

Bachelor Thesis:
**Possible Threats to PGP Key
Servers**

Tilman Holst

Department of Informatics
University of Hamburg

PGP Key-ID: 21AC6CC4

2006

supervised by

Prof. Dr. rer. nat. Joachim Posegga

SVS – Security in Distributed Systems
Department of Informatics
University of Hamburg

Possible Threats to PGP Key Servers

PGP® and Pretty Good Privacy® are trademarks or registered trademarks of PGP Corporation in the United States and other countries. All other brands or products are trademarks or registered trademarks of their respective holders.

For convenience of reading and writing all dates in this thesis are written in ISO 8601 format. I hope that it will remove the ambiguity found in the various local date conventions. For example the day GnuPG 1.0 was released is written as “1999-09-07”.

Table of contents

Acknowledgments.....	4
Preface.....	4
1 Motivation.....	5
1.1 Modus operandi of this thesis.....	6
1.2 Typographic Conventions.....	7
2 Introduction to PGP and PGP key servers.....	7
2.1 PGP (Pretty Good Privacy).....	8
2.2 Web of trust.....	8
2.3 Organization of a PGP public key.....	11
2.4 Meaning of “public” in public-key.....	12
2.5 PGP Key Servers.....	12
2.6 The global PGP Key Server Network.....	13
3 A brief history of PGP key servers.....	15
4 List of key server software.....	17
5 Problem classification.....	18
5.1 PGP class.....	19
5.2 Key server class.....	20
5.2.1 “Addition only” model.....	20
5.2.2 Store all keys until the end of days.....	20
5.2.3 Missing secure channel between key servers and clients.....	21
5.2.4 Implementation problems.....	22
5.3 Key server network class.....	23
5.3.1 “problem propagation”.....	23
5.3.2 missing secure channels.....	23
6 Example scenarios.....	24
6.1 Scenario 1: Man-in-the-middle.....	24
6.2 Scenario 2: DoS against SKS.....	25
6.3 Scenario 3: Submission of copyrighted material.....	26
6.4 Scenario 4: Flooding.....	27
7 Possible Solutions.....	28
7.1 PGP class.....	28
7.2 PGP key server class.....	29
7.2.1 “Addition only” model.....	29
7.2.2 Store all keys until end of days.....	29
7.2.3 Missing secure channel between key servers and clients.....	29
7.2.4 Implementation problems.....	30
7.3 PGP key server network class.....	31
7.3.1 “problem propagation”.....	31
7.3.2 missing secure channels.....	31
8 Conclusions.....	32
9 Glossary.....	33
10 Literature.....	34

Acknowledgments

I would like to thank the various people supporting me at the time of writing. This thesis would not have been written without them. First of all I would like to thank Olaf Gellert from PRESECURE Consulting GmbH who aroused my interest in PGP key servers and discussed the various issues with me. Martin Johns scientific assistant at SVS looked after my work and was very helpful, especially in structuring my work. Joachim Posegga head of the “security in distributed systems” group at the University of Hamburg (SVS) supervised this thesis and made it possible. Last but not least Klaus-Peter Kossakowski the managing director of the DFN-CERT Services GmbH supported my studies with advice and by providing a working and a test environment. I wish to thank my colleagues at the DFN-CERT for being very supportive in the last months. Thank you all.

Preface

My interest in PGP key servers was aroused in summer 2005, when the DFN-CERT started providing PGP encrypted mailing lists. One of the components involved is a PGP key server. Because only a subset of regular key server features is used I was thinking about writing a small key server program, easy to install and maintain, on my own. I asked Olaf Gellert if there are formal specifications about the key server protocol (called HKP) and he provided me with [HKP 2003] and [RFC 2440]. Unfortunately [HKP 2003] is an expired draft, but fortunately it was quite easy to implement. A few hours and 180 lines of Perl code later a working prototype of a stand-alone, delivery-only key server was finished – and my interest in PGP key servers grew. In one of the various discussions about my tiny key server and key servers in general the idea for this thesis was developed by Olaf Gellert and me.

1 Motivation

PGP key servers are used to store and distribute PGP public keys. Some key servers operate as stand-alone servers, e. g. for some enterprise. Others are connected with other key servers and form a key server network where submitted keys are distributed to each participating server.

The worldwide network of PGP key servers is highly decentralized, but synchronized with each other. This makes the key server network structure very robust. A key submitted once is replicated throughout the entire network and stored on all participating servers.

There are several problems arising from this distributed robustness combined with weaknesses in the principal design of current key servers in general:

- It is nearly impossible to remove a key - once distributed – within the network of key servers. Depending on the interpretation of the applicable privacy laws - especially German and European - this might be a problem for key server operators.
- Another problem is “key server poisoning”, e. g. placing illegal, unpleasant or copyrighted material inside the key server network. It is possible to do this by abusing photographic user-IDs or by using specially crafted fake keys.
- If a faked key is submitted, it is not easily possible to trace by whom that key was originally submitted. The first key server accepting the faked key must be identified and from there the original submission must be traced. A challenging task in a large network of key servers.
- Moreover it is imaginable to fill up a key servers storage with fake or bogus keys. To make it worse these keys are distributed throughout the key server network (called “problem propagation”). Keys containing just data garbage may be recognized by testing if they are cryptographically invalid, but a key that is valid and contains unwanted content is much harder to detect. The same problem exists for signatures.

- A variant of the above is to put some data into applicable parts of a key, e. g. user-ID or photographic user-ID. This may be used for arbitrary purposes ranging from bypassing censorship to misuse as an external data storage.
- Currently there is no written or official but a de facto standard for retrieving keys from a key server. The same problem exists for key exchange between servers.
- Another unsolved issue is that on one hand PGP users can use the web of trust to decide whether or not to trust each other. On the other hand no such trust model exists for key servers. This concerns the trust from server to server and from users to servers. Key servers are not integrated into the web of trust.
- A key server operator may alter or suppress certain keys or signatures without users noticing. Currently most key server networks provide a round robin DNS address – which is sometimes considered as a solution for that problem. But it does not solve that issue. It obscures the process of key retrieval for the user and his applications.
- Of course the whole set of network related threats to services operating without proper cryptographic protection also applies to key servers as well. This includes suppressing keys or altering them. This may happen while retrieving or submitting them.

The problems shown above exist in different classes, e. g. the key server or key server network class or the PGP class. These classes are elaborated in this thesis, threats and solutions are described according to this scheme. For example problems caused by PGP key servers should be dealt with in the key server class. Of course some work on other levels may be required.

1.1 Modus operandi of this thesis

In this thesis threats to PGP key servers and PGP key server networks will be shown and analyzed. A classification of the problem causes will be suggested. Example scenarios for the threats named above will be shown. Possible countermeasures on an ad hoc and on a general

level will be shown and evaluated. Examples will use the figures “Alice and Bob” and as introduced in [SCHNEIER 1996, p. 23].

This thesis focuses on key servers and the key server networks freely available, not on commercial services.

1.2 Typographic Conventions

The following typographic conventions will be used in this thesis.

Constant Width

Used for example commands like: `gpg --recv-keys 21AC6CC4`

2 Introduction to PGP and PGP key servers

To understand today's concerns about PGP key servers it is necessary to have a basic understanding about PGP itself. Knowing the internal design of PGP keys helps understanding the problems shown in this thesis.

