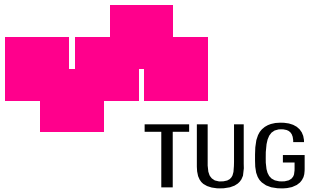
Informationstechnik Projekt (Technische Informatik)

BlueKey

Rainer Matischek, Stephan Weinberger

Institut für Technische Informatik
Technische Universität Graz
Vorstand: O. Univ.-Prof. Dipl.-Ing. Dr. techn. Reinhold Weiß



Betreuer: Dipl.-Ing. Dr. techn. Martin Schmid

Graz, im Januar 2004

Kurzfassung

Die vorliegende Arbeit stellt die Entwicklung von *BlueKey* vor, die in Form eines IT Projekts 2003/04 am Institut für Technische Informatik durchgeführt wurde.
BlueKey ist eine Erweiterung des am Institut für Technische Informatik (ITI) entwickelten LOMOT-Systems, welche es ermöglicht, mittels Bluetooth-fähiger PDAs und lokaler Bluetooth Access Points abgeschlossene Bereiche (beispielsweise durch motorische Zylinderschlösser gesicherte Türen) durch Senden entsprechender Kennungen (User-ID, PIN-Code, BT-MAC-Adresse etc.) an einen Zugangsserver zu öffnen und zu betreten.
In dieser Dokumentation werden Design und Implementierung der *BlueKey*-Clientsoftware für Palm-PDAs sowie des entsprechenden Servers erläutert.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

AP	(Bluetooth) Access Point
BT	Bluetooth
CDMA	Code Division Multiple Access
GUI	Graphisches User Interface
ITI	Institut für Technische Informatik (an der TU Graz)
IR	InfraRot (Datenübertragung)
LAN	Lokal Area Network
LOMOT	LOcation based Messaging and Object Tracking
OBEX	Object Exchange (Standard zum Datenaustausch zwischen mobilen Geräten)
OS	Operating System (Betriebssystem)
PDA	Personal Digital Assistent (Handheld-Computer)
RF	Radio Frequency
TCP/IP	Transmission Control Protocol/Internet Protocol
VM	Virtual Machine (Java-Laufzeitumgebung)

Kapitel 1

Einleitung

1.1 Motivation

In letzter Zeit finden Geräte mit Bluetooth-Unterstützung immer größere Verbreitung. Mobiltelefone und Personal Digital Assistents (PDAs) verfügen oft schon von Haus aus über eine derartige Schnittstelle. Da diese Geräte vom Benutzer meist ständig mit sich getragen werden, bietet es sich an, sie zur berührungslosen Identifikation des Trägers zu verwenden. Die geringe Reichweite von etwa 10 Metern erlaubt darüber hinaus auch eine relativ genaue Ortung des Gerätes, sodass auch ortsbezogene Anwendungen möglich sind.
Eine derartige Anwendung stellt das automatische öffnen von Türen dar. Hierbei können die Rechenleistung und Eingabemöglichkeiten eines PDA verwendet werden, um verschiedene Zugangsstufen – vom simplen Öffnen bei Annäherung bis hin zur Abfrage von Passwörtern oder Pincodes – zu implementieren.
Am ITI wurde bereits das LOMOT-System entwickelt, welches eine allgemeine Umgebung zur Positionsverfolgung von Objekten sowie zur Verwaltung orts-, geräte- und personenbezogener Daten bereitstellt. In der vorliegenden Arbeit soll LOMOT um die Funktionalität der Zugangskontrolle und des Türöffnens erweitert werden. Dazu werden sowohl ein Client-Programm für Palm-PDAs als auch eine Server-Applikation zur Kommunikation mit der LOMOT-Datenbank entwickelt.

1.2 Überblick über LOMOT

LOMOT steht für **LO**cation based **M**essaging and **O**bject **T**racking. Das System ermöglicht es die Position von Objekten, welche mit RFID Tags [AIM03] oder Bluetooth [Blu03] ausgestattet sind, zu verfolgen und ortsbezogene Services zur Verfügung zu stellen.
Passive Geräte (RFID Tags) werden hierbei durch Tag-Lesegeräte lokalisiert. Geräte welche über eine Bluetooth-Schnittstelle verfügen, können über LAN-Accesspoints aktiv TCP/IP-Verbindungen zu Servern des LOMOT-Systems Verbindung aufnehmen, wodurch nicht nur die Position erkannt wird, sondern auch die weiteren Dienste von LOMOT verwendet werden können.
Das LOMOT-System stellt zu diesem Zweck eine Datenbank zur Verfügung, in welche nicht nur Tracking-Daten eingetragen werden können sondern auch weitere Daten, beispielsweise

Authentifizierungsdaten, Informationen über die Umgebung des Tag Readers bzw. Accesspoints oder orts- oder objektbezogene Nachrichten abgelegt sind.
Die ursprüngliche Version von LOMOT wurde 2002 von M. Polaschegg im Zuge eines IT Projekts am ITI entwickelt [Pol03]. 2003 wurde das System von M. Thonhauser überarbeitet und um ein webbasiertes GUI erweitert [Tho03].

1.3 Überblick über *BlueKey*

BlueKey stellt eine Erweiterung des LOMOT-Systems dar, die es ermöglichen soll, authorisierten Benutzern bei Annäherung an eine Tür automatisch den Zugang zu gewähren. Die Identifizierung und Authentifizierung des Benutzers soll dabei in mehreren Stufen möglich sein:

- Annäherung eines registrierten Gerätes
- Authentifizierung mittels Username und Passwort
- Eingabe eines zusätzlichen Pincodes für spezielle Türen

Zur Nutzung des Systems muss der Benutzer ständig einen Bluetooth-fähigen PDA (in diesem Fall einen Palm mit Bluetooth-Modul) bei sich tragen, auf dem die *BlueKey*-Clientsoftware läuft. Diese baut über ortsfeste Bluetooth-Accesspoints eine TCP/IP-Verbindung zum *BlueKey*-Server auf, welcher die nötigen Abfragen in der LOMOT-Datenbank durchführt und die dem AP zugeordneten Türen öffnet. Die Lokalisierung des Benutzers erfolgt also über den (bekannten) Standort des Bluetooth-APs über den sich der PDA ins LAN eingewählt hat.
Weiters kann der Server dem Client Meldungen zurückschicken, die den Benutzer z.B. darauf hinweisen, dass für einzelne Türen ein Passwort oder Pincode erforderlich ist. Diese Daten können vom Benutzer am PDA eingegeben werden.

Kapitel 2

Analyse bestehender Systeme

An dieser Stelle möchten wir verschiedene Schließsysteme gegenüberstellen.

2.1 Konventionelle Schließsysteme

Mechanische Schlösser

Die altbewährten mechanischen Schlösser zeichnen sich durch einen niedrigen Preis bei gleichzeitig hoher Stabilität und Zuverlässigkeit aus. Moderne Kodierungsmethoden erlauben eine hohe Zahl an Kombinationen. Die rein mechanische Abtastung wird heute auch vielfach durch magnetische Kodierungen ergänzt.
Zu den größten Nachteilen zählen jedoch die mangelnde Flexibilität bei Schlüsselverlust oder Erweiterungen bzw. Änderungen der Zugangsberechtigungen sowie die fehlende Möglichkeit einer zentralen Steuerung und Protokollierung.

Elektronische Schlösser

Die Nachteile rein mechanischer Lösungen können durch den Einsatz elektronischer Zugangsmechanismen verbessert werden. Diese können verschiedene Schließmechanismen zumeist in Kombination mit mechanischen Schlössern steuern.

