

# Sicherheitsmechanismen in Java 2 Mobile Edition

## 1 Inhaltsverzeichnis

1 Inhaltsverzeichnis .....	1
2 Einführung .....	1
3 Security mit der Connected Limited Device Configuration (CLDC) .....	2
3.1 Überblick .....	2
3.2 Low level Security .....	2
3.2.1 Class file verification .....	2
3.3 Application level security .....	2
3.3.1 Sandbox model .....	3
3.3.2 Trusted applications model .....	3
3.3.2.1 Untrusted MIDlet Suites .....	3
3.3.2.2 Trusted MIDlet Suites .....	4
3.3.2.3 Beispiel für eine domain policy Datei .....	4
3.3.2.4 Authorization model .....	4
3.3.2.5 Trusted MIDlet Suites using X.509 Public Key Infrastructure (PKI) .....	5
3.3.3 Schützen von Systemklassen .....	5
3.3.4 Weitere Einschränkungen für das dynamische Laden von Klassen .....	5
3.4 End-to-end security .....	5
4 Security and trust services API (SATSA) .....	6
4.1 Allgemeines .....	6
4.2 Security Element (SE) .....	6
4.3 Features der SATSA .....	6
4.3.1 SATSA-APDU .....	7
4.3.2 SATSA-JCRMI .....	7
4.3.3 SATSA-PKI .....	7
4.3.4 SATSA-CRYPTO .....	7
4.4 Security bei der SATSA .....	7
5 Ausblick .....	8
6 Quellen .....	8

## 2 Einführung

Bei den bisherigen Vorträgen des Java 2 ME Vortragszyklus wurde vorwiegend die Funktionalität und die Möglichkeiten dieser Java Edition vorgestellt. Dabei wurde erklärt, wie mit Java 2 ME eigene Programme für mobile Geräte geschrieben werden können und was man dabei für besondere Möglichkeiten hat.

Doch kein Programmierer ist fehlerlos. Außerdem gibt es immer auch Programmierer, die schädliche Programme wie Viren entwickeln, und so rückt die Sicherheit immer mehr in den Vordergrund. Es treten neue Sicherheitsbedürfnisse auf, die eine Unschädlichkeit von Java 2 ME Programmen fordern. Dies geht von dem Wunsch, dass das mobile Gerät nicht abstürzt, über die Forderung von Zugriffsbeschränkungen, bis hin zur Garantie einer sicheren und vertraulichen Datenübertragung.

Diese Arbeit soll sich mit den Sicherheitsmechanismen bei Java 2 ME beschäftigen.

## 3 Security mit der Connected Limited Device Configuration (CLDC)

### 3.1 Überblick

Java 2 ME Entwickler können mit der *Connected Limited Device Configuration (CLDC)*<sup>1</sup> für jedes einzelne Programm individuelle Rechte definieren.

Das CLDC Sicherheitsmodell kennt dabei drei grundsätzlich unterschiedliche Sicherheitslevels:

- Die *Low level security*, manchmal auch *virtual machine security* genannt, stellt sicher, dass ein Java 2 ME Programm eine korrekte Java Semantik hat. Andernfalls sorgt dieses Sicherheitslevel dafür, dass eine fehlerhafte Klasse nicht zur Ausführung kommt und somit das Endgerät zum Absturz bringen oder in anderer Weise beschädigen kann.
- *Application level security* bedeutet, dass sichergestellt wird, dass das Programm nur auf klar definierte *libraries* und Systemressourcen zugreifen kann. Diese Zugriffsrechte werden von der Programmumgebung sowie vom mobilen Gerät festgelegt.
- *End-to-end security* garantiert, dass jede Datenübertragung, die vom Mobilien Gerät ausgelöst wird, auf dem kompletten Übertragungspfad vom Mobilien Gerät bis zum Übertragungspartner (zum Beispiel einen Internetserver) und zurück sicher ist. Um dies zu gewährleisten ist zum Beispiel eine Verschlüsselung der Daten nötig. Diese Dienste der End-to-End security sind außerhalb von CLDC, zum Beispiel in der *Security and Trust Services API (SATSA)* (Abschnitt 4) definiert.