The abbreviation “PGP” means “Pretty Good Privacy”, a software released by Philip Zimmerman in 1991. The term PGP is used with different meanings in the literature. Depending on the context, it could mean:

- Philip Zimmermann's suite of cryptographic software released in 1991.
- The whole family of cryptographic software and protocols related to the above.
- A message exchange format, defined by [RFC 1991] and [RFC 2440] (OpenPGP).

If used in this thesis the term PGP always means the whole family of PGP software and protocols.¹ If the term OpenPGP is used it is used to differentiate between PGP in general and OpenPGP.

¹ Please see the glossary for terms and abbreviations used in this thesis.

2.1 PGP (Pretty Good Privacy)

“Open-PGP software uses a combination of strong public-key and symmetric cryptography to provide security services for electronic communications and data storage. These services include confidentiality, key management, authentication, and digital signatures.” [RFC 2440, p. 2]

In the context of PGP the term “public-key” often means the users cryptographic public-key together with signatures from other users. The term “secret-key” just denotes the cryptographic secret-key. If the term “PGP key” is used it normally means a PGP public-key with additional signatures.

PGP is mostly used for securing email communications. It plays an important role in the communication of the worldwide computer security community. For example most security advisories are digitally signed using PGP. Data related to incident handling is encrypted with PGP. The DFN-CERT for example runs PGP encrypted mailing lists.

Another important realm is the distribution of software packages. Many software distributors sign their packages using PGP.

2.2 Web of trust

PGP uses a distributed approach in key management. Certification authorities as found in hierarchical trust models are unknown in PGP's “web-of-trust” model, where users create and distribute their own public keys. Users validate the keys of others on a peer-to-peer basis, stating the validity of a key to user relationship using so called “signatures” or more formally “certification of a user ID and public key”.

“[...] PGP uses a concept known as a 'web of trust', in which any party can certify the identity of any other party (in PGP parlance, 'sign their key'). A signature on a key or document can be trusted if, and only if, there is a path of signatures between the verifier's key and the key used to make the signature in question. The number, shortness, and quality of such paths determine how well the key, and therefore the signature, can be trusted.” [HOROWITZ 1995]

“The key management for PGP is ad hoc. Each user has a set of people he or she knows and trusts. The user exchanges public-keys with those friends...” [PFLEEGER 1996]

To make the “web-of-trust” work in a worldwide scenario public keys are often, but not always stored on a PGP public key server. Exchanging an almost complete set of keys and signatures manually even in a small group of persons is a tiresome and error prone task. Fortunately this is not necessary. Based on the assumption that “your friend is my friend” it is sufficient that someone you trust signed another one's key. PGP differentiates between owner trust (the confidence in the owner of a key to carefully check keys of other users when signing) and the validity of a key (confirmation that a key belongs to a given user).

The owner trust reaches from “ultimate” (at least your own key) over “full”, “marginal”, “none” to “unknown”. The owner trust for your own key is set to “ultimate” automatically. All other owner trust values must be set manually. Only the validity of a key is calculated by PGP. [ASHLEY 1999, p. 24]

The validity of a key is determined as follows: “a key K is considered valid if it meets two conditions:

1. it is signed by enough valid keys, meaning
 - you have signed it personally,
 - it has been signed by one fully trusted key, or
 - it has been signed by three marginally trusted keys; and
2. the path of signed keys leading from K back to your own key is five steps or shorter.

The path length, number of marginally trusted keys required, and number of fully trusted keys required may be adjusted. The numbers given above are the default values used by GnuPG.” [ASHLEY 1999, p. 24]

It is not necessary that each member of a given group has a collection of all keys and signatures issued. However enough keys and signatures are needed to establish a “trust path” to other group members. See figure 4 for an example cross-certification with three participants.

It is possible to simulate a hierarchical trust model with PGP. If a foreign key's (e.g. your certification authority) owner trust is set to ultimate, all keys signed by that key will be treated as valid. Please note: in the literature the term trust is used both for “owner trust” and “validity” of a key. To keep things clear in this thesis only the terms owner trust and validity are used.

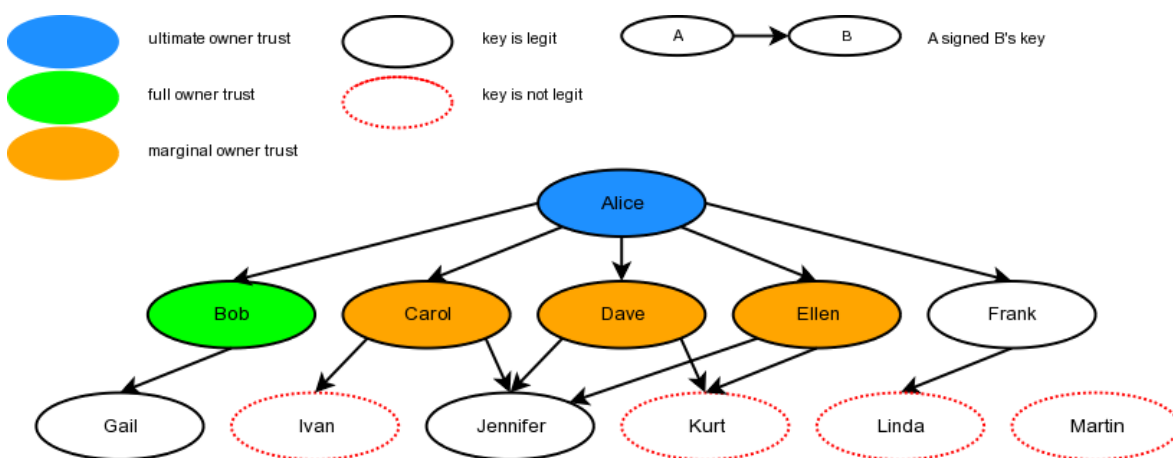


Figure 1: Alice's "web of trust"

Figure 1 shows Alice's personal “web of trust”. Alice signed the keys of Bob, Carol, Dave, Ellen, and Frank. Alice has set the owner trust for Bob to “full”, because she trusts him to carefully check keys before signing. The owner trust for Carol, Dave and Ellen is set to marginal, because Alice does not trust them to sign other keys as carefully as Bob. For all other keys no owner trust is set. PGP sets her own owner trust value automatically to “ultimate”. PGP sets the validity of all keys Alice signed to “full”. Gail's key is considered valid, because it was signed by Bob whom Alice trusts. Jennifer's key is signed by three keys whose owner trust value is set to marginal, this is enough to consider her key valid. Ivan's and Kurt's keys were not signed by enough marginally trusted keys, so they are not considered legit. Alice does not consider Linda's key valid, because she has not defined a trust value for Frank's key. Martin's key has no trust path to Alice, so it is not considered legit. Please see [ASHLEY 1999] for further details.

