Attacken mit manipulierten X.509 Zertifikaten

Projektarbeit 2, 2004
Zürcher Hochschule Winterthur

Studenten: **Hannes Zürcher**
zuerchan@zhwin.ch
Dominik Riedweg
riedwdom@zhwin.ch

Dozent: **Prof. Dr. Andreas Steffen**

Inhaltsverzeichnis:

1. Abstract 5
1.1. Einleitung 5
2. Aufgabenstellung 6
2.1. Pflichtenheft 8
3. Termine und Arbeitsteilung 9
3.1. Beschreibung der Arbeitspakete 9
3.2. Projektmeilensteine und Termine 9
3.3. Balkendiagramm (Gantt) 11
4. Übersicht über die Grundlagen 12
4.1. ASN.1 / DER-Kodierung 12
4.2. DER-Kodierung 12
4.3. X.509 13
5. Das Zertifikat-Tool (Cert) 15
5.1. Einleitung 15
5.2. Implementation 15
5.2.1. Cert.java: 15
5.2.2. ByteArray.java: 16
5.2.3. Die ASN.1 Strukturen wie Set.java, PrintableString.java und OID.java 16
5.2.4. ConfigFile.java 16
5.2.5. MessDig.java 16
5.3. Klassenübersicht 16
5.4. Sequenzieller Ablauf des Programms 17
5.5. Handbuch 18
5.5.1. Das Konfigurationsfile 20
6. Testen der Zertifikate 21
6.1. Übersicht über die Tests 21
6.2. Getestete Programme 21
7. Resultate 23
7.1. Standardzertifikat 23
7.2. Gefälschte Zertifikate 24
7.2.1. Objekt-Bezeichner 24
7.2.2. Bitstrings 27
7.2.3. Date 28
7.2.4. Common name 31
7.2.5. IA5String 32
7.2.6. Integer Header 33
7.2.7. NullType 34
7.2.8. Public Key 34
7.2.9. Sequence 36
7.2.10. Explicit Tags 37

8. Probleme und Verbesserungsmöglichkeiten 39
8.1. Probleme 39
8.2. Ausbaufähigkeit und Verbesserungen 40
9. Fazit 40
9.1. Schlusswort 40
10. Anhang 42
10.1. CD-Listing 42
10.2. Links 43
10.3. ASN.1 und DER Definitionen 44
10.4. Sourcecodes 45
10.4.1. Das Konfigurationsfile 45

Abbildungsverzeichnis:

Abbildung 1: Übersicht über die Arbeitspakete 9
Abbildung 2: Gantt-Diagramm 11
Abbildung 3: Ausschnitt der Klassenstruktur des Cert-Tools 17
Abbildung 4: Auszug aus dem Konfigurationsfile 20
Abbildung 5: Darstellung funktionierender Zertifikate in Windows 22
Abbildung 6: Das Standardzertifikat 24

1. Abstract

This project was released by the Zurich University of Applied Science Winterthur. It is about manipulating X.509 certificates to test the stability of different applications. First of all an application has to be developed to create and manipulate the certificates. Then the manipulated certificates must be tested. This application should be written in Java or C/C++. The tests are performed with “OpenSSL” and “strongSwan” under Linux and with the standard Windows certificate handler under Windows.

1.1. Einleitung

Bei dieser Arbeit handelt es sich um eine Projektarbeit (PA) der Zürcher Hochschule Winterthur. Es soll die Stabilität verschiedener Programme, welche X.509 Zertifikate verwenden, überprüft werden. Dazu muss ein Programm entwickelt werden, das es erlaubt die Zertifikate zu generieren und zu manipulieren. Die manipulierten Zertifikate werden unter Windows und Linux getestet.

2. Aufgabenstellung

Sichere Netzwerkkommunikation (SNK)

Projektarbeiten im Sommersemester 2004 - PA2 Sna01

Attacken mit manipulierten X.509 Zertifikaten

Studierende:

- Dominik Riedweg, IT3b
- Hannes Zürcher, IT3b

Partnerfirma:

- c't magazin für computer techik, Heise Verlag, Hannover

Termine:

- Ausgabe: Dienstag, 18.05.2004, 15.00 Uhr im E523
- Abgabe: Freitag, 2.07.2004

Beschreibung:
In vielen sicheren Netzwerkprotokollen (IPsec, SSL/TLS, S/MIME, ssh, etc.) wird die Benutzer- und/oder Rechnerauthentisierung auf der Basis von X.509 Zertifikaten durchgeführt. Weil diese Zertifikate als relativ komplexe ASN.1 Objekte (Abstract Syntax Notation #1) definiert sind und mittels der Distinct Encoding Rules (DER) binär codiert werden, waren sie bis vor kurzem vor Netzwerkattacken verschont geblieben. Dies hat sich geändert, seit in den letzten Monaten schwerwiegende Schwächen in den ASN.1 Parsern von OpenSSL und Microsoft aufgedeckt wurden.

Ihm Rahmen dieser Projektarbeit soll ein Tool geschaffen werden, welches es erlaubt, gezielt syntaktisch falsche X.509 Zertifikate zu generieren. Damit soll die Robustheit der ASN.1 Parser in diversen Applikationen getestet werden (strongSwan, OpenSSL, Microsoft Crypto Library, etc.)

Aufgaben:
Einarbeiten in die ASN.1 Syntax und die BER Codierung
Aufstellen möglicher X.509 Fehlerszenarien
Implementation des X.509 Zertifikatstools
Testruns mit manipulierten Zertifikaten
Auswertung der Resultate

Infrastruktur / Tools:
Raum: **E523**
Rechner: 2 Rechner unter Linux / Windows XP
SW-Tools: OpenSSL, strongSwan

Literatur / Links:

Microsoft Security Bulletin MS04-007
ASN.1 Vulnerability Could Allow Code Execution
eEye Digital Security
Microsoft ASN.1 Library Length Overflow Heap Corruption
NISCC Vulnerability Advisory
Vulnerability Issues in OpenSSL
IETF RFC 3280
PKIX - Certificate and CRL Profile
KSy-Script
Abstract Syntax Notation One (ASN.1)
ITU-T Recommendation X.509
The Directory: Public-Key and Attribute Certificate Frameworks
ITU-T Recommendation X.680
Abstract Syntax Notation One (ASN.1): Specification of basic notation
ITU-T Recommendation X.690
ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)
Linux strongSwan Distribution
www.strongswan.org
OpenSSL Home
www.openssl.org

Winterthur, 17. Mai 2004

Prof. Dr. Andreas Steffen

2.1. Pflichtenheft

Es muss eine Applikation geschrieben werden, welche es auf einfache Art und Weise erlaubt, sowohl korrekte, als auch manipulierte X.509 Zertifikate zu erstellen. Um das Manipulieren komfortabel zu ermöglichen, muss der ASN.1 Header unabhängig von den Daten geändert werden können.
Die Applikation soll in C++ oder in Java geschrieben werden.

Die erzeugten Zertifikate werden mit Windows und Linux getestet. Unter Linux werden die Programme Openssl und StrongSwan getestet. Bei Windows wird der Zertifikathandler des Betriebssystems getestet.

Das Endprodukt dieser Arbeit ist ein Paket mit manipulierten Zertifikaten und ein Programm zum Erstellen der manipulierten Zertifikate. Das Paket mit den Zertifikaten dient dazu, eine einfachere Überprüfung neuer Software zu ermöglichen

3. Termine und Arbeitsteilung

Die Projektarbeit wurde am 17. Mai 2004 ausgeben.
Abgabetermin ist der 2. Juli 2004.

3.1. Beschreibung der Arbeitspakete

Die Projektarbeit kann in verschiedene Arbeitspakete zerlegt werden. Da sich diese Arbeit jedoch schlecht aufteilen liess, wurden praktisch alle Arbeitspakete von beiden Studenten bearbeitet. Die folgende Tabelle gibt eine Übersicht über die verschiedenen Arbeitspakete und deren Aufteilung.

1	Pflichtenheft	Erstellen eines Pflichtenhefts	Beide
2	Einarbeiten	Einarbeiten in die X.509, ASN.1 und BER Spezifikationen	Beide
3	Fehlerquellen	Aufstellen möglicher Fehlerszenarien und dadurch die Anforderungen an das Cert-Tool anpassen	Beide
4	Feldtypen	Erstellen der verschiedenen Feldtypen (ASN.1 Strukturen) mit deren DER-kodierung, sowie auch deren Manipulation	Beide
5	X.509 Struktur	Erzeugen der X.509 Struktur und damit des Hauptprogramms	Domi
6	Signatur	Erzeugen einer gültigen Signatur und Anfügen derselben	Hannes
7	config.file	Erstellen der Konfigurationsdateien und damit Erstellen der manipulierten Zertifikate	Beide
8	Windowstests	Testen der Zertifikate mit dem Windows Zertifikathandler	Domi
9	Linuxtests	Testen der Zertifikate unter Linux. Das heisst mit den Programmen strongSwan und openssl	Hannes
10	Dokumentation	Erstellen der Dokumentation	Beide

Abbildung 1: Übersicht über die Arbeitspakete

3.2. Projektmeilensteine und Termine

Die Projektarbeit wurde am 17. Mai 2004 ausgeben. Die ersten zwei Wochen wurden für das Einarbeiten und das Erstellen verschiedener Fehlerszenarien reserviert. Danach musste innerhalb von drei Wochen das Tool zum Erstellen der Zertifikate implementiert werden. Somit blieben zwei Wochen für das Erzeugen von Zertifikaten und das Erstellen der Dokumentation. Für die Dokumentation musste mindestens

eine Woche eingerechnet werden, so dass für die Tests nur noch eine Woche übrig blieb.
Abgabetermin ist der 2. Juli 2004.

