Touch'n pay: ein NFC-Feldversuch

Mit der Near Field Communication (NFC) Technologie können mobile Geräte über kurze Distanzen (bis 10 cm) kommunizieren. NFC eignet sich daher sehr gut für mobiles Bezahlen oder für den bequemen Datenaustausch. In diesem Artikel beschreiben wir das Projekt touch'n pay, in welchem die technische Machbarkeit des Bezahlens mit NFC-Mobiltelefonen getestet wurde. Das Projekt touch'n pay läuft als Feldversuch in einem Hofladen im Kanton Zürich.

Ingo Bauersachs, Dominik Gruntz | dominik.gruntz@fhnw.ch

Unter dem Namen touch'n pay (www.touchnpay.ch) läuft seit Januar 2010 ein Feldversuch in einem Hofladen in Kilchberg (ZH), in welchem mit Hilfe von NFC-fähigen Mobiltelefonen eingekauft werden kann. Das Mobiltelefon wird ausserdem verwendet, um auch ausserhalb der Öffnungszeiten den Zugang zum Hofladen zu ermöglichen (Abb. 1).

Im Hofladen sind die Produkte mit NFC-Produktetiketten ausgezeichnet (Abb. 2). Wenn der Kunde sein Mobiltelefon an ein solches Produktetikett hält, wird das Produkt automatisch auf seinem elektronischen Kassenzettel eingetragen. Bei offenen Produkten wie Gemüse und Obst muss zusätzlich noch das Gewicht über die Tastatur des Mobiltelefons eingegeben werden. Produkte können auch wieder vom Kassenzettel gelöscht werden.

Sobald alle gewünschten Produkte in der Einkaufstasche liegen, muss nur noch ein *check-out*-Tag berührt oder das *Zahlen*-Menu auf dem Mobiltelefon gewählt werden. Damit wird der Bezahlvorgang ausgelöst. Das Programm verbindet sich dann mit dem Bezahlserver und übermittelt den zu belastenden Betrag zusammen mit der Telefonnummer des Kunden. Der Kunde kann wählen, ob dieser Betrag dem Postkonto (Handyzahlung PostFinance), der Kreditkarte oder dem ePay mWallet belastet werden soll. Der Kunde hat jederzeit die Möglichkeit, seine getätigten Einkäufe auf dem Mobiltelefon einzusehen.

Dieses Projekt ist von den Firmen e-24 AG und NEXPERTS GmbH mit Unterstützung der Fachhochschule Nordwestschweiz realisiert worden. Die auf dem Mobiltelefon laufende Applikation ist in Java geschrieben (Java Platform Micro Edition) und verwendet das *Contactless Communication API* (JSR 257) für den Zugriff auf die NFC-Funktionalität. Der Bezahlvorgang wird über einen Server der Firma e-24 abgewickelt. Damit die Sicherheit bei den Transaktionen gewährleistet ist, werden diese mit Hilfe eines Applets im Secure-Element (SE) des Mobiltelefons signiert.

In diesem Artikel möchten wir auf die einzelnen Teile dieser Lösung eingehen. Dazu werden wir zuerst kurz die NFC-Technologie und die Architektur unserer Lösung vorstellen. Wir werden dann aufzeigen, wie NFC-Tags ausgelesen und wie auf das Secure-Element zugegriffen werden kann. Am Ende beschreiben wir noch, wie diese Applikation *over-the-air* sicher auf das Mobiltelefon geladen werden kann.

NFC-Technologie

Near Field Technologie (NFC) ist eine kontaktlose Schnittstellentechnologie, welche eine einfache und schnelle Kommunikation über kurze Entfernungen zwischen RFID-Tags und einem speziell ausgerüsteten Mobiltelefon ermöglicht. Im Frequenzbereich 13.56 MHz werden über maximal 10 cm bis zu 424 kbit/s übertragen. Ein NFC-Mobiltelefon kann dabei sowohl als Lesegerät als auch als Tag dienen. Im Feldversuch werden Nokia-Geräte der Modellreihe 6131 NFC verwendet, welche uns freundlicherweise von Swisscom zur Verfügung gestellt worden sind.

