SCD

SECTION CONTROL DAEMON

by

Oliver Lorenz

Bachelor of Science

Studiengang Telematik / Netzwerktechnik

FH Kaernten

January 2011

ABSTRACT

SCD

SECTION CONTROL DAEMON

Oliver Lorenz

Studiengang Telematik / Netzwerktechnik

Bachelor of Science

Kapsch TrafficCom invented a new technique to increase data privacy on Section Control systems because current laws state that operating aforesaid systems infringes §1 of Austria's Data Privacy Law (DSG 2000) and therefore current systems are declared unusable.

The major issue with current systems is the storage of personal information like the license plate number because data will be recorded immediately after a vehicle enters a monitored section. Austria's Constitutional Court acquainted this approach illegal, because storage of personal data is only permitted in case of committing a contravention of law. Due to the fact that Section Control systems can significantly reduce the number of accidents[1] the Federal Ministry for Transport, Innovation and Technology enacted an ordinance which

[1] http://www.asfinag.at/verkehrssicherheit/section-control (german)

readmits the use of such systems.[2]

Kapsch's invention rests upon the use of encryption to protect personal data and to fulfil Austria's law standards. Object-oriented programming will be used to offer a simple interface to the cryptographic and hash functions, which can also be exchanged to different standards. This leads to improved privacy and data security because right after encryption the key is destroyed immediately. If a vehicle leaves a monitored section, the key can be regained and the measurement unit at the exit gate queries the unit at the entrance for the encrypted data. Right after data reception is completed, the software checks the recorded time stamps to decide whether there is a criminal offence or not. Only if there is an offence, the system decrypts the data and creates an incident report for this vehicle.

The main objective involves creating a software which implements the techniques stated in the patent specification. As a result it can be used as a proof-of-concept prototype for Kapsch to examine system and privacy improvements.

[2]BGBl. II Nr. 168/2007, 169/2007 and 176/2007

Contents

List of Figures

Chapter 1

Introduction

1.1 General

Section Control is a pseudo-Anglicism used in Austria to characterize systems used for average speed measurement on freeways. It should not be confused with traffic enforcement cameras that are used for speed control at single points whereas Section Control systems monitor a certain road passage. The main difference gets quite obvious when considering security aspects affecting road security. Traffic enforcement cameras have a scope of approx. 300 meters while Section Control systems can monitor arbitrary lengths - in Austria current section lengths vary from two to seven kilometres.

Figure 1.1 Road sign indicating Section Control surveillance in Austria.

1.2 Functionality

As mentioned above Section Control systems are based upon average speed measurement. An ordinary system consists of two or more infra-red cameras, usually mounted overhead and a unit used for time calculations. License plate recognition is achieved using optical character recognition (OCR).

Work flow:

1. Vehicle passes entrance: OCR cam reads the license plate number and stores the data along with the current time.

2. Vehicle is on its way to the exit measurement unit

3. Once it passed the exit unit the license plate number gets read in a second time and the system determines whether there is a criminal offence or not by calculating the time required (thus calculating the average speed for the monitored distance).

Refer to figure **??** for a graphical description.

1.3 Problems

The main problem regarding these systems are privacy issues due to the fact that personal data is being stored before a violation of law occurs. This is the reason why people rightly raised objections against judgements made based on data gained from Section Control systems.

According to Austria's "data privacy law" (DSG 2000) the storage of personal data is only permitted after committing an infringement. Now it gets quite obvious why these objections are justified and sustained. As a result an enactment from the