2.3 Organization of a PGP public key

A PGP key consists of several packets. Such a key usually contains packets of the following types if distributed: [RFC 2440; SCHMEH 2001]

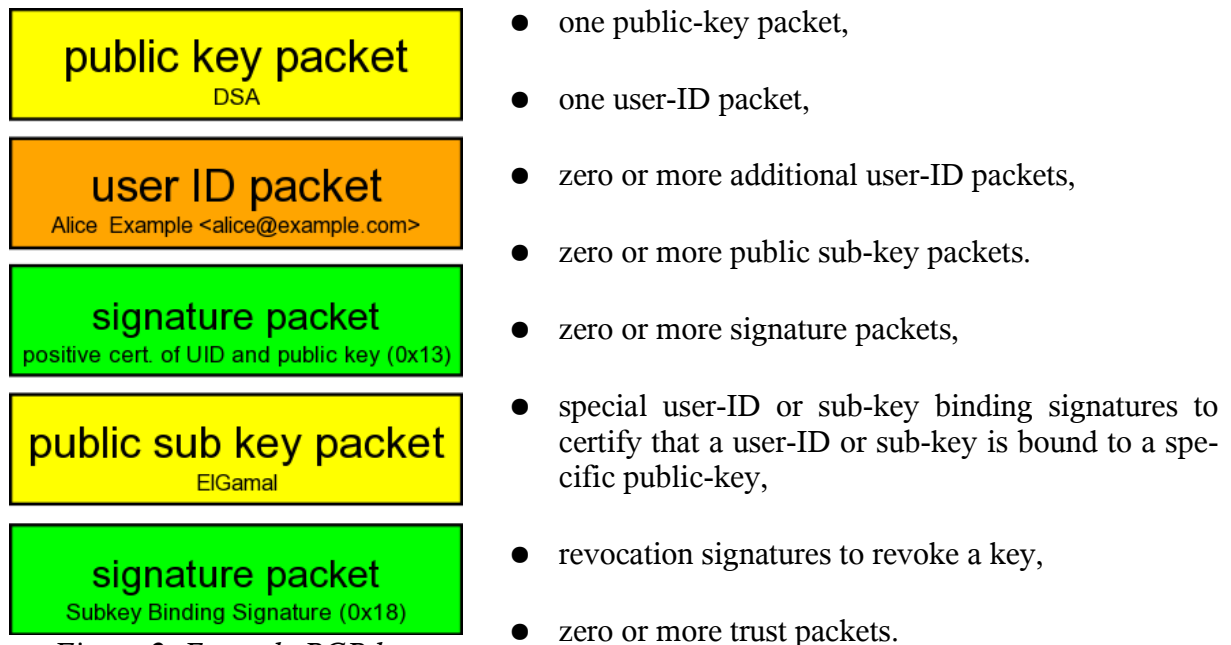


Figure 2: Example PGP key

For a more detailed listing of packet types please see [RFC 2440].

The collection of packets bound to a specific public-key is referred to as a PGP public-key. In the following the term “PGP key” will be used. An example PGP key can be found in figure 2.

A special signature type will be of importance in the following chapters: “revocation certificates”. If a key is no longer in use, is lost or compromised a revocation signature is released. It is good practice to check for new certificates (including revocations) from time to time. Using GnuPG for example this could be done with: “`gpg --refresh-keys`”.

Using GnuPG the sequence of packets could be listed: “`gpg --list-packets`”.

The format in which PGP keys are usually stored is called PGP key ring. A key ring could contain one or more PGP keys.

2.4 Meaning of “public” in public-key

It is necessary to keep in mind what the term “public” in public-key, esp. PGP public-key really means. It has a cryptographic meaning:

- One of two keys in a two-key scheme.

And it has a usage meaning:

- The key is shown to the public and therefore any number of copies may exist.
- It is stored in a public place either by its owner or by someone else.
- Various different versions of the published key could be stored in different places.
- Anyone can add, alter or omit packets or even whole keys, although the result may not be cryptographically valid.
- You cannot call back it once it is released. You can just add a revocation certificate, which must be distributed to all of your communication peers. However not necessarily all copies of the revoked key are updated, so old and therefore not revoked versions may still exist or could be forged by just omitting the revocation certificate.

2.5 PGP Key Servers

The task of a key server is to store and distribute PGP public-keys. Today this is mostly done through web enabled key servers, either used directly or through PGP software. It is important to keep in mind that it is not only cryptographic keys that are stored on the key server, but signatures too. New signatures and keys are received every day. Without the key servers it would be very difficult to distribute especially new signatures to keys. For convenience the term “key server” will be used in the following. Key servers are connected to other key servers and form a “key server network”. Within the network of key servers, PGP public-keys are distributed to each participating key server.

Most current key servers today use the “OpenPGP HTTP Keyserver Protocol” (HKP). It is a simple URL based parameter formalization using HTTP on TCP port 11371. Unfortunately there is no definitive standard of this protocol. A draft of an RFC (The OpenPGP HTTP Keyserver Protocol) does exist, but it expired in September 2003. [HKP 2003]

Key servers ease the exchange of PGP keys. The lengthy and error prone process shown in figure 4 could be simplified and made less error prone if a key server is used to store PGP keys in a central place. However the number of interactions to cross-sign keys in a certain group is – of course – not reduced. Just key retrieval and distribution is eased. Please see figure 3 for a simple example for key server use.

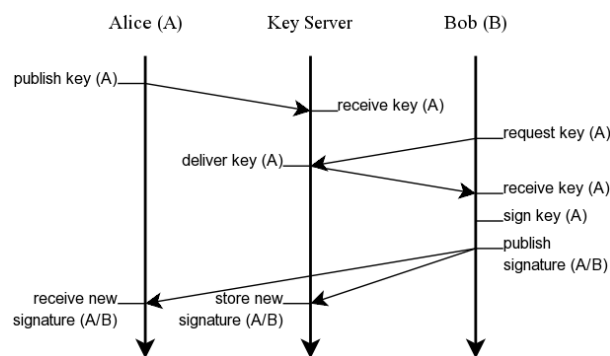


Figure 3: Simple key server use; “key (A/B)” means a signature made with key B is attached to key A.

2.6 The global PGP Key Server Network

Key servers may be connected with each other to synchronize their databases (e.g. exchange new keys or packets). At the moment of this writing there is just one single, global key server network of importance. All participating key servers store all available keys in their database. This is unusual for most distributed systems. This easy design leads to great robustness and availability. In the following sections problems arising from this distribution system will be shown.

Each key server in the network stores a large number of keys. For example 2.336.556 keys are stored on pgpkeys.pca.dfn.de at 2007-07-13.

