# GPG-agent based secure password cache for Subversion Version Control System

**BITS ZG629T: Dissertation**


by

Senthil Kumaran

2008HZ12370


**Dissertation work carried out at**

CollabNet Software Private Limited, Chennai


**BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE**
**PILANI (RAJASTHAN)**



October 2010

# GPG-agent based secure password cache for Subversion Version Control System

**BITS ZG629T: Dissertation**

by

Senthil Kumaran

2008HZ12370

**Dissertation work carried out at**

CollabNet Software Private Limited, Chennai

**Submitted in partial fulfillment of M.S. Software Systems**

**Under the Supervision of**
**Mr. Karthikeyakannan Swaminathan, Senior Director - Engineering**
**CollabNet Software Private Limited, Chennai**

**BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE**
**PILANI (RAJASTHAN)**

October 2010

# CERTIFICATE

This is to certify that the Dissertation entitled GPG-agent based secure password cache for Subversion Version Control System and submitted by Senthil Kumaran having ID-No. 2008HZ12370 for the partial fulfillment of the requirements of M.S. Software Systems degree of BITS, embodies the bonafide work done by him/her under my supervision.

**Place** _____

**Date** _____

**Signature of the Supervisor**
Mr. Karthikeyakannan Swaminathan
Senior Director - Engineering
CollabNet Software Private Limited
Chennai - 600 032.

***To my beloved...***

# Acknowledgements

I'm greatly indebted to my colleagues at CollabNet whose guidance and support has been a source of major comfort during hard times.

I sincerely thank the Free Software community for their excellent tools [1], documentation and support in this work.

I'm also greatful to the Subversion Developer Community [2] for the support and motivation that they've extended by providing a beautiful software.

[1] This report was generated using LaTeX,BiBTeX, Emacs and GIMP
[2] http://subversion.apache.org/

# Contents

**1 Introduction** **2**

1.1 Subversion Password Cache . . . . . 2

1.1.1 *NIX problem . . . . . 2

1.2 GPG-agent . . . . . 3

1.3 CollabNet and Apache . . . . . 3

**2 Analysis of Subversion System** **4**

2.1 Purpose of the Subversion System . . . . . 4

2.1.1 Suitable deployment of Subversion . . . . . 5

2.1.2 History of Subversion . . . . . 5

2.2 Other version control systems . . . . . 6

2.2.1 CVS . . . . . 6

2.2.2 Aegis . . . . . 6

2.2.3 Arch . . . . . 6

2.2.4 Bazaar . . . . . 7

2.2.5 BitKeeper . . . . . 7

2.2.6 Darcs . . . . . 7

2.2.7 Git . . . . . 8

2.2.8 Mercurial . . . . . 8

2.2.9 Monotone . . . . . 8

2.2.10 Perforce . . . . . 9

2.2.11 PureCM . . . . . 9

2.2.12 Vesta . . . . . 9

2.3 Using Subversion . . . . . 9
2.3.1 Repository and working copies . . . . . 9
2.3.2 Branching and Merging . . . . . 10
2.4 Joining the Subversion community . . . . . 10
2.4.1 Working on an open source project . . . . . 11
2.4.2 Coding style . . . . . 11
2.5 External libraries used by Subversion . . . . . 12
2.5.1 Apache Portable Runtime . . . . . 12
2.6 Subversion Architecture . . . . . 13
2.7 System Layers . . . . . 13
2.8 Credential caching mechanisms . . . . . 16

**3 Analysis of GPG-agent System 18**
3.1 GnuPG . . . . . 18
3.2 GPG-agent . . . . . 18
3.2.1 GPG-agent feature . . . . . 19
3.2.2 GPG-agent package . . . . . 19
3.2.3 GPG-agent dependencies . . . . . 20

**4 Implementation 23**
4.1 Assuan Protocol . . . . . 23
4.2 GPG-agent password cache . . . . . 24
4.2.1 simple_gpg_agent_first_credentials . . . . . 25

**5 Sample Run 27**
5.1 GPG-agent pinentry dialog . . . . . 27
5.2 Subversion listing . . . . . 27
5.3 Password cache . . . . . 27
5.4 gpg-connect-agent . . . . . 29

**6 Conclusion 31**

# List of Figures

2.1 Subversion Architecture . . . . . 14

5.1 Password pinentry dialog . . . . . 28
5.2 Directory listing shown after getting password . . . . . 28
5.3 Password cache . . . . . 29
5.4 Password pinentry dialog . . . . . 30

# Abstract

This dissertation solves the problem of security, in very large deployements of Subversion such as in a corporate environment where the user accounts in individual machines are created in a common server or home folders which are in turn mounted from an NFS share. In these kinds of environments when the passwords are cached in plaintext that becomes a major security hole which needs to be solved. Providing GPG-agent based password cache is a better mechanism to solve it by enforcing it as a corporate policy, limiting the usage of other types of less secure password stores.

# Chapter 1

# Introduction

Subversion is a popular free open source software based centralized version control system which is widely used both in open source and corporate project environments. Subversion is based on a client server model with three kinds of server access methods such as plain filesystem (file:///), over web servers (http:// or https://) and svnserve daemon which is a custom subversion protocol (svn://).

## 1.1 Subversion Password Cache

Authentication and authorization policies can be established over this client server model of subversion. In the client side there is a password cache which stores the user's password, which when an user talks to the subversion server will provide the cached passwords automatically to the server when challenged. This saves a lot of typing for the user. Subversion being a project developed for almost 10 years now, has implemented various mechanisms to store passwords in the client side. Though there exists various mechanisms for different operating systems, there is poor support for caching passwords in an encrypted format in *NIX like operating systems which my dissertation tries to solve.

### 1.1.1 *NIX problem

In windows Subversion uses the wincrypt API provided by the operating system and stores the password in an encrypted format. This is same across all windows flavors and hence generic in a windows system. On the other hand in Mac OSX subversion uses the Keychain services in order to store passwords in encrypted form. But in the case

of UNIXes like GNU/Linux kind of operating system there is no native APIs provided by the operating system itself which makes it hard to come up with a generic solution to store passwords in an encrypted form for these kind of operating systems. Of course there exists systems like GNOME Keyring, Kwallet, etc which are specific and tied to the desktop environment which the users have in their desktops. But a generic methodology to store passwords in an encrypted form is lacking in *NIX systems and GPG provides such a mechanism independent of the user's desktop environment.