NFC basiert auf denselben Standards wie RFID. Der wesentliche Unterschied zu RFID ist jedoch der, dass bei NFC der Benutzer den RFID-Leser bei sich trägt. Mit diesem Leser können Informationen aus nächster Umgebung ausgelesen werden. Im eingangs beschriebenen Feldversuch werden so die auf den NFC-Tags abgelegten Produktinformationen ausgelesen. Wird ein NFC-Tag mit dem Mobiltelefon berührt, so wird auch automatisch ein Programm auf dem Telefon gestartet, welches dann die Interaktion mit dem Benutzer übernimmt (z.B. Eingabe von Zusatzdaten bei offenen Produkten im Hofladen oder die Benutzerführung während des Bezahlvorgangs). Umgekehrt kann ein NFC-Mobiltelefon auch eine passive RFID-Karte emulieren. Diese Funktionalität wird für die Zutrittskontrolle in den Hofladen genutzt.

NFC unterscheidet folgende Betriebsarten:

- *Card-Emulation-Modus*: In dieser Betriebsart ist das NFC-Mobiltelefon passiv und emuliert nur eine kontaktlose Smartcard. Ein RFID-Leser, z.B. ein Kassensystem oder in unserem Fall das Türschloss des Hofladens, greift auf die emulierte Smartcard (des NFC-Mobiltelefons) zu. Diese Betriebsart funktioniert auch dann, wenn

Abbildung 1: Zutritt zum Hofladen

Abbildung 2: NFC-Produktetikett

das Mobiltelefon ausgeschaltet ist (solange der Akku nicht leer ist).

- *Reader/Writer-Modus:* In dieser Betriebsart wird das NFC-Mobiltelefon zum Leser und kann passive NFC-Tags auslesen und beschreiben. Auf diese Weise kann z.B. ein Smart-Label gelesen werden, welches eine URL enthält. Die referenzierte Seite kann danach direkt im Internet-Browser angezeigt werden. Damit entfällt das mühsame Abtippen der URL.
- *Peer-to-peer-Modus*: Die peer-to-peer Betriebsart ermöglicht es, Informationen zwischen zwei (aktiven) Geräten auszutauschen.

Wie die Smartcard enthält ein NFC-Gerät auch ein sogenanntes Secure-Element (SE), auf welchem Daten und Programme gesichert abgelegt werden können. Das SE ist entweder im Gerät fest verbaut (wie bei den im Feldversuch verwendeten Nokia-Geräten), oder es ist Teil der SIM-Karte des Mobilfunkanbieters. Bei einem Wechsel des Gerätes ist es von Vorteil, wenn die Daten und Programme mit der SIM-Karte auf das neue Gerät übernommen werden können.

Die Kommerzialisierung von NFC wird hauptsächlich von den im NFC-Forum zusammengeschlossenen über 60 Unternehmen vorangetrieben [NFC]. Die Standardisierung der Smartcard-Schnittstellen wird von der Vereinigung *GlobalPlatform* kontrolliert. Im Mobiltelefon Nokia 6131 NFC wird die *Card Specification 2.1.1* von März 2003 verwendet. Diese Spezifikation definiert bis auf Bit-Ebene, welche Befehle zur Verwaltung einer Smartcard in welchem Umfang (zwingend, empfohlen oder optional) unterstützt werden [GPCS].

Im Folgenden gehen wir auf die Implementierung des im touch'n pay Feldversuchs realisierten Bezahlvorgangs ein.

Architektur der touch'n pay Applikation

Sobald der Kunde alle Tags seiner Produkte ausgelesen hat, löst er die Zahlung aus. Dazu wird eine Meldung an den Bezahlserver geschickt, welche die Identifikation des Kunden, die Identifikation des Verkäufers sowie den zu bezahlenden Betrag enthält. Die Schwierigkeit dabei ist, dass auf dem Server zweifelsfrei festgestellt werden muss, ob die Anfrage auch wirklich von dem Kunden kommt, der in der Meldung als Kunde angegeben ist. Da der Bezahlvorgang von einem Java-Midlet initiiert wird und solche Midlets von einem Angreifer ausgelesen, dekompiliert und geändert werden können, muss sichergestellt werden, dass falsche Meldungen vom Server erkannt werden.

Wir haben dieses Problem so gelöst, dass die Meldung, bevor sie an den Bezahlserver geschickt wird, mit einem kryptologischen Hash-Prüfwert versehen wird. Dieser Prüfwert wird im SE berechnet und verwendet dazu einen im SE abgelegten privaten Schlüssel des asymmetrischen RSA-Kryptosystems.