3.3. Balkendiagramm (Gantt)

= Beide
= Dominik Riedweg
= Hannes Zuercher

Start date: 05 / 17 / 2004

Project: X.509 Zertifikate

Current Week							
Weeks **Tasks**	05/17 to 05/24	05/24 to 05/31	05/31 to 06/07	06/07 to 06/14	06/14 to 06/21	06/21 to 06/28	06/28 to 07/05
1) Pflichtenheft							
2) Einarbeiten in ASN.1, DER-Kodierung und x.509 Zertifikate							
3) Überlegungen zu möglichen Fehlerquellen							
4) Cert-Tool: Erstellen der verschiedenen Feld-Typen							
5) Cert-Tool: Erzeugen der x.509 Struktur							
6) Cert-Tool: Gültige Signatur erzeugen							
7) Cert-Tool: Erstellen von Konfigurationsdateien (bzw. Zertifikaten)							
8) Testen der Zertifikate unter Windows (Zertifikathandler des BS)							
9) Testen der Zertifikate unter Linux (strongSwan, openssl)							
10) Dokumentation							
Tasks Weeks	05/17 to 05/24	05/24 to 05/31	05/31 to 06/07	06/07 to 06/14	06/14 to 06/21	06/21 to 06/28	06/28 to 07/05
Current Week							

Abbildung 2: Gantt-Diagramm

4. Übersicht über die Grundlagen

Dieser Abschnitt gibt einen Überblick über die nötigen Techniken, welche zum verstehen, beziehungsweise erstellen, von X.509 Zertifikaten nötig sind. Allerdings enthält dieser Abschnitt nur eine grobe Beschreibung der Techniken. Die Verweise auf die entsprechenden OSI/ITU-T Spezifikationen finden sich im Anhang.

4.1. ASN.1 / DER-Kodierung

Die Abstract Syntax Notation #1 (ASN.1) ist eine formale Sprache zum abstrakten Beschreiben von Nachrichten. Sie dient als standardisierte Schnittstelle zwischen sehr vielen verschiedenen Anwendungen, welche Daten über ein Netzwerk versenden, beziehungsweise empfangen. Zum Beispiel benützen Anwendungen im Internet, bei Handys, für interaktives Fernsehen oder bei Voice Over IP diese Schnittstelle.
ASN.1 muss grundsätzlich noch kodiert werden, da es nur die syntaktischen Regeln für die Beschreibung von Datenfeldern definiert. Bei diesen Datenfeldern handelt es sich zum Beispiel um Datentypen oder zusammengesetzte Daten. Zusammengesetzte Daten können unter anderem Sequenzen und Strukturen sein. Die Kodierung definiert, wie die Daten in einen Datenstrom umgesetzt werden. Mögliche Kodierungen sind zum Beispiel DER oder BER.

Hier ein kleines Beispiel einer ASN.1 Syntax (nicht kodiert):

```
Report ::= SEQUENCE {
    author      OCTET STRING,
    title       OCTET STRING,
    body        OCTET STRING,
}
```

4.2. DER-Kodierung

Die syntaktischen Elemente der obenstehenden ASN.1 Nachricht (Report) sind:

-SEQUENCE
-OCTET STRING

Wird diese Nachricht nun DER-kodiert, so wird zuerst der Identifier selbst kodiert. Eine SEQUENCE wird mit 0x30 und der OCTET STRING mit 0x04 kodiert. Anschliessend wird die Länge der Daten bestimmt und diese mit den eigentlichen Daten ebenfalls DER-kodiert. Dieser Prozess wird mit den restlichen Feldern wiederholt und das Resultat wird in ein File geschrieben.

Hier die Pseudo-Kodierung des ASN.1 Syntax Beispiels (ohne Berechnung der Längenfelder, sowie ohne Daten; alle Zahlen in Hexadezimaldarstellung):

ASN.1:	SEQUENCE	author	title	body
DER:	30 Länge	04 Länge Daten	04 Länge Daten	04 Länge Daten

Da X.509 Zertifikate immer DER-kodiert sind, wird hier nur die DER-Kodierung behandelt.

4.3. X.509

Ein X.509 Zertifikat besteht im Wesentlichen aus den folgenden ASN.1 Feldern, welche DER-kodiert und so gespeichert werden. Zusätzlich wird noch eine Signatur erstellt und dem File angefügt.

```
Certificate ::= SEQUENCE {
    tbsCertificate          TBSCertificate,
    signatureAlgorithm      AlgorithmIdentifier,
    signature               BIT STRING
    }

TBSCertificate ::= SEQUENCE {
    version                 [ 0 ] Version DEFAULT v1(0),
    serialNumber            CertificateSerialNumber,
    signature               AlgorithmIdentifier,
    Issuer                  Name,
    validity                Validity,
    subject                 Name,
    subjectPublicKeyInfo    SubjectPublicKeyInfo,
    ssuerUniqueID           [ 1 ] IMPLICIT UniqueIdentifier OPTIONAL,
    subjectUniqueID         [ 2 ] IMPLICIT UniqueIdentifier OPTIONAL,
    extensions              [ 3 ] Extensions OPTIONAL
    }
```

Die TBSCertificate-Sequence (TBS steht für To Be Signed) stellt den zu signierenden Teil eine X.509 Zertifikates dar. Über diese TBS-Sequence wird ein Hash-Wert gebildet und mit dem Private-Key des Ausstellers verschlüsselt (den verschlüsselten Hash-Wert nennt man Signatur). Somit kann leicht festgestellt werden, ob der Inhalt der signierten Felder geändert wurde. Der Public-Key des Ausstellers muss dazu vorhanden sein. Deshalb werden „Selfsigned“ Zertifikate benötigt, die den Public-Key des Ausstellers beinhalten. Um die Signatur zu prüfen muss nur der verschlüsselte Hash-Wert mit dem Public-Key des Ausstellers entschlüsselt werden und mit dem errechneten Hash-Wert verglichen werden. Wenn

der errechnete mit dem entschlüsselten Hash-Wert übereinstimmt, ist das Zertifikat nicht manipuliert worden. Die restlichen Felder müssen nicht vor Änderungen geschützt werden, da es sinnlos ist sie zu fälschen. Ein falscher Hash-Algorithmus-Indentifier führt zu einer ungültigen Signatur. Andere Feldern in sinnvoller Weise zu fälschen ist kaum möglich, zum Beispiel ist es praktisch unmöglich die Felder so manipulieren, dass ein gewünschter Hash-Wert entsteht.

Es können noch einige zusätzliche Felder im Zertifikat stehen, sie sind jedoch optional und werden von vielen Parsern nicht verarbeitet, da ihr Wert sowieso nicht benötigt wird.

5. Das Zertifikat-Tool (Cert)

In diesem Kapitel werden das Programm und die Konfigurationsdateien beschrieben.

5.1. Einleitung

Am Anfang dieser Projektarbeit musste definiert werden, welche Manipulationen an einem Zertifikat möglich sein sollen. Es zeigte sich schnell, dass diese Manipulationen nicht einfach durch das Manipulieren eines gültigen Zertifikates erreicht werden können. Deshalb wurde entschieden ein Tool zu erstellen, welches sowohl das Manipulieren, als auch das korrekte Signieren, der Zertifikate erlaubt. Dieses Tool muss so geschrieben werden, dass es möglichst flexibel eingesetzt werden kann, um einerseits manipulierte, anderseits korrekte Zertifikate zu erstellen.

5.2. Implementation

Das Tool wurde komplett in Java geschrieben, da Java bereits über eine gute Unterstützung verschiedener kryptographischer Algorithmen verfügt. Dass die Zertifikate unter Linux und Windows getestet wurden, war ein weiterer Punkt, der für Java sprach. Mit Java kann das Tool auf beiden Betriebssystemen ausgeführt werden. Als Editor wurde Eclipse verwendet. Zusätzlich wurde entschieden, aus Zeitgründen auf eine graphische Benutzeroberfläche (GUI) zu verzichten, dafür aber Konfigurationsfiles zu unterstützen.
Damit auch inkorrekte ASN.1 Strukturen erzeugt werden können, konnten keine fertigen ASN.1 Klassen verwendet werden, sondern es mussten die entsprechenden Klassen selbst entwickelt werden. Es wurde eine Superklasse *ByteArray* erstellt, welche die Grundstruktur für die ASN.1 Struktur und die DER Kodierung bereitstellt. Die Klassen, welche die ASN.1 Strukturen repräsentieren, sind von dieser Klasse abgeleitet.

5.2.1. Cert.java:

Die Klasse *Cert* enthält das Hauptprogramm, das heisst die *main*-Methode. In dieser Methode wird die Struktur eines X.509 Zertifikates erstellt. Durch Abändern dieser Klasse könnten beliebige ASN.1/DER Strukturen erzeugt werden. Um ein Zertifikat zu erzeugen werden zuerst die Daten aus dem Konfigurationsfile gelesen. Diese Daten werden den entsprechenden Objekten übergeben, und damit die Daten des eigentlichen Zertifikates erzeugt. Diese Daten werden von der Klasse *MessDig* signiert und dann als Zertifikat in ein File geschrieben.

5.2.2. ByteArray.java:

Die Klasse *ByteArray* ist die Superklasse aller ASN.1 Strukturen. Sie enthält unter anderem zwei Arrays vom Typ Byte. Das erste Array wird benötigt, um den Header und das Längenfeld zu kodieren, das zweite um die Daten zu speichern. Zusätzlich werden einige Methoden und Variablen zur Verfügungen gestellt. Zum Beispiel die wird die *length*-Methode für alle ASN.1 Strukturen verwendet und deshalb in der Superklasse erstellt. Die *length*-Methode kodiert die Länge der Daten im Header. Dasselbe gilt für die *writeHex*-Methode, welche direkt hexadezimalkodierte Daten an die entsprechende Stelle schreibt.

5.2.3. Die ASN.1 Strukturen wie Set.java, PrintableString.java und OID.java

Diese Klassen erzeugen eine entsprechende ASN.1 Struktur. Der Konstruktor dieser Klassen verlangt ein Array vom Typ Byte und einen String. Das Array wird am Schluss in das File geschrieben; es dient also dazu, alle Strukturen geordnet zu buffern. Der String ist entweder *null,* wenn der Standartheader mit dem korrekten Längenfeld geschrieben werden soll, oder er enthält einen Hexadezimalwert, welcher ohne Überprüfung geschrieben wird. Dadurch ist es möglich, sowohl den Header, als auch das Längenfeld zu manipulieren. Die Klassen enthalten zusätzlich eine *writeDaten*-Methode. Diese ermöglicht es, die Daten zu schreiben. Dieser Methode wird ein String übergeben, welcher entweder die korrekten Daten oder einen Hexadezimalwert, welcher ohne Überprüfung geschrieben wird, enthält.