## 1.2 GPG-agent

GPG which is the acronym for GNU Privacy Guard is a free software alternative for Pretty Good Privacy (PGP) suite of cryptographic software. GPG is compliant with OpenPGP standard specification. This dissertation will explore and implement GPG-agent based password cache to store user password in Subversion client. The GPG-agent is a daemon which provides cryptographic services to any program. The GPG-agent based password stores can be extended to any program such as pidgin, network-managers, etc in a *NIX system which requires caching of passwords in the filesystem. This GPG-agent daemon is independent of operating system and desktop environments, hence this provides a generic solution for the problem of encrypted store in a *NIX system.

## 1.3 CollabNet and Apache

CollabNet Software Private Limited [1] is the primary sponsor of the Subversion Open Source project since 1999 and I am involved in developing Subversion by adding new features as a part of my employment with CollabNet, which adds another reason why this topic will be more aligned with my dissertation goals. Recently in December 2009, CollabNet donated the Subversion project to Apache Software Foundation [2] which is a huge non-profit Free Software based independent organization. Apache Foundation maintains lots of free software projects so that the individual software communities do not have the overhead of running a full fledged organization.

[1] http://www.collab.net/
[2] http://www.apache.org/

# Chapter 2

# Analysis of Subversion System

This chapter describes the Subversion system as deep as we need to understand the functionalities of the system and the client/server architecture with layered design. The authentication system for caching passwords both in an encrypted and unencrypted form is described in detail. Credential caching mechanisms, that are available now, are described and their features are explained in section 2.8.

## 2.1 Purpose of the Subversion System

In the book Version control with Subversion [13] authors of Subversion describe the system as a fantastic hammer, but we cannot view every problem as a nail. When we are deciding of the deployment of Subversion, we need to think, if the project needs Subversion at all.

Generally open source Subversion system handles a management of a versioned data (most often a source code of software). The system is free to use and these are its main features:

- Client-server architecture - a server handles storing of a data and clients use that data. There are several alternatives, how to get the data from the server, depending on protocol, security reasons, performance, etc. The system allows using the server and the client on one computer, as well as on different computers connected using a network.
- Keeping changes - besides the actual version of data every change made to the data is kept. The system allows restoring any of versions from the history, even if there is a newer version of the data stored on the server. We can look at the

data as an ordinary file system tree, which has one dimension more (the time). In other words the data stored on the server are sequences of snapshots, while we can see every snapshot as a file system tree.

- Ability to work in parallel - The Subversion system has been designed for parallel work of many clients. A client usually do the following: gets data from the server, changes the data and sends them back to the server. The system allows to do all these changes parallel by many users, without locking the data or even parts of them. Despite that, the system can keep the data in integrity and no change can be done without cognition of the user. The way how this is implemented, is called copy-change-merge solution and we can read more about this in [13].

### 2.1.1 Suitable deployment of Subversion

As mentioned before, the Subversion system is most often used to manage a source code of applications. It is useful utility for manage data, which are centralized on one place, but changed by many users from different places.

If a mistake is discovered, data can simply be restored to the original version. This can be done at any time in the future, even if the data are changed several times after that bad change. System stores changes in files data as well as in file properties (file property changes were not kept in CVS - Concurrent Version System [5]).

On the other hand, Subversion is not suitable to manage data, which are static or which we dont need to keep history of. There are better utilities to handle these situations.

### 2.1.2 History of Subversion

The history of Subversion began in 2000, when CVS had been a system used to manage software source code. CVS has many disadvantages, so CollabNet [4] decided to implement a new system, which will replace CVS.

CollabNet contacted author of the book Open Source Development with CVS, Karl Fogel, and he and others (Ben Collins-Sussman, Brian Behlendorf, Jaseon Robbins and Greg Stein) created a community of active contributors, who stayed behind Subversion development until recently. The community is now organized by The Apache Software Foundation [2], which manages many other open source projects.

Note: The Subversion project has been accepted as an Apache project just recently, it had been organized by Tigris.org before. The move was announced on Wednesday,

November 4th, 2009. More about that on [12] or [2].

## 2.2 Other version control systems

Some other version control systems (or configuration management systems) are shortly described later in this chapter. More information about comparison can be found on [16] or on [19].

### 2.2.1 CVS

CVS (Concurrent Version System) was the most used version control system to control source code a few years back. Subversion is straight coming out of CVS and does not want to break a good known and used way to manage source code. On the other hand authors tried to get the best of the CVS and remove or minimize its disadvantages.

The CVS system composes from one central repository and every change is uploaded back to repository. It does not recognize changes in a directory tree and revision numbers are related to files, not to whole repository (like it is in Subversion). Branching and merging are not usable in CVS, there are problems with binary files and locking is possible only explicitly by user. Despite many disadvantages CVS has been the most used SCM for a long time, but now there are many alternatives, which are shortly described in the next paragraphs. More about CVS on [5].

### 2.2.2 Aegis

Aegis is a transaction-based software configuration management system, but it is not used very often. It concentrates integrity and testing data and uses distributed way to share data. Its not suitable to use in a network and it is relatively complicated to manage the system.

Compared to Subversion it offers disconnected commits, peer-to-peer architecture and it uses a filesystem-based data backend (no SQL or embedded database are integrated). More about Aegis on [18].

### 2.2.3 Arch

Arch system fixes some problems, which CVS had. It is similar to Subversion when a directory tree is changing or new branches are created. Thanks to support of standard

protocols (FTP, SFTP and WebDAV over HTTP or HTTPS) it is easy to deploy and to use it on the Internet.

It is particularly useful for public free software projects, because it is easy to learn and to administer. It is a distributed system. More about Arch on [3].

### 2.2.4 Bazaar

The system is implemented in Python, it is very simple with basic configuration, though it is simple it is very scalable using many plug-ins (new storage formats can be plugged-in too). It is well portable and able to manage even large projects. There is no need to choose between central and distributed version control tools, Bazaar directly supports many work-flows with ease.

The system supports many best practices including re-factoring, pair programming, feature branching, peer reviews and pre-commit regression testing. With true rename tracking for files and directories, merging changes from others simply works better. More about Bazaar on [17].

### 2.2.5 BitKeeper

BitKeeper is a very powerful, capable and reliable version control system that supports copies, moving and renaming, atomic commits, change-sets, distributed repositories and propagation of change-sets, 3-way merging, etc. It is portable to all major UNIX flavors, and to Win32.

BitKeeper has a few drawbacks. It can only duplicate a repository and work against it and cannot work against a working copy of a snapshot. This makes large checkouts over a WAN very slow.

### 2.2.6 Darcs

An open source (GNU GPL) distributed version control system with a very simple repository creation and some interesting features (like spontaneous branches). An implementation in Haskel, not very supported and used language, speaks against its massive deployment. More about Darcs on [15].