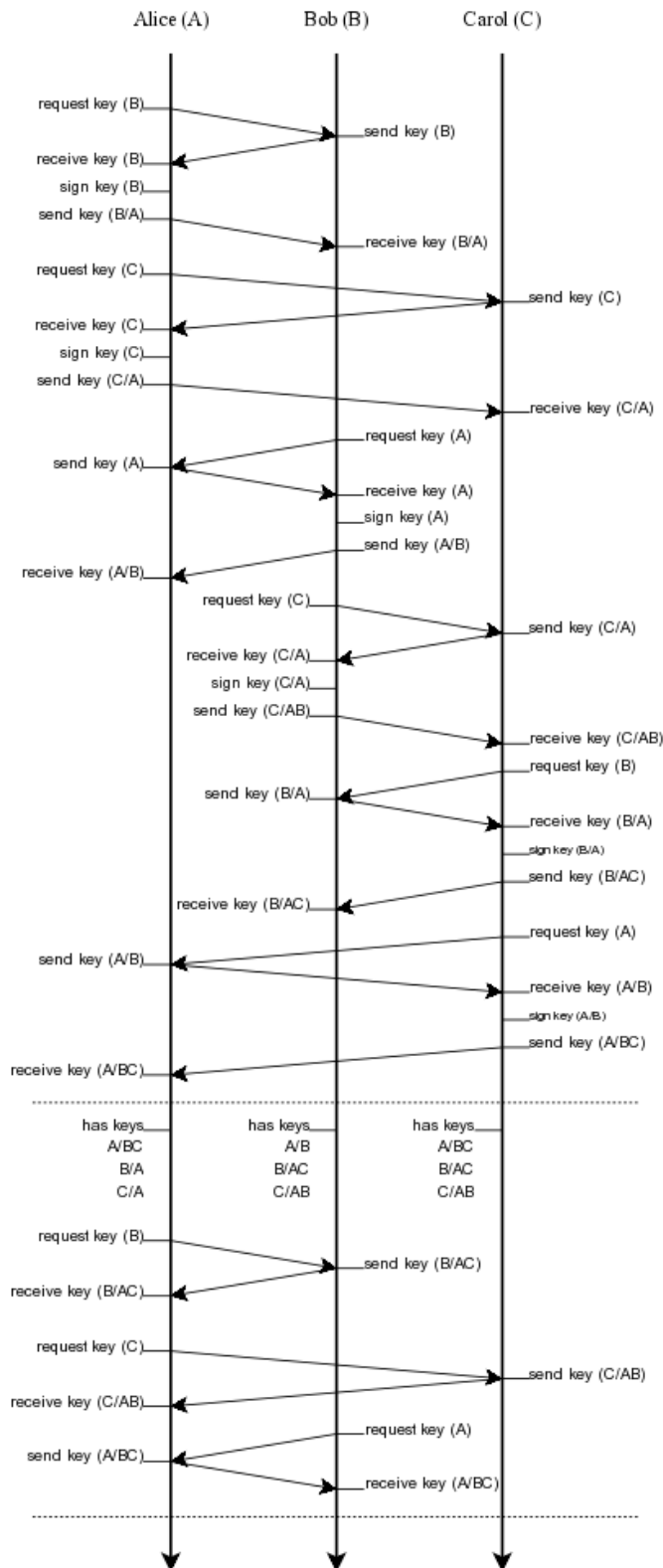


Figure 4: Cross-certification with 3 participants; "key (A/B)" means a signature made with key B attached to key A.

The most important fraction of the participating key servers runs the SKS key server software. They synchronize their databases using a gossip protocol² with practical set reconciliation on TCP port 11370. Older key servers (esp. PKS) use email for synchronization.

There is no central instance or hierarchy in the key server network. All participating servers are on equal terms. The network is irregularly meshed. This loosely coupled system has the advantage of surviving failing companies or a change of policies. As the whole database is stored on every participating system it is very hard to stop the entire key server network from operating.

New packets are distributed to gossip or mail-synchronization peers on a regular basis. A key (or packet) submitted to a key server is distributed to the entire network of participating servers. Please see figure 5 for an example of key and signature exchange with two synchronized key servers and three users involved.

3 A brief history of PGP key servers

To understand the conceptual design of today's key servers it is necessary to have basic knowledge about their history and development. The problem of key distribution is as old as public-key cryptography itself introduced in 1976 by W. Diffie and M. Hellman to the public [SCHNEIER 1996, p. 31]. In the case of PGP this challenge exists since 1991.

The first “key servers” were email gateways to regular PGP software. With the amount of PGP keys growing, the PGP software became a bottleneck, because it was never designed to handle all available keys and the format of the key ring files is not well suited for that purpose. [ANDERSON 1999]

On some FTP servers PGP key rings are made available [ANDERSON 1999]. Those key rings contain only a selection of keys considered to be important. This limited approach is used to distribute a core set of well connected keys to provide a basis for its users. This is not comparable with today's key servers because it is a manual process of selecting and publishing certain

² see [MINSKY 2002] for details

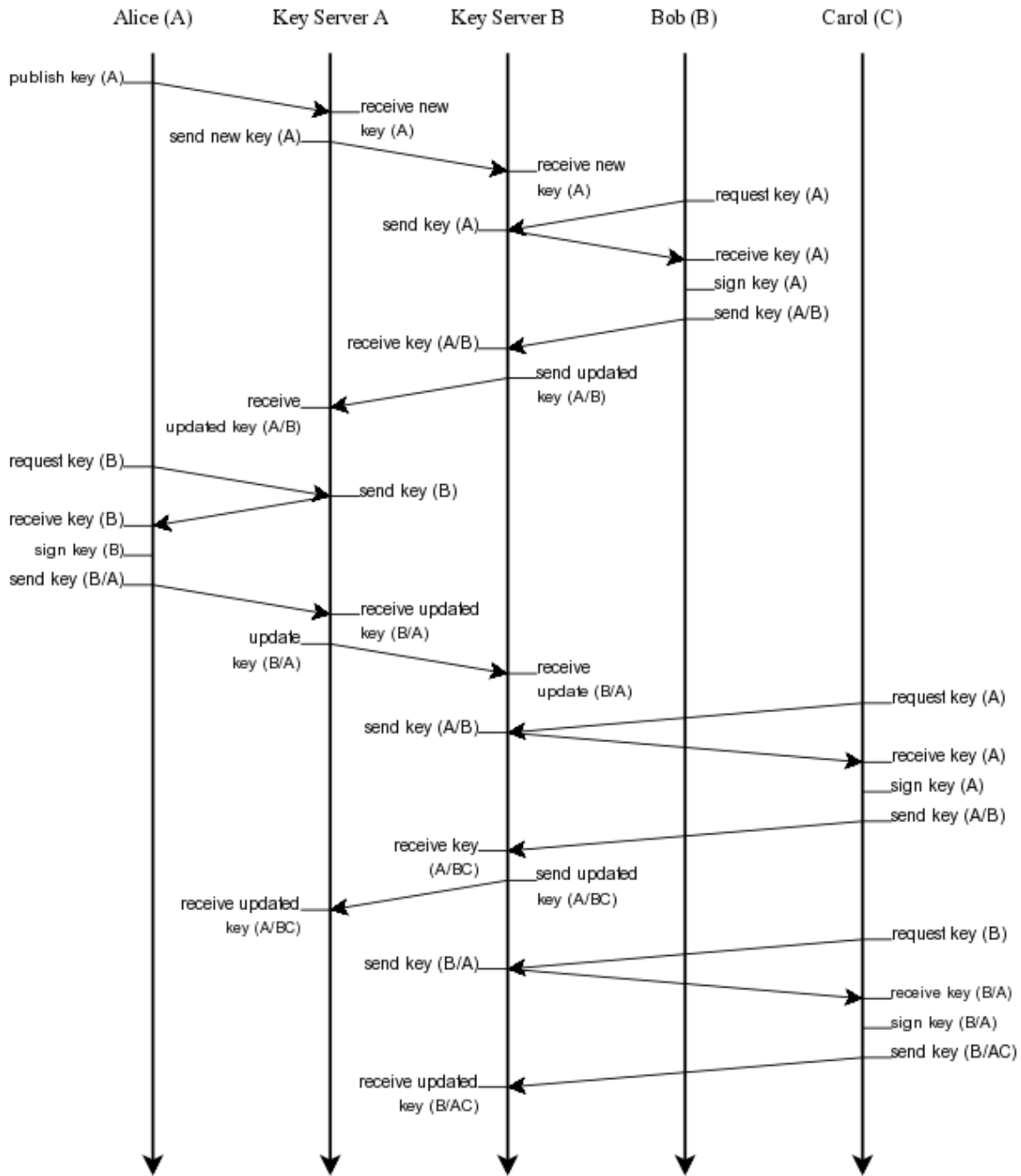


Figure 5: Cross signing using key servers
 “key (A/B)” means a signature made with key B is attached to key A.

keys. And moreover keys could just be retrieved. This service is still available on some FTP servers.