5.2.4. ConfigFile.java

Die Klasse *ConfigFile* dient dazu, die Daten aus dem Konfigurationsfile einzulesen und den entsprechenden Variablen zuzuweisen. Der Name des Konfigurationsfiles kann dem Programm als Argument übergeben werden. Wird kein Argument übergeben, wird das File „config.file“ als Standart genommen.

5.2.5. MessDig.java

Die Klasse *MessDig* vervollständigt das Zertifikat. Die Signatur wird berechnet, angefügt und das File geschrieben.

5.3. Klassenübersicht

Die untenstehende Abbildung zeigt eine Übersicht der im Cert-Tool verwendeten Klassen. Die Klasse *ConfigFile* ist auf dieser Darstellung nicht gezeichnet. Die Klasse enthält zu viele Variablen, so dass sie nicht mehr auf die Graphik passt.

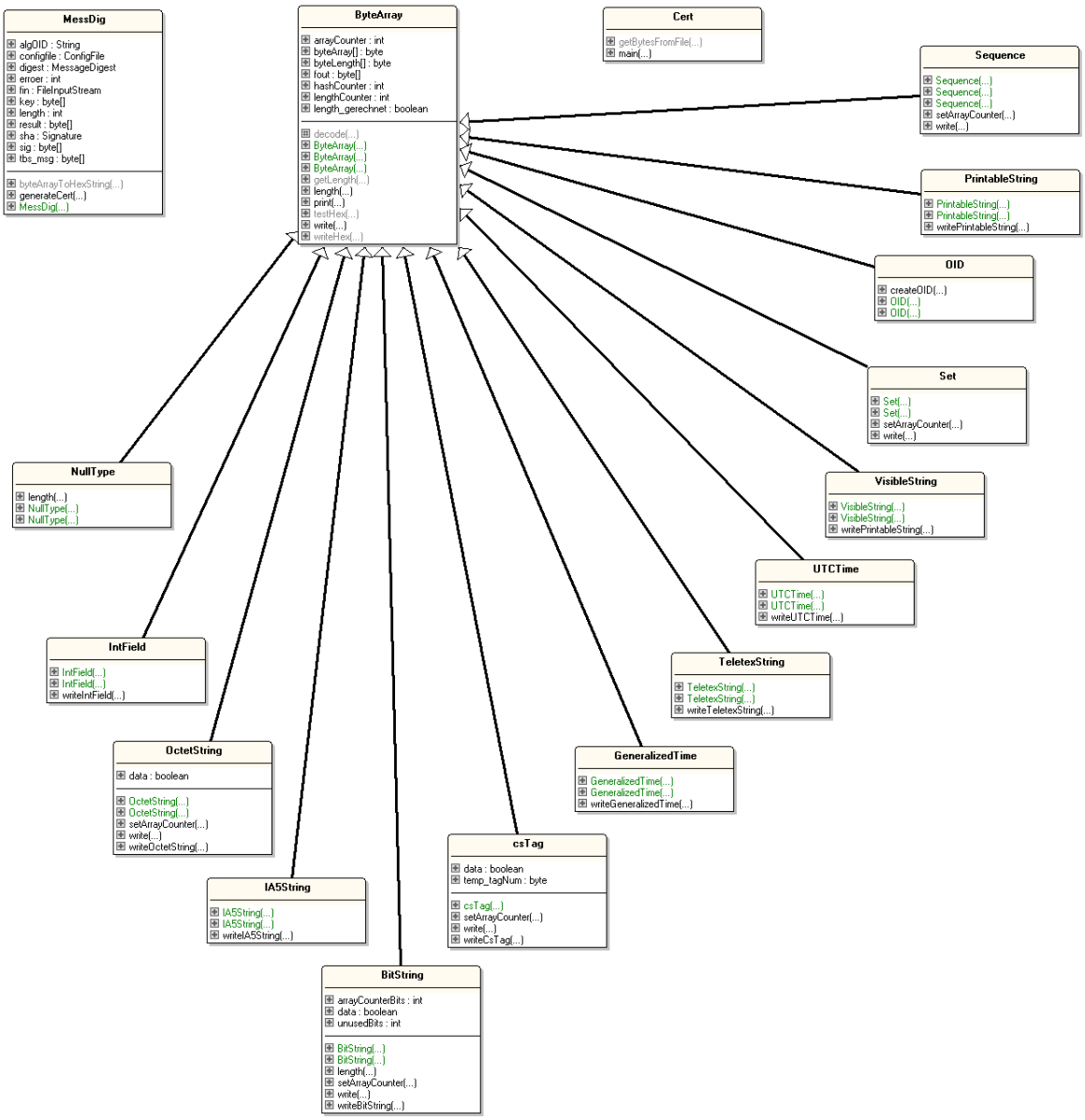

Abbildung 3: Ausschnitt der Klassenstruktur des Cert-Tools

5.4. Sequenzieller Ablauf des Programms

Die *main*-Methode befindet sich in der Klasse *Cert.* In dieser *main*-Methode wird zuerst die Klasse *ConfigFile* geladen und damit das Konfigurationsfile ausgelesen. Danach wird die Struktur eines X.509 Zertifikates erzeugt, indem Instanzen der entsprechenden Objekte erzeugt werden. Diese enthalten als Instanzvariablen Bytearrays, um den Header und die Daten aus dem Konfigurationsfile zu speichern. Nachdem die Struktur erzeugt und damit auch die Längen definiert wurden, werden die *write*-Methoden dieser Objekte aufgerufen. Dies führt dazu, dass die Daten in ein Array geschrieben werden. Dieses Array wird nun dem *MessDig* Objekt übergeben

und mit dem geladenen Privat-Key signiert. Danach wird das Zertifikat in ein File geschrieben.

5.5. Handbuch

Das *Cert*-Tool ist ein Java-Programm. Es wird mit dem Befehl „javac Cert.java“ kompiliert. Der Programmaufruf erfolgt über den Befehl „ java Cert config.file“, wobei „config.file“ durch das neu erstellte Konfigurationsfile ersetzt werden kann. Wird kein Konfigurationsfile angegeben, wird im selben Verzeichnis nach einem File mit dem Namen „config.file“ gesucht. Falls dieses File gefunden wird, verwendet es das Tool, sonst wird die Ausführung abgebrochen. Zusätzlich wurde ein Jar-File erstellt, das mit dem Befehl „java -jar Cert.jar <config.file>“ gestartet werden kann.

Der Public-Key, der in das Zertifikat eingefügt wird, kann sowohl aus einem File eingelesen, als auch durch die entsprechenden Einträge im Konfigurationsfile erzeugt werden. Der Private-Key, der benötigt wird um das Zertifikat zu signieren, muss in jedem Fall geladen werden. Die genaue Position des Private-Keys muss im Konfigurationsfile angegeben werden.

Das Zertifikat wird im Unterordner „Certs“ generiert. Dieser Ordner muss bereits bestehen, da sonst das Programm abgebrochen wird. Das neu erstellte Zertifikat erhält den Namen des Konfigurationsfiles mit der Endung „.der“.

Die Manipulationen entstehen durch Änderungen am *config.file,* welches im nächsten Kapitel separat beschrieben wird.

Häufig verwendete Befehle:
Um Zertifikate von bzw. in DER/PEM zu wandeln wurde OpenSSL verwendet. Die Parameter -inform und -outform müssen nur angegeben werden, falls nicht das Standartformat PEM verwendet wird:
openssl x509 -in <infile> -inform <DER/PEM> -out <outfile> -outform <DER/PEM>

Private- und Public-Keys wurde ebenfalls mit Openssl erstellt:
Der Private-Key wird erstellt mit:
openssl genrsa -out <keyfile> <numberofBits>

Der Public-Key wird mit folgendem Befehl aus dem Private-Key extrahiert:
openssl rsa -in <infile> -inform <DER/PEM> -out <outfile> -outform <DER/PEM> -pubout

Um die Kodierung der Keys zu ändern wird folgender Befehl verwendet:
openssl rsa -in <infile> -inform <DER/PEM> -out <outfile> -outform <DER/PEM>

Damit der Private-Key vom Cert-Tool zum Signieren verwendet werden kann muss er ins DER-kodierte PCKS#8-Format gebracht werden:
openssl pkcs8 -topk8 -in <infile> -inform <DER/PEM> -out <outfile> -outform DER -nocrypt

Da diese Befehle häufig gebraucht werden wurden einige Shell-Skripte geschrieben:

./generate_certs.sh :
Dieses Skript generiert Zertifikate für alle Konfigurationsfiles, die sich in diesem Ordner befinden und die Endung „.cfg“ aufweisen. Ausserdem wird, mit Hilfe von OpenSSL, jedes erzeugte DER-Zertifikat in das PEM-Format gewandelt, sofern dies möglich ist.

./generate_single_cert.sh <config.file> :
Dieses Skript ist erfüllt den gleichen Zweck wie *generate_certs.sh*. Als Parameter wird jedoch ein Konfigurationsfile übergeben. Es wird nur dieses Zertifikat generiert und in PEM gewandelt.

./Certs/openssl_verify_all.sh:
Es werden alle PEM Zertifikate in diesem Ordner mit OpenSSL verifiziert. Als CA-Zertifikat wird *./demoCA/cacert.pem* verwendet.

./Certs/openssl_verify_single.sh <certificate.pem> :
Das übergebene Zertifikat wird einzeln mit OpenSSL verifiziert. Auch hier wird ./demoCA/cacert.pem als CA-Zertifikat verwendet.