### 2.2.7 Git

Git is a free and open source distributed version control system influenced by commercial BitKeeper. It is designed to handle everything from small to very large projects with speed and efficiency. In the beginning it was designed to be used by many other version control systems, but now it is an independent project. This system is used by Linux distributions and the Linux kernel development. It supports a non-linear distributed development, branching, merging etc.

Following the UNIX tradition, Git is a collection of many small tools written in C, and a number of scripts that provide convenient wrappers. Git provides tools for both easy human usage and easy scripting to perform new clever operations. More about Git on [6].

### 2.2.8 Mercurial

A distributed system developed as a reaction to moving the source of the BitKeeper from open source to a commercial sphere. It was designed with the intention of being small, easy to use, and highly scalable. It is often marked as the fastest version system. Mercurial is a platform independent system written in Python and C. It offers many extensions and uses many commands known from Subversion. More about Mercurial on [8].

### 2.2.9 Monotone

Monotone is a distributed version control system with a different philosophy. Namely, change-sets are posted to a depot (with which the communication is done using a custom protocol called netsync), which collects change-sets from various sources. Afterwards, each developer commits the desirable change-sets into his own private repository based on their RSA certificates and possibly other parameters.

Monotone identifies the versions of files and directories using their SHA1 checksum. Thus, it can identify when a file was copied or moved, if the signature is identical and merge the two copies. It also has a command set that tries to emulate CVS as much as possible.

The Monotone architecture makes an implementation of many features easier. It is not without flaws, however. For example, Monotone is slow, and doesnt scale well to large code-bases and histories. More about Monotone on [9].

### 2.2.10 Perforce

Perforce is a centralized, commercial (non-free) solution for version control. It is very fast, very stable and robust. It scales very well, and has a good reputation. It requires an annual per-seat licensing, but it is also available for interested open source developers under a gratis license. More about Perforce on [10].

### 2.2.11 PureCM

PureCM project is a commercial, cross-platform SCM, which offers some features, which are not implemented in many other SCMs - a stream hierarchy view of the data, a commandline interface as well as a GUI interface in base, a checkpoint support for a parallel work, merging on the server, a visual folder diff, etc. More about PureCM on [11].

### 2.2.12 Vesta

Vesta is a mature software configuration management system that originated from an internal use by Digital Equipment Corporation (now Compaq/HP). It is a replacement for CVS which provides more than a mundane revision control systems. It is an open source software under the LGPL.

Vesta is a portable SCM system focused on supporting of development software systems of almost any size, from fairly small (under 10,000 source lines) to very large (10,000,000 source lines). More about Vesta on [14].

## 2.3 Using Subversion

This section takes a short look at an ordinary work with the Subversion version control system.

### 2.3.1 Repository and working copies

Subversion is a centralized (not distributed) system, where data are stored on the server, so-called repository. Repository is a directory tree with files and users can get the actual directory tree as well as any of the trees from history. Every change in the repository assigns a new incremental unique number to it. We call this number a

revision number and the revision number 0 means an empty directory (it is a convention used by Subversion).

We need to create so-called working copy for working with the data from the repository. The working copy is a copy of a part of the data from the repository on the client's file system. A client can make as many changes as he wants and then data gets copied back to the repository. An operation that provides copying the data back to the repository is called commit and it is designed as an atomic transaction. During the transaction data are checked if no newer data has been uploaded before (by another user). If there is a version conflict detected, the user (who is doing the commit transaction) is responsible to get all changes from both users to be applied and he is responsible to keep the data in an integrated state.

Subversion provides both copy-modify-merge and lock-modify-unlock ways of editing files collaborating with peer developers.

### 2.3.2 Branching and Merging

Branches and tags are well known from many other versioning systems, but were not implemented well in CVS. Both are actually copies of the actual version of, part of the repository.

Branches are used to develop new features or new versions of the software, tags are used to store the snapshot of the some version (for example an official releases).

Subversion does not force users to keep a strict directory tree structure, nevertheless the most often organization is the following:

- /trunk - the actual version, which is under development at all the time and which is updated from branches
- /branches - development branches, which are created, updated and removed in time
- /tags - tags are created in time, but not changed nor deleted any more

## 2.4 Joining the Subversion community

Subversion development was started by CollabNet and continues under Apache Software Foundation (ASF) now. Thanks to CollabNet for bootstrapping a contributor's community, which now is a middle-sized open source community with a specialization

to development and maintenance utilities and tools to improve the collaboration on the software development. Its motto sounds We're here to help you help us!

Anyone can join the community in many fields - analysis, design or implementation of the tools, general libraries, tests or documentation. The community likes to get new suggestions for new projects or to overtake some existing ones.

Joining the Subversion development means to join the Apache Subversion mailing list and sign the Individual Contributor License Agreement. Often form of contribution is bug logging or debugging, but in the case of this dissertation it is mandatory to join the development, to communicate and resolve problems with other contributors. Since I have already worked on some features of Subversion and acquired a "Full Committer" access to the Subversion repository I can commit my code directly to the development branch and discuss with other fellow committers across the world about my code.

### 2.4.1 Working on an open source project

For the implementation of this feature a new development branch was created and it available here [1] The new password store will be added to an official release in the future.

All communication on open-source project under ASF takes place only using mailing lists which are archived and available for the public to browse through. After this feature was introduced to the Subversion community it was well recieved and many Full Committers of the project are actively looking into the code in order to improve or suggest more ideas.

### 2.4.2 Coding style

Every new developer, who collaborates with ASF community, has to agree with conditions organized to establish an easier cooperation. Many projects are implemented in C/C++ language, which does not force programmer to follow general indent style, comments writing etc.

Reading or updating a source code in other coding style is difficult, so some rules (similar to GNU code style) had been established and every contributor should use them.

The common language of the documentation is English, which should be simple and single valued.

[1] http://svn.apache.org/repos/asf/subversion/branches/gpg-agent-password-store/

## 2.5 External libraries used by Subversion

Subversion uses some general libraries, which handle some parts of the system. These libraries are mentioned in the following paragraphs.

### 2.5.1 Apache Portable Runtime

The mission of the Apache Portable Runtime project (APR) is to create and maintain software C/C++ libraries that provide a predictable and consistent interface to underlying platform-specific implementations. The primary goal is to provide an API, which software developers may code in, and to be assured of predictable if not identical behavior regardless of the platform on which their software is built, relieving them of the need to code special-case conditions to work around or take advantage of platform-specific deficiencies or features.

APR was originally a part of the Apache HTTP Server, but it has been spun off into a separate project of the Apache Software Foundation, and it is used by other applications to achieve platform independence.