### 3.2 Low level Security

Für die *Java virtual machine (VM)* ist diese *Low level security* eine wichtige Grundlage. Es darf keinem Programm möglich sein, das Gerät auf dem die *virtual machine* läuft zu beschädigen oder zum Absturz zu bringen. Bei einer „normalen“ *Java VM* wird dies durch den *class file verifier* gewährleistet. Dieser stellt sicher, dass keine Klasse fehlerhafte Anweisungen, Anweisungen in falscher Reihenfolge oder Referenzen auf ungültige Speicherstellen enthält. Der *class file verifier* verhindert, dass eine in die *VM* geladene Klasse in irgendeiner nicht erlaubten Art und Weise zur Ausführung kommt.

#### 3.2.1 Class file verification

Da die normale *class file verification*<sup>2</sup> ein mobiles Gerät überfordern würde (Speicher und CPU) wird der *verifier* in zwei Phasen aufgeteilt:

1. Zuerst müssen die Klassen durch den *preverifier*, der der Klasse *stack map attributes* hinzufügt. Dies geschieht typischerweise auf einer Entwicklungsworkstation.
2. Bei der Ausführung muss die Klasse dann durch den *runtime verifier*. Die vom *preverifier* hinzugefügten *stack map attributes* werden dabei genutzt und so wird der aktuelle *verification process* effizienter.

### 3.3 Application level security

Um weitere Sicherheiten zu gewährleisten, reicht die durch den *verifier* in der *low level security* erreichte Korrektheit eines Java Programms bei weitem nicht aus. Es gibt noch viele potentielle Sicherheitslücken, die der *verifier* nicht schließt. Dazu gehört zum Beispiel der Zugriff auf externe Ressourcen wie das Dateisystem, Drucker, Infrarotgeräte, *libraries* oder das Netzwerk. *Application level security* bedeutet, dass ein Java Programm wirklich nur auf die ihm vom Gerät und der Programmumgebung erlaubten Systemressourcen, *libraries* und anderen Komponenten zugreifen kann.

---

<sup>1</sup> Weitere Details zur CLDC sind dem Vortrag „CLDC vs. CDC“ von Fabian Reiß (selbiger Java 2 ME Vortragszyklus) zu entnehmen.

<sup>2</sup> Weitere Details zur *Class file verification* sind dem Vortrag „Virtual Machine (JVM, KVM, CardVM)“ von Stefan Menz (selbiger Java 2 ME Vortragszyklus) zu entnehmen.

### 3.3.1 Sandbox model

Die *Application level security* wird mit der Metapher einer geschlossenen *sandbox* realisiert. Jedes Programm muss in einer solchen abgeschlossenen Umgebung laufen, in der das Programm nur auf die *libraries* zugreifen kann, die durch die Konfiguration (*CLDC*), Profile (z.B. *MIDP*)<sup>3</sup> und anderen Klassen des Gerätes (herstellerspezifische Klassen) definiert wurden. Java Programme können aus ihrer Sandbox nicht ausbrechen oder auf *libraries* oder Ressourcen, die nicht zur vordefinierten Funktionalität gehören zugreifen. Die *sandbox* stellt sicher, dass kein boshaftes oder fehlerhaftes Programm auf die Systemressourcen zugreifen kann.

Das *Sandbox model* benötigt folgendes:

- Die Klassen müssen überprüft sein und es muss sichergestellt sein dass sie ein korrektes Java Programm sind (*Class file verification*; 3.2.1).
- Auf dem Gerät muss das Downloaden, Installieren und Managen von Java Programmen so geschehen, dass der Entwickler eines Programms nicht den Standard *class loading* Mechanismus der *Java VM* verändern oder umgehen kann.
- Dem Programmierer muss ein geschlossenes, vordefiniertes Set von *Java APIs* zur Verfügung stehen (*CLDC*, Profile und herstellerepezifischen Klassen) (Schützen von Systemklassen; 3.3.3)
- Dieses Set von vordefinierten *Java APIs* muss geschlossen sein, das heißt, der Programmierer darf keine neuen *libraries* downloaden, die erweiterte Grundfunktionalität enthalten, oder auf irgendwelche Funktionen zugreifen, die nicht Teil der *Java libraries* sind, die in *CLDC*, Profilen und herstellerepezifischen Klassen angeboten wird.

### 3.3.2 Trusted applications model

Manche Java2ME Profile bieten auch andere Sicherheitslösungen an. Im *Mobile Information Device Profile (MIDP)* wird zum Beispiel seit der Version 2.0 auch das *trusted application model* angeboten.

