ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

A Software-Based Trusted Platform Module

Master Thesis

Domenic Schröder

Department of Computer Science
ETH Zurich

Supervisors:
Prof. Bernhard Plattner
Mario Strasser

Abstract

The Trusted Computing Group has specified different commands for a Trusted Computing Module (TPM). The latest specification is version 1.2 and there are some new functionalities. One of them is the delegation function.

Since 2004 there exists a TPM-Emulator which is in development. This thesis is about the further development of this TPM-Emulator, especially the delegation function and design a platform independent build and test framework for the TPM-Emulator.

Contents

1 Introduction

1.1 Acknowledgements

I would like to thank my supervisor Mario Strasser for supporting me in this master thesis. He helped a lot with competent answers to my questions. Also I want to thank Professor Bernhard Plattner that we could do this master thesis in his group and with his helpful suggestions.

1.2 Overview

In the area of trusted computing the Trusted Computing Group (TCG) has specified Trusted Platform Module (TPM). There are a lot of different opinions about the usability or danger of this TPM. Nevertheless there is an increasing amount of TPMs in use in mobile phones and personal computers. And more and more there are virtual machines running with only one TPM in the host system. A TPM is not designed to interact with multiple machines. For all this reason the TPM-Emulator project was initiated and developed further. This Emulator allows an easily usable TPM without the whole issues described above for testing and experimenting. This entirely in software running emulator is at the beginning of this thesis a linux application, which is used in different labs and universities.

The goal of this thesis is to contribute to the TPM-Emulator project by implementing the user management of the recent TPM specification 1.2 and create a platform independent build and test framework.

1.3 Chapter overview

In the first chapter we will introduce the Trusted Computing Group and their specification of a Trusted Platform Module(TPM). We will give an overview of the functionality of a TPM. However since there are already very good resources available about TPM and their inside and functionality we just summarize the most important facts needed for understanding the implementation details. As a beginning we present the currently existing TPM-Emulator. After that the delegation functionality will be explained in detail in Section 4, afterwards the design and implementation details of this user- and right management system. In Section 5 we will talk about the portability to other operating systems like Microsoft Windows and what the problems of platform independents applications are and how we did overcome these difficulties. In Section 6 we discuss the test framework, the implemented test and future possibilities for testing. At last we propose consecutively work.

2 Trusted Platform Modules

A Trusted Platform Module (TPM) consists of mainly three subsystems:

- the Root of Trust for Measurement,
- the Root of Trust for Storage and
- the Root of Trust for Reporting.

We will explain the functionality and the use of a TPM. We are aware that a TPM can be used for much more things than only in connection with an operation system in a PC, but for that we refer to a lot of good books about TPM. For this thesis we look explicitly on computers, because our TPM-Emulator is intended to run there. To get an overview we introduce the Trusted Computing Group and with that Trusted Computing. That will give the necessary information to look into more details to the TPM and the emulation of this module in C-code.

2.1 Trusted Computing Group

The Trusted Computing Groups (TCG)[1] is a international industry standards group. They publish and maintain the specification for trusted computing, among others the one for Trusted Platform Module (TPM).

The beginning of the TCG was in 1999. But it was not yet TCG, but Compaq, HP, IBM, Intel and Microsoft founded the Trusted Computing Platform Alliance (TCPA), the predecessor of the TCG. Quickly 200 other members joined this group. As this group had such a rapid growth in members the structure for decision taking did not fit any more.
In 2003 the Trusted Computing Group with different structures has been founded which continues the goals of the TCPA.
The description of them self is the following:

> The Trusted Computing Group (TCG) is a not-for-profit organization formed to develop, define and promote open, vendor-neutral, industry standards for trusted computing building blocks and software interfaces across multiple platforms. [1]

TPMs exists from different vendors in a lot of computers (IBM Thinkpad, Mac, ...) and are also integrated in mobile phones. For example Windows

[1] `http://www.trustedcomputinggroup.org/home`

Vista and its application uses the TPM to de- and encrypt the hard drive, BitLocker.

There are a lot of rumors about this module and his functionality and the intention of this group. We will not talk about if a TPM is “good” (whatever that means in this context) but about the technology used. For what this technology is used is over the scope of this thesis.

2.2 Technical overview

A TPM is passive in the sense that the operation system has to call the TPM and the TPM sends a response. As you can see in the Figure 1 some entity sends the TPM a header, a data part and an authentication of header and data. The TPM checks this and responds with the specified answer and authenticates this answer.

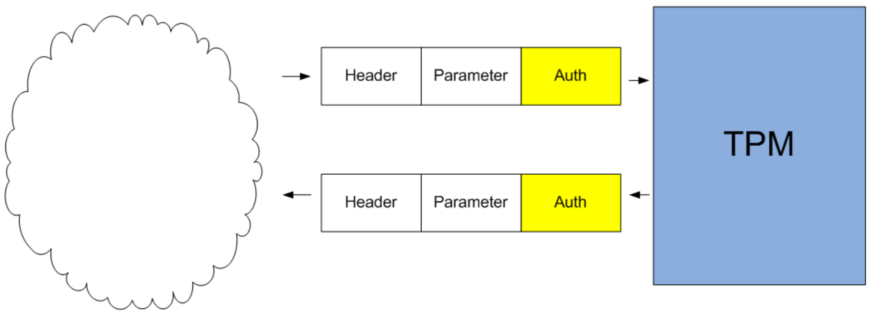


Figure 1: The communication of the TPM.

Despite all rumors a TPM cannot interrupt a running system or program and is not intended to do so. It is a cryptographic device, which is used as a root of trust for measurement, storage and reporting. It therefore can be used to secure application such that they only run if the system is in specific state.

In the following we will explain this citation:

> The TPM is a microcontroller that stores keys, passwords and digital certificates.[2]

The main components (see Figure 2) of a TPM are:

[2] `http://www.trustedcomputinggroup.org/developers/trusted_platform_module/faq`

- Cryptographic Co-Processor
- HMAC Engine
- SHA-1 Engine
- Opt-In
- Non-Volatile Memory
- Key Generation
- Random number generator
- Power detection
- Execution Engine
- Volatile Memory

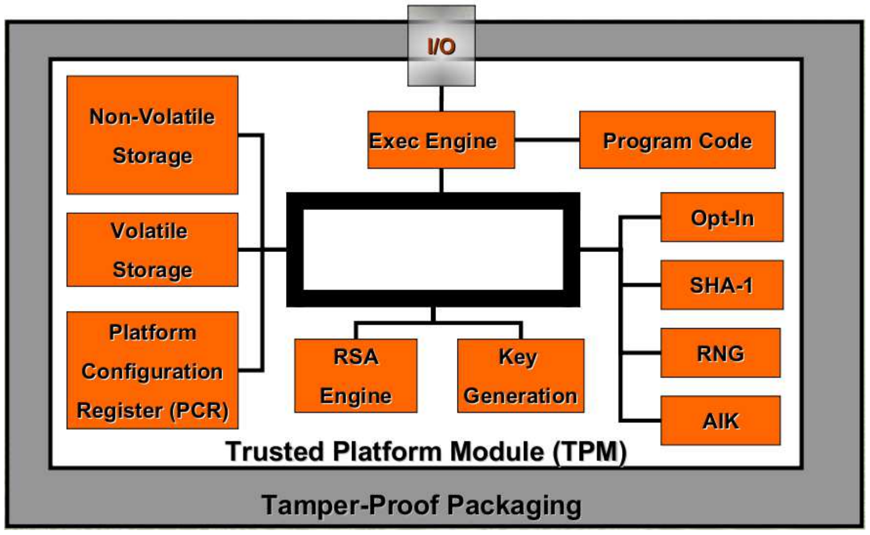


Figure 2: The components a TPM consists of.

A TPM is not designated to enhance the cryptographic of some other applications, but it has the capability to do cryptographic functions in order to be independent to create keys, calculates hashes and so on.

One of the enhancement a TPM has that it is independent of the machine. This means, that if the machine is corrupted, the TPM is not. Because of that, keys, passwords and digital certificates are saved in a secure environment, even when the machine is not. Important is, that the TPM does not give the passwords or keys out to a corrupted system. This is done with measuring and reporting and storing this values in the Platform Configuration Registers as described in the next chapter. The TPM can report its state with signing the data with its public key. By evaluating this data and the hash a remote entity can check in which state the machine is.

PCR

Platform Configuration Registers (PCR) are an important tool of the TPM. When something is measured on the machine, then it is reported to the TPM which stores this information in the PCR. There are at least 16 PCR. By getting new information, the PCR are not overwritten, but the old value will be hashed with the new value and afterwards safed in the PCR. With this method every change cannot be reverted.

A TPM uses this PCR to seal storage or keys or commands in the delegation function to a certain state in which the machine has to be. Or more exactly which values has been reported to the TPM. We see how important it is to have a trusted root of measurement in the machine. But this is out of the scope of this thesis.

3 TPM-Emulator

3.1 History

The initial release was 2004 as a semester thesis by Mario Strasser at ETH. At this time the TPM-Emulator was a kernel module and fully running in the kernel. In 2005 Heiko Stamer joined the project and contributed the direct anonymous attestation (DAA). At the end of 2006 the whole design was changed and rewritten as a user-space daemon (TPMD) which is the actual emulator and a kernel module (TPMD_DEV) that provides the character device `/dev/tpm` for low-level compatibility with TPM device drivers.[10]
For accessing the TPM-Emulator TCG has specified[17] a device driver library. This usual way for accessing has been written for linux. In 2007 an install-tool based on Makefiles has been done. In 2008 the support for OpenBSD has been added. This is only the kernel module, as all other things run as well in linux as in OpenBSD. In the year several bug-fixes has been made and missing functionalities has been added.