The range of a platform-independent functionality provided by APR includes:

- Memory allocation and memory pool functionality
- Atomic operations
- Dynamic library handling
- File I/O
- Command argument parsing
- Locking
- Hash tables and arrays
- Mmap functionality
- Network sockets and protocols
- Thread, process and mutex functionality
- Shared memory functionality
- Time routines

- User and group ID services

More about APR on [1].

## 2.6 Subversion Architecture

Generally, if we want to implement a new feature of the system, we should know the rest of the system. Figure 2.1 describe the Subversion architecture fastest. A reader will understand where important components are located, but detailed structure of them is not needed generally.

The server communicates with the client application using several Internet protocols, based on TCP/IP. User always works with his working copy, not with the repository directly. The repository itself can be directly accessed only by a repository administrator, using several routines, such as svnadmin, svnlook, etc. Client applications are graphic or command-line clients using the libsvn client library to process commands, libsvn wc library to work with working copy and libsvn ra library to access the repository. Client's application can be integrated in OS, like the TortoiseSVN does it in Windows.

We can recognize a layered architecture on the server too. Every layer has its own interface, while the most important interface for this dissertation is the client interface.

Right now Subversion supports password stores such as windows WincryptAPI, MacOSX KeyChain services, GNOME Keyring, KWallet which stores passwords and passphrases in an encrypted form. The aim of this dissertation is to add a password store support via GPG-agent which will store passwords and passphrases in an encrypted form in *NIX systems.

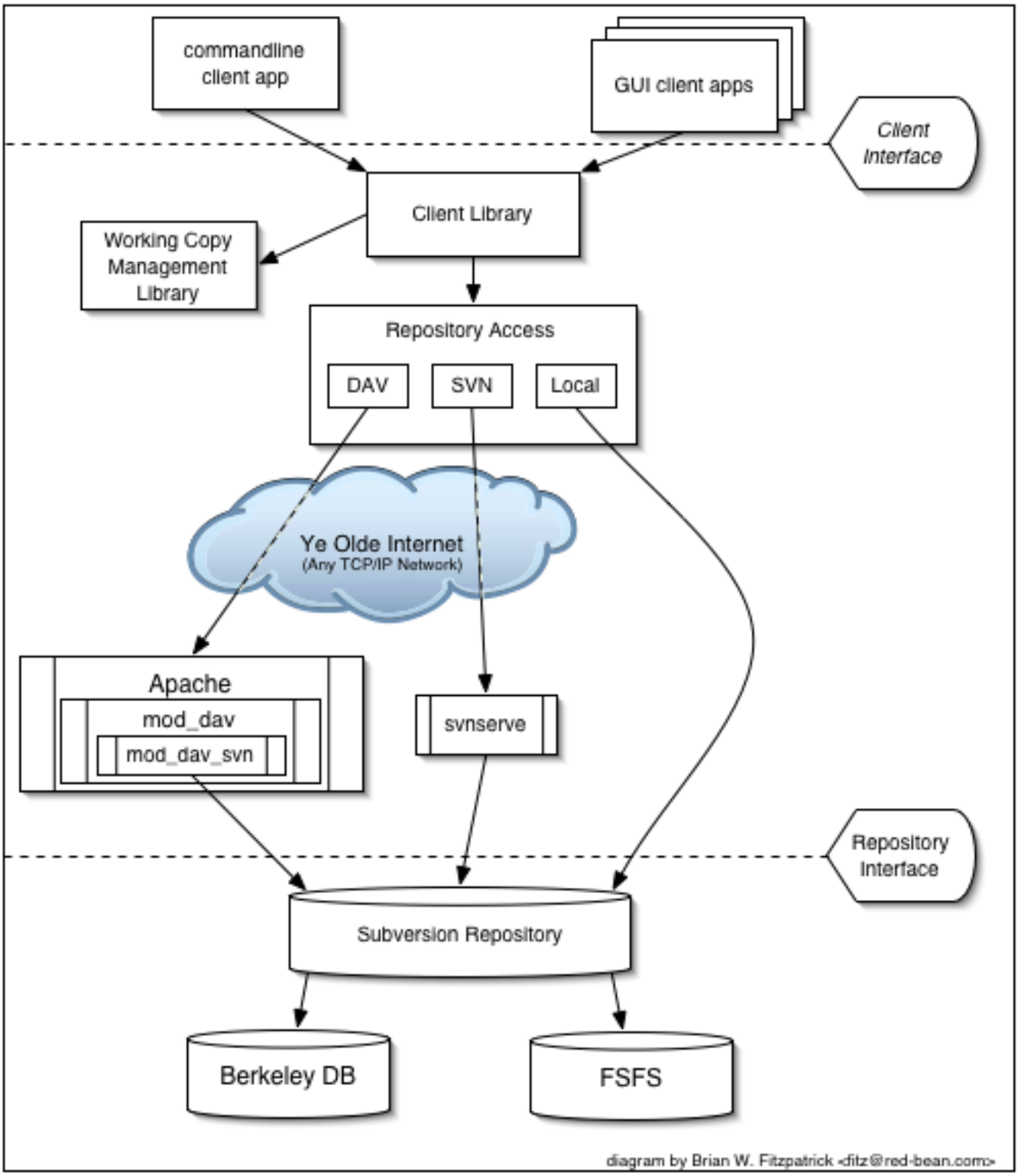
More information about other layers can be found in [13].

## 2.7 System Layers

We have mentioned the architecture divided into layers, while every layer or library has strictly defined interface. Every library is a part of exactly one layer, which can be the following:

- Repository Layer
- Repository Access (RA) layer

Figure 2.1: Subversion Architecture



- Client layer

Let's take a short look on all libraries:

- libsvn client - Primary interface for client programs
- libsvn delta - Tree and byte-stream differencing routines
- libsvn diff - Contextual differencing and merging routines
- libsvn fs - Filesystem commons and module loader
- libsvn fs base - The Berkeley DB filesystem backend
- libsvn fs fs - The native filesystem (FSFS) backend
- libsvn ra - Repository Access commons and module loader
- libsvn ra local - The local Repository Access module
- libsvn ra neon - The WebDAV Repository Access module
- libsvn ra serf - Another (experimental) WebDAV Repository Access module
- libsvn ra svn - The custom protocol Repository Access module
- libsvn repos - Repository interface
- libsvn subr - Miscellaneous helpful subroutines
- libsvn wc - The working copy management library
- mod authz svn - Apache authorization module for Subversion repositories access via WebDAV
- mod dav svn - Apache module for mapping WebDAV operations to Subversion ones

The fact that the word miscellaneous appears only once in the previous list is a good sign. The Subversion development team is serious about making sure that functionality lives in the right layer and libraries.