./Certs/openssl_verify_wildcards.sh <part_infile>:
Dieses Skript verifiziert alle Zertifikate im PEM-Format, deren Name mit <part_infile> beginnt (<part_infile>*.pem). Auch hier wird ./demoCA/cacert.pem als CA-Zertifikat verwendet.

./Certs/openssl_verify_wildcards_textout.sh <part_infile>:
Im Unterschied zum Shell-Skript ./Certs/openssl_verify_wildcards.sh wird hier das Zertifikat zusätzlich noch auf die Konsole ausgegeben.

5.5.1. Das Konfigurationsfile

Beim Konfigurationsfile handelt es sich um ein Textdokument. Es enthält alle Daten des Zertifikates und die Manipulationen an den Hadern und Längenfeldern. Der folgende Ausschnitt aus einem Konfigurationsfile zeigt, wie die Daten angegeben werden müssen.
Es gilt: Feldname: Daten : Header
Die Feldnamen sind vorgegeben und dürfen nicht verändert werden. Die Daten werden entweder unkodiert notiert oder als Hexadezimalwerte (mit führendem 0x) geschrieben. Die Header können nur als Hexadezimalwerte verändert werden, sonst wird automatisch der korrekte Wert geschrieben. Bei der Manipulation der Header ist zu beachten, dass die Länge auch übergeben werden muss (siehe Feld *version* in der untenstehenden Abbildung; 02 steht für Integer und 01 für die Länge). Um ein Feld nicht zu beschreiben, muss sowohl im Header, als auch im Datenfeld der Wert „0x“ stehen (siehe Feld *tag_3*). Eine Ausnahme bilden hier die Sequenzen und die Sets, die nur aus einem Header bestehen und entsprechend nur ein „0x“ im Datenfeld geschrieben werden muss. Ein allfälliger Eintrag im Header wird verworfen. Vorsicht ist bei den *Tag's* geboten: sie können sowohl einen Header wie auch Daten enthalten. Deshalb werden bei den *Tag's* beide Felder ausgewertet. Dies führt dazu, dass eine gültige Tag-Nummer im Header stehen muss. Die hexadezimalkodierten Zahlen dürfen nur kleingeschriebene Buchstaben enthalten. Kommentare beginnen mit den Zeichen „##“ und müssen auf einer eigenen Zeile stehen.

```
##this file contains the field information needed to generate x509
certificates
##it is used by the PA2_Sna_04_1 project.
      ##All comments on a separate line!!
      ##notation: FIELDNAME: VALUE: HEADER

      ##hex values should always be written in 2hex signs...
      ##0x05 and NOT 0x5 ...as 0x5 is interpreted as 0x50!!

##--------------certificate-Sequence {
version: 2 : 0x0201

serial: 3
algoid: 1 2 840 113549 1 1 4

tag_3: 0x : 0x
opt_seq: 0x
basicConstraints_seq: 0x
```

Abbildung 4: Auszug aus dem Konfigurationsfile

6. Testen der Zertifikate

In diesem Kapitel wird beschrieben mit welchen Programmen die Zertifikate getestet und wie die Zertifikate manipuliert wurden.

6.1. Übersicht über die Tests

Am Anfang der Projektarbeit wurden verschiedene Manipulationsmöglichkeiten gesucht. Diese Manipulationen werden hier beschrieben und das Cert-Tool musste flexibel genug gestaltet werden, um diese auch zu unterstützen.

1. Die Daten werden so verschachtelt, dass die Länge der inneren Daten grösser ist, als die Länge der umfassenden Sequenz.
2. Es werden Integer erstellt, welche grösser als 4 Bytes sind.
3. Es werden Längenfelder erzeugt, die grösser als 4 Bytes sind.
4. Der Public-Key wird grösser als 8192 Bit erzeugt.
5. Der Exponent des Pubilc-Keys ist eine sehr grosse Zahl.
6. Die Länge des Public-Keys ist nicht ein Mehrfaches von 8 Bits.
7. Das Datum wird im *GeneralizedTime*-Format angegeben.
8. Die Zeitverschiebung wird manipuliert.
9. Die Länge der Daten wird nicht angeben (EOC), was nicht der DER-Kodierung entspricht.
10. Das Zertifikat ist bis nach 2050 gültig.
11. Es werden Octetstrings erzeugt, die länger als 255 Bytes sind.

6.2. Getestete Programme

Es wurden die folgenden Programme getestet:

- StrongSwan
- Openssl
- Zertifikathandler von Windows XP (z.B. im Internet Explorer verwendet)

Das Testen der Zertifikate gestaltete sich unter Windows sehr einfach, da das Zertifikat durch eine Doppelklick geöffnet werden kann.

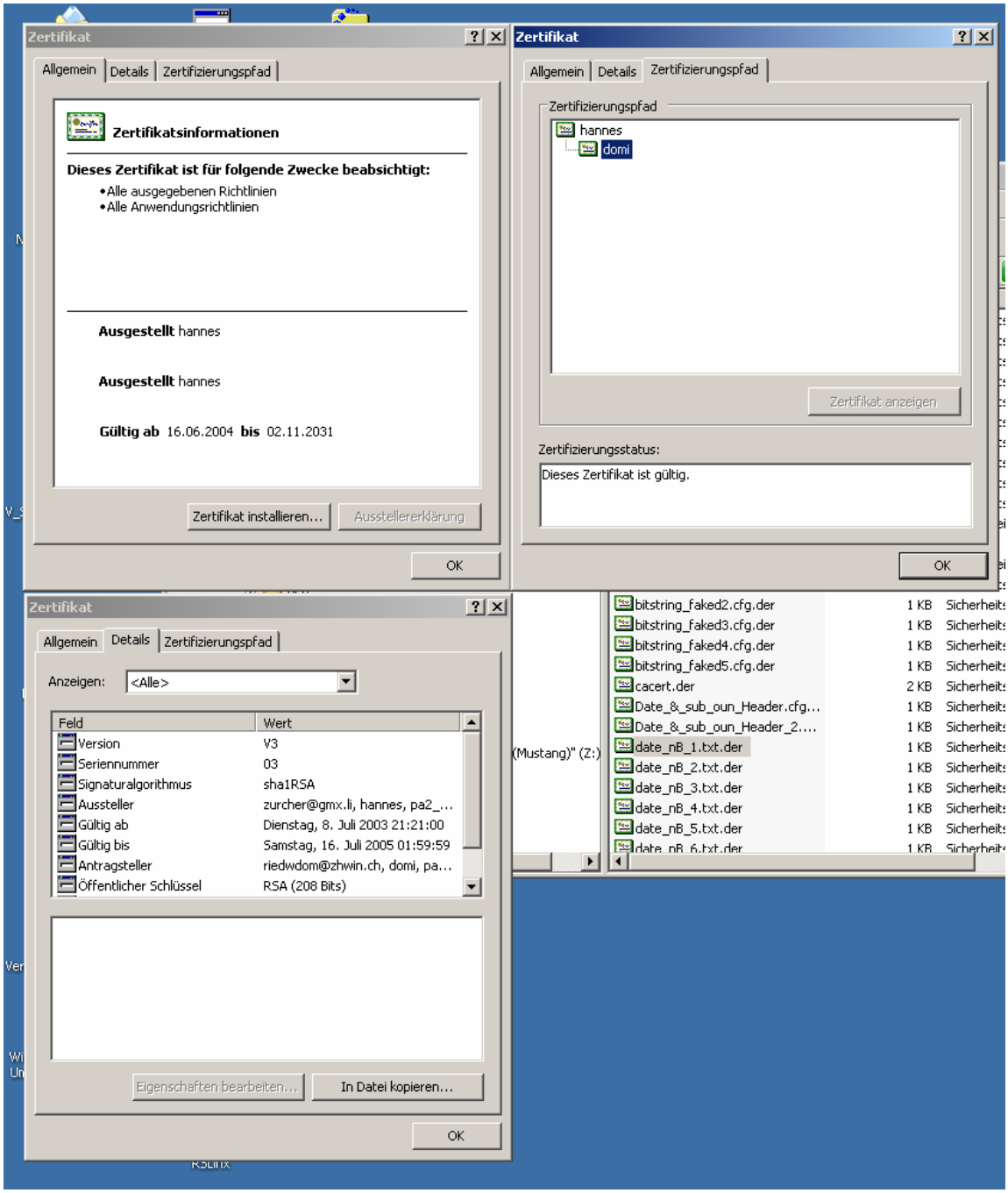

Abbildung 5: Darstellung funktionierender Zertifikate in Windows

StrongSwan lädt alle Zertifikate im Ordner „cacerts“ beim Starten, so dass alle Zertifikate in diesen Ordner kopiert wurden. Mit dem Befehl „ipsec auto --listall“ werden alle geladenen Zertifikate angezeigt.

Für den Test mit Openssl wurde der Befehl „openssl x509 –in FILENAME –inform der –noout –text“ verwendet. Zusätzlich wurde ein kleines Skript geschrieben, welches erlaubt, alle Zertifikate in einem Ordner zusammen zu überprüfen. Dieses Skript befindet sich auf der beiliegenden CD.

7. Resultate

In diesem Kapitel werden die von uns erstellten Zertifikate genauer beschrieben. Beim Erstellen der Zertifikate wurde ein Standardzertifikat verwendet (ohne Fehler). Alle Änderungen beziehen sich auf dieses Zertifikat (die Konfigurationsdatei zum Standartzertifikat ist *config.file*, siehe Anhang).
Zu den einzelnen Zertifikaten werden nur die Felder, welche gefälscht wurden, aufgelistet.

7.1. Standardzertifikat

Dies ist unser Standardzertifikat von welchem alle anderen abgeleitet wurden.