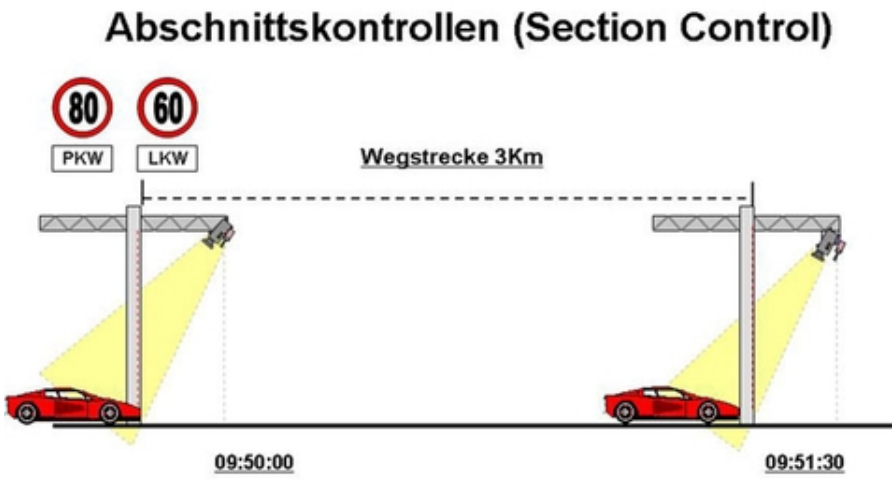

Figure 1.2 A basic Section Control system

Federal Ministry for Transport, Innovation and Technology is required *per* system installed.

The current situation is quite unsatisfying hence improved ways of processing personal data are required to further improve data privacy and the installation of new systems.

1.4 Objective

Austrian based company Kapsch TrafficCom wanted a prototype for a software which handles recording and storing of personal data retrieved from an OCR camera on freeways. With the assistance of cryptographic algorithms the recorded data is fully encrypted and impervious to abuse during a vehicle's travel to the section's exit. This procedure requires just a software update on current systems without the need of any exchange of hardware. Existing systems become compliant with Austria's law at very low costs on the one hand and a great increase in privacy on the other hand.

Chapter 2

Software Design

2.1 UML

UML or Unified Modeling Language is a modeling language in the field of software design and engineering. It includes rules, graphical notations and introduces important terms to the user to simplify collaboration during an entire software developing project. It was used during the development process and especially in the beginning to get a better understanding which objects exist, what their properties are and how they interact with each other.

2.2 Class Diagrams

The basic idea was a modular design due to the utilization of object-oriented programming language C++. In the beginning a list of involved objects was created and the properties needed were figured out. The result was converted into standardized UML class diagrams and formed the groundwork for the C++ implementation.

Note: Items marked with "+" specify *public elements* that can be accessed from both inside and outside the class, whereas "#" marked items (*protected elements*) can only be accessed from within the class or sub-classes respectively. Items tagged "-" define *private elements* that are only accessible within the class they are defined in. This schema is valid for variables as well as functions.

2.3 Used software

Windows:	.NET Studio 2008	IDE
Mac OS X:	OmniGraffle	Diagramming software
Linux:	vim	Text editor
	GNU gcc	Compiler
	GNU ld	Linker
	Subversion	Revision control
	Git	Revision control
	tail	File display utility
	netcat	OCR Cam simulation

2.4 Revision control

When designing software it is always important to have a working history that keeps track of applied changes to the project. In case of errors or misbehaving code it is very easy to compare (*"to diff"*) the current code to previous versions (*"revisions"*) and unveil the faulty code.

During application development programmers add snap-shots to the revision control (*"revisions"*) that can later be used to discard changes and revert to a working

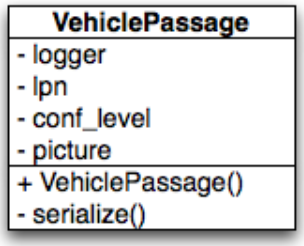

Figure 2.1 Class VehiclePassage representing a car

lpn the vehicles licence plate number

conf_level quality of optical character recognition (OCR)

picture picture from OCR cam (not implemented)