Codeeingabe

Die einfachste Form eines elektronischen Schlosses stellt die Eingabe eines Codes per Tastatur dar. Theoretisch sind hierbei sehr viele Kombinationen möglich, in der Praxis beschränkt man sich jedoch meist auf 4- bis 6-stellige Ziffernfolgen, da der Benutzer sich diese leichter merken kann. Dies bedeutet aber gleichzeitig eine relativ geringe Sicherheit. Ein systematisches Durchprobieren aller Möglichkeiten kann z.B. durch eine Wartezeit bei Fehleingabe erschwert werden. Die Sicherheit kann auch durch die Kombination mehrerer Codes (z.B. Username + Passwort) erhöht werden.
Zu den großen Vorteilen zählt jedoch, dass der Benutzer keine weiteren Gegenstände (Schlüssel, Karte, etc.) bei sich tragen muss.

Magnetkarte

Die Magnetkarte ist ein reines Speichermedium, auf dem ein Identifikationscode abgelegt ist.

Insofern ist sie dem mechanischen Schlüssel sehr ähnlich. Allerdings bieten Kartenlesegeräten den Vorteil, dass die Zugangsberechtigungen in elektronischer Form gespeichert sind und daher ohne Eingriff in die Hardware geändert oder erweitert werden können. Für eine erweiterte Sicherheit bei Verlust oder Diebstahl kann hier auch eine zusätzliche Codeeingabe vorgesehen werden. Ein Beispiel für ein solches Zugangssystem ist das „CAIN II Access-Control System“ des Lawrence Livermore National Laboratory [RJK88]: Dort wird neben der Benutzer-ID auch der DES-Key für die Verschlüsselung des PIN-Codes auf der Karte gespeichert.

Chipkarte

Chipkarten haben den Vorzug, über die reine Speicherung von Daten hinaus noch weitere Aufgaben durchführen zu können, da sie über einen eingebauten Prozessor verfügen können. Somit können die ausgetauschten Daten verschlüsselt oder bei jeder Benutzung andere Codes ausgetauscht werden. Daher bieten diese Systeme eine höhere Sicherheit als Magnetkarten.

2.2 Berührungslose Zugangssysteme

Biometrische Systeme

Biometrische Zugangsmechanismen verwenden zur Identifikation des Benutzers verschiedene (unverwechselbare) Körpermerkmale, wie z.B. den Fingerabdruck, die Muster der Netzhaut, Gesichts- oder Stimmmerkmale. Zu den Vorteilen zählen wiederum, dass der Benutzer keine weiteren Gegenstände zur Identifikation benötigt. Außerdem bieten biometrische Merkmale theoretisch eine hohe Fälschungssicherheit. Der Hauptnachteil ist jedoch, dass das maschinelle Erkennen der Merkmale sehr aufwändig ist. So ist es z.B. keineswegs trivial, das Kamerabild eines Gesichts von einem Foto dieses Gesichtes zu unterscheiden. Außerdem unterliegen auch die Merkmale selbst gewissen Schwankungen (z.B. andere Frisur bei der Gesichtserkennung oder Heiserkeit bei einem Stimmanalysesystem).
Aus diesem Grund sind biometrische Zugangsmechanismen noch nicht sehr weit verbreitet und unterliegen noch laufenden Weiterentwicklungen.

RFID Tags

RFID Tags erweitern die Möglichkeiten der Magnet- und Chipkarte um berührungslosen Datenaustausch. Für den Benutzer bedeutet dies vor allem einen größeren Komfort, da er sich nicht mehr händisch beim System anmelden muss. Je nach Sicherheitsanforderungen können Tag-basierte Systeme auch um die Eingabe eines Zugangscodes erweitert werden.
Die Technik der RFID Tags ist inzwischen sehr weit entwickelt und wird heute in vielen anderen Bereichen wie z.B. zur Lokalisierung von Gegenständen verwendet (Diebstahlschutz, Logistiksysteme, Lagerhaltung, etc).

RFID Tags in Kombination mit mechanischen Schlüssel

Weiters sehr bewährt hat sich die Kombination mechanischer Schlüssel mit eingebauten RFID Tags. Ein Beispiel hierfür ist das Electronic Control System (ECS) Schließsystem von EVVA [Sch02]. Dieses ermöglicht sowohl rein berührungslose Steuerung bzw. Überwachung der Schließvorgänge als auch die Möglich Türen mechanisch zu öffnen bei gleichzeitiger elektronischer Überwachung des Vorgangs. Die verwenden elektronischen Motorzylinder erhöhen außerdem die Zuverlässigkeit bei eventuellen Ausfällen des elektronischen Systems.

Beispiel proprietäre Funkübertragung

Ein automatisches Zugangskontrollsystem basierend auf Funkübertragung wurde von der Universität Uberlândia (Brasilien) entwickelt [PBFF99]. Die Benutzer erhalten einen RF-Transponder, der sowohl für Zugangskontrolle als auch zur automatischen Steuerung der Haustechnik verwendet wird. Die verwendete Übertragungstechnologie wurde in dieser Anwendung getestet. Zum Einsatz kommt in diesem System eine Datenübertragung mittels CDMA, die Spread-spectrum sowie Frequency-hopping verwendet. Der größte Vorteil dieser Technologie ist, dass viele Benutzer gleichzeitig mit dem Kontrollsystem kommunizieren können, ohne sich gegenseitig zu stören.
Da Bluetooth eine ähnliche Übertragungstechnologie nutzt, kann man diesen oben erwähnten Vorteil auch in einem Bluetooth-basierten Zugangssystem nutzen.

2.3 Zugangssysteme mittels Bluetooth

Prototyp Ski Access

Die Firma Skidata AG [Ski02] beschäftigt sich seit langem mit verschiedenen Zugangssystemen für Schilifte, Parksysteme und Gebäude. Derzeit sind noch am häufigsten Systeme mit Kartenlesegeräten im Einsatz (vgl. 2.1). Wegen der immer größer werdenden Verbreitung von Bluetooth-fähigen Mobiltelefonen entwickelte Skidata auch Systeme, die berührungslosen Zugang als auch bargeldloses Bezahlen mit dieser Technik ermöglichen sollen. Skidata präsentierte 2002 einen Prototyp eines Personenzutrittsystems, das mittels Bluetooth-Handy Zugang zu Schiliften ermöglichen soll. In dieser Version wird aber nur die Bluetooth Hardwareadresse des Handys zur Authentifizierung herangezogen, damit auf dem jeweiligen Handy keine zusätzliche Software laufen muß, was in der Praxis noch zu unsicher ist, denn diese Adresse kann man mit geeigneter Technik relativ einfach fälschen.

Wireless access control system mittels Bluetooth

Gegen Ende der Entwicklung dieses Projekts wurde eine weitere Veröffentlichung eines Zugangskontrollsystems basierend auf Bluetooth bekannt, das von der Lappeenranta University of Technology [HJP03] enwickelt wurde. Bei dieser Entwicklung werden vor allem die Sicherheitsaspekte bei Verwendung von Bluetooth als drahtlosen Zugangsmechanismus beschrieben:
Die Benutzer verwenden einen Bluetooth-fähigen PDA als Client (Personal Trusted Device, PTD), auf dem ein Programm zur Anmeldung läuft. Auf die verwendete Hard- und Software des Clients wird nicht genauer eingegangen. Dieser Client verbindet sich auf Befehl des Users („Türen öffnen“) über ein ihm bekanntes Bluetooth-fähiges „User Connection Module“ zum Administration Point. Dabei wird die interne Verschlüsselung von Bluetooth aber für ein großes Zugangssystem als nicht ausreichend erachtet. Der Grund dafür ist u.a. der von Bluetooth zum sogenannten Pairing verwendete PIN-Code: Dieser muss allen Devices, die sich mit dem System verbinden wollen bekannt sein und ansonsten geheim bleiben. Das ist in einem großen System in dem viele Benutzer mit ihren Devices ein- und ausgehen können nicht gesichert.
Die sichere Authentifizierung des Clients wird also auf die Applikations-Schicht ausgelagert. Hierfür wird eine Public Key Infrastructure (PKI) am Administration Point verwendet. Es wird also eine standardisierte asymmetrische Verschlüsselung mittels privaten+öffentlichen Schlüssel verwendet. Daher muss ein neuer Benutzer zuerst ein Schlüssel mit seinem Device erzeugen und beim Administration Point ein Zertifikat anfordern. In diesem verschlüsselten Zertifikat werden