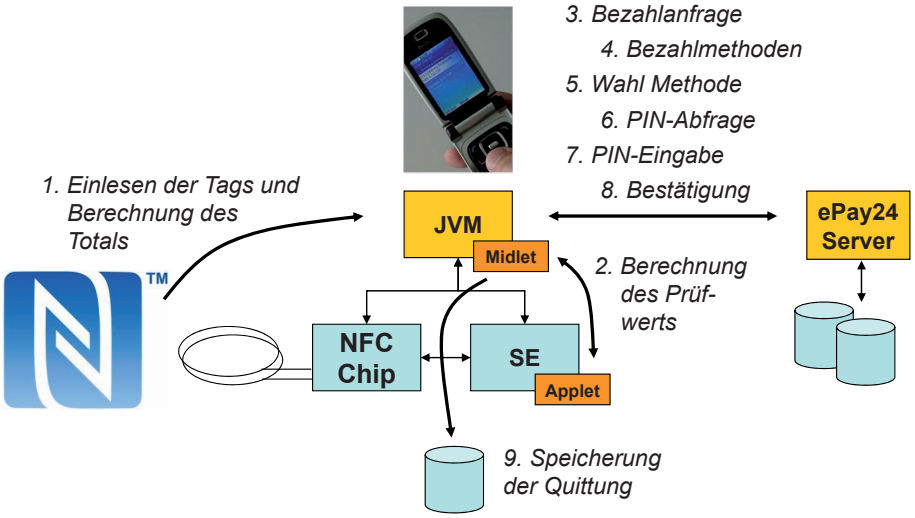

Abbildung 3: Ablauf des Bezahlvorganges

```
DiscoveryManager dm = DiscoveryManager.getInstance();
this.dm.addNDEFRecordListener(
  listener, // implementing NDEFRecordListener
  new NDEFRecordType(NDEFRecordType.EXTERNAL_RTD, "urn:nfc:ext:fhnw.ch:shop")
);
```

Listing 1: Registrierung eines Tag Listeners

Neben dem für das Hashing verwendeten privaten Schlüssel ist im SE auch die Identifikation des Kunden abgelegt (aktuell die MSISDN), denn aus einem Java-Midlet kann die Telefonnummer des Mobiltelefons nicht ausgelesen werden.

In Abbildung 3 ist der Ablauf des Bezahlvorgangs dargestellt. Nachdem die einzelnen NFC-Tags eingelesen (1) und die Gesamtsumme des Einkaufs berechnet worden ist, wird aus den Bezahldaten im SE ein kryptologischer Hash-Prüfwert berechnet (2). Die Bezahldaten werden zusammen mit der Nutzeridentifikation aus dem SE und dem Prüfwert an den Bezahlserver übermittelt (3). Der Bezahlserver meldet daraufhin dem Benutzer die möglichen Bezahlmethoden (4) und der Nutzer wählt eine dieser Methoden aus (5). Bei gewissen Bezahlmethoden (wie z.B. bei einer Abrechnung über die Kreditkarte) muss der Kunde zusätzlich ein PIN eingeben (6, 7). Nach erfolgter Bezahlung schickt der Server eine Bestätigung zurück (8) und der Einkauf kann im Record-Store des Midlets abgelegt werden (9).

Eine gesicherte Kommunikation zwischen Handy und Bezahlserver drängt sich aus Datenschutzgründen, nicht aber aus sicherheitstechnischen Gründen auf, denn die Daten können von einem Dritten nicht verändert werden, da die Meldung durch einen kryptologischen Prüfwert gesichert ist. Damit jedoch nicht öffentlich sichtbar ist, wer was eingekauft hat, werden die Daten sinnvollerweise über eine SSL-Verbindung an den Server übertragen.