3.2 Technical implementation

The TPM-Emulator is parted in three parts (Figure 3). The first one is the TPM-Emulator engine. This engine is the core of the TPM-Emulator. The public API consists of only 3 commands:

1. `tpm_emulator_init()`
2. `tpm_handle_command()`
3. `tpm_emulator_shutdown()`

The init command initializes the emulator. The handle command can be used to execute any of the TPM commands. So this is the main command to execute the commands which are given as numbers, as ordinals with all the parameters in one byte array. Afterwards the emulator_shutdown shuts down the emulator.

The cryptographic engine is a part which is only called by the TPM-Emulator engine. For this reason it is easy to exchange the actual cryptographic functions with others if necessary. The TPM-Emulator engine does the unmarshalling and the decoding to afterwards execute the commands and return the respond correctly marshalled and encoded. The key, data and state storage are also inside this engine.

The TPM-Emulator Daemon (TPMD) is the connection to the operation system where the functions to communicate, log and interact with the operation system is done. This part opens in linux a Unix domain socket to interact with other processes and waits until an application connects and sends the data to the TPM-Emulator engine. This part is also responsible to save non-volatile data into a file on the hard drive.

The TDDL is the Trusted Computing Group Device Driver Library and is specified as the usual way to interact with any TPM as you can see in Figure 3. This TDDL is programmed as a usual device driver library.

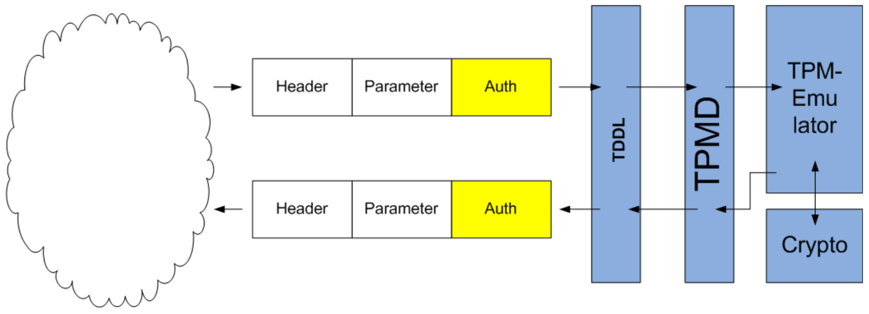


Figure 3: Overview of the TPM emulator package.

In short we can say that at the beginning of this master thesis the TPM-Emulator runs natively in linux and in Free- and Open-BSD. The TPMD is the only platform dependent part of the TPM. The building process is done with GNU Makefiles. For the cryptographic part the GNU Multiple Precision Arithmetic Library (GMP) is needed. This library is usually used in Linux.

4 Delegation

Authentication

> The process of providing proof of claimed ownership of an object or a subject's claimed identity. [3]

Authorization:

> Granting a subject appropriate access to an object. [4]

These two principles are mixed together in a TPM. The principle of authorization and authentication is implemented as easy as possible in a TPM. The only mechanism is a 160-bit value, the AuthValue. Everyone who can send this value is authenticated and also authorized to use the commands of a TPM (if they are available in the current state).
With this mechanism every process/user has to know this value to interact with the TPM. This violates the principle of least privileges and in TPM this shortcoming is addressed with the delegation model. This is new in the specification of the TPM 1.2

The Figure 4 gives an overview. First only the owner knows a value to connect in an authorized way to the TPM. He can create with his privileges different privileges to other entities.

4.1 Requirements Design

The basic requirements from the specification [3] are the following:

1. Consumer does not have to know and remember the TPM owner password.
2. Role based administration and separation of duty.
3. TPM should support multiple trusted processes.
4. Trusted processes may require restrictions.
5. Maintain the current authorization structure and protocols.

In the following we will look on these specifications and explain them:

[3] TPM Main - Part 1 Design Principles, V 1.2, p. 33
[4] TPM Main - Part 1 Design Principles, V 1.2, p. 33

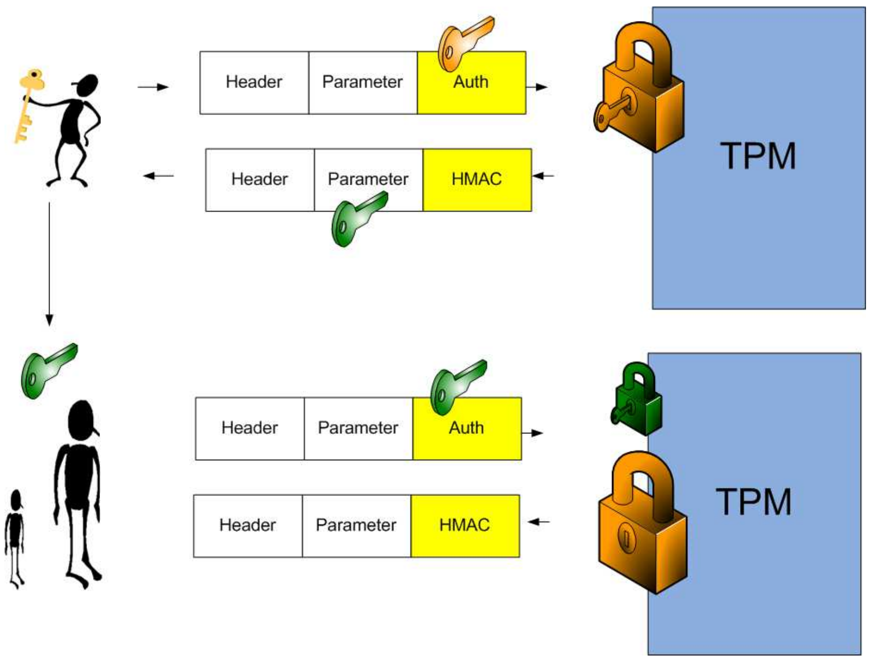


Figure 4: Owner can give privileges to any entity.

Not everyone is owner It es convenient to give a consumer not full control of the TPM, but only of the parts he needs. With that delegation model it is possible give away specific rights with just adding new delegations.

Role based Without giving full control in someones hand it is possible to give as much delegation as needed, but not more and revoke it at any time necessary. The roles are in TCG language so called families and the authorisations are called delegations.

Multiple trusted processes As it is possible through the principles of PCR-restrictions (Section 2.2) to separate the secrets of two trusted processes which do not run at the same time the TPM supports this with the delegation properties. Before this delegation model a trusted process needed the TPM owner AuthValue and with that gains knowledge of every data inside the TPM. That is because the owner has every right.

Least privilege The principle of least privilege: A process or entity has only access to the functions that it requires to complete a specific task. Every ordinal, this means every command can be delegated to any family and in connection with any PCR state.

Maintaining With this delegation model the existing functions, especially all the already existing authorisation protocols (OSAP, OIAP), will work without any changes. There are only new commands and functions which we will describe in Section 4.3.

4.2 Requirements Implementation

1. Unlimited delegations
2. Revoking any delegation without side effects
3. Delegations inserting possible by OEM
4. Delegations inserting possible without TPM owner

Unlimited delegations We want to have unlimited delegations. As the TPM has not unlimited space, there must be storage outside of the TPM. But it should not be changed outside of the TPM and old delegations should not be valid.

The solution is that an entity can create a delegation which is stored outside of the TPM and the entity inserts this delegation when it is needed. This raises different problems with verification and revocation which are solved with the verification count as described in Section 4.4.

Revoking An owner must have anytime the possibility to revoke any delegation. It must be, that this revocation does not affect other delegation. Old delegations must not be valid anymore. The owner should have the possibility to revoke any family of delegation. This must also revoke or invalidate any delegations which are stored outside of the TPM.

Delegation inserting It is convenient if it is possible to insert delegation even without an owner. That allows OEM to do the seeding of the table even during manufacturing. So before the TPM is turned on and an owner takes ownership, the delegation can be inserted by anyone.

Attacks that attempt to burnout the TPM's storage by writing many delegations are prohibited by only allowing a limited number of writes before an owner takes ownership.

4.3 Implementation

The previous requirements are fulfilled with the following implementation of two tables. One is the so-called "Family Table" and the other is the "Delegation Table". We will give an overview of this table and explain afterwards the functionalities of these parts.
The family table is a limited table. The limit is set by the manufacturer and can be interrogated by the function: `Get_capabilites`. A row of the family table consists of a family ID, the row label, family verification count and the family flags.

The delegation table consists of a label, a family ID, a verification count and the delegated capabilities. All of these entries are not sensitive. We will go into detail later. To enable unlimited delegations there is the possibility to create a so-called "blob", which consists of a delegation row which is encrypted by a key of the TPM only known to the TPM.

The owner of a TPM can delegate some of his rights to another entity. The owner of the platform can delegate commands with another 160-bit secret value.

4.4 Technical implementation delegation

First we illustrate the concept of the delegation implementation and go into the details of the implementation.