neben Benutzerdaten auch die BT Adresse des Clients sowie die IDs der zu öffnenden Türen gespeichert. Damit ist gesichert, dass nur dieses spezielle Device mit den richtigen Benutzerdaten/Zertifikat als zugangsberechtigt erkannt wird.
Das System ist weiters auch gegen ein eventuelles Abhören und Fälschen der gesendeten Authentifizierungsdaten abgesichert. Dafür wird vom Server nach erfolgter Validierung eine zufällige Sequenz erzeugt (Challenge), aus dieser mit dem asymmetrischen Schlüsselverfahren von Server und Client ein Hash-Code, der dann zur endgültigen Authentifizierung am Server verglichen wird (Abbildung 2.1).

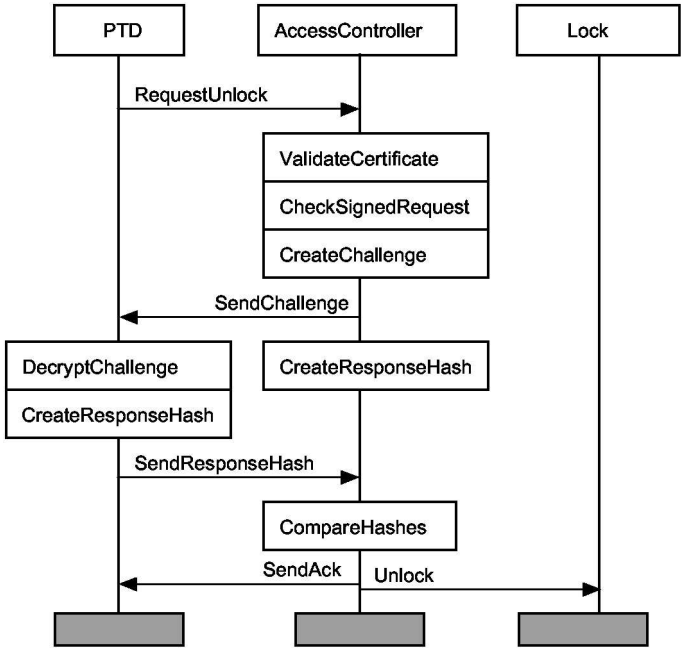


Abbildung 2.1: Sichere Authentifizierungssequenz mittels PKI

Die Überlegungen der Autoren zum Thema Sicherheit sind für ein Zugangsystem in der Praxis auf alle Fälle unerlässlich. Das hier vorgesehene Projekt *BlueKey* verwendet derzeit eine reine User/Password Authentifizierung mit optionaler proprietärer Verschlüsselung, was in einer derzeitigen Testumgebung als ausreichend empfunden wird. Es wäre aber in einer weiteren Ausbaustufe von *BlueKey* auch eine ähnliche Verschlüsselung mittels Public Key Verfahren vorzuziehen.

Kapitel 3

Design

3.1 Systemaufbau

Das System *BlueKey* besteht grundsätzlich aus 5 Komponenten (Abbildung 3.1), auf die nun kurz näher eingegangen wird:

- (1) Bluetooth-fähiger PDA, als Client bezeichnet
- (2) Bluetooth Access Point, dient zur Anbindung an das LAN
- (3) Server, dient zur Kommunikation mit Client und Datenbank
- (4) LOMOT Datenbank
- (5) Tür mit Motorzylinder oder ähnlichem Schließmechanismus, verbunden mit dem LAN

(1) Client

Der Client ist ein Bluetoothfähiges Device, das der Benutzer mit sich trägt und eine automatische Authentifizierung am System und in weiterer Folge das Öffnen berechtigter Türen ermöglicht. Dies kann ein RF-Tag (ohne Ein-/Ausgabemöglichkeit), ein PDA aber auch ein Mobiltelefon sein. Wesentlich ist, dass auf dem Client eine Software läuft, die die Kommunikation mit dem System entsprechend des Übertragungsprotokolls unterstützt (siehe Kapitel 3.4). Für dieses Projekt wurde aufgrund der Verfügbarkeit der Bluetooth-Komponenten (Hardware und Software) ein Palm OS PDA verwendet (siehe Kapitel 3.2).

(2) BT access point

Der Bluetooth Access Point (BAP) kann in zwei Kategorien unterteil werden: Handelsübliche LAN-Access Points und proprietäre APs, die an einer PC Hardware angeschlossen sind.
LAN-APs haben den Vorteil, dass sie von verschiedenen Herstellern erhältlich und einfach zu konfigurieren sind. Diese AP können aber ausschließlich als passive Verbindungsschnittstelle zwischen dem Client und dem LAN werwendet werden, d.h. der Client muß sich diese AP selbstständig suchen. Es gibt für diese LAN-APs leider keine einheitliche Schnittstelle um weitere Bluetooth-Funktionen zu nutzen.
Besser geeignet sind hierfür proprietäre „aktive“ APs (oder Smart Access Points), auf denen eigenständige Programme laufen können. Ein derartiger aktiver AP wird derzeit im Zuge eines weiteren

