own parts. A revocation signature issued by another key does not provide any information. The only exceptions are keys who are allowed to issue revocation on the key. This is done by adding the key ID of a key to the key's metadata. An implementation checks whether the revocation signature has been issued by either the primary key or a key mentioned in the metadata. If this is not the case, the revocation signature shall be ignored.

4.6 Representation

In this section we show how OpenPGP transferables can be represented. We show how a key and a message look like while transferred.

Since OpenPGP defines the structure of packets as a sequence of bytes, the primary representation of a transferable is also in byte format.

For many use-cases it is sufficient to store and transmit a transferable in binary form. For example, for storing the keys on a user's keyring or for transmitting over a channel which is able to transfer raw bytes.

Since there are channels with are not able to transfer raw bytes, there is a second way to represent OpenPGP transferables. It is possible to convert a binary transferable to ASCII-format. Such a representation of a binary OpenPGP transferable is called **ASCII-armored** [RFC4880, section 6].

An ASCII-armored OpenPGP transferable converts the raw bytes of a transferable to ASCII using the Base64 algorithm [RFC4648]. In addition, a

CRC-24 [RFC4880, section 6.1] checksum is appended to detect transmission errors. The resulting character sequence is furthermore prefixed by a header indicating the type of the transferable. Finally a footer is appended.

Figure 4.5 shows a sample *OpenPGP Transferable Public Key*. The internal structure of this key is explained in figure 4.2

An ASCII-armored OpenPGP message encrypted with the subkey contained in the *Transferable Public Key* shown in figure 4.5 is shown in figure 4.6. The internal structure of this message is explained in figure 4.3.

To process the actual transferable a OpenPGP implementation first removes the header and footer. Afterward a Base64 parser is used to recover the packet bytes while calculating the CRC-24 checksum. Finally, a (optional) comparison with the appended checksum is performed.

```
-----BEGIN PGP PUBLIC KEY BLOCK-----

mJMEVbmUchMFK4EEACMEIwQAf1vyIPtT3MQdbldMn/gDJUwqL6v3DlWN8nypa7On
wODvpamLg9UPs7l+gvJ7DVnqWoDe0QgssZF3a2em0v1NLRAAxAKi107ZQepZl4nD
lGaBLqL624OBqZtUMaKOR1VHq5RPnHtolmjQBXrbOAjwAHwc+b5jNaSthFjPWC6S
7BuAAY60JFN0ZWZhbiBNb3JlIChFQ0MgS2V5KSA8c3RlZkAyOTA0LmNjPojBBBMT
CgAnBQJVuZRyAhsDBQkcMgSABQsJCAcDBRUKCQgLBRYDAgEAAh4BAheAAAoJEKo6
18NmOtzfHP8CCOL1DU+Qu+k0BSq/oZay9BvmkIm35taDY7SirLErDZx4Hdn1OPrb
zzQyFdwZUQXXw+MAiGKzqsJMplnTZJZ3taEnAgiuAKdOYQKfggOKy2f81Cr6wRup
sxRbQiTQlHhruHhVp7QHml1sw9bCcPsPbVAX8vu8r9MHBJC6aw/VfAhscmTMcbiX
BFW5lHISBSuBBAAjBCMEAUKR6IQtxgtnbhsxfk2M6+UR8kV5YxxmpdwZEu/fzWuq
8tCnvHE9Eibo005MmhHG+aReLOig6spV6fkGjoLrq3PSAGPl7wQS5ST0zM7uXjUE
YfUtl2rCERif0dLLqUHe2YaLsZV7qH8NS6jl/XgXCb2mpwQEz2ayNF/57tYezfP1
d4GRAwEKCYipBBgTCgAPBQJVuZRyAhsMBQkcMgSAAAoJEKo618NmOtzfO/wCB3mO
s5XgbePJLUbAcBqD1q/Or4LZyGdsVmB/2CeuJZ41eOJKO1+4PbuGh5BQL6mNLa0G
cfZ1f8zErLVZZJ6T6tbVAgjM1g9xbJhjuoi+UF6GiVTsw4DHbKcjLCthVtOI0NJF
GfckfKlfBPIagDMnLXuDy6EP9daFRnBRkSiG8XiYorEmJw==
=sWck
-----END PGP PUBLIC KEY BLOCK-----
```