In the family table there are the following entries:

- tag
- family label
- verification count
- flags

In the delegation table there are the following entries:

- tag

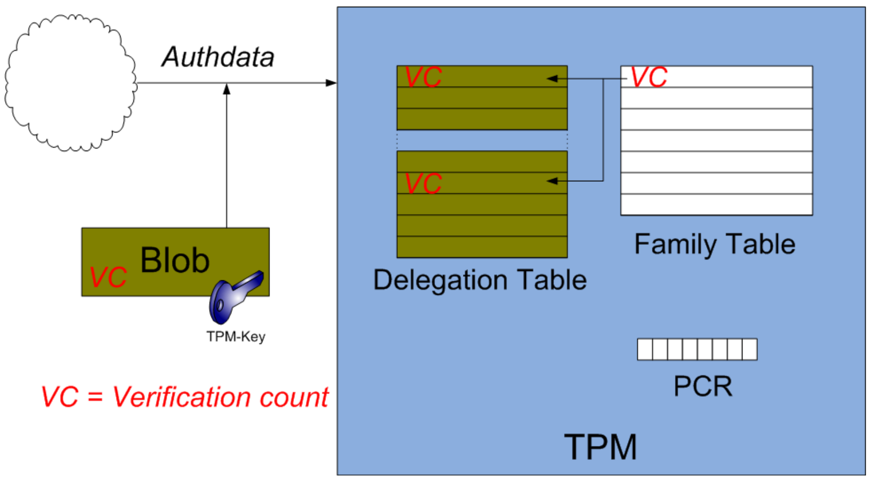


Figure 5: The delegation implementation as family and delegation table with the PCR information

- row label
- PCR Info
- permissions
- family ID
- verification count
- AuthValue

So every delegation belongs to a family. In the delegation is saved the Auth value, you can say the password, and also the PCR information, which means in which state the machine has to be to use this delegation.

We will show it on two examples. The first one is an entity which connects via the network with an AuthValue and a blob and the second one is an application which uses a key from the TPM. The commands are in Section 4.4 explained in more detail. We will mention them in the examples.

Network Connection First the owner creates a new family row with the ID 1 and creates for this family a delegation for the `TPM_Example1` command with e.g. the random AuthValue 0x010101..01. So the first command is the `TPM_Delegate_Manage` to insert a new family and to create the owner delegation the `TPM_CreateOwnerDelegation` command. This command returns a blob which the owner gives with the AuthValue to the entity. This entity can load the blob into the TPM with `TPM_LoadOwnerDelegation` and the TPM stores it in the delegation table. In our example it uses the blob directly with the AuthValue with the `TPM_DSAP` to get an authorized session. The TPM checks the blob. First if it is decrypted with the TPM key. Then it looks up the family in the blob, checks the verification count of the blob with the one in the family table. If this is all correct it is created a shared secret with the AuthValue with the entity. With this shared secret they will create a shared session to use the command. There it will be checked if the entity is authorized to use *this* command. The owner can revoke this delegation e.g. by disabling or deleting this family in the family table or just create a new delegation and increment the verification count.

Local application First the owner creates a new family row with the ID 2 (`TPM_Delegate_Manage`) and creates for this family with the command `TPM_CreateKeyDelegation` a new key delegation. To follow the requirements suggestion the AuthValue will be 0x111..11. This is because the application has no means to store this AuthValue in a secure way. The corrupted operation system and other corrupted applications will always find a way to get this AuthValue. But this time we add a PCR definition. This means that the machine has to been in a certain state where this process is running (e.g. a new booted operation system and the process to encrypt the hard drive is running). The `TPM_CreateKeyDelegation` returns a blob which the owner stores with the `TPM_LoadOwnerDelegation` in the TPM inside the delegation table. Furthermore the process can connect to `TPM_DSAP`, create an authorized session and will get the key if the machine is in the state saved in the PCR.

4.4.1 The commands

In the following we will describe the main functions we implemented to add the delegation functions to the TPM-Emulator. These commands are described in the specification[5] and we implemented them in C code.

TPM_Delegate_Manage()

This is the most fundamental function for the whole delegation functionality. The parameter opCode of this function decides if

- the function **enables/disables** the use of a family

- the function **locks** the tables until a owner is installed
- the function **creates** a new family
- or the function **invalidates** an existing family.

Enabling or disabling a family also enables respectively disables all the delegations belonging to this family.

TPM_Delegate_CreateKeyDelegation()

This command is to create a privilege to use a key. The output is a blob. This function does not test if the key shall be used with these permissions. This will be done if this blob is used calling `TPM_DSAP`.

TPM_Delegate_CreateOwnerDelegation()

With this command the owner can delegate the use of commands. The function will create a blob, which can be used afterwards to create a authorized session with `TPM_DSAP` or to load it in `TPM_Delegate_LoadOwnerDelegation()`. It is possible to increment the verification count. This would mean, that every other delegation of this family will be invalid and only this delegation can be used. This new delegation can be void, so every delegation is invalid.

TPM_Delegate_LoadOwnerDelegation()

This command takes a blob and saves the content internally in the delegation table. This command can be executed before an owner is installed.

TPM_Delegate_ReadTable()

This command is special on the first glance because there is no authorisation needed to execute this command. This command is used to verify the tables. It outputs the public part (e.g. not the AuthValue) of the family and delegation table.

TPM_Delegate_UpdateVerification()

This command is to update the verification count either in a blob or in the intern delegation table. To change a blob the old blob has to be given as a parameter and a new blob will be returned. For the intern delegation table only the index has to be given. Of course it will be checked if this family is not revoked or disabled.

TPM_Delegate_VerifyDelegation()

This command returns if the given blob is valid or not.

TPM_DSAP()

This command is the supplement for the existing commands in TPM version 1.1 `TPM_OSAP` and `TPM_OIAP`. This command creates the authorization session handler. Not with a TPM owner token, but with a delegated AuthValue. This AuthValue corespondents either to a inserted blob or to a row in the intern delegation table. This function generates a shared secret. With this shared secret and the authorization session handler the following commands can be used from the entity with the delegated token in the same way as a `TPM_OSAP` session. As discussed in Section 2.2 the PCR values are checked together with the AuthValue.

5 Platform Independent

5.1 CMake

CMake[8] is a cross-platform, open-source build system. It takes the code and does not build the application(Figure 6, but generates native makefiles and workspaces that can be used in the build system, in the environment of the choice of the developer.

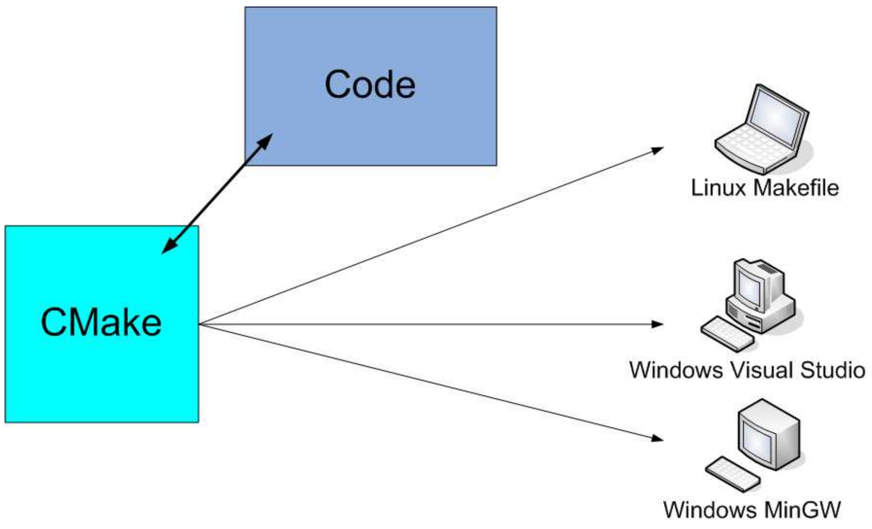


Figure 6: CMake takes the code and creates Make-/Projectfiles

5.1.1 The decision for CMake

At the start of this master thesis the TPM-Emulator has been build by Makefiles. They have been created and are now easy to use. It is possible to use Makefiles in Microsoft Windows by using for example Cygwin[6]. We asked ourself what requirements are needed and came to the following:

1. Must work in different operation systems.
2. Must be easy expendable to other operating systems.
3. The tool has be maintained for the next years.
4. It would be very nice to have possibilities to test.

CMake fulfills all of these requirements as described in the next chapter.

5.1.2 CMake introduction

The big advantage is that CMake generates native Makefiles and workspaces. So we can use the compiler environment of our choice. This leaves the decision of the compiler environment for future work on other operation systems to this developer.
The CMake application has at the moment the following binaries [5]

- Windows
- Mac OSX
- Linux i386
- SunOS Sparc
- IRIX64
- IRIX64 n32
- HUPX 9000
- AIX powerpc

This fulfills the first and the second part of the requirements very good as it is possible to compile it on other systems, too.

There are a lot of applications which uses CMake (e.g. Blender, Gammu, KDE, MySQL on Windows, MiKTeX). For example the developer of KDE switched from autotool to CMake. They have written a document why they did it and how it worked.[6] With this knowledge it is clear that CMake will be maintained for a long time.

In CMake there is a module CTest. In Section 6.2 we go into the details of CTest. This allows testing much better than we thought and the last requirement is coped.

CMake is capable of creating Makefiles respectively project files for:

- Makefiles (GNU, NMake, Borland, MinGW, Unix, etc)
- KDevelop, Eclipse

[5] `http://www.cmake.org/cmake/resources/software.html`
[6] `http://lwn.net/Articles/188693/`

- Visual Studio 6, 7, 8, 9 IDE
- Watcom WMake

In summary we see that with CMake the build process can be made much easier and CMake also supports to install files, to create packages and other pleasing functions. So we changed the Makefiles to CMake.

5.2 CMake implementation

We will introduce the main commands of CMake for this project. The commands in CMake are case insensitive and independent of space. We wrote the commands in uppercase letters with a space after the brackets. CMake uses usually a file called CMakeLists.txt. The following commands are in these files, or in subdirectories which are called by this first CMakeLists.txt.