## 2.8 Credential caching mechanisms

Many servers are configured to require authentication on every request. This would be a big annoyance to users if they were forced to type their passwords over and over again. Fortunately, the Subversion client has a remedy for this - a built-in system for caching authentication credentials on disk. By default, whenever the command-line client successfully responds to a server's authentication challenge, credentials are cached on disk and keyed on a combination of the server's hostname, port, and authentication realm.

When the client receives an authentication challenge, it first looks for the appropriate credentials in the user's disk cache. If seemingly suitable credentials are not present, or if the cached credentials ultimately fail to authenticate, the client will, by default, fall back to prompting the user for the necessary information.

The security-conscious reader will suspect immediately that there is reason for concern here. "Caching passwords on disk? That's terrible! You should never do that!"

The Subversion developers recognize the legitimacy of such concerns, and so Subversion works with available mechanisms provided by the operating system and environment to try to minimize the risk of leaking this information. Here's a breakdown of what this means for users on the most common platforms:

- On Windows, the Subversion client stores passwords in the %APPDATA%/Subversion/auth/ directory. On Windows 2000 and later, the standard Windows cryptography services are used to encrypt the password on disk. Because the encryption key is managed by Windows and is tied to the user's own login credentials, only the user can decrypt the cached password. (Note that if the user's Windows account password is reset by an administrator, all of the cached passwords become undecipherable. The Subversion client will behave as though they don't exist, prompting for passwords when required.)

- Similarly, on Mac OS X, the Subversion client stores all repository passwords in the login keyring (managed by the Keychain service), which is protected by the user's account password. User preference settings can impose additional policies, such as requiring that the user's account password be entered each time the Subversion password is used.

- For other Unix-like operating systems, no standard "keychain" services exist. However, the Subversion client knows how to store password securely using the

"GNOME Keyring" and "KDE Wallet" services. Also, before storing unencrypted passwords in the /.subversion/auth/ caching area, the Subversion client will ask the user for permission to do so. Note that the auth/ caching area is still permission-protected so that only the user (owner) can read data from it, not the world at large. The operating system's own file permissions protect the passwords from other non-administrative users on the same system, provided they have no direct physical access to the storage media of the home directory, or backups thereof.

# Chapter 3

# Analysis of GPG-agent System

## 3.1 GnuPG

GNU Privacy Guard (GnuPG or GPG) is a free software alternative to the PGP suite of cryptographic software. GnuPG is compliant with RFC 4880, which is the current IETF standards track specification of OpenPGP. Current versions of PGP (and Veridis' Filecrypt) are interoperable with GnuPG and other OpenPGP-compliant systems.

GPG is a part of the Free Software Foundation's GNU software project, and has received major funding from the German government. It is released under the terms of version 3 of the GNU General Public License.

GnuPG uses public-key cryptography so that users may communicate securely. In a public-key system, each user has a pair of keys consisting of a private key and a public key. A users private key is kept secret; it need never be revealed. The public key may be given to anyone with whom the user wants to communicate. GnuPG uses a somewhat more sophisticated scheme in which a user has a primary keypair and then zero or more additional subordinate keypairs. The primary and subordinate keypairs are bundled to facilitate key management and the bundle can often be considered simply as one keypair.

## 3.2 GPG-agent

Though gpg-agent has GPG ie., GNU Privacy Guard in its name, as far as this dissertation is concerned we are not using all the features of GPG, but we use the gpg-agent which is a tool developed as a part of gnupg development for caching passphrases/-passwords in a secure way.

gpg, the GNU Privacy Guard, provides a means for secure encryption and signing of all kinds of data, such as email, software distributions, or Debian packages. gnupg-agent safely stores your passphrase for use by gpg, giving you the convenience of not entering a passphrase frequently without the insecurity of a passphraseless key.

### 3.2.1 GPG-agent feature

When using gpg, you have a public key distributable to anyone, and a private key that you must keep secret and secure. To help protect your private key, gpg lets you set a passphrase, which gpg uses to encrypt the key. Every time you work with gpg, it decrypts your private key with your passphrase, and keeps it in memory only as long as necessary to perform the requested operations. However, if you encrypt or sign data often, you may find it a hassle to keep entering your passphrase, and may become tempted to make the passphrase less secure or even remove it entirely, making your key vulnerable to anyone who manages to get access to your files. Rather than succumbing to this temptation, we can use gnupg-agent to securely and conveniently store our passphrase. We are interested in this feature of gpg-agent where we store passphrases/passwords in a secure way in order to cache passwords in Subversion client.

### 3.2.2 GPG-agent package

The gnupg-agent package provides a daemon gpg-agent, designed to run as part of your login session. To make initial setup trivial, the gnupg-agent package includes an X11 startup script /etc/X11/Xsession.d/90gpg-agent, which automatically starts gpg-agent as part of any X session, and sets the appropriate environment variables so gpg knows about the running gpg-agent. If you don't use X, and want gpg-agent available in a console session, just

```
eval \$(gpg-agent)
```

in your shell startup script. Then, restart your session, and you should have gpg-agent running and the environment variable $GPG_AGENT_INFO set.

Applications which have support for gpg-agent will now automatically use the passphrase from gpg-agent if available and not timed out. However, you still need a way to enter the passphrase when gpg-agent does not already have it. To make this easier, install one of the pinentry programs, such as pinentry-gtk2 or pinentry-qt, and gpg will automatically use it to prompt for your passphrase when needed.

Most programs that invoke gpg to perform encryption or signing operations should continue to work with gpg-agent. You can ignore or turn off any passphrase-caching mechanisms in the programs themselves, in favor of gpg-agent. In some cases, however, you may need to explicitly tell the program to work with gpg-agent. For example, with the Enigmail extension to Thunderbird Icedove (highly recommended), you need to open OpenPGP-¿Preferences, go to the Advanced tab, and check "Use gpg-agent for passphrase handling".

Note: that neither passphrases nor gpg-agent make your private key secure to leave on a box where other people have root access, nor do they mean that you can assume your private key remains safe after a break-in.

Many applications use gpg-agent in order to cache passphrases in a secure way. The GNOME desktop environment provides an implementation called Seahorse which in turn implements gpg-agent and ssh-agent functionalities. The ssh-agent is a similar tool which helps to cache ssh passphrases in a secure way, but it is very well tied with ssh tools and does not help to come up with a generic solution, which is the reason why we have chosen gpg-agent for this dissertation.

### 3.2.3 GPG-agent dependencies

gpg-agent uses different libraries in a *NIX system in order to encrypt the passphrases/passwords which are given to it. The following is the list of libraries which gpg-agent is dependent on in a typical *NIX system,

- libgcrypt
- libpth
- libnsl
- libgpg-error
- libc

With the help of above libraries gpg-agent operates in a *NIX system. Of course, gpg-agent runs in Windows too, where we have same kind of dependencies in order to run gpg-agent. Let us have brief overview of the libraries listed above.