Figure 4.5: Sample ASCII-armored transferable OpenPGP public-key

```
-----BEGIN PGP MESSAGE-----

hMIDPeGobbImiCMSBCMEAM4p9dWWko6xc8FB7YU5ZgnqwQIwx/A2bJjzb4DSHUDP
qyzKf4IMucwhiis/VVJcP8hGRdlwUtxesvVWnBYorjnMAPBE723mnzqyvHmFSsjN
tUedloPPGOwEwmwknWMJ43kVuvz4NdX0UTpNVfh8ZL47Nh42RTuVlxYNMJh6germ
SdXdMK0R+Vq3Jg+BUfduja5ffnm0ThYzbcLT2pQA24A1YnvzYmjKinHCZgchim5m
jP0McdJ0Af+9/hRslpbqTN/6ygEKm4P39dzysgabRGlpLUXmMzRuzkR5P2CBWRbx
wb0z1a8+fQyu8mgpLk8K0NrpQawFrD9hvsMRMldrlvXnu7xcG3vpzTDKsgAi1uU4
3+Y94y4Ut45dJ1Why0QKnhigV6IIYYO0Gkg=
=3EoH
-----END PGP MESSAGE-----
```

Figure 4.6: Sample ASCII-armored transferable OpenPGP message

5 The Java Privacy Guard

In this chapter we give a overview of our implementation of the OpenPGP message format in Java. We show how we implemented the structures described in chapter 4. We illustrate the structure of our implementation. Furthermore, we give an introduction of our API.

As practical part of this thesis an implementation of the OpenPGP message format [RFC4880] in the *Java* programing language was done. This created a library titled **Java Privacy Guard**.

For cryptographic primitives like basic data types, encryption and digest algorithms we used the IAIK-JCE library.

The packet- and transferable-structure, as described in chapter 4, represents the basic structure of our implementation. All packets defined in [RFC4880, section 5] are implemented as one or more Java-classes. Additionally, all relevant packet-compositions are implemented in the form of transferable-classes as defined in [RFC4880, section 11].

Parsing and constructing of transferables and other internal data processing heavily relies on Java's stream architecture. This allows a fast and memory-saving flow of data.

To facilitate the use of our implementation we crated two APIs for access to the library functionality. The APIs are explained in the section following.

5.1 The APIs

In this section we give an overview of our two APIs. We highlight the difference between the two APIs and describe how they implement the OpenPGP message format. Furthermore, we show how the high-level API can be used

to perform basic OpenPGP operations. Figure 5.1 illustrates the structure of the high-level API.

The high-level API represents the interface used by users of our library. It mostly consists of transferable-classes and related toolbox-classes.

The low-level API represents the inner working of the library. It consists of all packet-classes. This API is used heavily in our implementation, specifically by the high-level API. Furthermore, it is possible to use it for behavior not covered by the high-level API. Figure 5.2 illustrates the structure of the low-level API.

To illustrate the functionality of the high-level API, figure 5.3 shows how to load an OpenPGP key. In this example a *PGPReader* is used to load a keyring. It automatically reads all keys from the file. In addition, it builds the class-structure needed to work with the keys. Afterwards it is possible to retrieve a specific *PGPKeyPair* identified by its User ID.

Figure 5.4 shows how to encrypt and sign data. First the demo data is stored in a byte array. This allows processing of generic data. Afterwards, a *PGPData* encrypter is created and configured. It is possible to encrypt the data for multiple keys. In addition, the symmetric algorithm has to be configured. The used asymmetric algorithm is derived from the type of the *PGPKeyPair* objects. Furthermore, the signing process is configured. Finally a *PGPWriter* object is used to post-process the encrypted data. Thanks to Java's stream architecture it is possible to process the ciphertext or just store it.