Zugriff auf die Tags: JSR 257

Die Preisinformationen sind in unserem Feldversuch auf den NFC-Tags im *NFC Data Exchange Format* (NDEF) abgelegt [NDEF]. NDEF definiert die Datenstrukturen für den Austausch von Informationen zwischen NFC-Geräten und NFC-Tags bzw. zwischen verschiedenen NFC-Geräten. Anwendungsdaten werden (zusammen mit Metainformationen) in einem oder mehreren NDEF-Records abgelegt. Diese Records ermöglichen den Transport verschiedener Datenformate. Wie auf dieser Basis typische Daten in den Tags repräsentiert und auf den Geräten interpretiert werden, wird in der *NFC Record Type Definition* (RTD) Spezifikation definiert. RTDs können einzelne Datensätze, wie z.B. eine URI, eine Telefonnummer, eine SMS oder ein Text sein. Für unsere Anwendung haben wir einen eigenen, proprietären Datentyp mit URN *urn:nfc:ext:fhnw.ch:shop* definiert, welcher den NFC-Konventionen folgt (es handelt sich um einen *NFC Forum External Type*).

Der Zugriff auf die NFC-Tags erfolgt in Java über das *Contactless Communication API* (JSR-257) [JSR257]. Dieses API unterstützt folgende Tag-Typen:

- *NDEF Tags*, deren Daten gemäss der NDEF-Spezifikation abgelegt sind;
- *RFID Tags*, deren Daten in einem proprietären Format abgelegt sind;
- *ISO14443 Karten*: Smartcards, auf die mit speziellen Kommandos zugegriffen werden muss;
- *visuelle Tags:* JSR-257 unterstützt auch das Lesen von visuellen 2D-Barcodes über die im Mobiltelefon eingebaute Kamera.

Das Contactless Communication API deckt den Reader/Writer-Modus ab. Sobald das Mobiltelefon in die Nähe eines Tags gehalten wird und ein Tag erkennt, wird eine Listener-Methode der Anwendung aufgerufen. Solche Tag-Listeners können in der Klasse *DiscoveryManager* registriert werden. Die Kommunikation mit den Tags erfolgt dann über das *Generic Connection Framework* (GCF). In Listing 1 ist angegeben, wie in einem Midlet ein Listener registriert werden kann, welcher auf unsere proprietären Tags reagiert.

Das Interface *NDEFRecordListener* definiert die Methode *recordDetected(NDEFMessage msg)*, deren Implementierung für unsere Anwendung in Listing 2 gezeigt wird. Die beim Lesen eines Tags

```
public void recordDetected(NDEFMessage msg) {
  NDEFRecord[] recArray = msg.getRecords();
  for(int i=0; i<recArray.length; i++){
    NDEFRecord rec = recArray[i];
    NDEFRecordType type = rec.getRecordType();
    if(type.getFormat() == NDEFRecordType.EXTERNAL_RTD
       && "fhnw.ch:shop".equals(type.getName())){
       addItem(rec.getPayload());
       return;
     }
  }
}
```

Listing 2: Implementierung des Tag Listeners

```
private static final byte[] SELECT = {
    0x00, // CLA Class
    0xA4, // INS Instruction
    0x04, // P1 Parameter 1
    0x00, // P2 Parameter 2
    0xA0, // Length
    0x63,0x64,0x63,0x00,0x00,0x00,0x00,0x32,0x32,0x31 // AID
};

String uri = System.getProperty("internal.se.url");
ISO14443Connection conn = (ISO14443Connection) Connector.open(uri));
byte[] result = conn.exchangeData(SELECT);
if (result[0] != 0x90 || result[1] != 0x00)
    throw new RuntimeException("could not select applet");
```

Listing 3: Absetzen eines SELECT-Befehls

aufgerufene Methode *addItem()* fügt das eingelesene Produkt dem Warenkorb hinzu. Dabei wird der Byte-Array, den die Methode *getPayload()* zurückliefert, mit Hilfe eines *DataInputStream* ausgelesen.

Java ME unterstützt die Möglichkeit, dass ein Programm beim Berühren eines NFC-Tags automatisch gestartet wird. Dazu muss das Programm in der Push-Registry registriert werden. Die Push-Registry ist Teil der Application Management Software, welche den Lebenszyklus aller Midlets steuert. Durch die Push-Registry muss eine Anwendung, die auf ein bestimmtes Ereignis wartet, nicht ständig aktiv sein. In unserer Anwendung erfolgt diese Registrierung dynamisch beim ersten Start der Applikation.