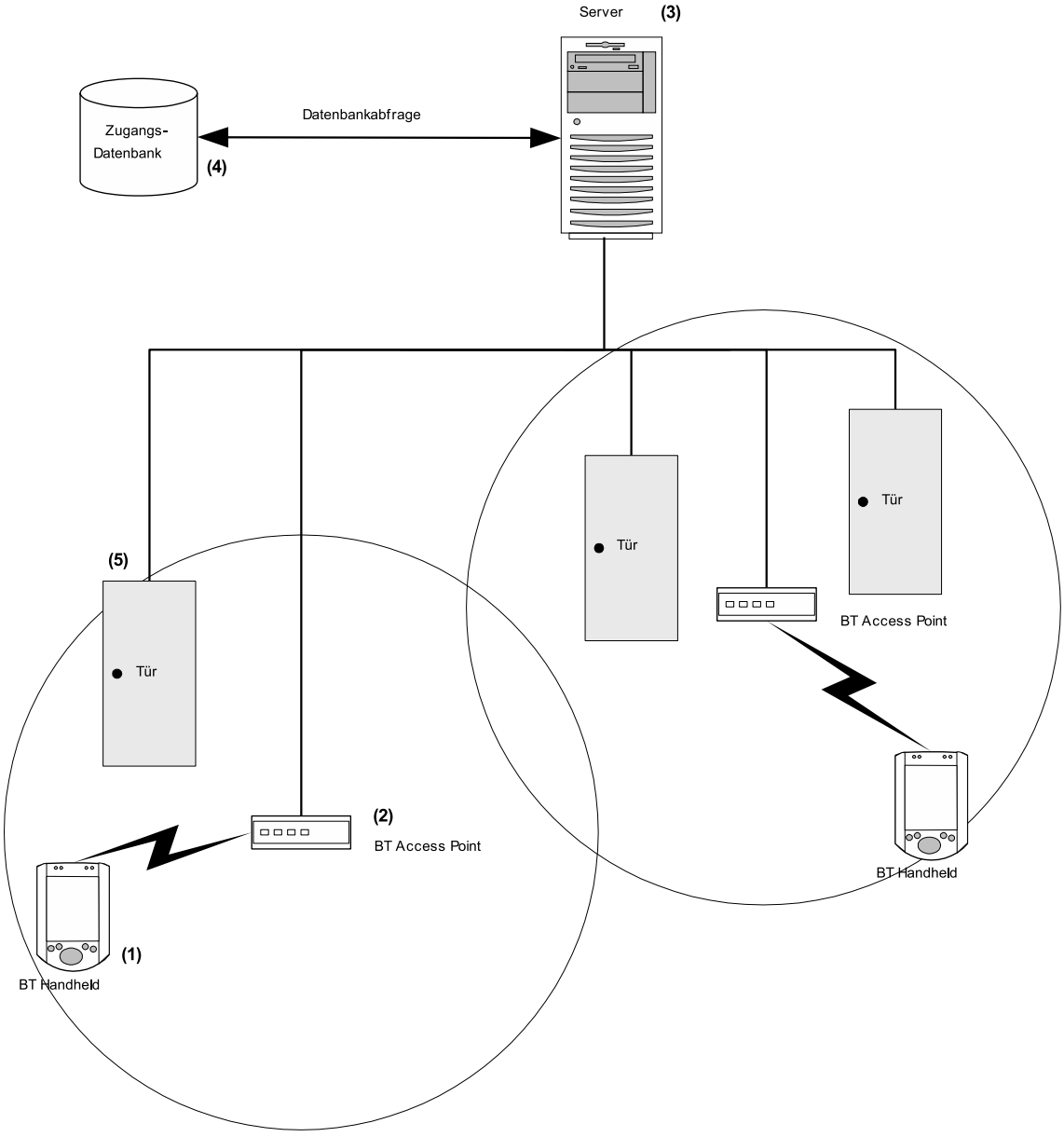


Abbildung 3.1: Systemkomponenten: (1) PDA, (2) AP, (3) *BlueKey* Server, (4) LOMOT-Datenbank, (5) Tür mit Motorzylinder

Projekts am ITI realisiert [SK04]. Für *BlueKey* werden folgende Funktionen des aktiven AP benötigt:

- Zyklisches Suchen nach Bluetooth-Devices im Empfangsbereich
- „Aufwecken“ eines neu erkannten *BlueKey* Clients mittels Senden einer Datei über OBEX

Die Verwendung eines solchen aktiven APs hat für das Projekt wesentliche Vorteile. Der wichtigste Punkt ist ein viel geringerer Energieverbrauch des mobilen Clients. Ein zyklisches Bluetooth-Inquery erfordert andauerndes Senden und Empfangen und somit permanenten Bluetooth Active Mode (ca. 40-60 mA) [Cha00]. Wenn diese Aufgabe ein AP übernimmt, muss der Client nur empfangsbereit bleiben, d.h. auf ein Inquery sowie auf ankommende OBEX Dateien reagieren.
Ein weiterer Vorteil ist, dass auf dem Client nicht permanent die Clientsoftware laufen muss. Wird also ein Bluetooth-fähiger PDA dafür verwendet, kann er für alle anderen Aufgaben weiter genutzt werden. Erst wenn er in den Empfangsbereich eines neuen APs gelangt und somit als *BlueKey* Client aktiviert wird, muss die Software aktiv werden.

(3) Server
Der Server dient als Schnittstelle zwischen Client, Datenbank und Türöffner. Der Server hat folgende Aufgaben: Er wartet auf eine TCP/IP-Verbindung, die vom Client aufgebaut wird, vergleicht die Authentifizierungsdaten mit der Datenbank, und veranlasst bei Bedarf ein Türsteuerungssystem zum öffnen zugewiesener Türen (siehe Kapitel 3.3).

(4) LOMOT Datenbank
Diese Datenbank enthält sämtliche Authentifizierungsdaten über zugangsberechtigte Benuzter/Gruppen und deren zugewiesene Türen. Weiters wird dort die Trackingereignisse (Positionsveränderungen) der Clients gespeichert. Die Verbindung zur Datenbank wird vom Server hergestellt (siehe Kapitel 3.3).

(5) Tür
Die Türen sind mit einem Motorzylinder versehen und über ein vernetztes Steuerungssystem mit dem LAN verbunden. Die genaue Realisierung dieser Steuerung ist nicht Teil des Projekts und wird daher nicht weiter beschrieben.

3.2 Client

Als *BlueKey*-Client wurde einen PDA basierend auf Palm OS gewählt, als Testgerät dient ein Palm Tungsten T. Ausschlaggebend für diese Entscheidung waren folgende Punkte:

- Bluetoothfähiger PDA, da einen handliches Device mit Touchscreen für diese Anwendung gut geeignet ist
- Die handelsüblichen PDAs (Palm OS, Pocket PC, etc) bleiben außerdem universell einsetzbar, d.h. es können neben *BlueKey* weiterhin alle anderen Anwendungen (Terminverwaltung, etc) verwendet werden
- Palm OS, weil es weit verbreitet ist und eine große Anzahl an Software teilweise auch mit Quellcode erhältlich ist

- Entwickler-Sourcen sowie das BT-API sind frei ferfügbar

Auf dem Palm ist eine von uns entwickelte Software für installiert, deren Aufbau nun kurz beschrieben werden soll.

Initialisierung/Parameter
Als erstes muss es eine Möglichkeit geben, folgende verschiedene Konfigurationsdaten zu speichern: Username und Passwort, IP-Adresse/Port des *BlueKey*-Servers. Diese müssen im allgemeinen nur editiert werden, beispielsweise wenn ein Benutzer einen Gastzutritt unter einer anderen Kennung in einen ihm normalerweise nicht zugewiesenen Zugangsbereich benötigt.

Programmablauf
Der Client geht nach erfolgter Initialisierung in Bluetooth-Empfangsbereitschaft. D.h. er muß auf ein Bluetooth-Inquery sowie auf ein ankommendes OBEX Objekt reagieren können.
Kommt der Client in den Empfangsbereich eines AP, wird er von diesem erkannt und per OBEX aufgeweckt (durch Empfangen einer vordefinierten Datei). Die BT-Adresse dieses APs wird nun als aktueller AP für die LAN-Verbindung gespeichert.
Nachdem der Client vom AP aufgeweckt wurde, baut er über diesen ins LAN kurz eine TCP/IP Verbindung zum Server auf (Abbildung 3.2). Es werden sämtliche Authentifizierungsdaten sowie die aktuelle Position (BT-Adresse des verbundenen APs) zum Tracking des Users gesendet. Eventuelle Antworten des Servers werden im Ausgabefeld angezeigt. Sollte für eine bestimmte Tür ein PIN-Code benötigt werden, kann dieser mit dem Ziffernfeld eingegeben und anschließend übertragen werden.

Datenübertragung
Der Client muss folgende Kommunikationsmechanismen unterstützen:

- Bluetooth OBEX Object Push Profile, um von einem aktiven AP aufgeweckt werden zu können (Empfangen einer Datei)
- Bluetooth LAN Access Profile, um eine PPP Verbindung über den AP ins LAN aufbauen zu können
- TCP/IP Protocol Stack, um mit dem Server eine Verbindung aufzubauen
- Kommunikation mit dem Server nach dem *BlueKey* Übertragungsprotokoll (siehe Kapitel 3.4)

User Interface (UI)
Das UI soll einfach aufgebaut sein und nur für den Benutzer wesentliche Felder anbieten, da die Größe der Displays mobiler Endgeräte sehr beschränkt ist. Daher sind im Hauptfenster der Anwendung nur 3 Bereiche zu finden: Ein Ziffernfeld für den PIN-Code, ein Button zur manuellen Übertragung und ein Ausgabefeld. Um eine möglichst einfache und schnelle Bedienung zu ermöglichen, sind die Eingabefelder noch groß genug gewählt, daß man sie eventuell auch mit der Fingerspitze bedienen kann (Abbildung 3.3).
Weitere Eingabemöglichkeiten, z.B. die der Benutzerkonfiguration, werden nicht sehr häufig benötigt und sind über das Menü erreichbar.