Dieses Model kann Programmen auch Zugriff auf (sicherheits-)kritischen *APIs* gewähren. Ob und wann ein Gerät bestimmt, dass einer *MIDlet Suite*<sup>4</sup> vertraut werden (*trusted MIDlet Suite*) und der Zugriff gewährt werden kann, entscheidet die *domain policy*. Jede *MIDlet Suite*, der nicht vertraut werden kann muss *untrusted* laufen. Falls ein Fehler beim Verifizierungsprozess auftritt, muss sie zurückgewiesen und darf nicht ausgeführt werden.

#### 3.3.2.1 Untrusted MIDlet Suites

Eine *Untrusted MIDlet Suite* ist eine *MIDlet Suite*, deren Ursprung und deren *Integrität* (Vollständigkeit und Unversehrtheit) der JAR Datei das Gerät nicht vertraut. Solche *MIDlet Suites* müssen in der *untrusted domain*, in der sie nur eine eingeschränkte Umgebung zur Verfügung hat, ausgeführt werden. In der eingeschränkten Umgebung ist der Zugriff auf geschützte *APIs* entweder nicht möglich oder nur dann, wenn der User diese Zugriffsrechte explizit erteilt. Jede *MIDP 1.0 MIDlet Suite* kann auch unter dieser *MIDP 2.0* Spezifikation laufen, allerdings als *untrusted MIDlet Suite*. Damit können *MIDP 1.0 MIDlet Suites* alle *APIs* und Funktionen von dieser *MIDP 2.0* Spezifikation, die nicht sicherheitskritisch sind (die keine Zugriffsrechte benötigen, also die *APIs* und Funktionen, die von *trusted* sowie von *untrusted MIDlet Suites* benutzt werden dürfen) nutzen.

Die *untrusted Domain* für *untrusted MIDlet Suites*, die keine explizite Bestätigung vom User benötigen, enthalten folgende *APIs* (*allowed permissions*):

- |                                 |                                 |
|---------------------------------|---------------------------------|
| - javax.microedition.rms        | (Record Management System APIs) |
| - javax.microedition.midlet     | (MIDlet APIs)                   |
| - javax.microedition.lcdui      | (User Interface APIs)           |
| - javax.microedition.lcdui.game | (APIs für Spiele)               |

<sup>3</sup> Weitere Informationen zu Profilen wie dem *MIDP* sind dem Vortrag „Profile (Basis, Personal, MIDP)“ von Thomas Bochtler (selbiger Java 2 ME Vortragszyklus) zu entnehmen.

<sup>4</sup> Weitere Informationen zu *MIDlet Suites* sind dem Vortrag „Profile (Basis, Personal, MIDP)“ von Thomas Bochtler (selbiger Java 2 ME Vortragszyklus) zu entnehmen.

- javax.microedition.media  
und javax.microedition.media.control (APIs für die Soundwiedergabe)

Die *untrusted Domain* für *untrusted MIDletSuites*, die eine explizite Bestätigung vom User benötigen enthalten unter Anderem folgende *APIs (user permissions)*:

- javax.microedition.io.HttpConnection (http Protokol)
- javax.microedition.io.HttpsConnection (https Protokol)

### 3.3.2.2 Trusted MIDlet Suites

Für *trusted MIDlet Suites* basiert die Sicherheit auf *protection domains*. Diese *protection domains* definieren die Zugriffsrechte, die einer *MIDlet Suite* erteilt werden können. Wer eine solche *protection domain* erstellt, legt ein Set von *allowed* und *user permissions* fest sowie den Mechanismus, mit welchem das Gerät eine *MIDlet Suite* als für diese *protection domain* vertrauenswürdig identifiziert. Mit dem Identifikationsmechanismus wird definiert, wann eine *MIDlet Suite* in diese *protection domain* aufgenommen und ihr damit die darin enthaltenen Zugriffsrechte auf geschützte *APIs* erteilt. Diese Mechanismen sind separat definiert, um eine für das Gerät, das Netzwerk und das Einsatzgebiet geeigneten Mechanismus auswählen zu können. Ein Beispiel dafür ist der Mechanismus, der auf der *X.509 Public Key Infrastructure (PKI)* basiert (siehe 3.3.2.4)