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 2 (0x2)
        Signature Algorithm: md5WithRSAEncryption
        Issuer:    C=ch,    ST=zurich,    L=winti,    O=zhwin,    OU=pa2_sna,
CN=hannes/emailAddress=zurcher@gmx.li
        Validity
            Not Before: Jul 13 00:00:00 2003 GMT
            Not After : Jul 15 23:59:59 2005 GMT
        Subject:    C=ch,    ST=zurich,    L=winti,    O=zhwin,    OU=pa2_sna,
CN=domi/emailAddress=riedwdom@zhwin.ch
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (2048 bit)
                Modulus (2048 bit):
                    00:be:da:07:da:e3:e5:b3:f5:22:31:a8:43:f0:cd:
                    e1:7d:1f:0a:7f:5b:ea:92:fc:3f:db:9d:9c:58:bd:
                    9f:97:7b:47:a9:63:c1:2e:dc:da:61:eb:50:4e:c1:
                    bb:89:b1:34:92:d3:ad:ba:35:7b:9a:f4:6b:fb:d0:
                    d3:0c:c8:3d:e0:db:86:dd:b9:b2:8d:71:80:6b:b5:
                    34:4f:82:d8:e2:0c:de:2f:27:eb:f4:e1:ae:a1:45:
                    29:23:db:72:dd:80:69:ca:47:4a:12:39:ac:f8:37:
                    6f:8c:65:87:46:6a:77:a2:f1:87:3c:dd:2b:73:ea:
                    6f:a3:a1:81:f9:64:d7:8a:47:d2:ec:bb:3a:22:80:
                    cc:fb:0f:b6:d6:96:f3:8b:2a:65:92:e7:a9:d3:34:
                    ff:74:7b:87:6d:39:85:3b:86:f9:a1:b9:3f:54:89:
                    54:fd:e2:b9:5f:5d:d4:50:f3:e4:43:ff:f9:07:fa:
                    8c:71:e8:8f:78:fd:4e:3f:9b:1e:20:16:ff:2c:0b:
                    2c:95:fc:d5:10:61:7d:fa:a2:16:8e:c7:ea:b9:31:
                    a8:fc:32:f7:48:c1:de:99:92:14:3a:71:1d:cb:67:
                    54:a9:4b:76:06:5d:37:0d:65:24:02:ee:a5:17:ad:
                    19:06:e9:27:ce:09:c9:4f:24:0f:6b:dd:83:18:b4:
                    0a:59
                Exponent: 65537 (0x10001)
    Signature Algorithm: md5WithRSAEncryption
```

```
        90:d0:bc:7e:41:7a:25:4c:51:c8:ac:e0:96:f0:36:b7:bd:dc:
        00:39:28:6d:13:14:54:cb:d0:62:53:7e:17:b6:45:9e:33:04:
        e9:57:e1:17:a8:a9:7c:a8:19:a6:57:1b:32:17:88:88:e1:b0:
        a3:29:39:6e:b0:df:af:a6:fb:79:d4:c2:af:43:72:4e:70:aa:
        aa:f2:00:4f:53:f0:3a:bc:a0:56:54:79:cf:03:4d:a4:cf:42:
        29:66:a9:73:f3:0b:ee:08:b6:6d:49:13:a5:96:81:7e:7c:59:
        d0:3f:c3:be:81:aa:39:5f:ca:f2:1c:7b:64:e6:59:85:0b:4a:
        fe:6c:a9:32:ab:7b:08:ba:64:dc:39:8a:43:3f:1c:de:de:d7:
        3c:20:37:ae:f6:df:01:ba:21:aa:87:f6:a2:46:98:bd:b2:64:
        08:98:80:3c:3c:7e:b5:75:83:4d:d0:3a:a9:4b:a7:07:bb:a1:
        cf:f7:39:5b:de:b8:c5:1c:2c:3f:33:2b:6a:02:4f:50:6d:15:
        d2:38:1f:9c:f2:6f:60:e0:83:00:8f:3f:87:7c:7e:82:3a:5f:
        30:f5:cd:bf:d5:7f:4f:9b:ea:65:88:39:1b:8d:30:96:22:53:
        b7:79:40:64:72:44:83:6f:a9:51:cd:4c:13:68:02:a8:d9:77:
        98:f5:0b:fe
```

Abbildung 6: Das Standardzertifikat

Dieses Zertifikat beinhaltet keine optionalen Felder.

7.2. Gefälschte Zertifikate

Im Folgenden werden die Zertifikate nach Testkriterien unterteilt (siehe 6.1 Übersicht über die Test). Die jeweiligen Zertifikate werden nicht einzeln, sondern in zusammengehörenden Gruppen, die sich mit dem gleichen Test beschäftigen, behandelt (zum Beispiel alle Zertifikate mit gefälschtem Datum, alle mit gefälschter Versionsnummer, etc.).

7.2.1. Objekt-Bezeichner

Ziele dieser Tests:
Es wurden verschiedene Objekt-Bezeichner verändert. Da die einzelnen Objekt-Bezeichner vermutlich mit derselben Methode oder Prozedur übersetzt werden, macht es kaum Sinn, jeden einzelnen Objekt-Bezeichner einzeln zu fälschen. Deshalb wurden wenige, jedoch sehr verschiedene, Objekt-Bezeichner gefälscht.

Es wurden ausserordentlich lange Objekt-Bezeichner erzeugt, genau so wie falsche Bezeichner mit vielen (führenden) Nullen oder viel zu grossen Werten innerhalb eines Bezeichners.

Da die Objekt-Bezeichner für die DER-Kodierung mathematisch umgerechnet werden müssen, wurde versucht die Stabilität dieser Umrechnung zu testen. Ausserdem wurden komplett falsche Objekt-Bezeichner verwendet, um zu sehen, wie mit den gelieferten Informationen umgegangen wird.

Erzielte Resultate:
oid_huge_*.cgf:
Windows akzeptiert unbekannte OID's problemlos. Es konnte keine Fehler in den berechneten Werten festgestellt werden. Es wird einfach die manipulierte OID-Nummer ausgegeben.

StrongSwan akzeptiert einige der unbekannten OID's, andere werden verworfen. Im Gegensatz zu Windows wird aber das ganze Zertifikat als ungültig erklärt.

Openssl akzeptiert alle unbekannten OID's. Es wird wie bei Windows die manipulierte OID ausgegeben.

oid_header_*.cfg
Openssl wandelt die Zertifikate mit manipulierten OID-Headern nicht ins PEM-Format, deshalb konnten sie nicht weiter getestet werden.

StrongSwan akzeptiert manipulierte OID-Header ohne Probleme, solange die Werte nicht ausgegeben werden. Bei der Ausgabe mit dem *--listall* Befehl kann es zu einem „Segmentation fault" kommen (oid_header_4.cfg). Dies ist auf ein manipuliertes Längenfeld zurückzuführen. Eigentlich ist dieser Fehler kein Problem, da Zertifikate im normalen Betrieb nicht ausgeben werden, ausser bei aktiviertem Pluto-Debugmode. Das Programm beendet sich dann automatisch und wird von einem Kontrollprozess neu gestartet. Wenn das File nun nochmals abgearbeitet werden soll, stürzt das Programm wieder ab. Dies wiederholt sich kontinuierlich. Wenn der Debugmode in diesem Stadium ausgeschaltet und strongSwan neu gestartet wird, bleibt das Problem bestehen. Das Zertifikat muss gelöscht oder der Computer neu gestartet werden.

Windows ist manipulierten OID-Headern gegenüber sehr tolerant. Die Manipulationen führen dazu, dass das ganze Subject-Feld verworfen wird. Über den Sinn eines Zertifikates ohne Empfänger lässt sich zwar streiten, der Parser scheint jedoch sehr robust zu sein.

Geänderte Felder in der Konfigurationsdatei:
oid_huge1.cfg:
algoid: 1 2 840 113549 1 1 4 2 3 4 4 4 4 403 404 49 49 94 3 3 4 506 05 38 300 494 494 3839 93 394783 2 2 43 39 599

oid_huge2.cfg:
Issuer_organizationalUnitName_Obj: 0 00 00 00 00 00 2 5 4 8 109 010 2920 020 00 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 00 0 0 0 0 0 0 000 00 000 00 00 000 00 00 00 0 0 0 0 0

oid_huge3.cfg:
Issuer_organizationalUnitName_Obj: 0 -800 0 0 0 0 000 00 000 00 00 000 00 00 00 0 0 0 0 0 0

oid_huge4.cfg:
Issuer_organizationalUnitName_Obj: 0 -800 -05 0 0 0 000 00 000 00 00 000 00 00 00 00 0 0 0 0 0 0

oid_huge5.cfg:
Subject_organizationalUnitName_Obj: 0x210242094823048230

oid_huge6.cfg:
Issuer_organizationalUnitName_Obj: 0x210242094823048230ffffffffffffffffffffff ... ffffff
Länge des OID´s : ca. 600Byte

oid_huge7.cfg:
Issuer_organizationalUnitName_Obj: 0x210242094823048230ffffffffffffffffffffff ... ffffff
Länge des OID´s : ca. 130 kByte

oid_header_1.cfg:
Subject_organizationName_Obj: 2 5 4 6 :0x

oid_header_2.cfg:
Subject_organizationName_Obj: 2 5 4 6 :0x80

oi_header_3.cfg:
Subject_organizationName_Obj: 2 5 4 6 :0x0680

oid_header_4.cfg:
Subject_organizationName_Obj: 2 5 4 6 :0x068fff ... ffffffffffffffffffffffffffffffffffffffffffffffff

oid_header_5.cfg:
Subject_organizationName_Obj: 2 5 4 6 :0x061ffffff...ffffff

oid_header_6.cfg:
Subject_organizationName_Obj: 2 5 4 6 :0x0601

7.2.2. Bitstrings

Ziel dieser Tests:
Da Bitstrings, wie ihr Name schon sagt, beliebigen binären Inhalt haben können, haben wir vor allem die Stabilität bezüglich der Unused-Bits getestet. Die Unused-Bits geben an, ob und wieviele Bits, des letzten Bytes, nicht genutzt werden.

BS_1-3 und BS_5 :
Die erzeugten BitStrings sind von der Grösse her inkorrekt. Ausserdem werden einige Bits dieses Feldes nicht benutzt (angegeben durch BitStringUnused). BS_3 erzeugt einen Bitstring mit Länge Null und 7 nicht benutzten Bits, was überhaupt nicht möglich ist.

BS_4 :
Dieser Hexwert bedeutet, dass 127 Längenfelder vorhanden sind und der Public-Key dementsprechen lang wird, die übergebenen Integer (Teile des Public-Keys) sind jedoch viel zu kurz.

Erzielte Resultate:
Windows akzeptiert alle Zertifikate ausser der Nummer 5. Beim Zertifikat Nummer 3 wird eine Public-Key-Länge von null Bits angegeben. Wie mit diesem Schlüssel gearbeitet werden soll, bleibt fraglich, das Zertifikat wird jedoch als korrekt akzeptiert.

StrongSwan akzeptiert nur einzelne Zertifikate. Die anderen als ungültig verworfen.

Openssl verwirft alle Zertifikate ausser der Nummer 3. Die Fehlermeldung deutet auf eine falsche Signatur hin, was jedoch nicht der Fall ist. Das die Nummer 3 akzeptiert wird, überrascht doch sehr, da genau dieses Zertifikat über keinen Public-Key verfügt.

Geänderte Felder in der Konfigurationsdatei:
BS_1.cfg:
subPubKeyBitString: 0x0301
BitStringUnused: 7

BS_2.cfg:
subPubKeyBitString: 0x030100ff
BitStringUnused: 7

BS_3.cfg:
subPubKeyBitString: 0x0300
BitStringUnused: 7
subjectPublicKeyInteger:0x:0x

subjectPublicKeyInteger2:0x:0x

BS_4.cfg:
subPubKeyBitString: 0x03fffffffffff ... fffff
Länge des subPubKeyBitString = ca. 125Bytes
BitStringUnused: 7

BS_5.cfg:
subPubKeyBitString:
BitStringUnused: 11
subjectPublicKeyInteger:0x:0x
subjectPublicKeyInteger2:0x:0x

7.2.3. Date

Ziel dieser Tests:
date_1-9: ändern die GMT-Felder.
date_10,11: ändern die Validity-Sequence
date_12-19: ändern die GeneralizedTime-Felder.

Es wird versucht ungültige Zeiten mit Hilfe von Zeitverschiebungen zu erzeugen. Ausserdem wird getestet was geschieht falls eines der beiden Felder nicht geschrieben wird.
Auch ungültige Daten wie zum Beispiel der 29.02.2005 oder unmögliche Zeitverschiebungen werden getestet.

Erzielte Resultate:
Openssl scheint die Zeitverschiebung nicht zu berücksichtigen. Das Zertifikat date_19.cfg.der sollte eigentlich am 06.06.2006 um 06:06:06+3600 auslaufen, die Zeitverschiebung +3600 wird jedoch ignoriert.

Openssl-Output:

```
        ...
    Validity
      Not Before: Jun  1 12:00:00 2000
      Not After : Jun  6 06:06:06 2006
    ...
```

Allgemein ist openssl bei den manipulierten Daten sehr tolerant. Ungültig Daten werden akzeptiert. Überraschend ist das Zertifikat Nummer 6, welches zwar als korrekt anerkannt wird, bei der Ausgabe der Daten jedoch kein „notBefore“-Feld hat (BadTimeValue).

Eine Zeitverschiebung wird von Windows eingerechnet. Zugrosse Werte werden ab einer bestimmten Stelle abgeschnitten und der Rest verworfen. Die errechneten Werte werden ins Windowsformat umgerechnet. Dadurch sind auch Jahreszahlen wie 10000 möglich. Ungültige Daten führen dazu, dass das Zertifikat verworfen wird. Eine Division als Zeitverschiebung ist nicht gültig, das Zertifikat wird verworfen. Falls eines der Datumsfelder nicht korrekt erkannt wird, ist das ganze Zertifikat ungültig.

StrongSwan kann mit allen Zertifikaten umgehen. Die Ausgabe wird jedoch immer nach einem Zertifikat mit manipuliertem Datum abgebrochen. Beim Zertifikat Nummer 3 wird das „notBefore“ Datum einfach wegelassen.

Geänderte Felder in der Konfigurationsdatei:
date_1.cfg:
Validity_notBefore: 030713000000-999999999
Validity_notAfter: 050715235959Z

date_2.cfg:
Validity_notBefore: 030713000000-99999999999999999999 ... 99999
Validity_notAfter: 050715235959Z

date_3.cfg:
Validity_notBefore: 030713000000/0
Validity_notAfter: 050715235959Z

date_4.cfg:
Validity_notBefore: 0x : 0x
Validity_notAfter: 050715235959Z

date_5.cfg:
Validity_notBefore: 050715235959Z
Validity_notAfter: 050715235959Z

date_6.cfg:
Validity_notBefore: 990000000000Z
Validity_notAfter: 050715235959Z

date_7.cfg:
Validity_notAfter:
Validity_notBefore: 030713000000Z
date_8.cfg:
Validity_notBefore: 030715235959+9

Validity_notAfter: 491231235959+9

date_9.cfg:
Validity_notBefore: 031231235959+
Validity_notAfter: 0507152359592

date_10.cfg:
Val_seq: 0x30ff

date_11.cfg:
Val_seq: 0x3000

date_12.cfg:
GeneralizedTime: true
Validity_notBefore: 20000601120000-0000
Validity_notAfter: 20080601120000-0000

date_13.cfg:
GeneralizedTime: true
Validity_notBefore: 20000601120000/0000
Validity_notAfter: 20080601120000-0000

date_14.cfg:
GeneralizedTime: true
Validity_notBefore: 20000601120000-1.5
Validity_notAfter: 20080601120000-0000

date_15.cfg:
GeneralizedTime: true
Validity_notBefore: 20000601120000-0000
Validity_notAfter: 99991231235959+9999999 .. 999999999

date_16.cfg:
GeneralizedTime: true
Validity_notBefore: 20000601120000Z
Validity_notAfter: 20050229120000Z

date_17.cfg:
GeneralizedTime: true
Validity_notBefore: 20000601120000Z
Validity_notAfter: 20090202125550,988988888...888888888 + 9999999... 99999999

date_18.cfg:
GeneralizedTime: true
Validity_notBefore: 20000601120000Z
Validity_notAfter: 00000101000000-5

7.2.4. Common name

Ziel dieser Tests:
commonname_1,2,3,6:
Es werden verschiedene Sonderzeichen getestet. Auch die Länge des Feldes wird verändert.

commonname_4,5:
Es werden sehr lange Strings erzeugt, die als Subject und einmal als Issuer eingefügt werden, sodass bei einer Überprüfung die Beiden extrem langen Strings (~430kByte) miteinander verglichen werden müssen.

Erzielte Resultate:
Windows akzeptiert alle diese Zertifikate.

Openssl, wie auch StrongSwan, akzeptieren ebenso alle Zertifikate.

Geänderte Felder in der Konfigurationsdatei:
commonname_1.cfg:
Subject_commonName_Str: 0x850000000ff

commonname_2.cfg:
Subject_commonName_Str: \n : 0x13

commonname_3.cfg:
Subject_commonName_Str: flsdkfj slekfjies ... föweärkewoü … emrf,.nm\n\n\n\t\t\t
Länge des Feldes= ca. 450Byte

commonname_4.cfg:
Subject_commonName_Str: uuuuuuuuuuuuu ... uuuuuuuuuuuuuuuuuuuuuuuuuu
Länge des Feldes = ca. 430 kByte

commonname_5.cfg:
Issuer_commonName_Str: uuuuuuuuuuuuu ... uuuuuuuuuuuuuuuuuuuuuuuuuu
Länge des Feldes = ca. 430 kByte

commonname_6.cfg:

Subject_commonName_Str: \n : 0x1301

7.2.5. IA5String

Ziel dieser Tests:
Testen auf Sonderzeichen und sehr lange Strings. Auch zu kurze Header oder falsche Längenfelder wurden getestet.

Erzielte Resultate:
Windows akzeptiert alle diese Zertifikate.

Openssl kann ausser dem Zertifikat Nummer 1 und 5 keines ins PEM-Format wandeln. Die anderen werden akzeptiert.

StrongSwan erkennt 4 der 5 Zertifikate und gibt diese auch korrekt aus.

Geänderte Felder in der Konfigurationsdatei:
ia5string_1.cfg:
Subject_email_Str: \n

ia5string_2.cfg:
Subject_email_Str: \n : 0x1601

ia5string_3.cfg:
Subject_email_Str: \n : 0x16ff

ia5string_4.cfg:
Issuer_organizationalUnitName_Str5: jrjklsdfs...kldrörktjwmr
Länge dieses Feldes= ca. 500kByte

ia5string_5.cfg:
Subject_email_Str: slsopdjfofj l335§% ... §$5&$%/&% 6 456öok
Länge dieses Feldes= ca. 350Byte

7.2.6. Integer Header

Ziel dieser Tests:
Integer können in der BER-Kodierung beliebig lang werden, in der DER-Kodierung ist die Länge jedoch begrenzt. Es wird getestet, ob die Dekodierung strickt das DER-Format einhält. Ausserdem werden die Header geändert.

Erzielte Resultate:
Windows scheint die Serienummer modulo 65536 zu rechnen, integer_2.cfg wird als Serienummer 2 ausgegeben. Das Zertifikat Nummer 1 wird korrekt erkannt, obwohl die Länge offensichtlich falsch ist. Die anderen Zertifikate werden nicht aktzeptiert.

Openssl wandelt nur die Nummern 1 und 2 ins PEM-Format. Diese Zertifikate werden korrekt erkannt.

StrongSwan gibt auch nur ein Zertifikat aus. Dieses wird korrekt erkannt.

Geänderte Felder in der Konfigurationsdatei:
integer_1.cfg:
subjectPublicKeyInteger:0x872947829... 78923498cc88787d : 0xffffffffffffffff...fffff

integer_2.cfg:
serial: 65538

integer_3.cfg:
serial: 0xffffffffffffffffffffffff...fffffffffff5eebe4cb242390ef7de2f

integer_4.cfg:
serial: -1

integer_5.cfg:
serial: 65538 : 0x11

7.2.7. NullType

Ziel dieser Tests:
Reaktion der Parser auf fehlende oder ungültige Null-Type-Felder.

Erzielte Resultate:
Windows und openssl akzeptieren alle drei Zertifikate. StrongSwan akzeptiert nur eines dieser Zertifikate.

Geänderte Felder in der Konfigurationsdatei:
null_1.cfg:
null-1: 0x

null_2.cfg:
null-2: 0x

null_3.cfg:
null-2:0xaa

7.2.8. Public Key

Ziel dieser Tests:
Da beliebig lange Public-Keys verwendet werden dürfen, wurde hier versucht, sehr grosse und sehr kleine (unsinnig kleine) Integer zu verwenden. Ausserdem werden sehr grosse gültige Keys verwendet, um Zertifikate gegenseitig zu validieren. Dies könnte einen enormen Rechenaufwand verursachen.

Erzielte Resultate:
Es konnten einige der gewünschten Zertifikate nicht erstellt werden, da die Standard Java Crypto Library keine RSA-Key's akzeptiert, welche grösser als 2048 Bits sind. Nicht erstellt werden konnten: pubkey_9.cfg und pubkey_12.cfg. Diese Zertifikate wurden deshalb auch nicht getestet.

Windows akzeptiert die meisten dieser Zertifikate. Die angegebenen Schlüssellängen sind teilweise erstaunlich, bringen Windows jedoch nicht in Bedrängnis.

StrongSwan kann das Zertifikat pubkey_11.cfg nicht ausgeben (d. h. der Prozessor blieb während der Testphase voll ausgelastet, so dass die Ausgabe abgebrochen wurde). Von den restlichen Zertifikaten werden nur zwei als gültig anerkannt.

Openssl anerkennt die Signatur beim Zertifikat Nummer 11 nicht als gültig. Die andern Zertifikate werden als gültig anerkannt.

Geänderte Felder in der Konfigurationsdatei:
Falls der Public-Key durch Integer festgelegt wird, muss das Feld PublicKey_file immer leer sein, sonst wird das angegebene File verwendet!

pubkey_1.cfg:
subjectPublicKeyInteger:0x872947829 ... 000000000000002
Länge dieses Feldes = ca.500Bytes
subjectPublicKeyInteger2:0x872947829842 ... 0000000002
Länge dieses Feldes = ca.500Bytes

pubkey_2.cfg:
subjectPublicKeyInteger:0x872947829 ... 000000000000002
Länge dieses Feldes = ca.500Bytes
subjectPublicKeyInteger2:0x8729...00002 : 0x02effff...fff
Länge dieses Feldes = ca.500Bytes + ca. 150Bytes Header

pubkey_3.cfg:
subjectPublicKeyInteger:0x : 0x
subjectPublicKeyInteger2:0x : 0x

pubkey_4.cfg:
subjectPublicKeyInteger:0xffffffffffffffffffff ... fffffff
Länge dieses Feldes = ca. 16.7MByte
subjectPublicKeyInteger2:65537

pubkey_5.cfg:
subjectPublicKeyInteger:0x872947829842893741242898419a…8923498cc88787d
subjectPublicKeyInteger2:65

pubkey_6.cfg:
subjectPublicKeyInteger:0x8729478298428937412428984…3498cc88787d
subjectPublicKeyInteger2:65232323232323

pubkey_7.cfg:
subjectPublicKeyInteger:0x87294
subjectPublicKeyInteger2:65232323232323

pubkey_8.cfg:
subjectPublicKeyInteger:0x01
subjectPublicKeyInteger2:0x01

pubkey_9.cfg:
PublicKey_file: ./demoCA/huge_pub.der : 0x
Der angegebene Key ist 16400Bit lang (mit openssl generiert)

pubkey_10.cfg:
subjectPublicKeyInteger:0x1
subjectPublicKeyInteger2:0x133333424242423423948723948237489237982374892374239847238947239479247239487234987234987234

pubkey_11: sehr grossen Public-Key Integer (16.7MByte)
pubkey_11_toverify: hat das Subject von pubkey_11 als Issuer

pubkey_12: sehr grossen, gültigen Public-Key (16400Bit)
pubkey_12_toverify: hat das Subject von pubkey_12 als Issuer und wurde mit dem passenden PrivateKey zum PublicKey des Zertifikates pubkey_12 signiert.

7.2.9. Sequence

Ziel dieser Tests:
Es wurde versucht die Länge einer Sequence so zu manipulieren, dass einige Datenfelder nicht vollständig darin enthalten sind, oder dass die Sequence länger ist als das Zertifikat.

Erzielte Resultate:
Keines der getesteten Programme konnte auch nur eines dieser Zertifikate verarbeiten; sie wurden alle kontrolliert verworfen.

Geänderte Felder in der Konfigurationsdatei:

seq_1.cfg:
tbs_seq: 0x30820185

seq_2.cfg:
tbs_seq: 0x30820190

seq_3.cfg:
tbs_seq: 0x30820195

seq_4.cfg:
tbs_seq: 0x30820199

seq_5.cfg:

tbs_seq: 0x30820181

seq_6.cfg:
start_seq: 0x:0x20820299
tbs_seq: 0x30820181

7.2.10. Explicit Tags

Ziel dieser Tests:
Es wurde versucht die Tag-Felder zu manipulieren, so dass ein Parser andere Felder an dieser Stelle erwartet.

Erzielte Resultate:
Keines der getesteten Programme konnte auch nur eines dieser Zertifikate verarbeiten; sie wurden alle kontrolliert verworfen.

Geänderte Felder in der Konfigurationsdatei:
tag_1.cfg:
version_tag: 0xffffffffffffff...ffffffffffffffffffffffffffffff
Länge dieses Feldes= ca.170Byte

tag_2.cfg:
version_tag: 0x0

tag_3.cfg
version_tag: 0x03

tag_4.cfg:
version_tag: 99999999999999999999999999999999999999999999

tag_5.cfg:
version_tag:0x00ffff

tag_6.cfg:
tag_3: : 3

tag_7.cfg:
tag_3: : 3

tag_8.cfg:
tag_3: : 99

tag_9.cfg:
tag_3: 0x0000 : 3

8. Probleme und Verbesserungsmöglichkeiten

In diesem Kapitel werden die Probleme des *Cert-Tools*, sowie mögliche Verbesserungen beschrieben.

8.1. Probleme

Ein Problem des *Cert-Tools* ist, dass für die Manipulation der Zertifikate Kenntnisse über ASN.1 und DER nötig sind. Das *Cert-Tool* erlaubt es nicht, eine Übersicht über das generierte Zertifikat zu erhalten, dazu ist ein externes Tool, wie der ASN1Viewer (siehe CD), nötig.

Ein weiteres Problem ist die fehlende Unterstützung der optionalen Strukturen. Einige Strukturen sind zwar vorgesehen und können auch manipuliert werden, andere jedoch müssen direkt als Hexadezimalwerte eingegeben werden, was relativ mühsam ist.

Eine Unschönheit in der Programmierung ist sicherlich, dass das Byte Array, in welches das ganze Zertifikat geschrieben wird, mit einer festen Grösse erzeugt wird. Diese feste Grösse führt theoretisch zu einer Einschränkung der möglichen Zertifikate, allerdings wurde diese Grenze bisher nie erreicht. Alle anderen Arrays werden in der benötigten Grösse erzeugt.

Bei sehr grossen Zertifikaten (einige Megabytes) tritt eine „Java Out of Memory Exception“ auf. Diese kann verhindert werden, wenn das Programm mit dem Befehl „java -Xmx512m Cert IHR-CONFIG-FILE” aufgerufen wird.

Der „ASN.1 Viewer“, der benötigt wird um die erzeugten Zertifikate darzustellen, stürzt bei einigen manipulierten Zertifikaten ab.

Viele der manipulierten Zertifikate lassen sich nicht in das PEM-Format wandeln. Diese können zwar theoretisch im DER-Format von *openssl* ausgeben werden, aber nicht für eine Verbindung benützt werden.

Es konnten einige der gewünschten Zertifikate nicht erstellt werden, da die Standard Java Crypto Library keine RSA-Key’s akzeptiert, welche grösser als 2048 Bits sind. Dieses Problem könnte behoben werden, indem eine andere Crypto Library (wie zum Beispiel BouncyCastle) verwendet wird.

8.2. Ausbaufähigkeit und Verbesserungen

Sinnvoll wäre wohl, wenn die Struktur des Zertifikates nicht mehr starr programmiert wäre, sondern während der Laufzeit aus dem Konfigurationsfile ausgelesen würde. Dies würde zusätzliche Möglichkeiten der Manipulation eröffnen. Für diese aufwändige Änderung fehlte jedoch die Zeit.

9. Fazit

Leider ist es uns nicht gelungen eine der drei getesteten Applikationen zum Absturz zu bringen (ausser StrongSwan im Debugmode). Ein Test, wie der von uns durchgeführte, kann jedoch nie abschliessend sein.

Der Zertifikathandler von Windows scheint sehr robust zu sein. Zusätzlich ist er erstaunlich tolerant gegenüber manipulierten Zertifikaten.

Der Parser von StrongSwan ist einiges strikter als der von Windows. Dies führt dazu, dass viele Zertifikate als ungültig betrachtet werden. Die Ausgabe der Zertifikate scheint kleine Schwächen zu haben. Es konnten zwei Zertifikate erzeugt werden, die StrongSwan im Debugmode zum Absturz bringen.

OpenSSL hatte gegenüber den andern getesteten Programmen davon profitiert, dass die Zertifikate zuerst ins PEM-Format gewandelt werden mussten. Da diese Wandlung nicht für alle Zertifikate funktionierte, konnte OpenSSL nicht mit allen Zertifikaten getestet werden. Zudem wurden nur die Ausgabe und der Parser getestet. Es bleibt zu Testen, wie Programme, welche die openssl-Funktionen verwenden, mit den Rückgabewerten umgehen, zum Beispiel bei einem gültigen Zertifikat, welches kein Startdatum beinhaltet.

9.1. Schlusswort

Die Zeit, welche für diese Projektarbeit zur Verfügung stand, war sehr knapp bemessen. Dies führte dazu, dass die Tests schlussendlich nicht im gewünschten Umfang gemacht werden konnten. Das Cert-Tool erlaubt zwar schnell manipulierte Zertifikate zu erstellen, jedoch dauerte die Entwicklung des Tools relativ lange. Ein leistungsfähiges Tool zu entwickeln erfordert jedoch immer Zeit und deshalb bei der Funktionalität Abstriche in Kauf zu nehmen, wäre aus unserer Sicht falsch gewesen.

Wir finden es etwas schade, dass uns am Schluss dieser Projektarbeit ein Tool zur Verfügung steht, mit welchem in kurzer Zeit viele Zertifikate erzeugt werden könnten, uns diese Zeit jedoch fehlt.

Trotzdem sind wir mit dem Resultat dieser Projektarbeit zufrieden. Aus unserer Sicht wurde das Pflichtenheft eigentlich eingehalten. Es steht sowohl ein Tool zur Verfügung, um Zertifikate zu manipulieren, als auch ein Paket mit manipulierten Zertifikaten. Es fehlt jedoch die Möglichkeit mit Private-Keys, welche grösser als 2048 Bits sind, zu signieren. Diese Änderung sollte nicht allzu kompliziert sein und kann in kurzer Zeit implementiert werden.

.

10. Anhang

10.1. CD-Listing