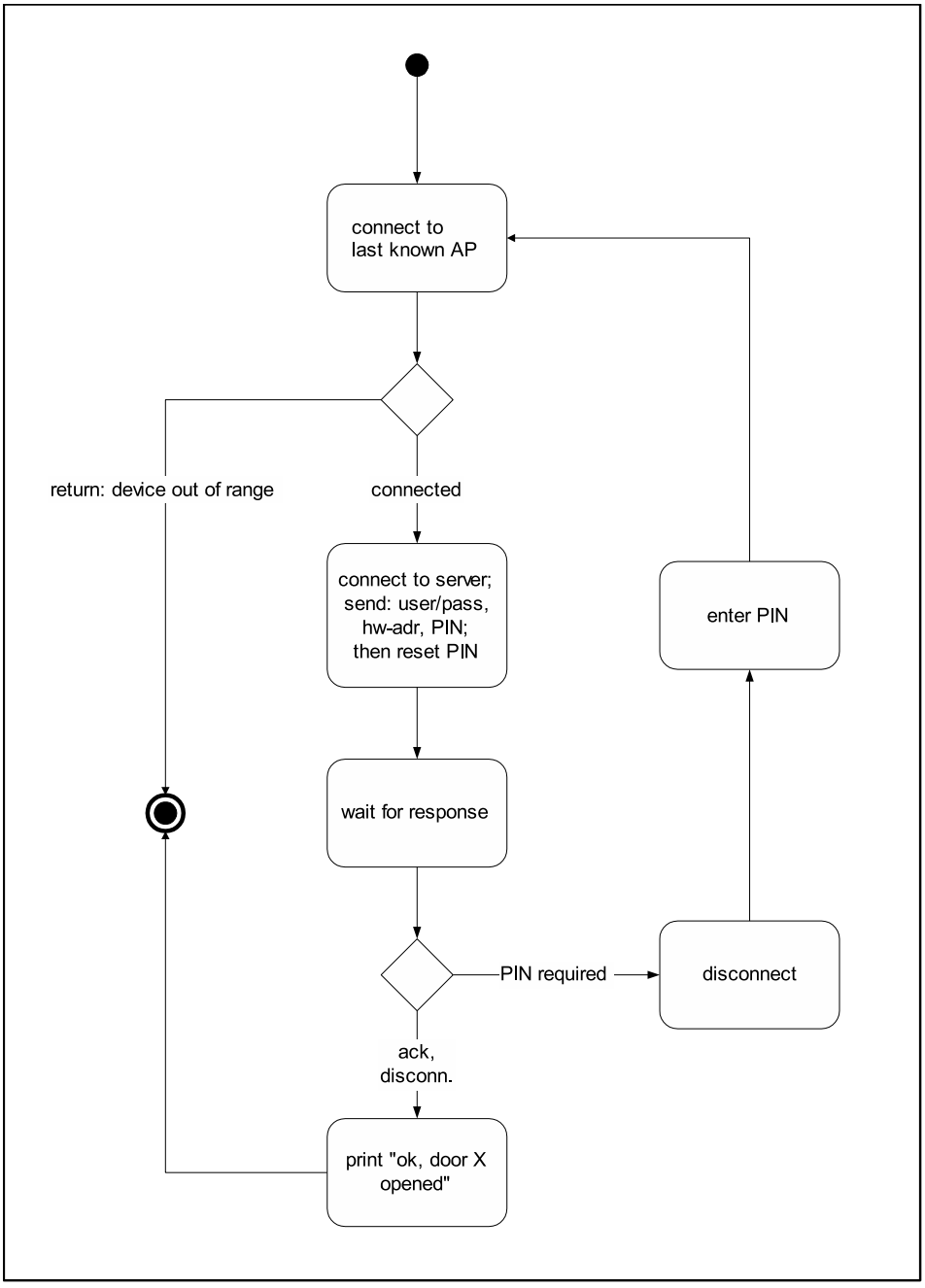


Abbildung 3.2: Palm Client: Flussdiagramm des Vorgangs zum Öffnen der Türen nach dem Aufwecken des Palms

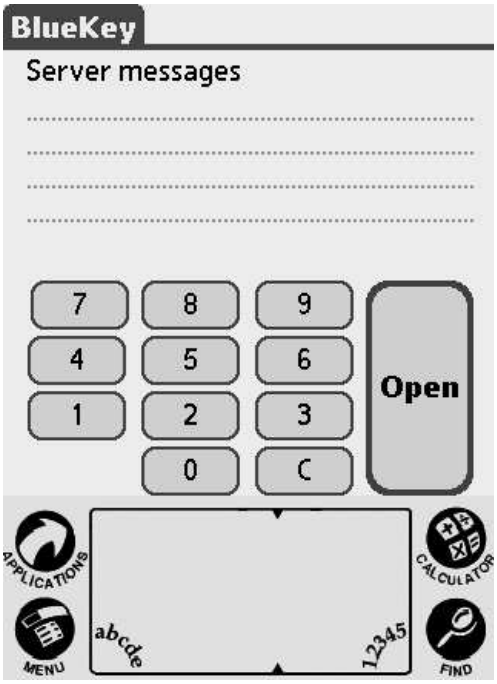


Abbildung 3.3: Palm Client: User Interface

3.3 Server

Der Server wartet auf Verbindungen der Clients, macht die erforderlichen Abfragen in der LOMOT-Datenbank und veranlasst das Öffnen der Türen. Zusätzlich werden diese Ereignisse in der Datenbank protokolliert.
Es sollen mehrere Clients gleichzeitig mit dem Server Kontakt aufnehmen können, daher fiel die Designentscheidung, den Server multi-threaded aufzubauen, wobei jeder Thread eine Verbindung verwalten soll. Darüberhinaus soll der Server nur jene Daten von den Clients anfordern, die tatsächlich benötigt werden, um Türen öffnen zu können.

Zungangskontrolle

Im ersten Schritt wurden die für die geforderte Funktionalität maßgeblichen Tabellen in der Datenbank ermittelt. Die Zugangsrechte werden in LOMOT üeber sogenannte „Services“ verwaltet, welche in der Tabelle *LOMOT.services* (Tabelle 3.1) eingetragen sind und jeweils einen bestimmten Zugang erlauben. Diese Services werden über *LOMOT.user_rights* (Tabelle 3.2) und *LOMOT.group_rights* (Tabelle 3.3) den einzelnen Benutzern zugeordnet und ggf. durch ein Passwort geschützt. Da die Benutzer datenbankintern nicht mit ihrem Namen, sondern über eine numerische ID verwaltet werden ist es auch noch nötig, diese in *LOMOT.users* (Tabelle 3.4) zu ermitteln.