### libgcrypt

libgcrypt is a general purpose cryptographic library based on the code from GnuPG. It provides functions for all cryptograhic building blocks: symmetric ciphers (AES, DES, Blowfish, CAST5, Twofish, Arcfour), hash algorithms (MD4, MD5, RIPE-MD160, SHA-1, TIGER-192), MACs (HMAC for all hash algorithms), public key algorithms (RSA, ElGamal, DSA), large integer functions, random numbers and a lot of supporting functions.

### libpth

Pth is a very portable POSIX/ANSI-C based library for Unix platforms which provides non-preemptive priority-based scheduling for multiple threads of execution (aka "multi-threading") inside event-driven applications. All threads run in the same address space of the server application, but each thread has it's own individual program-counter, run-time stack, signal mask and errno variable.

The thread scheduling itself is done in a cooperative way, i.e., the threads are managed by a priority- and event-based non-preemptive scheduler. The intention is that this way one can achieve better portability and run-time performance than with preemptive scheduling. The event facility allows threads to wait until various types of events occur, including pending I/O on filedescriptors, asynchronous signals, elapsed timers, pending I/O on message ports, thread and process termination, and even customized callback functions.

Additionally Pth provides an optional emulation API for POSIX.1c threads ("Pthreads") which can be used for backward compatibility to existing multithreaded applications.

### libnsl

Functions in libnsl library provide routines that provide a transport-level interface to networking services for applications, facilities for machine-independent data representation, a remote procedure call mechanism, and other networking services useful for application programs.

### libgpg-error

Libgpg-error is a small library with error codes and descriptions shared by most GnuPG related software.

### libc

Any Unix-like operating system needs a C library: the library which defines the "system calls" and other basic facilities such as open, malloc, printf, exit...

The GNU C library is used as the C library in the GNU system and most systems with the Linux kernel.

The GNU C library is primarily designed to be a portable and high performance C library.

It follows all relevant standards (ISO C 99, POSIX.1c, POSIX.1j, POSIX.1d, Unix98, Single Unix Specification). It is also internationalized and has one of the most complete internationalization interfaces known.

# Chapter 4

# Implementation

Subversion has a layered architecture where the APIs specific to functionalities and features are organized. For implementing the new password cache for Subversion we are using the following code,

- subversion/libsvn_subr/cmdline.c
- subversion/libsvn_subr/simple_providers.c
- subversion/libsvn_subr/username_providers.c
- subversion/libsvn_subr/auth.c
- subversion/libsvn_auth_gpg_agent/version.c
- subversion/libsvn_auth_gpg_agent/gpg_agent.c

The code responsible for adding gpg-agent support is given as a separate library in subversion with the help of libsvn_auth_gpg_agent.

## 4.1 Assuan Protocol

In an ideal world, Assuan is irrelevant. Assuan's primary use is to allow a client to interact with a non-persistent server. Using Assuan, this is accomplished by forking a subprocess and communicating with it via, for example, a pipe or unix domain socket. This method is neither elegant nor efficient especially when there is a lot of data spread across several transactions: not only is there a penalty for an increased number of context switches, but also a significant amount of data is memcpy-ed from

the client to a file descriptor and from the file descriptor to the server. Despite these and other disadvantages, this type of client/server communication can be useful: the client is completely separate from the server; they are in different address spaces. This is especially important in situations where the server must have a known degree of reliability and data must be protected: as the Assuan protocol is well defined and clients cannot corrupt the servers' address space, auditing become much easier.

Assuan was developed for use by the GNU Privacy Guard, GnuPG, to prevent potentially buggy clients from unwittingly corrupting sensitive transactions or compromising data such as a secret key. Assuan permits the servers, which do the actual work, e.g. encryption and decryption of data using a secret key, to be developed independently of the user interfaces, e.g. mail clients and other encryption front ends. Like a shared library, the interface is well defined and any number of front ends can use it; however, unlike a shared library, the client cannot see or touch the server's data. As with any modular system, Assuan helps keep the servers small and understandable help to make code more understandable and less error prone.

Assuan is not, however, limited to use with GnuPG servers and clients: it was design to be flexible enough to meet the demands of almost any transaction based environment with non-persistent servers.

The gpg-agent should be started by the login shell and set an environment variable to tell clients about the socket to be used. Clients should deny to access an agent with a socket name which does not match its own configuration. An application may choose to start an instance of the gpgagent if it does not figure that any has been started; it should not do this if a gpg-agent is running but not usable. Because gpg-agent can only be used in background mode, no special command line option is required to activate the use of the protocol.

gpg-agent uses the Assuan protocol server internally which is run as a daemon. In order to communicate with the Assuan based server we need to issue commands using an UNIX socket.

## 4.2 GPG-agent password cache

In order to create a simple password cache provider in subversion we need to implement the following 3 functions which forms a part of svn_auth_get_*_simple_provider() routine.

- simple_*_first_creds (mandatory)

- simple_*_next_creds (optional)
- simple_*_save_creds (mandatory)

In our implementation we have implemented the above using simple_gpg_agent_first_creds and simple_gpg_agent_save_creds in libsvn_auth_gpg_agent/gpg_agent.c

In gpg-agent implementation, we specifically do not require a save credentials function explicitly, since, gpg-agent has inbuilt support for caching new credentials when it encounters it for the first time. Hence in this case we are just returning with TRUE for save credentials.

### 4.2.1 simple_gpg_agent_first_credentials

This function forms the core of this implementation where a helper function called password_get_gpg_agent has been implemented. This is the function responsible and much of the code related to this feature lies in this function.

In password_get_gpg_agent() the following is the sequence of operations,

- Identify and store environment variables such as $GPG_AGENT_INFO, $GPG_TTY, $TERM
- Open a UNIX socket to the gpg-agent daemon.
- Pass 'OPTION' command to tell the gpg-agent daemon who we are and check the return stream of data from the daemon.
- Based on Subversion's interactive mode, pass 'GET_PASSPHRASE' option to obtain the stored password or set the new password in the gpg-agent's secure store.
- Once the password is retrieved from the gpg-agent's store we can then pass it on to the other layers of Subversion's API where the password is required.

Thus using the above sequence of operations the gpg-agent password store for subversion is implemented using the C language. The entire development of this was carried out as a separate branch in Apache's Subversion repository which is available at [1]

[1] http://svn.apache.org/repos/asf/subversion/branches/gpg-agent-password-store/

The patch which gives this feature to Subversion which was committed to the branch could be found in [7] This patch has been accepted and actively reviewed by the dynamic Subversion Open Source Community developers.