### 3.3.2.3 Beispiel für eine domain policy Datei

Mit *domain* wird eine *domain* definiert, die die nachfolgenden *permissions* enthält. *Allowed Permissions* werden mit *allow*, *user permissions* mit *blanket*, *session* und *oneshot* definiert. Dabei wird der User bei *blanket* einmal pro installierte Funktion; bei *session* einmal pro *MIDlet Suite* und bei *oneshot* bei jeder Benutzung der Funktion nach der *permission* gefragt. Mit *alias* können mehrere *permissions* zu einer *Permission* gruppiert werden.

```
domain: O=„MIDletUnderwriters, Inc.“, C=US
allow: javax.microedition.io.HttpConnection
oneshot (oneshot): javax.microedition.io.CommConnection
alias: client_connections
    javax.microedition.io.SocketConnection,
    javax.microedition.io.SecureConnection,
    javax.microedition.io.HttpConnection,
    javax.microedition.io.HttpsConnection
domain: O=AcmeWireless, OU=SoftwareAssurance
allow: client_connections
allow: javax.microedition.io.ServerSocketConnection,
javax.microedition.io.UDPDatagramConnection
oneshot (oneshot): javax.microedition.io.CommConnection
domain: allnet
blanket (session): client_connections
oneshot: javax.microedition.io.CommConnection
```

### 3.3.2.4 Authorization model

Die Grundlage der Standard *Authorization* einer *MIDlet Suite* ist das Zusammenspiel folgender Elemente:

- Einer *protection domain*, die ein Set von *allowed* und *user permissions* ist.
- Einem *permissions* -Set, das eine *MIDlet Suite* mittels ihrer Attribute *MIDlet-Permissions* und *MIDlet-Permissions-Opt* anfordert.
- Einem *permissions* -Set für jede geschützte *API* oder Funktion auf dem Gerät, die eine Vereinigung aller *permissions* ist, die durch jede *API* für geschützte Funktionen definiert ist.
- Dem User, der weitere *permissions* erteilen kann.

### 3.3.2.5 Trusted MIDlet Suites using X.509 Public Key Infrastructure (PKI)

Einer der Mechanismen, der *MIDlet Suites* authentifiziert und gegebenenfalls als *trusted MIDlet Suite* signiert, so dass das Gerät der *MIDlet Suite* vertrauen kann, basiert auf der *X.509 Public Key Infrastructure*. Dieser Mechanismus läuft folgendermaßen ab<sup>5</sup>:

- Die *MIDlet Suite* wird durch die Signierung des Java Archivs (JAR Datei) in den Zustand *protected* versetzt.
- Die Signatur und die Zertifikate werden dem Programm *descriptor (primary key)* als Attribute angehängt.
- Das Gerät benutzt diese Attribute um die Signatur zu prüfen.
- Das Gerät schließt die Authentifizierung ab indem es ein *root certificate* benutzt, das eine *protected domain* des Gerätes begrenzt.

### 3.3.3 Schützen von Systemklassen

Alle Systemklassen sind geschützt und dürfen nicht verändert werden. Diese Bedingung ist nötig, da es möglich ist, Java Programme dynamisch auf die *Java VM* herunterzuladen. Falls aber die heruntergeladenen Programme die Möglichkeit hätten, das Klassenset, das in den *packages java.\** und *javax.microedition.\** sowie in profil- und herstellerspezifischen *packages* angeboten wird zu überschreiben oder zu erweitern, würde eine potentielle *Application level* Sicherheitslücke in der *Java VM* entstehen. Deshalb muss *CLDC* sicherstellen, dass der Programmierer diese geschützten Systemklassen nicht überschreiben, verändern oder erweitern kann.

### 3.3.4 Weitere Einschränkungen für das dynamische Laden von Klassen

Das dynamische Laden von Klassen ist in der *CLDC* zwar nur vom Mechanismus her spezifiziert und die meisten Details sind dann implementierungsabhängig, aber eine zentrale Sache muss doch festgelegt sein. Ein Javaprogramm darf standardmäßig nur Klassen von seinem eigenen Java Archive (JAR Datei) herunterladen. Diese Einschränkung stellt sicher, das Java Programme auf dem Gerät sich nicht in andere einmischen oder von anderen Klassen stehlen können. Außerdem stellt dies sicher, dass kein Programm von Drittanbietern sich Zugang zu *private* oder *protected* Java Klassen, die der Gerätehersteller oder der Hersteller eines Services als Systemprogramme bereitstellt hat, verschaffen kann.