The first web enabled key server, called PKS (public key server) was developed by Marc Horowitz in 1995. Today the development of PKS is continued as a SourceForge project. It

uses HKP and allows searching for keys as well as uploading and downloading. For reasons of performance no checking of cryptographic validity is performed. The PKS server software had problems handling keys with multiple sub-keys due to their database format. In the past keys stored on PKS servers were damaged and made unusable. PKS servers are synchronized with each other by sending emails containing new keys and signatures.

The Synchronizing Key Server (SKS) is developed by Yaron Minsky. The project started probably in 2002. SKS introduced a new and more reliable synchronization scheme for the key database. It uses a gossip protocol with practical set reconciliation described in [MINSKY 2001] and [MINSKI 2002] for synchronization with other (currently only SKS) key servers. Like the PKS key server implementation SKS does not check for cryptographic validity. Today it is the implementation most public PGP key servers run.

Currently approximately 30 SKS key servers form a global synchronized network. It is also connected with some PKS key servers. Together they form the global PGP key server network. Synchronization between SKS and PKS servers is done via the email based synchronization introduced by PKS.

4 List of key server software

Several PGP key server implementations do exist, but only three of them are in widespread use (SKS, PKS, PGP Global Directory). The following list tries to give a basic overview of the projects found during this work.

- **CryptNET Keyserver:** Written by V. Alex Brennen in C and released under the GPL. The most recent release is 0.2.3 from 2005-05-08. <http://cryptnet.net/fsp/cks/>
- **CryptoEx Enterprise Keyserver:** Status unknown. CryptoEx is now part of the PGP Corporation, the key server is no longer available for new customers. <http://www.cryptoex.com>
- **onak keyserver:** Written by Jonathan McDowell and released under the GPL. The project is written in C and started in 2003. Its last release is version 0.3.2 from 2005-03-25. <http://www.earth.li/projectpurple/progs/onak.html>

- **OpenPKSD**: A project started by Hironobu Zuzuki in 2001 and released under the GPL. It is written in Ruby and uses a Postgres database. No official release is currently available.
- **PHKP**: Written by Remko Troncon in 2006 and released under the GPL. It is written in PHP and uses GnuPG. The current version is 0.1. <http://el-tramo.be/software/phkp/>
- **PGP Global Directory**: The key server operated by the PGP Corporation. [PGP 1], [PGP 2],[PGP 3] <http://keyserver.pgp.com>
- **PKS**: The first “modern” key server written by Marc Horowitz in 1995. PKS is written in C. The last release is 0.9.6 from 2003-02-06. This software is release under a BSD license. <http://pks.sourceforge.net/>
- **SKS**: Written by Yaron Minski, released under the GPL. The most recent version is 1.0.10 from 2005-08-21. SKS is written in objective Caml. <http://www.nongnu.org/sks/>
- **Veridis FileCrypt Keyserver**: Available under a “public license” for non-commercial use and a commercial license. <http://veridis.com>

5 Problem classification

It is necessary to separate the different problems into general problems, problems of single key servers and problems affecting an entire key server network. Problems in the class of PGP itself will only be discussed as it is necessary in the context of PGP key servers.

To distinguish between classes of problems it is necessary to separate cause and effect of the problems. The classification shown in this document is always based on the cause of a problem. A single weakness could cause various effects. This will be shown in the example scenarios.

Three classes of problems could be pointed out:

- the PGP class,
- the PGP key server class,

- and the PGP key server network class.

Of course some problem causes are located in more than one class or could influence another class. Sometimes a problem of the key server class has an impact on the key server network class. This type of problem will be called “problem propagation”. In principal problem causes could be further separated into conceptual and implementation problems.

In general problems should be solved in the originating class. Conceptual problems should be solved on the conceptual level, implementation problems should be fixed in the implementation. Legal issues are a special case, they must be dealt with at the key server class, because there legal actions take place, but depending on the exact issue any other class could be affected.

5.1 PGP class

There are privacy concerns on using public PGP key servers. Signatures and user-IDs reveal a lot of personal information. This information could be abused for building user profiles or group profiles. This is quite easy as all necessary information is available. A lot of signature based analysis has been done in the last years. However in most cases the main focus is on a general level, for example [PENNING 2006] or [CEDERLÖF 2004]. “High scores” of well connected keys seem to be popular [STREIB 2004]. Calculations of the “mean shortest distance” (MSD) and rankings based on this value are often found in the PGP community. [PENNING 2006a], [HARRIS 2006]

Another general concern is that publishing your PGP key on a key server includes publishing your user-ID, which usually includes a valid email address. When Philip Zimmerman released PGP in 1991 one of its main uses was to secure email communication. It is no coincidence that most user-IDs look like regular email addresses. That turns a key server into a good harvesting ground for spammers. Fortunately it is limited to only $2.336.836^3$ PGP keys.

3 <http://pgpkeys.pca.dfn.de:11371/pks/lookup?op=stats> (2006-07-14)

There are security concerns too. In the case of RSA for example it is possible to make a factorization attack on the modulus n to determine its factors, p and q . Although this is a very minor possibility this could be used to decrypt secret messages.

For the above reasons it is considered rude to upload foreign keys to a key server if it is not already stored there.

It would have been possible to put the problems shown above into the following “key server class”, because without the key servers these problems would not exist in the same dimension. They are placed in their own class, because it is a conceptual decision of PGP's developers what to publish in a PGP key and what not.

5.2 Key server class

In this class problems are discussed whose cause is the general design or implementation of a single key server. The following basic problems of recent key servers (SKS, PKS) could be traced back to their base model of operation.

5.2.1 “Addition only” model

A public PGP key server stores and distributes all available keys. Under regular circumstances this is an “addition only” policy. The idea of storing all keys for all times seems to be a simple, but robust approach. Even with optimistic assumptions on the development of PGP usage it should be no problem to provide enough storage capacity for all keys in use. This idea has a major disadvantage – it covers just regular use of PGP keys and does not cover abuse of keys, e.g. for data storage. See scenario 3 for an example of this problem.

5.2.2 Store all keys until the end of days

It is not *easily* possible to remove a key from the database or put it into a blacklist of some kind, because a hash value different from the key-ID is needed.

Based on the “addition only” model many public PGP key servers display a warning message like the following:⁴

“Please note: The email addresses contained in your keys will also be published and are accessible to anybody through the worldwide network of keyservers. It is therefore possible that keys and their user IDs are used for malicious purposes, e.g. SPAM.