PROJECT (TPM-Emulator)

This command sets the name of the project.

SET (Tpm_dir ${PROJECT_SOURCE_DIR}/tpm)

The "SET" command is used to define variables. We use them e.g. to set the used directories.

IF (UNIX) / IF (WIN32)

The variables unix, win32, apple and cygwin are set automatically by CMake. There are other variables which describe the system in which the build system is running.
We use this command to use the corresponding files for the different operation systems and to find special libraries if they are needed with the commands `SET ( System "Win32" )` and
`SET ( System_dir ${PROJECT_SOURCE_DIR}/windows )` . We wrote a CMake module to easy extend the `find_library` command (see Appendix)).

MESSAGE (STATUS "System is ${System}")

This command outputs a status message, which is just displayed. There are the following different states

- STATUS

- FATAL_ERROR
- SEND_ERROR

FIND_LIBRARY (Gmp_library NAMES gmp)

This command searches in the paths for libraries of the operation systems and on the ones inputed to CMake for gmp and saves the name in the variable "Gmp_library".

INCLUDE_DIRECTORIES (${Tpm_dir})

This adds the given directory to those searched by the compiler to include files.

FILE (GLOB Tpm "${Tpm_dir}/*.[h—c]")

This creates an array of all the header and code files in the chosen directory. The recommended way to import files is to write them in the CMakeLists.txt and not to import them with a regular expression. The reason for this is, that if a file is deleted or added the CMakeLists.txt is not changed and therefore is not built again. CMake adds the CMakeLists.txt as a target. So if the CMakeLists.txt is changed, the project respectively the Makefile will be rebuilt and the new files are added. We choose the command `FILE` because by implementing another operation system it is convenient to just add a whole directory with all its files without copying every file name in the CMakeLists.txt.

ADD_DEFINITIONS (" -Wall . . . ")

This adds the flags to the compilation of the source files.

ADD_EXECUTABLE (${Tpm} . . .)

This command takes code files and with all the previous definitions creates the needed target. One of the big advantage of CMake is, that it creates the targets by checking the included file in the source and header files of the code. By changing one file not every other has to be recompiled. Usually this is done manually and it is a lot of work.

For further information we refer to [9], the man page and to the complete CMake-Code for the TPM-Emulator in Appendix.

5.2.1 Difficulties of platform independence

There are popular difficulties of platform independence applications. Most of them are well-known to developers like endianness, the addressing-problem, differences in compilers and problems with libraries. Usually the interprocess-communication and the logging is platform dependant, as well as the user- and right management. As the reader can see in Section 5.3 we changed the whole interprocess-communication (Section 5.3.2) and the logging system (Section 5.3.3). As the TPM-Emulator runs as an application, the user management is easy to manage.

5.3 Windows

As a proof of concept we ported the TPM-Emulator to Microsoft Windows XP. There were different problems arising by porting. Some of them will be explained in detail. The main problems are the compilers, the differences in saving log entries and the decision which compiler should be used. The decision which design is used for Microsoft Windows were sometimes easy, sometimes difficult. We will show one of such a decision in detail: The decision how the interprocess communication should be done.
First we will see which build environment has been chosen.

5.3.1 Build Environment

As written before in Section 5.1.2 CMake is capable of using a lot of different compilers (it even creates project files for Visual Studio). There are some differences from the Borland Compiler to the GNU Compiler. As there were already some commands which only work with the GNU compiler we decided not to rewrite these commands but to use the GNU Compiler (gcc).

To use the gcc we decided to use MinGW Makefiles[7]. MinGW is a "Minimalistic GNU for Windows" to use gcc and GNU Binutils in the development of native Microsoft Windows applications. There are other environments for using gcc in Microsoft Windows as Cgwin but the advantages of MinGW were the following:

- It is easy to install. It is just a binary and includes everything you need to just compile the Makefiles. This is different to e.g. Cygwin, where you have to choose from different packages until you have everything you need.
- MinGW does not need additional libraries. So there is no unnecessary overhead for linking or augmenting the final binary.

As the TPM-Emulator should be easy to compile for a developer and easy to use for testing for a normal user this is the best choice. Every developer can with the CMake build system use the build environment he knows, but a normal user can just follow the instruction or use the precompiled executable.

5.3.2 Connection Sockets/Pipes

The implementation for linux for the interprocess communication uses the unix sockets. They are available for linux, unix, Apple, Free-BSD and others. But not for Microsoft Windows. Microsoft Windows has sockets, but "only" the usual network sockets. The starting point is, that we have a server, the TPM-Emulator, and different clients who will send requests which the TPM-Emulator must answer to this specific client.

There are different possibilities for Interprocess Communication in Windows (IPC)[15]. First we list them and explain about the pros and cons and then present our choice for the TPM-Emulator in Microsoft Windows.

Clipboard

Everyone knows the clipboard but nobody would think of it as an interprocess communication possibility, but in fact it is. But it is only usable with user interaction and we mention it just for completeness.

Dynamic Data Exchange

Dynamic Data Exchange (DDE) can be thought as extension to clipboard. It is old and not efficient, so it is only used when an existing application only supports DDE.

File handler

Every file can be used for interprocess communication. It is implementable at every operating system. As this is only the use of a file as a memory location, there are better methods for shared memory. One problem of using files for IPC for our TPM-Emulator would be the synchronization.

Mailslots

Mailslots behave, like the name says, as mails on the internet. A server creates a mailslot and another process sends messages to the mailslot server.

These are only one way communication, perfect for broadcast but as there are no confirmations and responds, this is no good solution for interaction from client to server.

Named Pipe

Named pipe exists in unix etc. Is in Windows a client-server communication, the passive mode is similar to the unix domain sockets. They are handled like a file (as in unix before). They are not permanent, so this is what we want. To connect you have to know the path, like `//.//pipe//‘‘name of the pipe’’`. They are inter- and intramachine interprocess communitations. There exists different modes like half duplex or full-duplex.

Most of the IPC are not suitable for the TPM-Emulator. At last the decision was between sockets and pipes. As the Microsoft Windows sockets uses different commands than Unix sockets reusing the code is not possible neither with sockets nor named pipes. Sockets are more intended for communication with different machines, although named pipes are also capable, so named pipes are a little bit better suited for our purpose.

5.3.3 Logging

In every unix-like operating system one of the easiest and best possibility to write a log is to use the syslog. As the TPM-Emulator will be mainly used for developers to debug and test their application, a good logging system is essential.
Microsoft Windows XP knows an Event Logger[16]. We adapted the function in the TPM-Emulator to write to the Event Logger instead of syslog. With the command `init_log()` the TPM-Emulator register an Event Source (the default is “Application”). After initialization the function `tpm_log()` takes as parameter the priority and the message. It writes a new event into the registered source. The ability to write debug messages is implemented as it was in syslog.

It is also possible to start the TPM-Emulator not as service but as application. For this use case there is the possibility to write the log to the console where the TPM-Emulator is started and not only in the Event Log.

5.4 Driver - TDDL

TCG specifies a way to access a TPM, the so-called TPM Device Driver Library (TDDL). It is the user mode interface. It ensures that different implementations of the TCG software stack can communicate with any TPM and it provides a operation system independent interface for TPM applications. The TDDL is the connection to the kernel mode. There is only one instance of a TDDL, as the multithreeaded components are applied higher in the stack. So this enforces a single threaded access to the TPM. There are only a few command sets we implemented. See also Section 5.4.2.

5.4.1 Linux

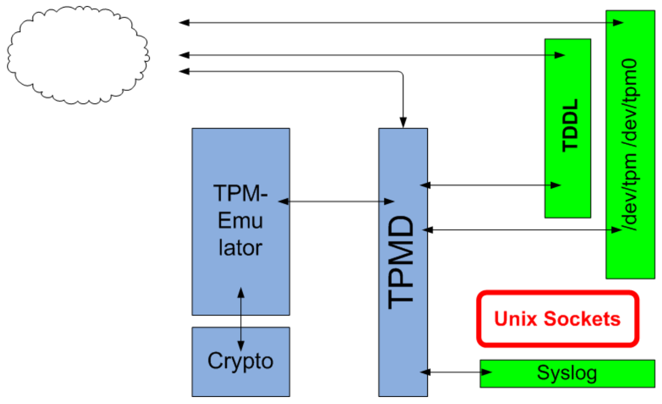


Figure 7: Linux implementation for the TPM-Emulator

In linux the `libtddl.so` library has been done by Mario Strasser in 2006 and uses the unix sockets instead of the hardware access as it would be for a TPM-Chip. By using this library every linux program will access the TPM-Emulator instead of a TPM-Chip.