## 3.4 End-to-end security

Typischerweise ist ein Gerät, das den *CLDC* Spezifikationen entspricht Teil einer *End-to-end* Lösung (wie zum Beispiel das Netzwerk eines Payment - Terminals). Solche Netzwerke bieten im Allgemeinen eine Vielzahl von erweiterten Sicherheitsmechanismen an (wie zum Beispiel Verschlüsselung) um die Übertragung von Daten und Code zwischen dem Server und dem Endgerät sicherzustellen. Da die Vielfalt solcher Netzwerke aber unglaublich groß ist, enthält die *CLDC* Spezifikation keinen einzigen *End-to-End* Sicherheitsmechanismus. Deshalb sind all die *End-to-End* Sicherheitsmechanismen implementierungsabhängig und außerhalb der *CLDC* spezifiziert.

Ein solcher *End-to-End* Sicherheitsmechanismus ist die *Security and trust services API (SATSA)*.

---

<sup>5</sup> Grobe Skizze; auf die Details dieses Mechanismus wird im Rahmen dieses Vortrags verzichtet. Weitere Informationen sind den Quellen (Abschnitt 6) zu entnehmen.

## 4 Security and trust services API (SATSA)

### 4.1 Allgemeines

Die Idee der *Security and Trust Services API (SATSA)* ist es, eine Sammlung von *APIs* bereitzustellen, die für Sicherheit und *trust services* garantieren. Diese *API* ist optional. Mit der *SATSA* wird ein Sicherheitselement, das sogenannte *Security element (SE)* eingeführt. Ein *SE* ist eine Komponente in einem Java 2 ME Gerät, das folgendes verspricht:

- Vertrauliche Daten, wie private Passwörter des Users, Zertifikate von Services oder persönlichen Informationen werden sicher gespeichert
- Verschlüsselungsmethoden, um Zahlungsprotokolle, die Übertragung anderer vertraulicher Daten sowie die Sicherstellung deren *Integrität* (Vollständigkeit und Unversehrtheit) zu unterstützen.
- Eine sichere Ausführungsumgebung um Sicherheitsfunktionen zu aktivieren oder anzupassen

Auf diese Eigenschaften wird sich ein Java Programm verlassen können und so werden Dinge wie die Identifikation des Users, banking und payment ermöglicht.

### 4.2 Security Element (SE)

*Security elements (SE)* gibt es in verschiedenen Formen. Meistens sind es *Smart cards* die eingesetzt werden, um ein Sicherheitselement zu realisieren. Sie sind weit verbreitet in Handys wie zum Beispiel die *SIM cards*. Auf den *SIM cards* sind die Authentifizierungsdaten gespeichert und sobald sie im Handy ist, ist der Zugriff auf das Netzwerk freigegeben.

Ein Sicherheitselement kann aber genauso im Gerät selber realisiert sein. Bei einer solchen Realisierung macht man sich eingebettete Chips oder spezielle Sicherheitsfeatures der Hardware zu nutzen.

Oder aber man realisiert das Sicherheitselement vollständig mit Software.

Im Grunde ist es egal, wie man das *SE* realisiert, obwohl einige *APIs* für *Smart cards* optimiert sind.

### 4.3 Features der SATSA

Sicherheitselemente haben verschiedene Software und Hardwarecharakteristiken. Die Funktionen der *SATSA* basieren auf folgenden Kriterien:

- Der benötigte Speicherplatz für Endgeräte, dessen Ressourcen begrenzt sind.
- Die Bandbreite der Nutzungsmöglichkeiten des Sicherheitselements.
- Die Flexibilität und die Erweiterbarkeit der *API*.