JavaCard Transaktions-Signierungs Applet

Auf die im SE abgelegten JavaCard Applets wird mit den im *Security and Trust Services API* (SATSA) definierten Methoden zugegriffen. Java ME Applikationen können so die sichere Ausführungsumgebung und den sicheren Datenspeicher des SE benutzen. Das Teilpaket SATSA-APDU unterstützt dabei die Kommunikation mit dem SE auf der Basis von *Application Protocol Data Unit* (APDU) Befehlen [APDU]. Die Struktur dieser Befehle ist in der Norm ISO/IEC 7816-4 festgelegt [ISO7816]. Auf den Series-40 Geräten von Nokia (die wir im Feldversuch verwendet haben) ist der Zugriff auf das SE auch mit der im *Contactless Communication API* (JSR-257) spezifizierten *ISO14443Connection* möglich [LaRo10].

Um mit einem im SE abgelegten JavaCard Applet kommunizieren zu können, muss dieses zuerst selektiert werden. Dazu wird ein APDU-Select-Befehl mit dem Application Identifier des gewünschten Applets an das SE übermittelt. In Listing 3 ist angegeben, wie ein solcher Select-

```
public class TXSigningApplet extends Applet {
  private final static byte INS_INIT = 0x01;
  private final static byte INS_SIGN = 0x02;
  private final static byte INS_MSISDN = 0x04;

  private byte[] msisdn;
  private byte[] key;
  private boolean initialized = false;

  public static void install(byte[] b, short off, byte len) {
    new TXSigningApplet().register(b, (short)off+1, b[off]);
  }

  public void process(APDU apdu) {
    // Return 9000 on SELECT
    if (selectingApplet()) { return; }
    byte[] buf = apdu.getBuffer();
    switch (buf[ISO7816.OFFSET_INS]) {
    case (byte) INS_GET_MSISDN:
      apdu.setOutgoing();
      apdu.setOutgoingLength((byte) msisdn.length);
      apdu.sendBytesLong(msisdn, (short)0, // offset
                         (byte) msisdn.length);  // length
      break;
    case (byte) INS_INIT: cmdInit(apdu); break;
    case (byte) INS_SIGN: cmdSign(apdu); break;
    default:
      ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
    }
  }
}
```

Listing 4: Implementierung des TXSigning-Applets (Methoden cmdInit() und cmdSign() fehlen)

```
private static final byte[] GET_MSISDN = {
  0x80, // CLA Class
  0x04, // INS Instruction
  0x00, // P1 Parameter 1
  0x00, // P2 Parameter 2
  0x10  // LE maximal number of bytes expected in result
};

byte[] result = conn.exchangeData(GET_MSISDN);
int len = result.length;
if (result[len-2] != 0x90 || result[len-1] != 0x00)
  throw new RuntimeException("could not retrieve msisdn");
byte[] data = new byte[len-2];
System.arraycopy(result, 0, data, 0, len-2);
String msisdn = new String(data).trim();
```

Listing 5: Abfrage der Telefonnummer im SE

Befehl aus dem Midlet über eine *ISO14443Connection*-Verbindung an das SE übermittelt wird.

Das JavaCard Applet *TXSigningApplet,* welches eine Meldung mit einem kryptologischen Hash-Prüfwert ergänzt und danach verschlüsselt, ist nach den üblichen JavaCard-Regeln aufgebaut:

- die Klasse ist abgeleitet von der Klasse *javacard.framework.Applet;*
- die Initialisierung des gesamten Speichers wird bei der Deklaration von Objektvariablen oder im Konstruktor vorgenommen;
- die Klasse enthält eine statische *install*-Methode, welche die *register*-Methode der Basisklasse aufruft;
- in der *process*-Methode werden die eingehenden APDU-Befehle gemäss dem Instruction-Byte an die zugehörige Methode weitergeleitet, welche dann die Antwort zurückschickt.

In Listing 4 ist ein Auszug aus der Implementierung dieses Applets angegeben. Das *TXSigning*-Applet unterstützt folgende Befehle:

- *INS_INIT*: Mit diesem Befehl wird das Applet initialisiert. Bei der Initialisierung werden der private Schlüssel sowie die Kundennummer (aktuell die MSISDN) angegeben. Diese Funktion kann genau einmal aufgerufen werden.
- *INS_MSISDN*: Mit diesem Befehl kann die Kundennummer (MSISDN) ausgelesen werden.
- *INS_SIGN*: Mit diesem Befehl wird aus den Daten mit SHA1 eine Prüfsumme bestimmt. Diese Prüfsumme wird danach mit dem privaten Schlüssel verschlüsselt und zurückgegeben.