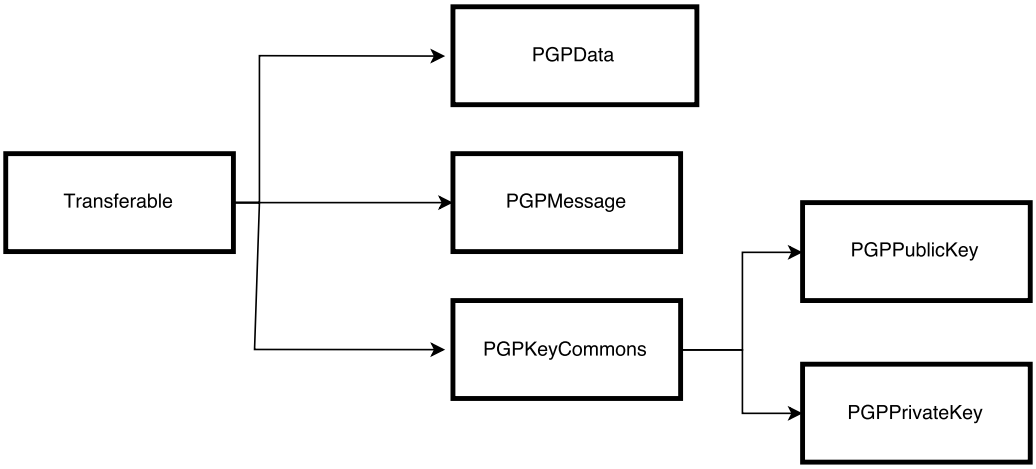


Figure 5.1: The high-level API: Transferable hierarchy

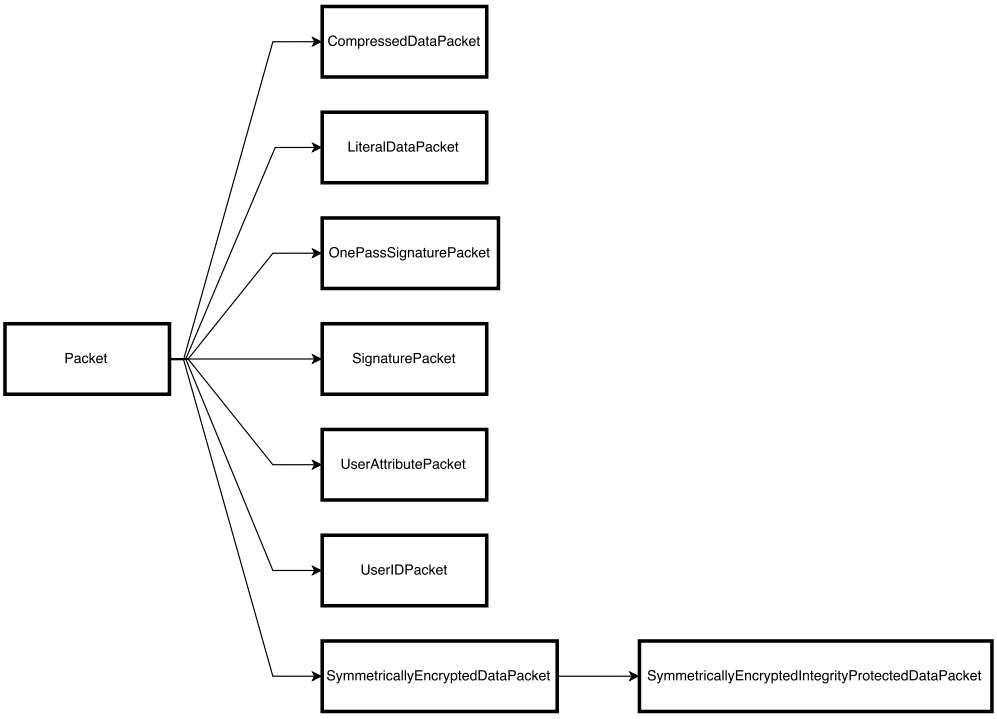


Figure 5.2: The low-level API: Packet hierarchy

```
// initialize PGPReader and load keyring:
PGPReader reader = new PGPReader(in);
PGPKeyRing keyRing = new PGPKeyRing(reader);

// get keypair by User ID
PGPKeyPair key = keyRing.getKeyPair("st <crypto@2904.cc>");
```

Figure 5.3: Loading an OpenPGP key using JavaPrivacyGuard

```
// prepare plaintext:
byte[] plaintext = "hello".getBytes();
ByteArrayInputStream is = new ByteArrayInputStream(msg);

// initialize PGP Encrypter:
PGPData data = new PGPData();
data.setData(is);

// set encryption parameters:
PGPEncryptionInfo encInfo = new PGPEncryptionInfo();
// encrypt for recipient
encInfo.addEncryptionKey(receiver.getPub());
// encrypt for sender, too
encInfo.addEncryptionKey(sender.getPub());
// set symmetric cipher
encInfo.setSymmetricCipher(SymmetricKey.AES128);
data.setEncryptionInfo(encInfo);

// set signing parameters:
PGPSigningInfo sigInfo = new PGPSigningInfo();
sigInfo.setSigningKey(sender.getPriv());
sigInfo.setHashAlgoID(DigestCommons.SHA256);
data.addSigningInfo(sigInfo);

// prepare target for encryption
PGPWriter pgpout = new PGPWriter(out);

// perform encryption
data.encryptTo(pgpout, 2048);
pgpout.close();
```

Figure 5.4: Encrypting and signing data using JavaPrivacyGuard

5.2 Implementation Security

In this section, we give an overview of the security measures of our implementation. Security concerns, as long as they affect the actual design of the OpenPGP standard, are discussed in chapter 6 and in [RFC4880, section 14].

Basic security measures like constant-time implementations are taken into account by our implementation.

We use the IAIK-JCE toolbox to perform cryptographic operations and some other tasks. The toolbox is developed with a focus on security and well tested. No other external libraries have been used.