# Chapter 5

# Sample Run

Following screenshots show a sample run of the gpg-agent based password caching in Subversion using a Debian based GNU/Linux distribution.

## 5.1 GPG-agent pinentry dialog

When any of the subversion operation requires a password, a pinentry dialog opens up and asks for the password, which is shown in figure 5.1 for 'svn ls' command.

## 5.2 Subversion listing

Once the correct password is given and the password gets cached in gpg-agent the list of directories is shown in the command line which is shown in figure 5.2.

## 5.3 Password cache

Figure 5.3 shows the password cache ie., " /.subversion/auth/svn.simple" which mentions the password store as gpg-agent. We can note that the password is not shown here, only the username is available since the password is cached in a secure store ie., gpg-agent.

Figure 5.1: Password pinentry dialog

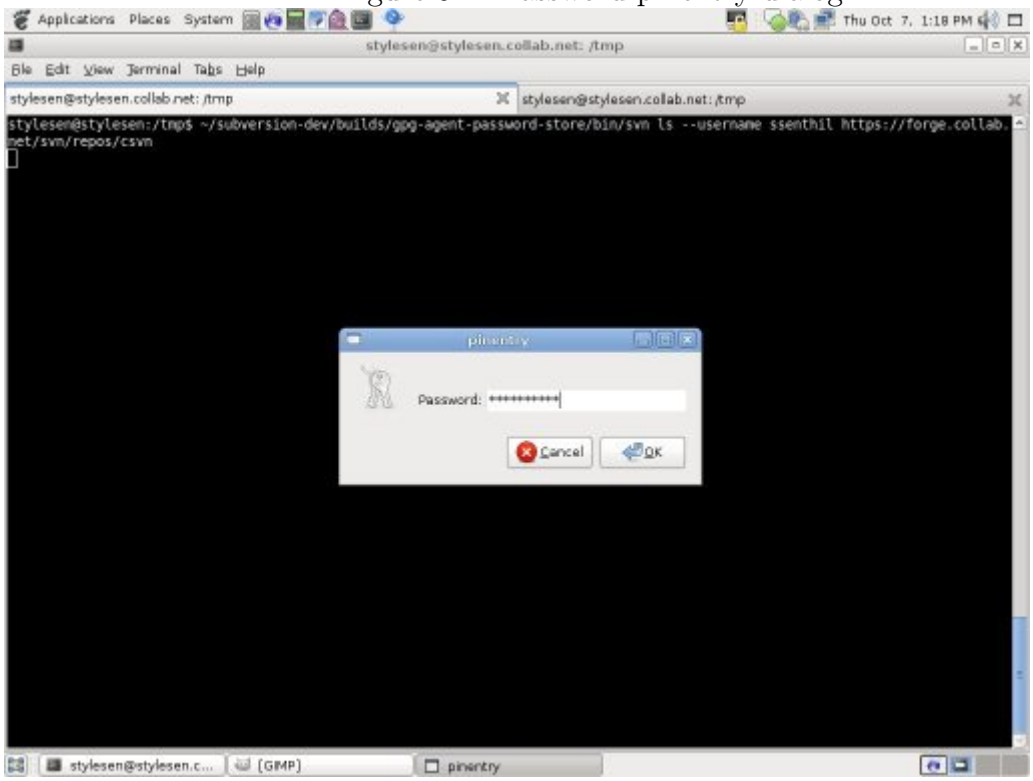



Figure 5.2: Directory listing shown after getting password

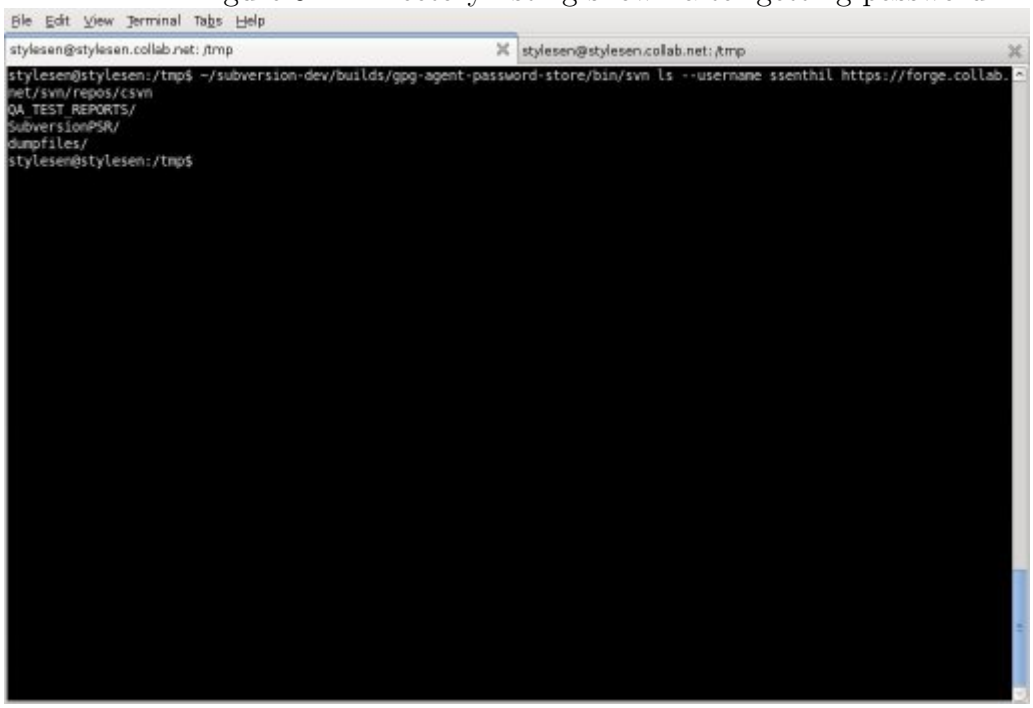
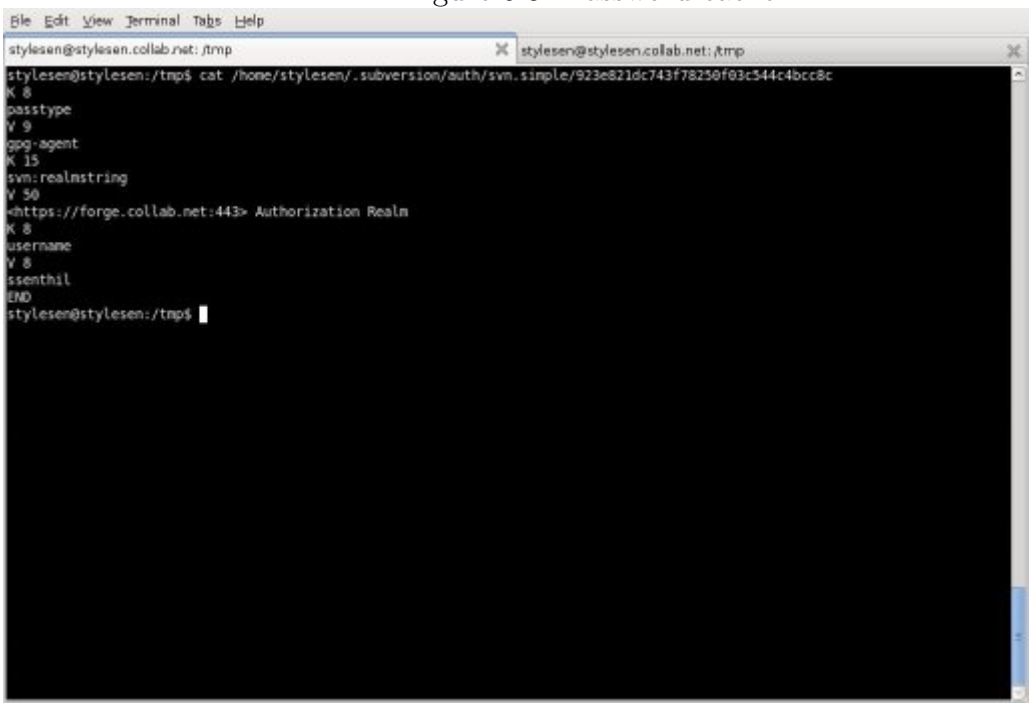