It is impossible to remove a published key from the keyserver network.

By submitting your key you confirm that these facts are known to you.”

As stated above once a PGP public-key is released the user is no longer in control of that key. A key server does not check if a key is submitted by a legitimate user e.g. the owner of the key.

The statement that removing a key from the key server network is not possible, is not true. It is a difficult and tiresome task, but it is possible. In Germany for example privacy laws could be used to force a (german) key server operator to remove a key. Although that has never happened before.

However old, expired, or revoked keys could be necessary to check signatures made using those keys. So it is a good idea to keep them.

5.2.3 Missing secure channel between key servers and clients

If communicating with users or their clients, there is no authentication of the key server. Depending on the client software used a default key server is used or not. As it is not possible to ensure secure communication between client and server the communication is open for all kinds of man-in-the-middle attacks.

It is possible for an attacker or eavesdropper to:

- notice whose keys are requested,

⁴ <http://pgpkeys.pca.dfn.de/> (2006-07-14)

- manipulate or suppress keys that are received or submitted,

A possible solution would be to establish “secure channels” between clients and key servers.

A “secure channel” is defined by [COULOURIS 2001, p. 61] as follows:

“Secure channels: Encryption and authentication are used to build secure channels as a service layer on top of existing communication services. A secure channel is a communication channel connecting a pair of processes, each of which acts on behalf of a principal.

- Each of the processes knows reliably the identity of the principal on whose behalf the other process is executing. [...]
- A secure channel ensures the privacy and integrity (protection against tampering) of the data transmitted across it.
- Each message includes a physical or logical time stamp to prevent messages from being replayed or reordered.”

The authentication of a client that is just requesting keys towards the server is not of the same importance as the authentication of the server. It should be possible to use something similar to the anonymous methods of TLS, because it should be of no importance who is requesting a key.

5.2.4 Implementation problems

Keys stored on a PKS or SKS key server use approximately twice the space of the original key for storage. Fortunately a current dump of all keys is just about three gigabytes large, that is about six gigabytes for the key server database. However, as it is a linear factor this is no major problem.

In the OpenPGP standard [RFC 2440] there is no size limitation for PGP keys. However SKS for example limits the submission size for the HKP interface to 5MB. For reasons unknown the limit for email synchronization is 10 MB. This could be because of different transport

formats (URL-escaping and base64-encoding). But it is fairly easy to circumvent these limits. It is possible to submit only certain packets of a key to sum up beyond that limit. As PKS and SKS servers internally work with packets, it is effectively just a packet size limit. Unfortunately the synchronization to other key servers is based on packets too. The cause of the problem is that key servers are not considered in the RFC. It is the key server's dilemma that they could not enforce limits without losing RFC compliance. The current size limitations on SKS servers are generously chosen – and therefore they could be abused like in example scenario 3.

Because keys and packets are not tested for cryptographic validity, broken or bogus packets may be stored – and distributed. That behavior can be abused to store arbitrary data on these servers. The data can be placed anywhere in the key. One of the easiest way to store arbitrary data on key servers is to use the packets for photographic-IDs. Large files can be split up and packed into multiple photo-IDs.

5.3 Key server network class

There are two main problems in the key server network class: “problem propagation” and missing secure channels.

5.3.1 “problem propagation”

Problems may be transmitted from peer to peer in the key server network. That happens the same way as legitimate keys are transmitted. A special case is the “sync back”-problem: keys or packets removed from one server and not filtered out will sync back with the next synchronization from one of the servers peers.

5.3.2 missing secure channels

From eavesdropping to alteration, spoofing to flooding all kinds of problems have their cause here.

Some problems from the previous section also have an effect into this class. This “problem propagation” is due to the fact that new packets are replicated immediately to all peers and so step by step to the entire key server network.

The synchronization is based on packets. Limitations will add other possibilities for attacks. Size limitations will make flooding more attractive, because keys or additional packets (e.g. valid signatures) may be kept out.

The worst case of limiting synchronization is that legitimate new keys or revocations are blocked.

There is no secure channel between key servers. No validation of peers takes place. So using techniques like IP-spoofing the synchronization protocol could be abused to inject fake keys directly into a given key server (and its peers of course...).

The worst case here is taking a key server and possibly some of its peers down by flooding them with fake keys.

6 Example scenarios

6.1 Scenario 1: *Man-in-the-middle*

Alice wants to send an encrypted message to Bob. But Bob's PGP key (secret key) was copied (read: stolen) by Mallory. Bob notices and publishes a revocation certificate for his key stating that the key is compromised. So Bob sends this revocation certificate to a key server. This key server distributes the key to its peers in the key server network. Alice uses PGP for some time now and updates her PGP keyring from time to time using “gpg --refresh-keys”. Mallory wants Alice to continue using Bob's old (compromised) key. So Mallory alters the key Alice is receiving from the key server. He has two possibilities:

- He just removes the packet containing the revocation certificate from the key Alice is receiving.

- Alternatively he could use an old version of Bob's PGP key without the revocation certificate.

After refreshing her keyring Alice still has a valid (non-revoked) key from Bob. Maybe she has checked the key's fingerprint a while ago. Perhaps she even signed Bob's key! From her point of view everything is just perfect. Without doubts she sends her encrypted message to Bob.

What happens now?

- Mallory can decrypt the message Alice sent.

Of course Bob would notice, that Alice missed the revocation certificate, but does he get her message? If Mallory could alter keys Alice is receiving he could also manipulate her email. For example he could decrypt Alice message and encrypt it to Bob's new public key. Fortunately he could not forge Alice's signature, so Bob will probably not trust this fake email.

In this scenario Alice and Bob did nothing wrong. Of course Bob could have given his new key and the revocation of the old one directly to all people he is communicating with. But this is not possible for large groups. Because of that Alice and Bob relied on key servers for key distribution and retrieval. This trust is not adequate as there is no secure communication between them and the key servers. Even the distribution of public-keys should be secured by all parties involved.

6.2 Scenario 2: DoS against SKS

Using “`gpg --send-key`” it is possible to supply a large (100 MB) key and try to send it to a key server. The submission fails. But the database transaction logs of the Berkeley DB used by SKS start filling up the free disk space (20 MB each!). These logs are generated for each try that failed. So after a few hours or even faster using multiple client instances the storage partition runs out of free space. To make it worse the key server must be stopped to remove the transaction logs.

The worst case assumption is that a single key server could be filled up with database transaction logs. No new keys or packets could be stored. This could be used to bring down single key servers or as a variant of scenario 1, to keep out revocation certificates on a specific key server.

Another problem is that SKS uses quite an old version of Berkeley DB (version 4.1). To recover the database from the problem shown above the same version of the according utilities is needed. Unfortunately they are not included in recent Linux distributions (e.g. Suse Linux). The database utilities needed are just included for recent versions of Berkeley DB. This is clearly not an SKS problem, but an administration problem that could be solved easily by installing the proper version.

This problem is a combination of problems of different origins. SKS does not handle the submission of large keys in a proper way and leaves the transaction logs. Older versions of the Berkeley DB tools are not included in recent distributions and that shortcoming is not resolved by default.