Another issue are weak defaults for ciphers and other algorithms. Because of the generic nature of a library our implementation does not set any defaults. Thus the issue of bad defaults is shifted to the user of the library.

6 Concerns

In this chapter we give an overview of known concerns about the OpenPGP standard. A detailed security evaluation of OpenPGP was out of scope of this thesis. We give an overview of current keylength recommendations. Furthermore, we discuss the security of algorithms used by OpenPGP. In addition, we highlight some complicacy concerning the security of keys itself. Finally, we discuss the usability of OpenPGP.

It should be noted that at the time of writing of this thesis the used schemes were undergoing a security evaluation [Hes].

6.1 Keylengths

The following section gives a brief overview of the question of key lengths. It furthermore summarizes the current recommendations for keylengths.

Various algorithms are used in OpenPGP, whose strengths are not strictly defined in the standard. Only a recommendation of hash sizes and key lengths is given [RFC4880, section 14]. The recommended values at the time of writing of the OpenPGP standard are shown in table 6.1.

Date	Asymmetric	Hash	Symmetric
2010 (Legacy)	1024	160	80
2011–2030	2048	224	112
$>$ 2030	3072	256	128
$>>$ 2030	7680	384	192
$>>>$ 2030	15360	512	256

Table 6.1: Strengths for algorithms as recommended in [RFC4880, section 14]

The current NIST and BSI recommendations for cipher and hash strengths confirm the recommendations given [12] [15].

6.2 Algorithms

In this chapter we discuss the security of the algorithms used by OpenPGP.

Most of the algorithms used by OpenPGP are still considered secure [Hes]. The two exceptions are the MD5 [XLF13] and SHA-1 [Ste+12] algorithms used for digital signatures and message authentication.