Auf diesen Kriterien basierend werden in der *SATSA* folgende Funktionen für Java 2 ME Plattformen angeboten:

- *Smart card* Kommunikation:  
*Smart cards* stellen eine sichere programmierbare Umgebung bereit. Sie sind die am weitesten verbreiteten Sicherheitselemente um eine Vielzahl von Sicherheits- und *trust Services* zu entwickeln. Diese *Services* können laufend per Installation von neuen oder verbesserten Programmen auf den neusten Stand gebracht werden. In der *SATSA* werden zwei Zugriffsmethoden für *Smart cards* angeboten: Die eine Methode basiert auf dem *APDU* Protokoll, die andere auf dem *Java Card RMI* Protokoll. Beide Methoden ermöglichen Java 2 ME Programmen die Kommunikation mit *Smart cards*, was die auf ihr entwickelten Sicherheitservices testet. (→ *SATSA-APDU*, 4.3.1 und *SATSA-JCRMI*, 4.3.2)
- *Services* für digitale Signaturen und das Userzertifikatesmanagement  
Der Service für digitale Signaturen ermöglicht es einem Java 2 ME Programm digitale Signaturen zu erstellen, die auf dem *Cryptographic Message Syntax (CMS)* basieren. Digitale Signaturen werden benötigt, um Enduser zu authentifizieren oder um einer Transaktion die Verschlüsselung per *public key* zu erlauben. Die Identität des Users ist im Normalfall mit dem *public key* Zertifikat

an den *public key* gebunden. Die Zertifikate können mit dem Zertifikatsmanagementservice gemanagt werden. (→ *SATSA-PKI*, 4.3.3)

- Library zur Verschlüsselung von Daten  
Dieses *cryptography library* ist ein Teil der Java 2 ME *cryptography API*. Damit können einfache *cryptographic operations* wie das Verifizieren von Signaturen, die Ver- und Entschlüsselung von Daten durchgeführt werden. Damit kann ein Java 2 ME Programm sichere Datenkommunikation und Datenschutz sowie das Managen solcher Daten realisieren. (→ *SATSA-CRYPTO*, 4.3.4)

In den folgenden Abschnitten 4.3.1 bis 4.3.4 ist kurz zusammengefasst, was die einzelnen *SATSA packages* für Methoden und Funktionen enthalten.

### 4.3.1 SATSA-APDU

Mit dem *SATSA-APDU (Application Protocol Data Unit) package* kann ein Java 2 ME Programm eine *APDUConnection* erstellen, um mit einer *Smart card* Anwendung zu kommunizieren. Eine solche *APDUConnection* unterstützt den Austausch von *APDU* Befehlen. Jede dieser *APDUConnections* hat einen logischen Kanal, dessen Verwaltung ebenfalls die *API* übernimmt. Es ist sogar möglich, mehrere *APDUConnections* gleichzeitig existieren zu lassen.

### 4.3.2 SATSA-JCRMI

Das *SATSA-JCRMI (Java Card over Remote Method Invocation) package* ermöglicht es einem Java 2 ME Programm, eine *JavaCardRMICConnection* zu erstellen. Damit wird eine *Java Card RMI session* für das Java Card Programm eröffnet. Jede *JavaCardRMICConnection* hat einen logischen Kanal, dessen Verwaltung ebenfalls die *API* übernimmt. Auch hier können mehrere *JavaCardRMICConnection* gleichzeitig existieren. Sogar Plattformen, die standardmäßig kein *J2SE RMI* unterstützen ist eine solche *JavaCardRMICConnection* möglich. Dazu muss für jedes *remote object* ein so genannter *static stub* erzeugt werden, dessen Interface diese *API* bereitstellt.

### 4.3.3 SATSA-PKI

Mit der *SATSA-PKI (Public Key Infrastructure)* steht eine ähnliche *PKI* zur Verfügung, wie die oben schon erwähnte *X.509 PKI* (siehe 3.3.2.4). Das *SATSA-PKI package* stellt einen *CMSMessageSignatureService* bereit um Nachrichten mit einem *private key* zu signieren. Außerdem wird ein *UserCredentialManager* angeboten, um Anfragen zur Anmeldung von Zertifikaten an eine Zertifikatregistrierungs-Autorität zu senden oder um Zertifikate einem Zertifikat *store* hinzuzufügen oder zu entfernen.

### 4.3.4 SATSA-CRYPTO

Das *SATSA-CRYPTO (Cryptography) package* bietet Methoden an, um Signaturen zu überprüfen sowie Verschlüsselungscode zur Ver- und Entschlüsselung von Daten. Außerdem enthält es eine *Key factory* um sichere *public keys* zu erstellen sowie eine Methode um hersteller-unabhängige geheime Schlüssel Objekte für verschlüsseltes Material zu erstellen.