```
.
|-- Cert.jar
|-- Certs
|   |-- openssl_verify_all.sh
|   |-- openssl_verify_single.sh
|   |-- openssl_verify_wildcards.sh
|   `-- openssl_verify_wildcards_textout.sh
|-- Doku
|   `-- X509 PA2.doc
|-- config.file
|-- configfiles
|   |-- BS_1.cfg
|   |-- ...
|   |-- ...
|   |-- ...
|   |-- tag_8.cfg
|   `-- tag_9.cfg
|-- demoCA
|   |-- cacert.der
|   |-- cacert.pem
|   |-- index.txt
|   |-- newcerts
|   |   `-- 01.pem
|   |-- openssl.cnf
|   |-- private
|   |   |-- cakey.der
|   |   |-- cakey.nocrypt.p8
|   |   |-- cakey.p8
|   |   |-- cakey.pem
|   |   |-- cakey_nodes.der
|   |   |-- cakey_nodes.nocrypt.der.p8
|   |   |-- cakey_nodes.nocrypt.p8
|   |   |-- cakey_nodes.pem
|   |   `-- cakey_public_nodes.pem
|   |-- privkey.pem
|   |-- serial
|   |-- testcert.cert
|   |-- testcert.der
|   |-- testkey.der
|   |-- testkey.pem
|   |-- testkey_public.der
|   |-- testkey_public.pem
|   `-- testreq.pem
|-- generate_certs.sh
|-- generate_single_cert.sh
|-- src
```