A solution would be to fix the submission issue in SKS and include a hint in the administration documentation. Software distributors should include an option to install the right versions of utilities if they offer different library versions. Administrators should be aware of these problems and check their installations.

6.3 Scenario 3: Submission of copyrighted material

In this scenario the impact of legal issues to the key server network will be shown with an example of abusing the key server network as a file-sharing system.

Mallory likes music. Especially “Sad But True” from Metallica. He wants to share it with his friends Marvin and Mallet. So he creates a PGP key with the user-ID “Metallica - Sad But True (0e05504fb624f8eb83939c7be8a286bf)”. The md5-hash is included to make it easier to search for specific files. Mallory now adds the song in a photo-ID packet to the key. Then he submits it to a key server. The keyserver replicates the new key to his peers in the key server

network. The PGP key containing “Sad But True” is now stored on many servers. Marvin and Mallet (and anyone else too!) can now download the song from each of them.

Using current key server software it is very hard to remove it from the network. If deleted it is received again upon the next synchronization. It is possible to define input filters e.g. for SKS, but currently this is a manual task.

Some lawyers or copyright owners will be unhappy with this situation. Key server operators may decide to stop the service. Perhaps they just stop synchronizing - that could lead to problems as shown in scenario 1. Perhaps some key server operators face a lawsuit. Maybe they have to pay compensations. Almost certainly they will be forced to prevent such things in the future, so key server developers will implement countermeasures.

The worst case would be that key servers are shut down because of this threat.

There is no easy solution for this problem, as size limitations on keys for example would just open a new attack scenario. Perhaps a size limitation for packets would be a better approach. But it is easy to split up data to several packets. The discarding of photographic IDs is not a good solution too, because it would make a useful service useless. A common approach is blacklisting or the use of Bayesian filters but it would not be a complete solution, as blacklists are reactive and Bayesian filters do not cover 100% of the problem. If the content is packed or encrypted Bayesian filters are useless too.

6.4 Scenario 4: Flooding

Current key servers store every key and packet they receive. The whole network may be flooded with bogus keys. They do not need to be cryptographically valid, because this is not checked at the key servers. Unfortunately key removal is as easy as cleaning the Augean stables and it is not an option to flush out all keys. ;-)

Checking keys for cryptographic validity does not solve the problem, because cryptographically valid keys could be used to flood key servers. And it offers another point of attack, be-

cause validation of keys is a time consuming operation, which could be abused for denial of service attacks, if not properly implemented.

A current key database is approx. 5-6 GB large. Flooding the key server network will increase it dramatically. Some servers may run out of storage and stop their service.

It is hard to filter out unwanted keys while keeping legitimate keys. If fake keys are created in a way that they look like legitimate keys, e.g. with cross-signatures, multiple user-IDs etc., it will be almost impossible to tell the difference. One possible solution would be that new keys need a valid signature from a key already known, but this is not a good solution, because it would construct an obstacle especially for new users. And without a central instance, who should decide who is a legitimate key server user?

7 Possible Solutions

In this chapter possible solutions to the problems described in chapter 5 (different classes of problems) and shown in chapter 6 (example scenarios) will be presented and evaluated. The classification as introduced in chapter 5 is used here too.

7.1 PGP class

One of PGP's development goals was to secure email communications. Therefore in many cases an email address is included in the user-ID field. It should be obvious, that if a key is published (see chapter) all information contained within is published too. To change that a change to PGP's model of operation would be necessary. However if someone is concerned about privacy issues regarding his name and email address he should not publish his key. Perhaps it could be a solution not to include personal information in a PGP public-key. Unfortunately this will complicate the communication with others.

The problem of robots collecting email addresses from key servers should be addressed in the key server class.

7.2 PGP key server class

7.2.1 “Addition only” model

Without an instance – central or distributed – that decides what keys should be allowed to be added to the key server network it is not possible to address that issue. Therefore all keys are accepted and stored.

7.2.2 Store all keys until end of days

A possible solution for the problem of “initial submission” could be a signed message of its owner or a dedicated signature. The problem of “key removal” could be solved using the same approach. This would of course work only if the key owner is still in possession of the according secret key. A dedicated signature stating that it is allowed to distribute that key is located in the PGP class. It would be naive to assume that everyone (key servers and users) would obey to the users wish.

Issuing a revocation does not solve the problem of key removal, because the key is not deleted on the servers. They just add the revocation to the key. It would be possible to delete revoked keys from the server and then to blacklist them. The blacklisting is necessary to avoid the problem of resubmission – without the revocation.

7.2.3 Missing secure channel between key servers and clients

There are three possible approaches to introduce secure channels to key servers:

1. Key servers sign their response messages using PGP. It is important not to sign the requested key, because no validation has been done! Clients could now check the integrity of the message received and check if the message was created recently. If key server operators sign the server's key the key servers could be integrated into the web-of-trust.
2. TLS is used to secure the communication between key servers and clients.

The first approach has a drawback, if the message is encrypted using the client's public key, the key server operator and an eavesdropper could tell who the recipient is. If the message is not encrypted it is possible for an eavesdropper to tell whose key was received. The key server operator knows it anyway. Using anonymous keys on the client could solve this problem.

The second approach is unlike the first different from PGP's own trust model. However it should be easy to implement, because current key requests are already handled using HTTP. Perhaps a simple wrapper would be enough to secure the channel. Because this approach is against PGP's way of a trust model it almost certainly will have acceptance problems:

3. TLS is used to secure the communication between client and key server. Outgoing messages from the key server and as necessary requests from the client are signed using PGP.

This approach would solve the problems of the first approach but introduce a second authorization of the server (the first for the channel, the second for the message). Moreover it would be the most complex approach.

As it is always possible to tell who is the recipient of a PGP message, the first approach is preferable. Furthermore it would solve the problem of the missing secure channel with PGP's own techniques.

From the inclusion of timestamps another very general problem becomes visible: there is no global clock for key servers or PGP in general. Unfortunately it is missing for most distributed systems. The clock problem could be solved if existing time synchronization methods (e.g. NTP) were used. However this would solve a lot of other problems in the PGP class like timestamps in the future or timestamps of signatures before the key was created.

7.2.4 Implementation problems

The space consumption of keys stored on PKS and SKS servers is just a minor issue, but it might hurt in some scenarios, for example in a flooding attack.

There is no easy solution to the problem of missing or too generous size limitations for keys or packets. If chosen too tight it might keep legitimate keys or parts of them out, if chosen too large abuse as shown in scenario 3 is possible. This problem could be addressed in the PGP class, because size limitations on PGP keys should be defined there.

Currently keys are not tested for cryptographic validity. It would be a first step to introduce these tests to prevent the submission of bogus keys. But it is easy to circumvent these tests by submitting masses of valid keys. Checking each incoming key could lead to performance problems, which could be abused for denial-of-service attacks. This should be addressed by proper scheduling procedures.

7.3 PGP key server network class

7.3.1 “problem propagation”

To prevent problems from spreading throughout the key server network it is necessary to check incoming keys and packets on every channel. Keys submitted by users should be checked as keys and packages received from synchronization peers should be checked too. Because there is no instance that could judge all keys or packages this problem could only be resolved in the key server class or by introducing a shared rating system or something similar.