## 4.4 Security bei der SATSA

Die Zugriffsrechte auf die *SATSA* sind nicht generell erteilt. Damit ein Java 2 ME Programm, die *SATSA* nutzen kann, müssen ihm erst die Zugriffsrechte auf die *SATSA* gewährt werden. Ausgenommen davon ist die *SATSA-CRYPTO*. Der Zugriff auf sie ist generell möglich, es bedarf keinem expliziten Recht. Um die Benutzung der Ressourcen des Sicherheitselements zu schützen wird in der *SATSA* ein *Access Control Model* angeboten, dass dem SE ermöglicht fein abgestufte Zugriffsrechte zu erteilen. Dieses *Access Control Model* ist vor allem für die *SATSA-APDU* und die *SATSA-JCRMI* gedacht. So kann genau spezifiziert werden, auf welche Funktion der *Smart card* zugegriffen werden darf.

## 5 Ausblick

Abschließend möchte ich noch aufzeigen, wie verwundbar doch jede Sicherheitsmechanismen sind, wenn der Wille nur groß genug, und/oder der User unbesorgt genug ist. Folgendes meldete der Heise Verlag am 15.06.2004:

„Erster Handy-Wurm entdeckt

Der Wurm EPOC.Cabir ist in der Lage, sich über Bluetooth auf Smartphones der Nokia Series 60 mit Symbian-Betriebssystem zu verbreiten. Er verschickt sich als Installations-Datei (.SIS) an alle Geräte in der Nähe -- sogar Drucker --, auf denen Bluetooth aktiviert ist, und versucht sich in das Verzeichnis "APPS" zu kopieren. Allerdings muss hierzu der Benutzer den Empfang bestätigen. Anschließend startet der Wurm und kopiert sich in ein standardmäßig nicht sichtbares Verzeichnis, von dem aus er bei jedem Neustart des Handys aktiviert wird. Zwar ist Cabir derzeit nur als Proof-of-Concept-Wurm einzustufen, der keinen Schaden anrichtet -- außer die Batterie durch die Bluetooth-Verbindungen leer zu saugen -- allerdings glauben die Virenforscher bei Network Associates, dass andere Virenautoren diesen Schädling als Basis für weitere Entwicklungen heranziehen werden. In-The-Wild, also in freier Wildbahn, ist Cabir bislang noch nicht gesichtet worden.

Forscher und Hacker warnen seit langem vor Viren und Würmern für Handys. Insbesondere seitdem sehr viele Geräte Java-Applikationen ausführen können und Smartphones an die Leistungsfähigkeit von PDAs herankommen, ist es nur eine Frage der Zeit, bis die Schädlingsschwelle auf den Mobilfunkbereich überschwappt.“

Doch trotz dieser Meldung sollte man jetzt nicht in Panik verfallen. So rasend schnell, wie bei Computern werden sich gefährliche Viren und Würmer im mobilen Bereich sicherlich nicht verbreiten. Bei der Entwicklung der mobilen Geräte und deren Netzwerke wurde mit großer Sorgfalt gerade auch im Bereich Sicherheit umgegangen. Die mobile Technik ist nicht so „gewachsen“ wie die Computer, die ursprünglich ja gar nicht für eine Kommunikation in einem solch großen Netz wie dem Internet vorgesehen waren. Aus der Handy-Wurm Meldung geht hervor, dass sich solche unerwünschten Programme nicht unbemerkt installieren können und die Sicherheitsmechanismen greifen (der User musste den Empfang erst bestätigen).

Als Fazit kann man sagen, dass die Sicherheitsmechanismen den Anforderungen gerecht werden mobile Geräte auf denen Java 2 ME Anwendungen ausgeführt werden, ausreichend geschützt sind.

## 6 Quellen

J2ME(TM) Connected Limited Device Configuration (CLDC) Specification 1.1 Final Release (JSR 139)  
(<http://jcp.org/aboutJava/communityprocess/final/jsr139/index.html>)

Mobile Information Device Profile Specification 2.0 Final Release (JSR 118)  
(<http://jcp.org/aboutJava/communityprocess/final/jsr118/index.html>)

Security and Trust Services API for J2ME(TM) Specification 1.0 Public Review Draft (JSR 177)  
(<http://jcp.org/aboutJava/communityprocess/review/jsr177/index.html>)

Erster Handy-Wurm entdeckt (15.06.2004, dab)  
(<http://www.heise.de/newsticker/meldung/48252>)