In Listing 5 sieht man, wie aus dem Midlet mit der Funktion *INS_MSISDN* die Kundennummer abgefragt werden kann.

OTA-Loader

Ein letztes Problem ist, wie das Applet in das SE des Mobilgeräts abgespeichert und wie dieses mit der Identifikationsnummer des Kunden und dem privaten Schlüssel initialisiert werden kann. Diese Installation und Provisionierung des Applets kann mit einem NFC Writer erfolgen. Dazu ist es jedoch nötig, dass das Mobiltelefon bei einer Servicestelle vorbeigebracht wird.

Interessanter ist es, diese Installation „Over-the-Air" (OTA) vorzunehmen. Dazu wird via SGM/UMTS eine Verbindung zu einem speziellen Java-Midlet aufgebaut, welches auf das SE zugreifen kann. Dieses Midlet übernimmt dabei die Rolle eines Proxy: Auf der einen Seite greift es über die interne Schnittstelle des Handys auf das SE zu, auf der anderen Seite wird die Internet-Verbindung des Handys genutzt, um die Befehle von einem OTA-Server an die Smartcard weiterzuleiten und deren Antwort wiederum zum OTA-Server zurück zu senden.

Das Java-Midlet mit dem OTA-Loader kann mit einer Dienstmeldung installiert und gestartet werden. Das dabei verwendete JAD-file wird per-

```
void seCommand() throws IOException, ContactlessException{
  short b0 = (short)( is.read() & 0xFF );
  short b1 = (short)( is.read() & 0xFF );
  short apduLength = (short)((b0 << 8) + b1);
  int n = 0; byte[] apdu = new byte[apduLength];
  while(n < apduLength){
    int read = is.read(temp, n, apduLength-n);
    if(read > -1) n += read; else throw new IOException();
  }
  //send to SE
  byte[] result = seConn.exchangeData(apdu);
  byte[] length = new byte[]{ (byte)(result.length & 0xFF), (byte)(result.length & 0xFF) };
  os.write(length);
  os.write(result);
  os.flush();
}
```

Listing 6: Hauptschleife des OTA Proxies

sonalisiert, d.h. die im JAD-file definierte Server-URL enthält Parameter, die den Kunden identifizieren. Nach dem Start wird eine Verbindung zum OTA-Server aufgebaut und die pendenten Befehle werden ausgeführt. Dieses Midlet implementiert im Wesentlichen ein Proxy, welches die Daten vom Server in das SE schreibt. Listing 6 zeigt, wie ein SE-Kommando vom OTA-Server gelesen und in das SE geschrieben wird.

Der Zugriff auf das SE erfolgt mit APDU Befehlen, welche in der *GlobalPlatform Card* Specification definiert sind [GPCS]. Um diese Befehle zu erzeugen, haben wir das *SourceForge* Projekt *GlobalPlatform* verwendet [GP]. Dieses Projekt verfolgt mit seinen beiden Teilprojekten *GlobalPlatform-Library* und *GPShell* das Ziel, die *GlobalPlatform Card Specification* als API bereitzustellen. Die Herausgeber der Spezifikation und das Projekt haben nichts miteinander zu tun; das *SourceForge*-Projekt hat lediglich den Namen übernommen. Die *GlobalPlatform-Library* codiert die Befehle der *Card Specification* und sendet diese über die Smartcard-API direkt an den Smartcard-Reader. Damit wir diese Befehle über einen anderen Kommunikationskanal zum SE übertragen können, haben wir anstelle der Smartcard-API Aufrufe einen Callback-Mechanismus eingebaut.

Für den Datenaustausch zwischen dem Server und dem SE wird das *Secure Channel Protocol* (SCP 02) verwendet. Die Daten werden dabei mit dem 3DES Algorithmus mit Schlüsseln der Länge 112bit verschlüsselt. Weder das Proxy-Midlet noch ein Man-In-The-Middle können daher die Daten, die an das SE gesendet werden, verstehen. Als zusätzliche Sicherheit könnte die Verbindung zwischen dem OTA-Midlet und dem OTA-Server mit SSL verschlüsselt werden.