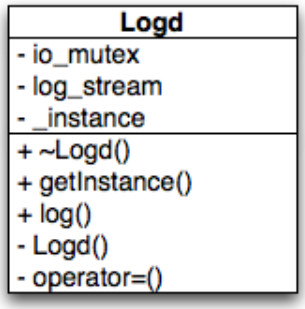

Figure 2.2 Class Logd used for event logging purposes

log() used to write log entries to a file

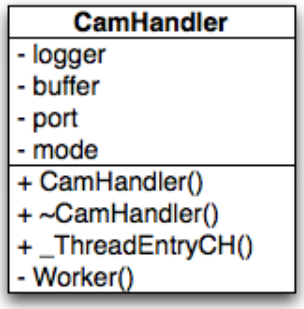

Figure 2.3 Class CamHandler used to communicate with the OCR cameras

Worker() used to read data sent from on OCR cam

buffer queue to enable asynchronous mode of operation

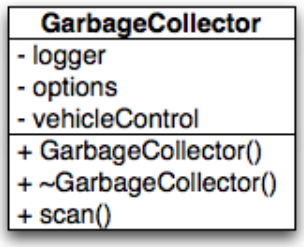

Figure 2.4 Class GarbageCollector used to remove already exceeded vehicles

vehicleControl to get the list of vehicles

scan() reads the list of known vehicles and removes already exceeded ones (where *now* greater than *minTS*)

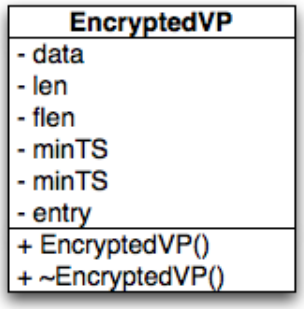

Figure 2.5 Class EncryptedVP holds data of encrypted VehiclePassages

data encrypted data

len data length

flen fill length used for padding

entry timestamp created when passing an OCR cam

minTS calculated minimum time required to pass the monitored section

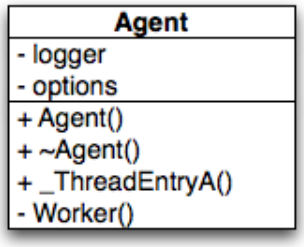

Figure 2.6 Class Agent provides options for remote controlling the Section Control daemon

copy of the code. During the beginning of the project Subversion[1] was used to provide these services, but due to the lack of proper branching functionality git[2] was used henceforth.

Branching is used for parallel development where many people contribute to files under revision control. The ability to branch offers a way to isolate changes to an independent tree without affecting the codebase. This feature is especially useful when the programmer wants to implement new experimental functionality which is normally done in the following way:

1. Achieve a stable master branch
2. Create a new branch and switch to that branch
3. Write and commit the changes
4. Change back to master branch
5. Merge the changes
6. Delete the testing / experimental branch

Consider serious problems while adding new features in the testing branch. While the errors prevent the testing tree from working properly the master branch is still fully functional and thus unaffected from changes made to the testing branch. See **??** for an example.

2.5 Used libraries

Despite the fact that platform independence was not a required objective during the work on the Section Control daemon it was tried to achieve this goal to keep the

[1] http://subversion.apache.org

[2] http://git-scm.com

software as portable as possible. Changing the underlying infrastructure should not affect the software. This is important when a company maintains various systems using different operating systems but still want to use the software without the need of serious changes.

- **Boost C++ libraries**

  Boost provides portable C++ libraries which are either already included in the upcoming C++ standard (Technical report 1) or will be incorporated into the technical report 2. They offer another abstraction layer for operating system specific functions like threading, networking (ASIO), date and time functions, file system manipulation, program options and data serialization. Further details will be given in later chapters.

- **PolarSSL cryptographic C library**

  The PolarSSL libraries offer a simple API to the cryptographic and hash functions used in the Section Control daemon. Implementing cryptographic algorithms like AES is very sensitive, error-prone and it takes a long time to test and verify the code. To not reinvent the wheel the cryptographic library was used for AES encryption, MD5 as well as SHA-256 hashing.