Die Zuordnung der Access Points zu Türen findet sich in *LOMOT_LOCATION.gates_control* (Tabelle 3.5). Dort wird der Tür auch gleichzeitig ein (oder mehrere) Service zugewiesen. Ein

Field	*Type*	*Null*	*Default*	*Description*
id	int(11)	No		Index (automatisch inkrementiert)
servicename	text	No		Bezeichnung des Services
pass	tinyint(1)	No	0	Flag, ob Passwort erforderlich

Tabelle 3.1: Datenbanktabelle LOMOT.services

Field	*Type*	*Null*	*Default*	*Description*
userid	int(11)	No	0	Benutzerindex (Schlüssel in users)
serviceid	int(11)	No	0	zugeordnetes Service (Schlüssel in services)
pass	text	No		Passwort

Tabelle 3.2: Datenbanktabelle LOMOT.user_rights

Field	*Type*	*Null*	*Default*	*Description*
groupid	int(11)	No	0	Index (automatisch inkrementiert)
serviceid	int(11)	No	0	Servicenummer (Schlüssel in services)
pass	text	No		Passwort

Tabelle 3.3: Datenbanktabelle LOMOT.group_rights

Field	*Type*	*Null*	*Default*	*Description*
id	int(11)	No		Index (automatisch inkrementiert)
username	varchar(20)	No		Benutzername (eindeutig)
firstname	text	No		Vorname
lastname	text	No		Nachname
street	text	No		Strasse
plz	tinyint(4)	No	0	Postleitzahl
city	text	No		Stadt
phone	text	No		Telefonnummer
groupid	int(11)	No	0	zugeordnete Gruppe (schlüssel in user_groups)

Tabelle 3.4: Datenbanktabelle LOMOT.users

Field	*Type*	*Null*	*Default*	*Description*
gateid	int(11)	No	0	Türkennung
apid	decimal(14,0)	No	0	AccessPoint - Kennung
aptypeid	smallint(6)	No	0	AccessPoint - verwendete Technik
serviceid	int(11)	No	0	benötigtes Recht (Schlüssel in services)

Tabelle 3.5: Datenbanktabelle LOMOT_LOCATION.gates_control

Field	*Type*	*Null*	*Default*	*Description*
id	int(11)	No		Index (automatisch inkrementiert)
room1	int(11)	No	0	zugeordneter Raum
room2	int(11)	No	0	zugeordneter Raum
pincode	int(10)	No	0	Pincode, falls Tür gesichert wird

Tabelle 3.6: Datenbanktabelle LOMOT_LOCATION.gates

Benutzer ist also genau dann zutrittsberechtigt, wenn er und die Tür gleichzeitig über das selbe Service verfügen.
Weiters kann einer Tür noch ein spezieller Pincode zugewiesen sein, welcher in *LOMOT_LOCATION.gates* (Tabelle 3.6) eingetragen ist. Bevor die Tür geöffnet wird, muss also noch überprüft werden, ob dieser Eintrag leer ist oder mit dem vom Benutzer eingegebenen Code übereinstimmt.

Bereits in der Designphase stellte sich heraus, dass die Rechteverwaltung in dieser Form zwar sehr flexibel ist, jedoch den Anforderungen von *BlueKey* nicht entgegenkommt. Für eine Zugangskontrolle auf Benutzerebene wäre eine direkte Zuordnung eines Passwortes zu einem Benutzer sinnvoller. Dieses Verhalten lässt sich in LOMOT nur dadurch erreichen, dass für jede Tür, die ein Benutzer öffnen können soll, ein eigenes Service mit dem (immer gleichen) Passwort des Benutzers angelegt wird. Vorteilhaft ist hier aber sicher die hohe Flexibilität der Rechtevergabe im System.

Protokollieren

Die Tür-Ereignisse sollen in *LOMOT_TRACKING.log_gates* (Tabelle 3.7) protokolliert werden. Um die Anzahl der Datenbankeinträge nicht unnötig wachsen zu lassen, wurde jedoch beschlossen, nur erfolgreiche Zutrittsversuche zu loggen. Andernfalls hätte jeder vorbeigetragene Client einen (oder, wenn einem AP mehrere Türen zugeordnet sind, sogar mehrere) Einträge erzeugt, selbst wenn der Benutzer die Tür gar nicht benutzen wollte.

Field	*Type*	*Null*	*Default*	*Description*
id	bigint(20)	No		Index (automatisch inkrementiert)
deviceid	decimal(14,0)	No	0	Geräteadresse
devicetypeid	smallint(6)	No	0	Technik des Devices
gateid	int(11)	No	0	GateID (Schlüssel in gates)
apid	decimal(14,0)	No	0	AccessPoint - Adresse
aptypeid	smallint(6)	No	0	AccessPoint - Technik
userid	int(11)	No	0	Benutzer-ID(Schlüssel in users)
gateaction	text	No		Aktionsresultate
gatedate	date	No	0000-00-00	Datum des Ereignisses
gatetime	time	No	00:00:00	Zeitpunkt des Ereignisses

Tabelle 3.7: Datenbanktabelle LOMOT_TRACKING.log_gates

3.4 Übertragungsprotokoll

Anforderungen

Beim Entwurf des Übertragungsprotokolls zwischen Client und Server sollte auf die Eigenheiten der beteiligten Geräte sowie der teils drahtlosen, teils kabelgestützten Kommunikation Rücksicht genommen werden. Weiters soll das Protokoll von Anfang an eine leichte Erweiterbarkeit vorsehen, die es für eine Vielzahl an unterschiedlichen Client-Architekturen (PDAs, BT-Tags, RF-Tags/Reader, ...) verwendbar macht. Es ergeben sich daraus ganz spezielle Anforderungen, die ein geeignetes Übertragungsprotokoll erfüllen muss.

Energieaufwand

Drahtlose Datenübertragung ist insbesondere für batteriebetriebene Geräte eine verhältnismäßig energieaufwändige Kommunikationsform. Daher sollte nach Möglichkeit darauf geachtet werden, den Kommunikationsaufwand und im Speziellen die Dauer der Verbindung zu minimieren.
Wie bereits beim Design der Software beschrieben soll die Kommunikation vom Client ausgehen, da bei Palm OS das Offenhalten eines Sockets zwangsläufig auch eine ständig aktive PPP-Verbindung voraussetzt. Daher ist es sowohl einfacher als auch energetisch sinnvoller, den Client sowohl die PPP-Verbindung zum AP als auch die TCP/IP-Verbindung zum *BlueKey*-Server selbst aufbauen zu lassen.

Um die weitere Kommunikation so kurz als möglich zu halten soll einerseits dem Server gleich beim Verbindungsaufbau mitgeteilt werden, über welche Daten der Client aktuell verfügt und andererseits die Auswahl der tatsächlich zu übertragenden Daten dem Server überlassen werden. Dieser kann nämlich z.B. je nach aktueller Position des Clients leicht entscheiden, ob eine Übertragung sämtlicher Daten überhaupt erforderlich ist (z.B. ist die Übertragung eines Pincode sinnlos, wenn sich beim betreffenden AP gar keine Pincode-geschützten Türen befinden). Auch andere Anwendungen (z.B. personenbezogenes Messaging, für dass die Position des Client-PDA irrelevant ist) lassen sich so optional recht einfach und energieeffizient implementieren.
Erreicht wurde dies durch die Definition verschiedener Fähigkeiten („Capability“) (Tabelle 3.8), über die der Client verfügen kann. Beim Verbindungsaufbau sendet die Client-Software eine Li-

Code	*Beschreibung*
POSN	Gerät kann die Position (Adresse des AP) erkennen
DISP	Gerät hat ein Display und kann Meldungen anzeigen
USER	Es wurde ein Username eingegeben
PASS	Es wurde ein Passwort eingegeben
PINC	Es wurde ein Pincode eingegeben
CRYP_xxx	Das Gerät beherrscht die Verschlüsselung nach Algorithmus xxx
CRYP_PLAIN	Unverschlüsselte Daten (default)

Tabelle 3.8: Client-Capabilities

ste der vorhandenen Capabilities und der Server fordert daraufhin gezielt die Übertragung der benötigten Daten an.