5.4.2 Windows

For this master thesis we wrote a TDDL for windows. In Microsoft Windows XP most program use the `ifxtpm.dll`.
`Ifxtpm.dll` is from Infineon Technologies AG and belongs to their TPM Software Driver. This library is provided in software based on Infineon's TPM software stack. Several manufacturers are infineon-based even if they are not from Infineon itself. So by exchanging this library in Microsoft

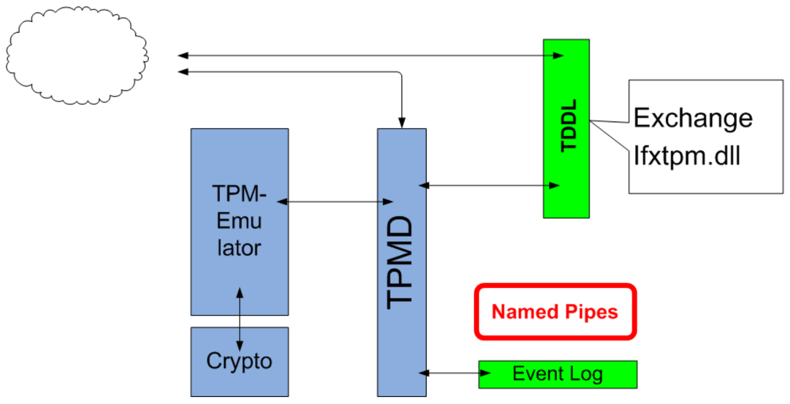


Figure 8: Windows implementation for the TPM-Emulator

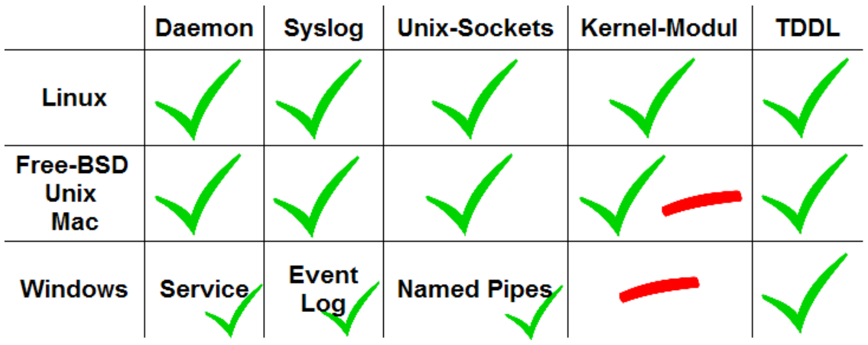
Windows XP in most cases the TPM-Emulator will be used instead of a TPM-Chip. If this will not work it is possible to run an application which tells which libraries are used. By replacing the TPM library every TPM holding the specification of the TDDL by the TCG will work.
In the following we look on the implementation of the device driver library. The main commands are to open and close the device, transmit and receive data and get the capabilities of the TPM. Actually, by open and close the library opens respectively closes the named pipe to the TPM-Emulator. The receiving and the sending of data is consequentially only sending the bytes over the named pipe to the TPM-Emulator. The TPM-Emulator then marshals the data according to the data of `tpmd.c` (Section 2.2). With this, the library is as small as possible.
This library writes important events and errors to the Event Log, as the TPM-Emulator does.

5.4.3 Apple/Free- and Open-BSD

It is easy to port the TPM-Emulator to Apple, Free- and Open-BSD. Most of the code will work just out of the box. Only for the kernel module there has to be done some adjustment. But CMake will work without problems, so the building process is easy.

5.4.4 All together

The TPM-Emulator runs now in Linux, Mac, Free-BSD, Open-BSD and in Microsoft Windows. Only in Linux is a written device driver interface



	Daemon	Syslog	Unix-Sockets	Kernel-Modul	TDDL
Linux	✓	✓	✓	✓	✓
Free-BSD Unix Mac	✓	✓	✓	✓ —	✓
Windows	Service ✓	Event Log ✓	Named Pipes ✓	—	✓

Figure 9: Comparison of OS

as a kernel module. It is possible to change this written module for Mac, Free-BSD and Open-BSD. But this is not done yet.

6 Testing

As this master thesis consists to a big part in software engineering testing is important. We chose tests which are not only self-written, but are as much as possible from third parties to really check if our implementation is compatible to the specification. This is possible because the main requirements come from the TCG as described in Section 2.1.

6.1 TrouSerS

TrouSerS is a open-source TCG Software Stack. TrouSerS was launched by IBM in 2004 and consists of a TCG Software Stack (TSS) and a TSS Core Services daemon. After installing these parts it is possible to use all the TPM-Commands from TrouSerS and the TCG tests. These tests are for testing hardware TPM. We tested the newly implemented delegation functions and all tests succeeded in our TPM-Emulator.

6.2 CTest

CTest[12] is a part of CMake, which adds the functionality of testing to CMake. It can be used to run tests, but also to perform memory check, coverage and to submit the results do a CDash[13] or Dart[14] dashboard system. At the moment the feature to export the tests are not important for us, but could be used in the future. We use CTest to run tests for developers and to add a simple way for others to add tests by their own.

The syntax is the same as in CMake in Section 5.1.2. It is possible to outsource the tests and include the additional file, but as there are only a few tests at the moment we put these tests at the end of the main CMakeLists.txt.

With the command `ENABLE_TESTING()` the CTest part of CMake is loaded and enabled. It is now possible to add tests, test directories or to start scripts. Adding a test is done with the following command
`ADD_TEST( SimpleTest ${PROJECT_SOURCE_DIR}/tests/simpletest paramter1)`.
With this command a new test "SimpleTest" is added. CTest will start the executable `/project_source_dir/tests/simpletest` with the parameter1 which can be defined as a variable in the CMakeLists.txt or read from a file. As we produced Makefiles with CMake we start the test by just executing `make test` in our build directory. The output will be analog to Figure 10.

```
$cd /tpm-emulator/build_linux
$make test
Running test...
Start processing tests
Test project
/tpm-emulator/build_linux
  1/  2 Testing SimpleTest              ***Failed
  2/  2 Testing SimpleTest2                Passed

50% tests passed, 1 tests failed out of 2

The following tests FAILED:
          1 - SimpleTest (Failed)
```

Figure 10: CTests running

6.3 Other

All of these tests explained before do not run in Microsoft Windows XP, becuase TrouSerS is a linux tool. As the only changes to the TPM-Emulator are in the file `tpm_emulator_config.h` and `tpmd.c` we do not have to test the whole emulator part of the emulator, but only these interfaces. We wrote the TDDL (Section 5.4.2) for windows and this had to be tested too.
We tested all of this with the instrument it will be used afterwards: A third party application, the TPM/J[11]. TPM/J is a cross-platform object-orientated AP using Java to access the TPM. It was developed as part of the research project on Trusted Computing at MIT. It is intended to be used as a platform independent tool for any programmer to access the TPM with handy Java classes. By using TPM driver object to access the TPM they were able to support multiple platforms without requiring the programmer to change any code.[7] We chose this tool to do the tests as it is open-source and therefore easy to debug. It is possible to print all communication from the tool to the TPM to the terminal and see what is going on.

We could access from this Java application the TDDL which connected to the TPM-Emulator instead of a real TPM and it responded back to the application. Without changing anything on the application but by only replacing the device driver library, the ifxtpm.dll, it is now possible to interact

[7]http://tpmj.sourceforge.net

with the TPM-Emulator.

This proves that the TPMD is running correctly, that the TDDL is running as expected for the TPMD and for the third party application. As the TDDL has only a small interface to the TPMD we could test every command and the application always reacted as it would if there were a real TPM.

7 Conclusion

7.1 Summary

The aim of my master thesis was to design and implement the delegation function into the TPM-Emulator and to develop a platform independent build and test framework.
We can state that this main goal is reached as the delegation function is now fully implemented and tested with a third-party tool to test “real” TPM, TrouSerS and all tests succeeded.

We changed the build system to a platform independent, CMake, and implemented CTest for further testing. Going further we built as a proof of concept a TPM-Emulator in Microsoft Windows XP, wrote the TPMD, adjusted the libraries and wrote a TDDL and tried with a third-party application from MIT and the TPM-Emulator reacted as expected.

7.2 Outlook

A future work will be to insert the tests from the Frauenhofer Institute.
Another will be to use the TPM-Emulator in a virtual machine. There are different problems with that, but should be possible without changing the design of the TPM-Emulator.
The portability to other Microsoft Windows Version like Vista and Windows 7 can be done. Other very interesting thesis including the TPM-Emulator are to include the TPM-Emulator into a virtual machine and design how to anchor this TPM-Emulator instances into the host system and this TPM-Chip.

References

[1] TCG - Trusted Computing Group
Homepage with all the ressources and specifications.
http://www.trustedcomputinggroup.com

[2] Kent Yoder et al.
TrouSerS - Open-source TCG Software Stack.
http://trousers.sourceforge.net/.

[3] Trusted Computing Group
Part 1 - Design Principles
http://www.trustedcomputinggroup.org/resources/tpm_main_specification

[4] Trusted Computing Group
Part 2 - TPM Structures
http://www.trustedcomputinggroup.org/resources/tpm_main_specification

[5] Trusted Computing Group
Part 3 - Commands
http://www.trustedcomputinggroup.org/resources/tpm_main_specification

[6] Cygwin.
Website of Cygwin, a Linux-like environment for Windows.
http://www.cygwin.com

[7] Minimalistic GNU for Windows
http://www.mingw.org/

[8] CMake.
Website of CMake, cross-platform, open-source build system
http://www.cmake.org

[9] CMake Documentation for CMake 2.8
Website of CMake, the documentation for version 2.8
http://www.cmake.org/cmake/help/cmake-2-8-docs.html

[10] Mario Strasser et al.
Software-based TPM Emulator for Linux.
http://developer.berlios.de/projects/tpm-emulator.

[11] TPM/J - Trusted Computing at MIT.
http://projects.csail.mit.edu/tc/tpmj/.

[12] CTest: Testing to from CMake
http://www.cmake.org/Wiki/CMake_Testing_With_CTest

[13] CDash: Web-based software testing server
http://www.cdash.org/

[14] Dart: Open-source, distributed, software quality system
http://public.kitware.com/Dart/

[15] MSDN - Interprocess Communication
http://msdn.microsoft.com/en-us/library/aa365574(VS.85)
.aspx

[16] MSDN - Event log
http://msdn.microsoft.com/en-us/library/aa385780(VS.85)
.aspx

[17] Trusted Computing Group.
TPM Software Stack (TSS) Specification, Version 1.2, p. 42.
http://www.trustedcomputinggroup.org/developers/software\
_stack/specifications.