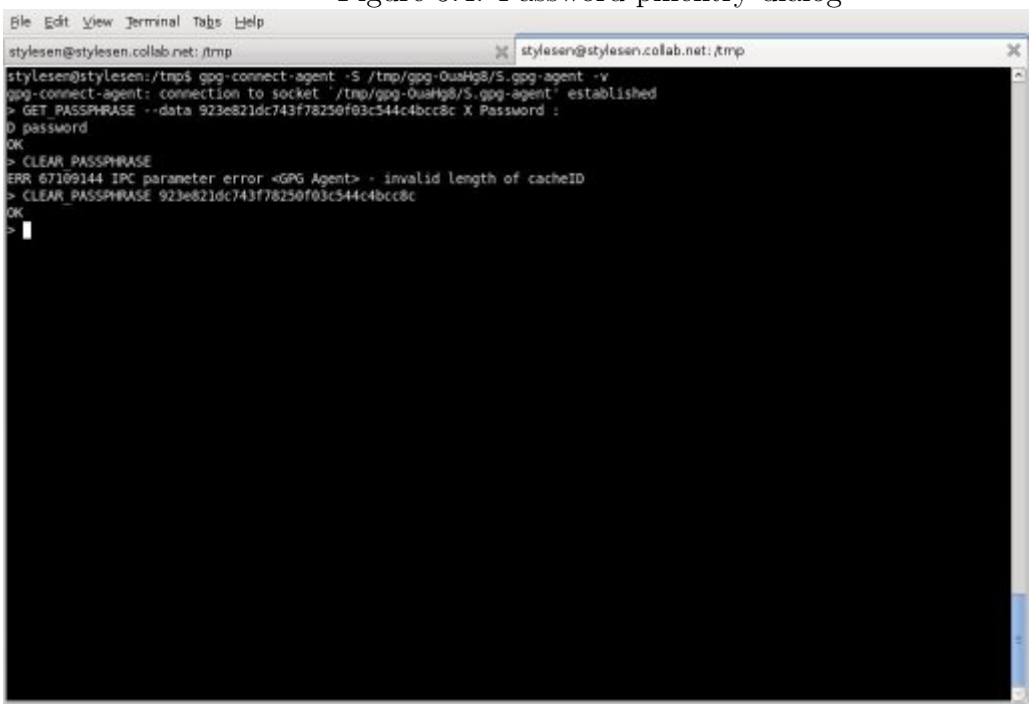
Figure 5.3: Password cache




## 5.4 gpg-connect-agent

gpg-connect-agent is a tool which allows us to examine the gpg-agent secure password store which is shown in the figure 5.4. We can see the cached password in the store, though it is shown in plaintext, it needs special privileges by the user in order to view this. We can see that '923e821dc743f78250f03c544c4bcc8c' matches with the previous screen shown in figure 5.3, which is nothing but the md5sum hash of the repository realm name which we use as the cache id to store passwords and we can map it with the Subversion password cache.

Figure 5.4: Password pinentry dialog



# Chapter 6

# Conclusion

This dissertation implements a gpg-agent based password cache for subversion clients. GPG-agent is generic enough that it could be used as a secure password store for any kind of application. This provides a platform independent solution for the problem of password caching, since it works on all *NIX and also windows systems.

Further work could try to use Assuan APIs and the library itself in order to connect to the gpg-agent daemon. Right now the implementation just uses the Assuan protocol and not the APIs. The connection to server is done with the help of raw unix socket functions.

# Bibliography


[1] Apache portable runtime project. http://apr.apache.org/.

[2] Apache software foundation homepage. http://www.apache.org.

[3] Arch project homepage. http://www.gnu.org/software/gnu-arch/.

[4] Collabnet inc. http://www.collab.net.

[5] Cvs project homepage. http://www.nongnu.org/cvs/.

[6] Git project homepage. http://git-scm.com/.

[7] Implementation patch of gpg-agent password store. http://svn.apache.org/viewvc?view=revision&revision=1005065.

[8] Mercurial project homepage. http://mercurial.selenic.com/.

[9] Monotone project homepage. http://www.monotone.ca/.

[10] Perforce project homepage. http://www.perforce.com.

[11] Purecm homepage. http://www.purecm.com/.

[12] Tigris homepage. http://www.tigris.org.

[13] C. Michael Pilato Ben Collins-Sussman, Brian W. Fitzpatrick. *Version Control with Subversion: For Subversion 1.6*. O'Reilly Media, r3784 edition, 2009.

[14] Compaq. Vesta project homepage. http://www.vestasys.org/.

[15] Software Freedom Conservancy. Darc project homepage. http://darcs.net/.

[16] Shlomi Fish. Available vcs alternatives. http://better-scm.berlios.de/alternatives/.

[17] Canonical Limited. Bazaar project homepage. http://bazaar.canonical.com/en/.

[18] Peter Miller. Aegis project homepage. http://aegis.sourceforge.net.

[19] David A. Wheeler. Comments on open source software. http://www.dwheeler.com/essays/scm.html.