Netzwerkauslastung

Ein weiteres Problem stellt die Tatsache dar, dass ein Bluetooth-Piconet auf maximal 8 gleichzeitig kommunizierende Geräte (d.h. 1 AP und 7 PDAs) beschränkt ist. Da in einer Umgebung, in der LOMOT eingesetzt wird, damit zu rechnen ist, dass es eine große Anzahl an BT-Geräten gibt, ist diese Beschränkung ein ernstzunehmender Faktor. Auch aus diesem Grund ist ein Protokoll, dass die Übertragungsdauer möglichst kurz hält in diesem Anwendungsfall von Vorteil. So kann der Übertragungskanal schnell aufeinander folgend von verschiedenen Clients genutzt werden.

Erweiterbarkeit und Portierbarkeit

Um sowohl die Implementierung auf unterschiedlichen Plattformen als auch die spätere Erweiterung zu erleichtern sollte von Anfang an ein Plain-Text-Datenformat (ASCII) verwendet werden. So ist sichergestellt, dass die ausgetauschten Nachrichten auf möglichst vielen Architekturen verarbeitet werden können, was bei Binärdaten nicht unbedingt der Fall ist (man denke z.B. an unterschiedliche Byte-orders auf verschiedenen Prozessoren).

Sicherheit

Für den drahtlosen Teil der Übertragung (vom PDA zum AP) bietet Bluetooth durch Frequency Hopping und Verschlüsselung prinzipiell bereits ein hohes Maß an Sicherheit, sodass die Kommunikation zwischen Client und AP nicht einfach abzuhören ist. Dieser Mechanismus baut jedoch auf einem Pincode auf, den alle Benutzer kennen müssen („shared secret“) und ist daher in Umgebungen mit vielen Benutzern weniger geeignet [HJP03], weswegen im vorliegenden Projekt darauf verzichtet wird. Weiters läuft die Verbindung vom AP zum Server über ein LAN, das u.U. auch noch für andere Dienste genutzt wird. Daher sollte das Protokoll zusätzlich auf jeden Fall über die Möglichkeit einer weiteren Verschlüsselung der Datenfelder verfügen.
Hier sollte der Einsatz verschiedener Algorithmen möglich sein, um je nach Exposition des LAN und Art der eingesetzten Clients einen geeigneten Kompromiss zwischen der Stärke der Verschlüsselung und der begrenzten Rechenleistung der PDAs wählen zu können.
Die Art der Verschlüsselung wird ebenfalls vom Server – je nach Fähigkeiten des Clients – festgelegt. Außerdem obliegt dem Server die Generierung eines ev. erforderlichen Schlüssels für

die Verbindung.

Für die vorliegende Arbeit wurden der Einfachheit halber nur einfache Verschlüsselungsstrategien implementiert, die nicht robust gegenüber Attacken sind, bei denen sich ein „feindliches“ BT-Gerät als AP ausgibt. Dieses könnte den Client aufwecken, seinen Kommunikationsversuch mit dem Server abfangen und manipulieren und so sämtliche Daten als Plaintext anfordern.
Eine derartige Attacke könnte durch den Einsatz von Public Key Verfahren unterbunden werden. Der Server müsste dem Client dann seinen Public Key senden worauf der Client die Daten mit einer Kombination von Public Key und z.B. dem eingegebenen Passwort oder einem zusätzlichen „Private Key“ verschlüsseln könnte. Da der Server das Passwort/den Private Key aus der Datenbank lesen kann, kann er die Daten auch wieder entschlüsseln. Ein etwaiger Angreifer hätte jedoch keinen Zugriff auf den Private Key, da dieser in keinem Fall übertragen wird. Ein ähnliches Verfahren wird in [HJP03] vorgestellt.
Für die vorliegende Arbeit wurde jedoch auf die Implementierung dieser starken Verschlüsselung verzichtet. Das Protkoll bietet aber genügend Erweiterungsmöglichkeiten, um sie in weiterführender Arbeit hinzuzufügen.

Protokolldefinition

Insgesamt ist das Protokoll sehr einfach gehalten. Die Befehle beginnen immer mit der Zeichenkette `BLUEKEY_`, gefolgt vom Befehlsbezeichner und optionalen Argumenten. Da die Kommunikation mit Ausnahme des Verbindungsaufbaus immer vom Server angestoßen wird, verfügen Client und Server über einen sehr verschiedenen Befehlssatz. Die Server-Kommandos (Tabelle 3.9) dienen hauptsächlich zum Anfordern von bestimmten Daten, während die Client-Nachrichten (Tabelle 3.10) großteils die Antworten darauf darstellen. Weiters verfügen sowohl Client als auch Server über eine allgemeine Bestätigung (`BLUEKEY_ACCP`), eine allgemeine Fehlermeldung (`BLUEKEY_FAIL`) sowie ein Kommando zum Abbrechen der Verbindung (`BLUEKEY_QUIT`). Das Kommando zum Verbindungsaufbau (`BLUEKEY_HELO`) liefert neben der BT-Adresse des PDA auch sofort eine Liste der am Client verfügbaren Capabilities. Eine derartige Liste wird generell Comma-separated übertragen. Der Server kann dann mit `BLUEKEY_CRYP` einen Verschlüsselungsalgorithmus auswählen und mit `BLUEKEY_SEND` die Datenübertragung anstoßen. Jede Nachricht gilt implizit als `ACCP`. Beispiele für mögliche Kommunikationsabläufe sind im Anhang (A) angeführt.

Message	*Beschreibung*
`BLUEKEY_ACCP`	Aktion erfolgreich; optional kann noch eine Liste von zusätzlich sinnvollen Capabilities angegeben werden
`BLUEKEY_FAIL [command/capability]`	Fehler bei der Interpretation der Nachricht oder bzw. der Client verfügt nicht über eine geforderte Capability
`BLUEKEY_CRYP alg key [key2 ...]`	Festlegen des Verschlüsselungsalgorithmus der auf alle folgenden Datenfelder anzuwenden ist
`BLUEKEY_SEND capability[,capability2 ...]`	Aufforderung zur Übertragung der aufgelisteten Capabilities. Der Server wartet, bis alle geforderten Meldungen eingelangt sind. Kann der Client einen geforderten Datensatz nicht liefern muss er dies mit `BLUEKEY_FAIL` melden
`BLUEKEY_DISP message`	Das Gerät soll die Textmeldung auf seinem Display ausgeben
`BLUEKEY_QUIT`	Verbindung beenden

Tabelle 3.9: Server-Kommandos

Message	*Beschreibung*
`BLUEKEY_HELO version id capability1,capability2,...`	Verbindungsaufbau mit Bekanntgabe der höchsten unterstützten Protokollversion (derzeit 1.0), der eigenen ID (BT-Hardwareadresse) sowie der vorhandenen Capabilities
`BLUEKEY_ACCP`	Aktion erfolgreich
`BLUEKEY_FAIL [command/capability]`	Fehler; der Client kennt das Kommando nicht oder verfügt nicht über eine geforderte Capability
`BLUEKEY_POSN id`	Positionskennung (BT-Hardwareadresse des AP)
`BLUEKEY_USER username`	Username
`BLUEKEY_PASS password`	Passwort
`BLUEKEY_PINC pin`	Pincode
`BLUEKEY_QUIT`	Verbindung beenden

Tabelle 3.10: Client-Kommandos

Kapitel 4

Implementierung

4.1 Client, Palm OS

Entwicklungsumgebung

Als Entwicklungsumgebung wurde „Metrowerks CodeWarrior 9.1“ [Met03] verwendet, da diese für Palm OS Programmierung am häufigsten eingesetzt wird. Für die Bluetooth-Unterstützung musste außerdem noch das „Palm OS 5 SDK, R2“ [Pal03] nachinstalliert werden, das auch ausführliche Palm OS Dokumentationen enthält.
Wertvoll für die Programmierung waren daraus folgende Dokumente: „Palm OS Companion.pdf“ (UI und Event Handling), „Palm OS Companion2.pdf“ (Network, Bluetooth, OBEX), sowie „Palm OS Reference.pdf“ (vollständige Befehlsreferenz).
Als gute Unterstützung in den Einstieg in die Palm OS Programmierung diente die „Palm OS 5 Programming Bible“ [Fos03], die auch viele nützliche Code-Beispiele enthält.
Als Vorlage für eine Bluetooth-fähige Palm OS Applikation wurde „Discovery.c“ [Pal03] verwendet, das u.a. ein UI mit Event Handling, sowie ein fertig implementiertes Bluetooth-Inquery enthält (das aber aufgrund des aktiven APs derzeit nicht benötigt wird).