MD5 is already deprecated in the current OpenPGP standard [RFC4880, section 14], and implementations are not allowed to create signatures using MD5.

OpenPGP is not using authenticated ciphers to detect modifications to encrypted messages. Instead an own scheme is used, supporting only the SHA-1 hash algorithm [RFC4880, section 5.13]. Following the finding of sufficient serious attacks on *SHA-1* [Ste+12], it should be considered to upgrade this scheme to a hash algorithm of the SHA-2 family or SHA-3, or by using a mode of operation to achieve authenticated encryption. The OpenPGP standard makes some suggestions to allow such extensions [RFC4880, section 13.11], but fails to suggest authenticated ciphers or modes of operation.

6.3 The Case On Usability

In this section we discuss the usability of OpenPGP implementations. Furthermore, we review the implications the OpenPGP standard itself brings in terms of usability.

The usability of OpenPGP implementations is a critical factor. As usability evaluations of such implementations [WT99] have shown, "effective security requires a different usability standard, and will not be achieved through the user interface design techniques appropriate to other types of consumer software".

This is often hard to achieve due to the nature of non-transparent encryption and asynchronous communication [RFC4880, section 2.1], since the sender's and receiver's OpenPGP implementations never directly talk to each other.

To establish an end to end encrypted communication it is therefore necessary to manually create the key-pairs, and to establish a manual key exchange as describe in section 3.2. This results in a number of security and usability problems.

After generating the key-pair, the receiver has to transmit it to the sender over a potentially insecure channel. The sender then has to verify the integrity and authenticity of the key, which isn't easily possible.
One way of doing this is by transmitting the key's fingerprint over a trusted channel (for example via phone or by printing it on a business card).
Another way of doing this is the *Web of Trust*, as described in chapter 3.2.

An issue of the Web of Trust concerns the definition of trust. Since the OpenPGP standard [RFC4880] does not specify the precise meaning of a certification, it is unclear what means of identity-verification have to be performed before issuing a certification. A possible solution to this problem is the publication of textual description of one's certification policy. In addition, it is suggested to always follow common best practice.

Another issue of the Web of Trust is the fact that it reveals the social structures of OpenPGP users. A suggested countermeasure is to certify keys of unrelated people. In the FLOSS community this is done at so called keysigning parties.

All suggested measures and the Web of Trust itself require interaction by the user. This makes using OpenPGP harder and has a strong effect on its adoption-rate, compared to transparent encryption systems [Gre14].

Another usability issue applies to the fact that many modern email clients are web-based. Since OpenPGP implementations are programs running on the client, access from web-applications is limited. Webbrowser integrations are a possible solution to this problem.

7 Conclusion

In this thesis we presented the *OpenPGP Message Format*. We have discussed its functionality and security. We have shown how OpenPGP uses asymmetric cryptography to provide a secure key- and message-exchange over insecure networks. Furthermore we discussed the trust model of OpenPGP, the *Web of Trust*. We have shown how it can be used to authenticate the sender of a message.

In addition we have shown how we used the structures defined in the *OpenPGP Message Format* standard to implement OpenPGP using the Java programming language and IAIK-JCE.

Bibliography

[12] *NIST Report on Cryptographic Key Length and Cryptoperiod*. Tech. rep. NIST, 2012 (cit. on p. 29).

[15] *BSI Cryptographic Key Length Report for Digital Signature*. Tech. rep. BNetzA, BSI, 2015 (cit. on p. 29).

[Gre14] Matthew Green. *What's the matter with PGP?* 2014. URL: http://blog.cryptographyengineering.com/2014/08/whats-matter-with-pgp.html (visited on 02/20/2015) (cit. on pp. 9, 31).

[HAC] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography*. 1st. Boca Raton, FL, USA: CRC Press, Inc., Oct. 1996. ISBN: 0849385237 (cit. on p. 3).