7.3.2 missing secure channels

Using authentication of peers or their messages could prevent spoofing based attacks. To prevent replay attack either a time stamp should be included or some kind of challenge-response authentication. The best solution would be to introduce secure channels between servers in the key server network in a way similar to that shown in section .

8 Conclusions

In this thesis problems with today's PGP key servers have been shown. A classification of problems based on their cause has been proposed. Example threat scenarios to single key servers and to the whole key server network have been presented. Possible solutions – as far as possible – have been given.

During the last six months all of my example scenarios were carefully tested using SKS (version 1.0.10) in a dedicated environment kindly provided by the DFN-CERT.

Public PGP key servers are an important part of the PGP infrastructure. They enable the “web-of-trust” to scale by providing a robust and convenient way to receive and publish keys and signatures. There is no secure channel between users and key servers or between key servers themselves. Without it is difficult to establish trust relationships. The key server infrastructure, although very robust is vulnerable to certain attacks. Secure channels or appropriate alternatives should be implemented between key servers themselves and towards users.

I think this thesis gives a good overview of possible problems and solutions. A promising approach would be to integrate key servers into the web of trust. PGP itself could be used to secure the communication between key servers and towards users.

*“However, I think the whole key server stuff has to be re-thought;
I have some ideas and probably create a white paper.”*

--Werner Koch [Koch 1999, file: doc/DETAILS]

9 Glossary

Alice, Bob, Carol, and others: Fictive persons used for describing cryptographic protocols.

DFN-CERT Services GmbH: The Computer Emergency Response Team for the German Research Network (DFN). The first German CERT founded in 1993. <http://www.dfn-cert.de>

HKP: The OpenPGP HTTP Keyserver Protocol. Expired 2003. <http://ietfreport.isoc.org/all-ids/draft-shaw-openpgp-hkp-00.txt>

PGP key: a PGP public-key together with signatures. Also known as “public-key certificate” in general [SCHNEIER 1996, p. 185].

PGP key ring: a collection of PGP keys, either secret or public keys. It is the usual format in which PGP keys are stored and distributed.

PGP key server: stores and distributes PGP public-keys.

PGP key server network: a loosely coupled and synchronized network of PGP key servers. Each participating servers stores all available keys.

PRESECURE Consulting GmbH: The company Olaf Gellert is working for, Klaus-Peter Kossakowski is its managing director.

Revocation certificate: a special PGP signature which is used to revoke a PGP key.

Signature: a peer-to-peer certification of the validity of a key-to-user relationship.

SVS: Security in distributed systems group, located at the university of Hamburg. Joachim Posegga is the head of this group, Martin Johns works there as a scientific assistant.

Web-of-trust: PGP's trust model. Each user can decide on its own which entities he trusts. If another trusted user signs another users key, this key could be considered valid.

10 Literature

- [ANDERSON 1999] R. Anderson, B. Crispo, J. Lee, C. Manifavas, V. Matyáš, Jr., F. Petitcolas: The Global Internet Trust Register, 1999 Edition
- [ASHLEY 1999] J. M. Ashley: The GNU Privacy Handbook,
[http://www.gnupg.org/\(en\)/documentation/guides.html#gph](http://www.gnupg.org/(en)/documentation/guides.html#gph)
(2006-08-24, various formats), 1999
- [CEDERLÖF 2004] J. Cederlöf: Dissecting the Leaf of Trust,
<http://www.lysator.liu.se/~jc/wotsap/leafoftrust.html> (2006-08-24),
2004
- [COULOURIS 2001] G. Couloris, J. Dollimore, T. Kindberg: Distributed Systems,
Harlow 2001
- [DFN-PCA 2004] Das Vertrauensnetz von Pretty Good Privacy,
<http://www.dfn-pca.de/bibliothek/bulletins/info/tools/gnupg/web-of-trust/1.0/> (2006-08-24), 2004
- [HARRIS 2006] J. Harris: Keyanalyze Results, <http://keyserver.kjssl.com/~jharris/ka/>
(currently unavailable, 2006-08-24)
- [HOROWITZ 1995] M. Horowitz: A PGP Public Key Server,
<http://www.mit.edu/afs/net.mit.edu/project/pks/thesis/paper/thesis.html>
(2006-07-01)
- [KOCH 1999] W. Koch et al.: GNU Privacy Guard (source code),
<ftp://ftp.gnupg.org/gcrypt/gnupg/gnupg-1.4.5.tar.bz2> (2006-08-25)
- [MINSKI 2001] Y. Minski, A. Trachtenberg, R. Zippel: Set Reconciliation with
Nearly Optimal Communication Complexity, IEEE International
Symposium on Information Theory. Washington, DC; 24th-29th June
2001, http://ipsit.bu.edu/documents/isit2001_short.ps (2006-08-24)
- [MINSKI 2002] Y. Minski, A. Trachtenberg: Practical Set Reconciliation,
<http://ipsit.bu.edu/documents/BUTR2002-01.ps> (2006-08-24), 2002
- [PENNING 2006] H. P. Penning: Analysis of the strong set in the PGP web of trust,
<http://www.cs.uu.nl/people/henkpen/henkpenpgp/pathfinder/plot/>,
(2006-08-24)

- [PENNING 2006a] H. P. Penning: PGP pathfinder & key statistics,
http://www.cs.uu.nl/~henkp/henkpgp/pathfinder/mk_path.cgi?STAT=21AC6CC4
- [PFLEEGER 1996] C. R. Pfleeger: Security in Computing (Second Edition), 1996
- [PGP1] New PGP Global Directory, PGP Corporation 2005,
http://download.pgp.com/products/pdfs/PGP-GlobalDirectory_DS_050308_F.pdf, (2006-07-10)
- [PGP 2] PGP Global Directory - Key Verification Policy, PGP Corporation 2005, <https://keyserver.pgp.com/vkd/VKDVerificationPGPCom.html>, (2006-07-10)
- [PGP 3] Global Directoy FAQ, PGP Corporation 2005
<https://keyserver.pgp.com/vkd/VKDHelpPGPCom.html>, (2006-07-10)
- [RFC 1991] D. Atkins, W. Stallings, P. Zimmermann: RFC 1991 - PGP Message Exchange Formats, 1996
- [RFC 2440] J. Callas, L. Donnerhacke, H. Finney, R. Thayer: RFC 2440 - OpenPGP Message Format, 1998
- [RAVEN 2000] K. Raven: Das OpenPGP Web of Trust,
<http://kai.iks-jena.de/pgp/gpg/weboftrust.html> (2006-08-24)
- [SCHMEH 2001] K. Schmech: Kryptographie und Public-Key-Infrastrukturen im Internet, 2. Auflage, Heidelberg 2001
- [SCHNEIER 1996] B. Schneier: Applied Cryptography, Protocols, Algorithms, and Source Code in C, Second Edition, 1996
- [STREIB 2004] M. D. Streib: Key Analysis 11 Apr 2002,
<http://dtype.org/keyanalyze/200204.php> (2006-08-24)
- [YAMANE 2003] S. Yamane, J. Wang, H. Suzuki, N. Segawa, Y. Murayama: Rethinking OpenPGP PKI and OpenPGP Public Keyserver,
<http://arxiv.org/abs/cs.CY/0308015> (2006-08-25), 2003