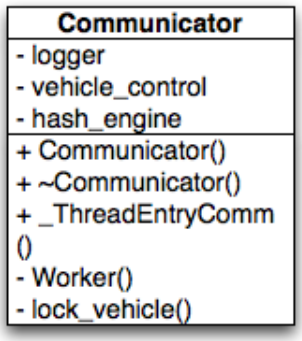

Figure 2.7 Class Communicator used to communicate with other Section Control daemons (e. g. entrance or exit)

vehicle_control to get items from VehicleControl's list of vehicles

hash_engine to calculate hashes used to identify vehicles

lock_vehicle lock the vehicle so the gargabe collector cannot remove it accidentally

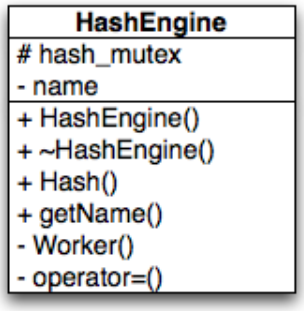

Figure 2.8 Class HashEngine provides an abstract API for hash functions

name the name of the hash method (e.g. "MD5" or "SHA-1")

Hash() returns the hash of the input data

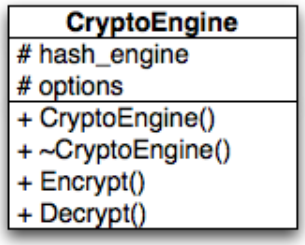

Figure 2.9 Class CryptoEngine provides an abstract API for cryptographic functions

name the name of the cryptographic algorithm (e.g. "AES-256" or "DES")

Encrypt() encrypts input data

Decrypt() decrypts input data

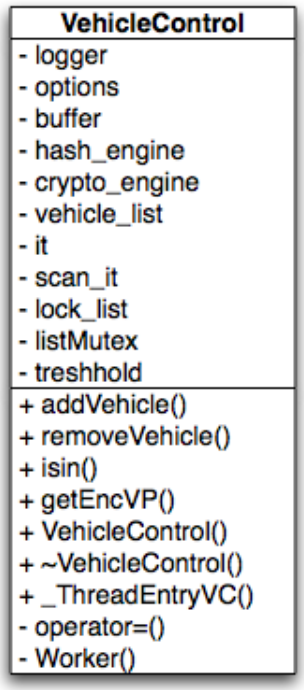

Figure 2.10 Class VehicleControl managing stored vehicle data

vehicle_list list containing all stored vehicles

treshhold minimum OCR quality required

hash_engine access to hash functions

crypto_engine access to cryptographic functions

getEncVP() get an encrypted VehiclePassage from the list

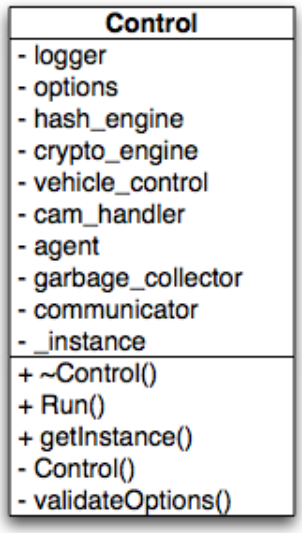

Figure 2.11 Class Control is used as a link between all components

Chapter 3

Implementation

3.1 Module Design

Listing 3.1 The Singleton pattern

```
static Logd* getInstance() {
        if( !_instance )
        _instance = new Logd();
        return _instance;
}
```

3.2 Functional interaction

3.3 Caveats

Chapter 4

Software tests

Chapter 5

Conclusion

Appendix A

Appendix

Listing A.1 Git revision control

```
\$ git status
# On branch master
nothing to commit (working directory clean)
\$ git branch experimental
$ git checkout experimental
Switched to branch 'experimental'
$ edit files ...
git commit -a
[experimental f4e4220] edit main
1 files changed, 7 insertions(+), 0 deletions(-)
$ git checkout master
Switched to branch 'master'
$ git merge experimental
Updating ed66c24..f4e4220
Fast-forward
```

```
main.c |    7 +++++++
1 files changed, 7 insertions(+), 0 deletions(-)
$ git branch -d experimental
Deleted branch experimental (was f4e4220).
```