```
|   |-- BitString.java
|   |-- ByteArray.java
|   |-- Cert.java
|   |-- ConfigFile.java
|   |-- GeneralizedTime.java
|   |-- IA5String.java
|   |-- IntField.java
|   |-- MessDig.java
|   |-- NullType.java
|   |-- OID.java
|   |-- OctetString.java
|   |-- PrintableString.java
|   |-- Sequence.java
|   |-- Set.java
|   |-- TeletexString.java
|   |-- UTCTime.java
|   |-- VisibleString.java
|   `-- csTag.java
`-- tools
    `-- InstallAsn1Viewer.bin

8 directories, 141 files
```

10.2. Links

ASN.1
ASN.1 Vulnerability Could Allow Code Execution
eEye Digital Security
Microsoft ASN.1 Library Length Overflow Heap Corruption
NISCC Vulnerability Advisory
Vulnerability Issues in OpenSSL
IETF RFC 3280
PKIX - Certificate and CRL Profile
KSy-Script
Abstract Syntax Notation One (ASN.1)
ITU-T Recommendation X.509
The Directory: Public-Key and Attribute Certificate Frameworks
ITU-T Recommendation X.680
Abstract Syntax Notation One (ASN.1): Specification of basic notation
ITU-T Recommendation X.690
ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)
Linux strongSwan Distribution
www.strongswan.org
OpenSSL Home
www.openssl.org

Java
java.sun.com
BouncyCastle
www.bouncycastle.org
ASN.1 Viewer
www.obj-sys.com/asn1viewer.shtm

10.3. ASN.1 und DER Definitionen

ASN.1 wird durch drei OSI-Standarts definiert:

ISO 8824/ITU X.208 spezifiziert die Syntax (zum Beispiel welches Datenfeld wo in der Nachricht steht oder mit was für einem Datenfeld).

ISO 8825/ITU X.209 spezifiziert die Basic Encoding Rules für ASN.1 (zum Beispiel wie und wo die Längenfelder zu den jeweiligen Datentypen geschrieben werden).

ISO 8825-1/ITU X.690 spezifiziert mehrere Encoding Rules für ASN.1 (Basic Encoding Rules (BER), Canonical Encoding Rules (CER) und Distinguished Encoding Rules (DER)).

ISO 9594-8/ITU X.509v3 spezifiziert die X.509 Zertifikate

10.4. Sourcecodes

10.4.1. Das Konfigurationsfile

```
##this file contains the field information needed to generate x509
certifiactes
##it is used by the PA2_Sna_04_1 project.

      ##All comments on a separate line!!
      ##notation: FIELDNAME: VALUE : HEADER

      ##hex values should always be written in 2hex signs...
      ##0x05 and NOT 0x5 ...as 0x5 is interpreted as 0x50!!
      ##if a field should not be written at all set 0x as header- and data-
value

##--------------certificate-Sequence{
start_seq:

##--------------TBSCertificate-Sequence{
tbs_seq:

## ------------versionsTag must be 0 (and must be written in the header
field)
version_tag: : 0

##------------version if optionals are present it shoud be 2, else nothing
(0x : 0x)
version: 2


##------------serial should be unique
serial: 2

##-----------md5 use this one or the sha1 but not both
algoid: 1 2 840 113549 1 1 4

##----------sha1
##algoid: 1 2 840 113549 1 1 5

## ----------the nulltype
null-1:

##---------------Issuer-Sequence{
Issuer_seq:
Issuer_organizationName_Obj: 2 5 4 6
Issuer_organizationName_Str: ch

Issuer_organizationalUnitName_Obj: 2 5 4 8
Issuer_organizationalUnitName_Str: zurich

Issuer_organizationalUnitName_Obj2: 2 5 4 7
Issuer_organizationalUnitName_Str2: winti

Issuer_organizationalUnitName_Obj3: 2 5 4 10
Issuer_organizationalUnitName_Str3: zhwin

Issuer_organizationalUnitName_Obj4: 2 5 4 11
Issuer_organizationalUnitName_Str4: pa2_sna
```

```
Issuer_commonName_Obj: 2 5 4 3
Issuer_commonName_Str: hannes

Issuer_organizationalUnitName_Obj5: 1 2 840 113549 1 9 1
Issuer_organizationalUnitName_Str5: zurcher@gmx.li
##----------------}Issuer-Sequence


##-------------Validity
##-------------Validity-sequence
Val_seq:

##  --------------Validity UTC; UTC or GeneralizedTime must be used, not
both together
GeneralizedTime: false
Validity_notBefore: 030713000000Z
Validity_notAfter: 050715235959Z

##---------------GeneralizedTime as example
##GeneralizedTime: true
##Validity_notBefore: 20000601120000-0000
##Validity_notAfter: 20080601120000-0000

##-------------Subject-Sequence{
Sub_seq:
Subject_organizationName_Obj: 2 5 4 6
Subject_organizationName_Str: ch

Subject_organizationalUnitName_Obj: 2 5 4 8
Subject_organizationalUnitName_Str: zurich

Subject_organizationalUnitName_Obj2: 2 5 4 7
Subject_organizationalUnitName_Str2: winti


Subject_organizationalUnitName_Obj3: 2 5 4 10
Subject_organizationalUnitName_Str3: zhwin

Subject_organizationalUnitName_Obj4: 2 5 4 11
Subject_organizationalUnitName_Str4: pa2_sna

Subject_commonName_Obj: 2 5 4 3
Subject_commonName_Str: domi

Subject_email_Obj: 1 2 840 113549 1 9 1
Subject_email_Str: riedwdom@zhwin.ch
##}-------------Subject-Sequence

##--------------Subject Public Key Info
SubjectPubKey_AlgOID: 1 2 840 113549 1 1 1

##--------------Nulltype
Null-2:

##---------------next 4 fields will only be used if the PublicKey_file-
field is an empty string
subPubKeyBitString:
BitStringUnused: 0
subjectPublicKeyInteger:0x87294782984289374124289841 9a87878f878923498cc8878
7d
subjectPublicKeyInteger2:65537
```

```
##---------------TAG[3] Optional Extensions
##---------------not written in the standart certificate
tag_3: 0x : 0x
opt_seq: 0x
basicConstraints_seq: 0x
##basicConstraints: 2 5 29 19
basicConstraints: 0x : 0x
basicConstraints_OS: :0x
basicConstraints_seq2: 0x

crl_seq:  0x
crl_OID:  0x : 0x
crl_OS: : 0x
crl_seq2: 0x
crl_seq3: 0x
crl_tag1: 0x : 0x
crl_tag2: 0x : 0x
crl_tag3: 0x : 0x

##---------------Hash; SHA-1 or MD5 should be used, not both together
##hash_algoid: SHA-1
hash_algoid: MD5

##---------------Signature; SHA-1 or MD5 should be used, not both together
##sig_algoid: SHA1withRSA
sig_algoid: MD5withRSA

##---------------The Private-Key to sign the certificate
PrivKey_file: ./demoCA/private/cakey_nodes.nocrypt.der.p8

##-------------PublicKey_file header HAS to be 0x (none) to work!!
##-------------PublicKey_file has to be DER-encoded!!
##-------------if the Public-Key is not in a file, use the integers above
and write an empty string here
PublicKey_file: ./demoCA/testkey_public.der : 0x

##--------------end
```