[18] Steven Kinney Trusted Platform Module Basics

[19] Silberschatz, Galvin, Gagne System Concepts - Sixth edition Wiley, 2003

[20] Norbert Pohlmann, Helmut Reimer
Trusted Computing - Ein Weg zu neuen IT-Sicherheitsarchitekturen vieweg, 2008

[21] Jonathan M. McCune, Bryan Parno, Adrian Perrig, Michael K. Reiter, Hiroshi Isozaki
Flicker: An execution Infrastructure for TCB Minimization
http://www.ece.cmu.edu/~jmmccune/papers/mccune_parno_
perrig_reiter_isozaki_eurosys08.pdf

A Appendix

Manual

We will show how to compile the TPM-Emulator in Windows.

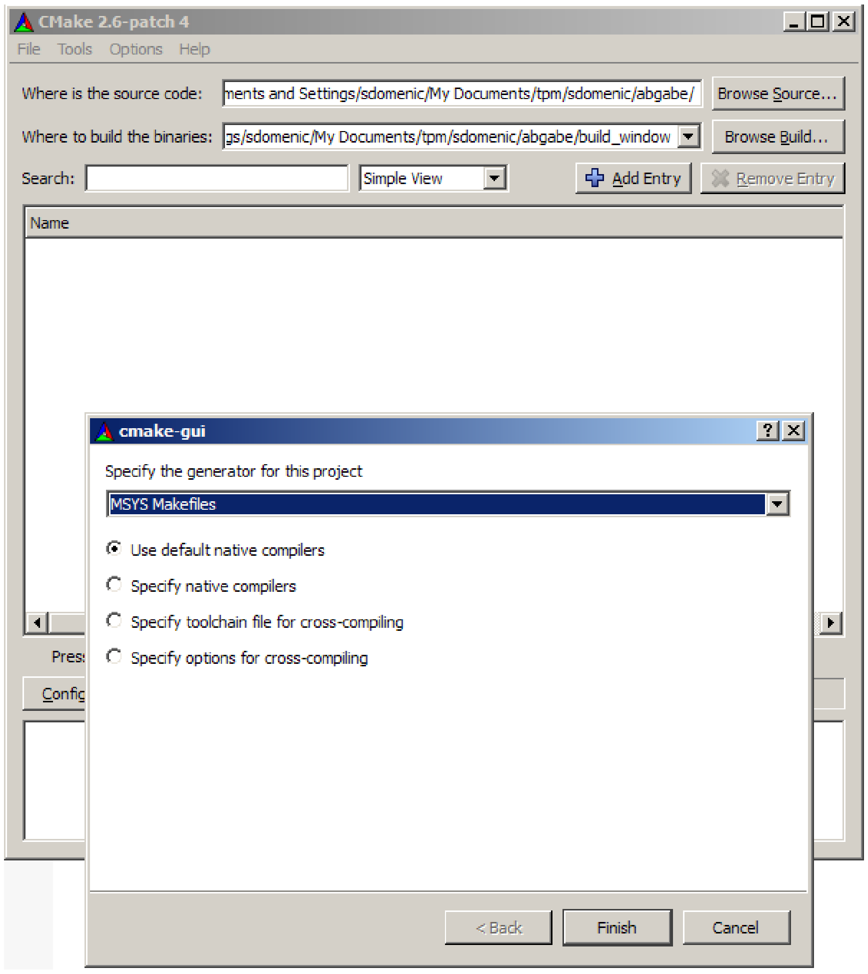
First MinGW and CMake has to be installed. As in Figure 11 create MinSys Files by choosing a build directory. We propose to use a directory with the prefix "build_". If you call it otherwise the command `make dist` will add your build directory too by creating a package. After pressing configure two times you can generate the Makefiles or change the library path or other variables. This is not necessary.

Now you can generate the Makefiles in your chosen build directory, in the Figure it is build_window.

The next step is to open MinGW and change to the build directory. The command `make` will create the tpmd and if available in the operating system the TDDL as you can see in Figure 11.

The file `ifxtpm.dll` has to be copyied to `C:/Windows/System32`. We do not do it automatically, because if there is a TPM in use, it is not clear what will not work anymore, so we leave this to the user who knows what he is doing.
The last step is to execute the `tpmd.exe` in the command line or with a link. By the first start you have to start in clear mode. For further information we refer to the Readme from the linux tpmd. The output will be as in Figure 13.

To create packages with all the source file you can use the command `make dist` from CPack. The output as you can see in Figure 14 will be a compressed file with everything from the project directory without any directory with the prefix "build_" and without svn or other version control tools.



Where is the source code: ments and Settings/sdomenic/My Documents/tpm/sdomenic/abgabe/

Where to build the binaries: gs/sdomenic/My Documents/tpm/sdomenic/abgabe/build_window

Search:

Simple View

Specify the generator for this project

MSYS Makefiles

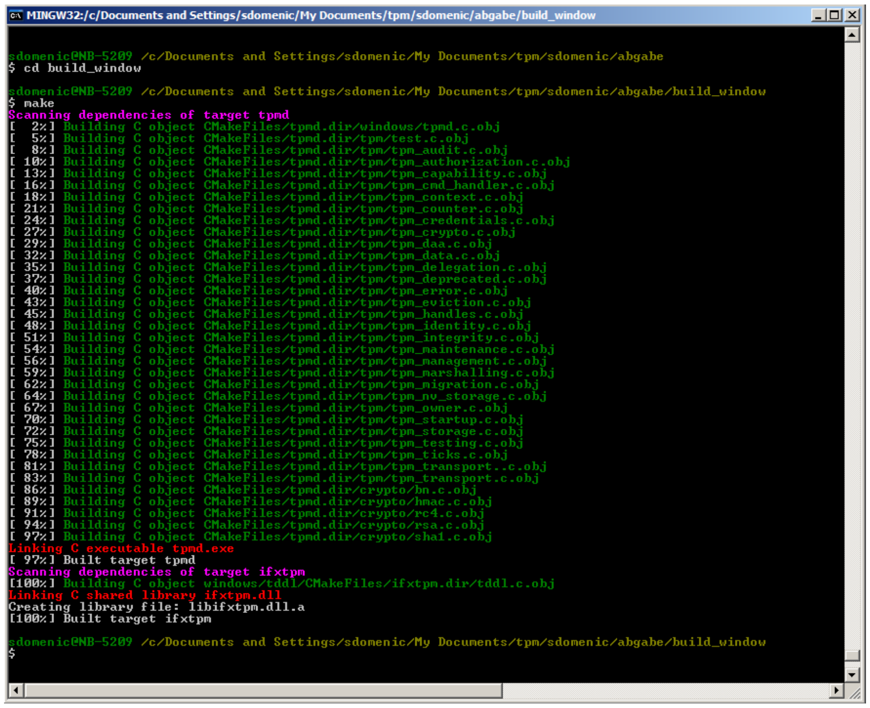
[✓ Use default native compilers]