[Hes] Sandra Hesse. "Security Evaluation of Cryptographic Schemes in OpenPGP." Master Thesis, TU Braunschweig. to appear. URL: http://www.ibr.cs.tu-bs.de/theses/schuerm/openpgp-evaluation.html (cit. on pp. 29, 30).

[IAIKJCE] *The IAIK Provider for the Java Cryptography Extension (IAIK-JCE)*. 2015. URL: https://jce.iaik.tugraz.at/ (cit. on p. 4).

[JCA] *Java Cryptography Architecture (JCA) Reference Guide*. 2015. URL: https://docs.oracle.com/javase/8/docs/technotes/guides/security/crypto/CryptoSpec.html (cit. on p. 4).

[PGP10] *PGP Marks 10th Anniversary*. 2001. URL: https://www.philzimmermann.com/EN/news/PGP_10thAnniversary.html (cit. on p. 5).

[PGPMAN] *PGP 2.6.3i User's Guide*. 1994. URL: http://www.pgpi.org/doc/guide/2.6.3i/ (cit. on p. 8).

[RFC3156] M. Elkins et al. *MIME Security with OpenPGP*. RFC 3156. RFC Editor, Aug. 2001. URL: http://www.rfc-editor.org/rfc/rfc3156.txt (cit. on p. 9).

[RFC4648] S. Josefsson. *The Base16, Base32, and Base64 Data Encodings*. RFC 4648. RFC Editor, Oct. 2006. URL: http://www.rfc-editor.org/rfc/rfc4648.txt (cit. on p. 21).

[RFC4880] Callas, J., Donnerhacke, L., Finney, H., Shaw, D., and R. Thayer. *OpenPGP Message Format*. RFC 4880. RFC Editor, Nov. 2007. URL: http://www.rfc-editor.org/rfc/rfc4880.txt (cit. on pp. 4–9, 12–16, 19, 21, 22, 24, 28–31).

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. RFC 5280. RFC Editor, May 2008. URL: http://www.rfc-editor.org/rfc/rfc5280.txt (cit. on pp. 10, 11).

[RFC5751] Ramsdell, B. and S. Turner. *Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification*. RFC 5751. RFC Editor, Jan. 2010. URL: http://www.rfc-editor.org/rfc/rfc5751.txt (cit. on pp. 10, 11).

[RFC6091] N. Mavrogiannopoulos and D. Gillmor. *Using OpenPGP Keys for Transport Layer Security (TLS) Authentication*. RFC 6091. RFC Editor, Feb. 2011 (cit. on p. 10).

[Ste+12] Marc Martinus Jacobus Stevens et al. *Attacks on hash functions and applications*. Mathematical Institute, Faculty of Science, Leiden University, 2012 (cit. on p. 30).

[Ulrich2011] Alexander Ulrich et al. "Investigating the OpenPGP web of trust." In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 6879 LNCS. 2011, pp. 489–507 (cit. on p. 8).

[WT99] A. Whitten and J.D. Tygar. "Why Johnny can't encrypt: A usability evaluation of PGP 5.0." In: *Proceedings of the 8th USENIX Security Symposium* (1999), pp. 169–184. URL: https://dl.acm.org/citation.cfm?id=1251435 (cit. on p. 30).

[XLF13] Tao Xie, Fanbao Liu, and Dengguo Feng. "Fast Collision Attack on MD5." In: *IACR Cryptology ePrint Archive* 2013 (2013), p. 170. URL: http://eprint.iacr.org/2013/170 (cit. on p. 30).

[Zimmermann] *No Regrets About Developing PGP*. 2001. URL: https://www.philzimmermann.com/EN/essays/PRZ_Response_WashPost.html (cit. on p. 5).

List of Figures

This document is set in Palatino, compiled with pdfLaTeX2e and Biber.

The LaTeX template from Karl Voit is based on KOMA script and can be found online: https://github.com/novoid/LaTeX-KOMA-template