Mit der Umwandlung der SIM zur Mehrzweck-Smartcard könnte der Zugriff auch über die bestehenden GSM-Kanäle erfolgen, analog zur Aktualisierung der SIM-Karte durch den Mobilfunkbetreiber (MNOs, Mobile Network Operators). Ein bekanntes Beispiel für ein Update der SIM Karte ist die Erneuerung der Roaming-Partner-Liste. Diese Variante konnte im Rahmen unseres Feldversuches nicht umgesetzt werden da entsprechende Mehrzweck-Smartcards noch nicht verfügbar sind.

Zusammenfassung

Wir haben in diesem Artikel die technische Realisierung des touch'n pay Feldversuchs dargestellt, welcher noch bis Ende Oktober 2010 läuft. Das Projekt touch'n pay hat den ersten Preis des *NFC Forum Global Competition* Wettbewerbs 2010 in der Kategorie für den weltweit besten kommerziellen NFC-Service gewonnen [A10a] und ist an der Contactless Intelligence 2010 Konferenz für den *Monkey Award* in der Kategorie *NFC in the High Street* nominiert worden [A10b]. Der Monkey Award wird von Visa an Firmen oder Projekte vergeben, welche kontaktlose Technologien fördern.

Dieser Feldversuch ist ein weiterer von vielen in der Vergangenheit bereits durchgeführten NFC-Feldversuchen. Die NFC-Technologie scheint reif zu sein, doch die breite Einführung von NFC wird durch ein Henne-Ei Problem verzögert: Anwendungen wird es erst dann geben, wenn genügend NFC-Geräte verfügbar sind, und die Gerätehersteller und Mobile Operators werden erst dann NFC-Geräte auf den Markt bringen, wenn diese von den Kunden auch verlangt werden.

Es gibt jedoch in letzter Zeit wichtige Zeichen, dass die Gerätehersteller in naher Zukunft doch NFC-Geräte auf den Markt bringen könnten. So hat Nokia kürzlich angekündigt [NW10a], dass 2011 alle neuen Nokia Smartphones mit NFC ausgestattet sein sollen (wobei aus der Ankündigung nicht hervorging, wie das SE implementiert sein wird). Auch Apple scheint seit längerem aktiv mit NFC zu arbeiten. Apple hält mehrere NFC-Patente und hat kürzlich einen NFC-Experten als *Mobile Commerce Product Manager* angestellt [NW10b]. Auch in der Schweiz gibt es einige NFC-Aktivitäten welche im Report *Mobile Contactless Payment und Mobile Ticketing: Ein Schweizer Statusbericht* von der KPMG und ETHZ [KPMG10] zusammengefasst sind.

Wie auch immer sich NFC entwickeln wird: Wir sind bereit!

Referenzen

[A10a] NFC-Forum, 2010 Global Competition; http://www.nfc-forum.org/events/2010_competition.

[A10b] The Contactless Intelligence Monkey Awards 2010, London, 2010; http://c-i.tv/index.php?id=394.

[APDU] Alexander Shevelev, APDU Command Liste; http://cheef.ru/docs/HowTo/APDU.table.

[GP] GlobalPlatform, SourceForge Projekt; http://sourceforge.net/projects/globalplatform.

[GPCS] GlobalPlatform. Card Specification 2.1.1., 2003; http://www.globalplatform.org/specificationscard.asp.

[ISO7816] ISO, Identification cards & Integrated circuit cards Part 4: Organization, security and commands for interchange, ISO/IEC 7816-4, 2005.

[JSR257] JSR-257, Contactless Communication API, Final Release, Technical Report, Syn Microsystems, Inc., 2006; http://jcp.org/en/jsr/detail?id=257.

[KPMG10] KPMG Mobile Contactless Payment and Mobile Ticketing; http://tinyurl.com/395uysa.

[LaRo10] Josef Langer, Michael Roland, Anwendungen und Technik von Near Field Communication (NFC), Springer Verlag, 2010.

[NDEF] NFC Forum, NFC Data Exchange Format (NDEF), Refv. 1.0. Technical Specification, 2006; http://www.nfc-forum.org/specs/spec_list/.

[NFC] NFC-Forum; http://www.nfc-forum.org/home/.

[NW10a] NFCWorld, All new Nokia smartphones to come with NFC from 2011, 2010; http://tinyurl.com/37d7ncj.

[NW10b] NFCWorld, Apple hires NFC expert as mobile commerce product manager, 2010; http://tinyurl.com/24zuasu.