Specify native compilers

Specify toolchain file for cross-compiling

Specify options for cross-compiling

Figure 11: CMake-gui in Microsoft Windows



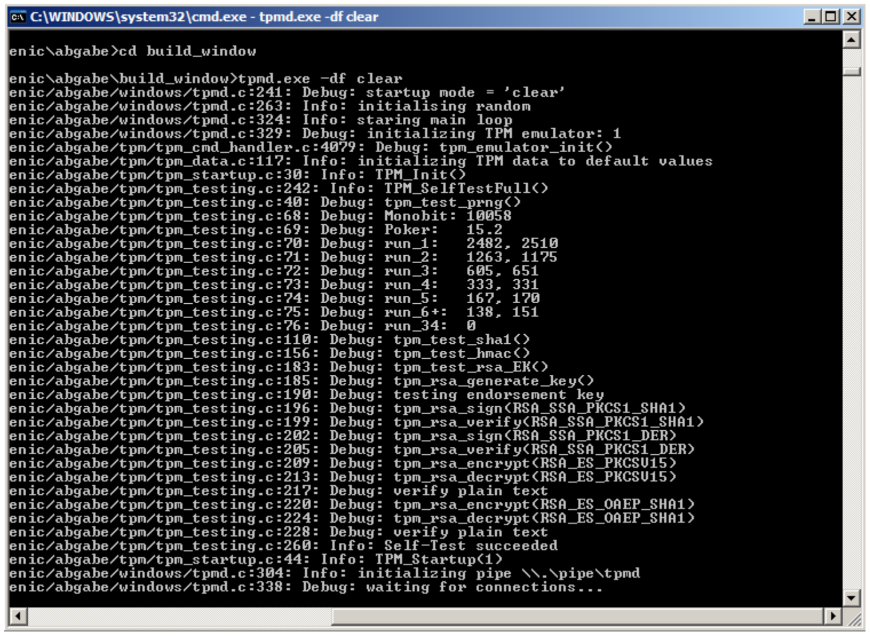
```
MINGW32:/c/Documents and Settings/sdomenic/My Documents/tpm/sdomenic/abgabe/build_window

sdomenic@NB-5209 /c/Documents and Settings/sdomenic/My Documents/tpm/sdomenic/abgabe
$ cd build_window

sdomenic@NB-5209 /c/Documents and Settings/sdomenic/My Documents/tpm/sdomenic/abgabe/build_window
$ make
Scanning dependencies of target tpmd
[  2%] Building C object CMakeFiles/tpmd.dir/windows/tpmd.c.obj
[  5%] Building C object CMakeFiles/tpmd.dir/tpm/test.c.obj
[  8%] Building C object CMakeFiles/tpmd.dir/tpm/tpm_audit.c.obj
[ 10%] Building C object CMakeFiles/tpmd.dir/tpm/tpm_authorization.c.obj
[ 13%] Building C object CMakeFiles/tpmd.dir/tpm/tpm_capability.c.obj
[ 16%] Building C object CMakeFiles/tpmd.dir/tpm/tpm_cmd_handler.c.obj
[ 18%] Building C object CMakeFiles/tpmd.dir/tpm/tpm_context.c.obj
[ 21%] Building C object CMakeFiles/tpmd.dir/tpm/tpm_counter.c.obj
[ 24%] Building C object CMakeFiles/tpmd.dir/tpm/tpm_credentials.c.obj
[ 27%] Building C object CMakeFiles/tpmd.dir/tpm/tpm_crypto.c.obj
[ 29%] Building C object CMakeFiles/tpmd.dir/tpm/tpm_daa.c.obj
[ 32%] Building C object CMakeFiles/tpmd.dir/tpm/tpm_data.c.obj
[ 35%] Building C object CMakeFiles/tpmd.dir/tpm/tpm_delegation.c.obj
[ 37%] Building C object CMakeFiles/tpmd.dir/tpm/tpm_deprecated.c.obj
[ 40%] Building C object CMakeFiles/tpmd.dir/tpm/tpm_error.c.obj
[ 43%] Building C object CMakeFiles/tpmd.dir/tpm/tpm_eviction.c.obj
[ 45%] Building C object CMakeFiles/tpmd.dir/tpm/tpm_handles.c.obj
[ 48%] Building C object CMakeFiles/tpmd.dir/tpm/tpm_identity.c.obj
[ 51%] Building C object CMakeFiles/tpmd.dir/tpm/tpm_integrity.c.obj
[ 54%] Building C object CMakeFiles/tpmd.dir/tpm/tpm_maintenance.c.obj
[ 56%] Building C object CMakeFiles/tpmd.dir/tpm/tpm_management.c.obj
[ 59%] Building C object CMakeFiles/tpmd.dir/tpm/tpm_marshalling.c.obj
[ 62%] Building C object CMakeFiles/tpmd.dir/tpm/tpm_migration.c.obj
[ 64%] Building C object CMakeFiles/tpmd.dir/tpm/tpm_nv_storage.c.obj
[ 67%] Building C object CMakeFiles/tpmd.dir/tpm/tpm_owner.c.obj
[ 70%] Building C object CMakeFiles/tpmd.dir/tpm/tpm_startup.c.obj
[ 72%] Building C object CMakeFiles/tpmd.dir/tpm/tpm_storage.c.obj
[ 75%] Building C object CMakeFiles/tpmd.dir/tpm/tpm_testing.c.obj
[ 78%] Building C object CMakeFiles/tpmd.dir/tpm/tpm_ticks.c.obj
[ 81%] Building C object CMakeFiles/tpmd.dir/tpm/tpm_transport..c.obj
[ 83%] Building C object CMakeFiles/tpmd.dir/tpm/tpm_transport.c.obj
[ 86%] Building C object CMakeFiles/tpmd.dir/crypto/bn.c.obj
[ 89%] Building C object CMakeFiles/tpmd.dir/crypto/hmac.c.obj
[ 91%] Building C object CMakeFiles/tpmd.dir/crypto/rc4.c.obj
[ 94%] Building C object CMakeFiles/tpmd.dir/crypto/rsa.c.obj
[ 97%] Building C object CMakeFiles/tpmd.dir/crypto/sha1.c.obj
Linking C executable tpmd.exe
[ 97%] Built target tpmd
Scanning dependencies of target ifxtpm
[100%] Building C object windows/tddl/CMakeFiles/ifxtpm.dir/tddl.c.obj
Linking C shared library ifxtpm.dll
Creating library file: libifxtpm.dll.a
[100%] Built target ifxtpm

sdomenic@NB-5209 /c/Documents and Settings/sdomenic/My Documents/tpm/sdomenic/abgabe/build_window
$
```

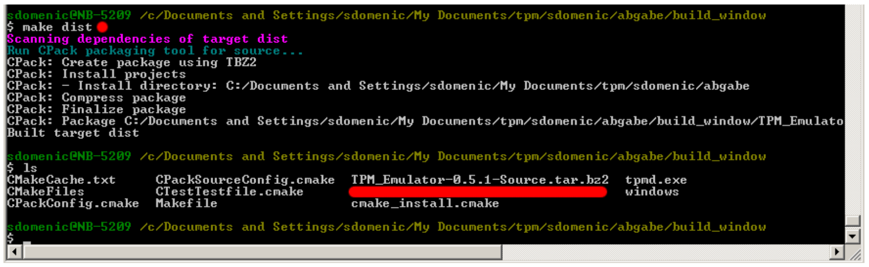
Figure 12: Build with make in Microsoft Windows



```
enic\abgabe>cd build_window

enic\abgabe\build_window>tpmd.exe -df clear
enic/abgabe/windows/tpmd.c:241: Debug: startup mode = 'clear'
enic/abgabe/windows/tpmd.c:263: Info: initialising random
enic/abgabe/windows/tpmd.c:324: Info: staring main loop
enic/abgabe/windows/tpmd.c:329: Debug: initializing TPM emulator: 1
enic/abgabe/tpm/tpm_cmd_handler.c:4079: Debug: tpm_emulator_init()
enic/abgabe/tpm/tpm_data.c:117: Info: initializing TPM data to default values
enic/abgabe/tpm/tpm_startup.c:30: Info: TPM_Init()
enic/abgabe/tpm/tpm_testing.c:242: Info: TPM_SelfTestFull()
enic/abgabe/tpm/tpm_testing.c:40: Debug: tpm_test_prng()
enic/abgabe/tpm/tpm_testing.c:68: Debug: Monobit: 10058
enic/abgabe/tpm/tpm_testing.c:69: Debug: Poker:   15.2
enic/abgabe/tpm/tpm_testing.c:70: Debug: run_1:   2482, 2510
enic/abgabe/tpm/tpm_testing.c:71: Debug: run_2:   1263, 1175
enic/abgabe/tpm/tpm_testing.c:72: Debug: run_3:   605, 651
enic/abgabe/tpm/tpm_testing.c:73: Debug: run_4:   333, 331
enic/abgabe/tpm/tpm_testing.c:74: Debug: run_5:   167, 170
enic/abgabe/tpm/tpm_testing.c:75: Debug: run_6+:  138, 151
enic/abgabe/tpm/tpm_testing.c:76: Debug: run_34:  0
enic/abgabe/tpm/tpm_testing.c:110: Debug: tpm_test_sha1()
enic/abgabe/tpm/tpm_testing.c:156: Debug: tpm_test_hmac()
enic/abgabe/tpm/tpm_testing.c:183: Debug: tpm_test_rsa_EK()
enic/abgabe/tpm/tpm_testing.c:185: Debug: tpm_rsa_generate_key()
enic/abgabe/tpm/tpm_testing.c:190: Debug: testing endorsement key
enic/abgabe/tpm/tpm_testing.c:196: Debug: tpm_rsa_sign(RSA_SSA_PKCS1_SHA1)
enic/abgabe/tpm/tpm_testing.c:199: Debug: tpm_rsa_verify(RSA_SSA_PKCS1_SHA1)
enic/abgabe/tpm/tpm_testing.c:202: Debug: tpm_rsa_sign(RSA_SSA_PKCS1_DER)
enic/abgabe/tpm/tpm_testing.c:205: Debug: tpm_rsa_verify(RSA_SSA_PKCS1_DER)
enic/abgabe/tpm/tpm_testing.c:209: Debug: tpm_rsa_encrypt(RSA_ES_PKCSV15)
enic/abgabe/tpm/tpm_testing.c:213: Debug: tpm_rsa_decrypt(RSA_ES_PKCSV15)
enic/abgabe/tpm/tpm_testing.c:217: Debug: verify plain text
enic/abgabe/tpm/tpm_testing.c:220: Debug: tpm_rsa_encrypt(RSA_ES_OAEP_SHA1)
enic/abgabe/tpm/tpm_testing.c:224: Debug: tpm_rsa_decrypt(RSA_ES_OAEP_SHA1)
enic/abgabe/tpm/tpm_testing.c:228: Debug: verify plain text
enic/abgabe/tpm/tpm_testing.c:260: Info: Self-Test succeeded
enic/abgabe/tpm/tpm_startup.c:44: Info: TPM_Startup(1)
enic/abgabe/windows/tpmd.c:304: Info: initializing pipe \\.\pipe\tpmd
enic/abgabe/windows/tpmd.c:338: Debug: waiting for connections...
```

Figure 13: Starting the tpmd in Microsoft Windows



```
sdomenic@NB-5209 /c/Documents and Settings/sdomenic/My Documents/tpm/sdomenic/abgabe/build_window
$ make dist
Scanning dependencies of target dist
Run CPack packaging tool for source...
CPack: Create package using TBZ2
CPack: Install projects
CPack: - Install directory: C:/Documents and Settings/sdomenic/My Documents/tpm/sdomenic/abgabe
CPack: Compress package
CPack: Finalize package
CPack: Package C:/Documents and Settings/sdomenic/My Documents/tpm/sdomenic/abgabe/build_window/TPM_Emulato
Built target dist

sdomenic@NB-5209 /c/Documents and Settings/sdomenic/My Documents/tpm/sdomenic/abgabe/build_window
$ ls
CMakeCache.txt      CPackSourceConfig.cmake  TPM_Emulator-0.5.1-Source.tar.bz2  tpmd.exe
CMakeFiles          CTestTestfile.cmake                                         windows
CPackConfig.cmake   Makefile                 cmake_install.cmake

sdomenic@NB-5209 /c/Documents and Settings/sdomenic/My Documents/tpm/sdomenic/abgabe/build_window
$
```

Figure 14: Creating packages with CPack in Microsoft Windows.

CMake: CMakeList.txt

```
PROJECT( TPM_Emulator)

CMAKE_MINIMUM_REQUIRED( VERSION 2.6 )

###############
#CONFIGURATION#
###############
SET( CMAKE_ALLOW_LOOSE_LOOP_CONSTRUCTS TRUE )

SET( ${PROJECT_NAME}_MAJOR_VERSION 0 )
SET( $Major ${${PROJECT_NAME}_MAJOR_VERSION} )
SET( ${PROJECT_NAME}_MINOR_VERSION 5 )
SET( ${PROJECT_NAME}_BUILD_VERSION 1 )



#######
#CPack#
#######

## Implement the package system ##
SET( CPACK_PACKAGE_VERSION_MAJOR ${${PROJECT_NAME}_MAJOR_VERSION} )
SET( CPACK_PACKAGE_VERSION_MINOR ${${PROJECT_NAME}_MINOR_VERSION} )
SET( CPACK_SOURCE_GENERATOR "TBZ2" )
#SET ( CPACK_SOURCE_PACKAGE_FILE_NAME "${PROJECT_NAME}-${Major}-source" )
SET( CPACK_SOURCE_IGNORE_FILES "/build.*/;/.bzr/;~$;/.svn/;${CPACK_SOURCE_IGNORE_FILES}" )
#A "better" command
ADD_CUSTOM_TARGET( dist COMMAND ${CMAKE_MAKE_PROGRAM} package_source )
INCLUDE( CPack )


SET( Crypto_dir ${PROJECT_SOURCE_DIR}/crypto )
SET( Tpm_dir ${PROJECT_SOURCE_DIR}/tpm )
#CHECK_INCLUDE_FILES(tpm_version.h _TPM_VERSION_H_)

SET( CMAKE_MODULE_PATH ${PROJECT_SOURCE_DIR}/cmake_modules )
IF( UNIX )
        SET( Tpmd_user "tss" )
        SET( Tpmd_group "tss" )
        SET( Store_dir "/var/lib/tpm" )
        IF( APPLE )
                SET( System "Apple" )
        ELSE( APPLE )
                SET( System "linux" )
                SET( System_dir ${PROJECT_SOURCE_DIR}/linux )
        ENDIF( APPLE )
ELSE( UNIX )
        IF( WIN32 )
                SET( System "Win32" )
                INCLUDE( FindADVAPI )
                IF ( Advapi32_found )
                                LINK_LIBRARIES( advapi32 )
                else ( Advapi32_found )
                        MESSAGE( FATAL_ERROR "Advapi32 not found, abort" )
                ENDIF( Advapi32_found )
                SET( CMAKE_LIBRARY_PATH "${PROJECT_SOURCE_DIR}/gmp" )
                SET( CMAKE_INCLUDE_PATH ${PROJECT_SOURCE_DIR}/gmp )
                SET( System_dir ${PROJECT_SOURCE_DIR}/windows )
                #Worked with msys Makefiles
        ELSE( WIN32 )
                SET( System "Unknown" )
                MESSAGE( WARNING "System unknown, try linux" )
                SET( System_dir ${PROJECT_SOURCE_DIR}/linux )
        ENDIF( WIN32 )
ENDIF( UNIX )

MESSAGE( STATUS "System is recognised as ${System}." )

INCLUDE( FindGMP )
IF( Gmp_found )
        MESSAGE( STATUS "GMP ready to use" )
        INCLUDE_DIRECTORIES( "${Gmp_h}" )
        LINK_DIRECTORIES ( "${Gmp_h}" )
ELSE( Gmp_found )
        MESSAGE( FATAL_ERROR "GMP not found, abort" )
ENDIF( Gmp_found )

###############
#The real work#
###############
#include dir
INCLUDE_DIRECTORIES( "${PROJECT_SOURCE_DIR}/crypto" )
INCLUDE_DIRECTORIES( "${PROJECT_SOURCE_DIR}/tpm" )
INCLUDE_DIRECTORIES( ${System_dir} )
MESSAGE( STATUS  "${PROJECT_SOURCE_DIR}/${System}")
ADD_DEFINITIONS( "-Wall -Werror -Wno-unused -Wpointer-arith
-Wcast-align -Wwrite-strings -Wsign-compare -Wno-multichar" )

#To turn of Werror uncomment the following 2 lines
#MESSAGE( Turned off werror!!)
#ADD_DEFINITIONS("-Wall  -Wno-unused -Wpointer-arith -Wcast-align
#-Wwrite-strings -Wsign-compare -Wno-multichar")
ADD_DEFINITIONS("-g -I.. -I. -O2 -fno-strict-aliasing")

#TARGET_LINK_LIBRARIES
LINK_LIBRARIES( gmp )

#Importing like this you have to call cmake yourself by adding or removing files!
file( GLOB Crypto "${Crypto_dir}/*.[h|c]" )
file( GLOB Tpm "${Tpm_dir}/*.[h|c]" )
file( GLOB Tpmd "${System_dir}/*.[h|c]" )


#######
#Do it#
#######
ADD_EXECUTABLE( tpmd ${Tpmd}  ${Tpm} ${Crypto}  )

#ADD_DIR to store files:
```

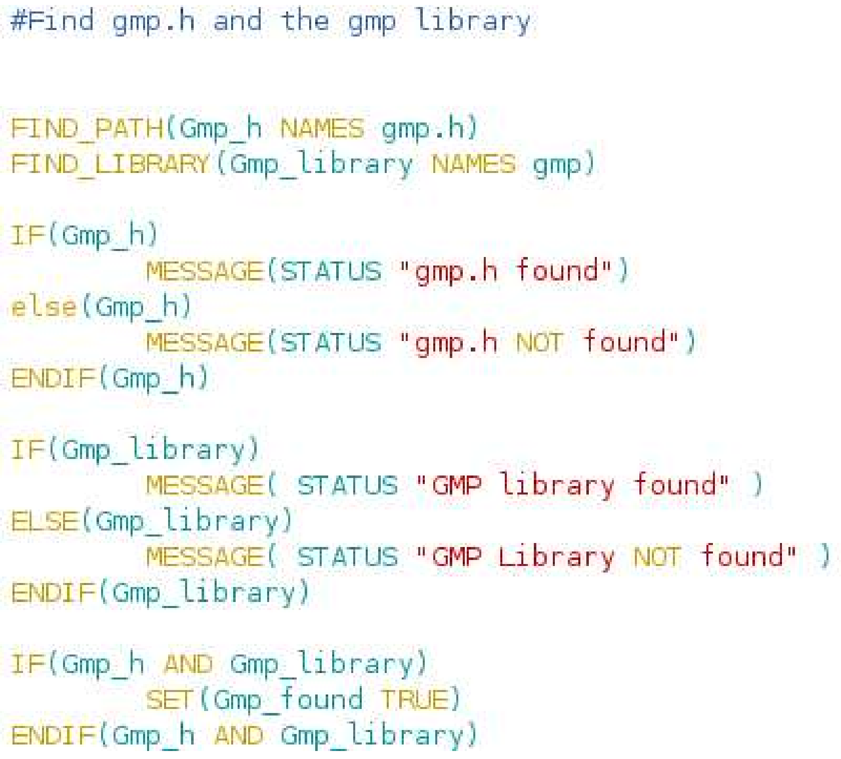
```
FILE( MAKE_DIRECTORY "${Store_dir}" )
INSTALL( TARGETS tpmd DESTINATION /usr/sbin
        PERMISSIONS OWNER_EXECUTE OWNER_WRITE OWNER_READ
        GROUP_EXECUTE GROUP_READ
        WORLD_EXECUTE WORLD_READ
        )

######
#TDDL#
######
#This calls the CMakeLists.txt in the system/tddl
MESSAGE ( STATUS "Search for TDDL in ${System_dir}" )
IF( EXISTS "${System_dir}/tddl" AND IS_DIRECTORY "${System_dir}/tddl" )
        ADD_SUBDIRECTORY( ${System}/tddl )
ENDIF()

#########
#Testing#
#########

ENABLE_TESTING()
ADD_TEST( SimpleTest ${PROJECT_SOURCE_DIR}/tests/simpletest )
ADD_TEST( SimpleTest2 ${PROJECT_SOURCE_DIR}/tests/simpletest return_true )
```

CMake: Module for Windows to find library



```
#Find gmp.h and the gmp library


FIND_PATH(Gmp_h NAMES gmp.h)
FIND_LIBRARY(Gmp_library NAMES gmp)

IF(Gmp_h)
        MESSAGE(STATUS "gmp.h found")
else(Gmp_h)
        MESSAGE(STATUS "gmp.h NOT found")
ENDIF(Gmp_h)

IF(Gmp_library)
        MESSAGE( STATUS "GMP library found" )
ELSE(Gmp_library)
        MESSAGE( STATUS "GMP Library NOT found" )
ENDIF(Gmp_library)

IF(Gmp_h AND Gmp_library)
        SET(Gmp_found TRUE)
ENDIF(Gmp_h AND Gmp_library)
```