Vorbereitungen

Die Entwicklung eines UI für Palm OS wird von der Entwicklungsumgebung integriert und soll hier nicht weiter erläutert werden. Etwas komplizierter ist das Palm OS Event Handling für das UI, also z.B. Abfrage der Buttons, Menübefehle oder Scrollbalken. Um solche Events muss sich der Programmierer selbst kümmern, die eingesetzte Entwicklungsumgebung stellt hier keine automatischen Hilfestellungen (z.B. leere Prototypen o.ä.) zur Verfügung. Man kann aber die vielen frei verfügbaren Palm OS Programmbeispiele als Vorlage nehmen und diese um eigene Funktionen erweitern. Da die oben erwähnten Dokumentationen die Entwicklung eines UI einer Palm OS Anwendung ausführlich beschreiben, wird nun nicht weiter darauf eingegangen.

Die folgenden Kapitel beschreiben daher ausschließlich die Implementierung der Kommunikations-Programmteile, die für den *BlueKey* Client benötigt werden. Allen voran soll hier die Erstellung von Netzwerk- und Verbindungseinstellungen beschrieben werden, da dieses Thema sehr dürftig dokumentiert ist und keine Beispiele verfügbar waren. Als Nachschlagewerk konnte hierfür nur „Palm OS Reference.pdf“ [Pal03] als reine Befehlsreferenz herangezogen werden.

4.1.1 Initialisierungen

Folgende Initialisierungen sind beim Start der Applikation notwendig:

- OBEX Registrierung, damit der Client vom AP aktiviert werden kann
- Verbindungeintrag (Connection-Setting) für Bluetooth AP erstellen
- Netzwerkeinstellungen (Network-Settings) erstellen, für die Netzwerkverbindung ins LAN über den vorher definierten Verbindungeintrag

OBEX Registrierung

OBEX ist ein plattformübergreifender Standard für den Austausch von Dateien (Objekten) zwischen verschiedenen Endgeräten über IR oder Bluetooth. Eine weit verbreitete Anwendung ist z.B. der Austausch von Adressdaten zwischen Mobiltelefonen in Form von vcf-Dateien (business card format). OBEX wird aber auch häufig zum einfachen Datenaustausch zwischen PDA und PC verwendet. Palm OS verwendet für OBEX Senden und Empfangen den „Exchange Manager“, der seit Palm OS 4.0 auch Bluetooth unterstützt. Zur Programmierung stehen eigene Libraries zur Verfügung, für BT wird "BtExgLib.h" benötigt.
Der *BlueKey* Client benötigt OBEX für zwei Aufgaben: Die Anwendung muss auf eine vom Access Point ankommende Datei reagieren (vom Palm OS gestartet werden) und danach die BT Adresse des Senders ermitteln.

Damit Palm OS die Anwendung bei Erhalt eine bestimmten Datei benachrichtigen kann, muss sich diese zuerst nach dem Installieren beim OS registrieren. Zur Unterscheidung, um welchen Dateityp es sich handelt bzw. welche Applikation benachrichtigt werden muss, wird die Dateinamenserweiterung herangezogen. Standardmäßig wird z.B. bei Erhalt einer "dateiname.vcf" Datei die Adressen-Applikation gestartet, während bei "dateiname.txt" die Memo-Anwendung gestartet wird.

Diese Registrierung wird in der Funktion *PilotMain* vorgenommen, die vom OS immer nach System-Events aufgerufen wird. Die OBEX Registrierung erfolgt gleich nachdem die Palm OS Anwendung auf dem Palm Device installiert wird:

```
Funktion: UInt32 PilotMain( UInt16 cmd, MemPtr cmdPBP, UInt16 launchFlags)
[..]
case sysAppLaunchCmdSyncNotify:
 // Nach Installieren der Anwendung: OBEX-Datei-Typ registrieren um
 // vom OS bei Empfang dieses Dateityps notified zu werden
 error=ExgRegisterDatatype(appFileCreator, exgRegExtensionID,
               BT_OBEX_WAKEUP_FILETYPE, BT_OBEX_WAKEUP_FILETYPE_DESCR, 0);
 error=ExgSetDefaultApplication (appFileCreator, exgRegExtensionID,
               BT_OBEX_WAKEUP_FILETYPE);
 if (error) ErrDisplay("Registering DataType failed");
 break;
[..]
```

Erklärungen: Das Event *sysAppLaunchCmdSyncNotify* tritt bei der Installation der Anwendung auf, der registrierte Dateityp "blk" ist in *BT_OBEX_WAKEUP_FILETYPE* definiert.

Verbindungeintrag

Damit sich der Palm OS Client über einen Bluetooth AP in ein LAN Netzwerk verbinden kann, muss zuerst eine Verbindung eingerichtet werden. Palm OS stellt hierfür sogenannte Connection Settings (Verbindungs-Profile) zur Verfügung, die man u.a. in den Palm OS Einstellungen editieren kann. Dort werden beispielsweise auch Modem-Verbindungen über Mobiltelefone definiert um einen Palm über dieses ins Internet zu verbinden. Ein solcher Verbindungseintrag kann dann in den Netzwerkeinstellungen ausgeählt werden (siehe Absatz „Netzwerkeinstellungen“)

Dieses Connection Setting wird mit der Funktion *CreateBTConnectionEntry* erstellt und unter dem Profilnamen *„BlueKey_Connection“* abgespeichert. Unter diesem Namen ist die Verbindung auch in den Palm OS Verbindungs-Settings sichtbar und editierbar. Im weiteren Betrieb des *BlueKey* Clients wird dann in diesem Profil der jeweils aktuelle Bluetooth AP eingetragen.
Bekannte Probleme: Leider funktioniert das neu erstellte Profil erst, wenn man nach dem ersten Programmaufruf in den Palm OS Verbindungs-Einstellungen für diese Connection einmalig ein beliebiges Bluetooth-Device sucht und dieses zuweist. Erst danach kann beim nächsten Aufruf von *BlueKey* die Verbindung problemlos verwendet werden. Ansonsten bricht jeder Verbindungsaufbau mit der Meldung "Fehler: (0x1F08)" ab.
Die wichtigsten Ausschnitte dieser Funktion, die zu Programmstart aufgerufen wird, sollen hier gezeigt werden (gekürzt):

```
Funktion: CncProfileID CreateBTConnectionEntry()

// Definitionen
  UInt16 deviceKind = kCncDeviceKindSerial;  // Smart Access Point/PC
  Char myConnectionName[32] = "BlueKey_Connection\0";
  UInt32 port = sysFileCVirtRfComm;  // Verbindung ueber Bluetooth
  UInt32 baud = 115200;
[..]
// Oeffnen der Connection Manager profile database
  err = CncProfileOpenDB();
[..]
// Profilname erstellen
  err = CncProfileSettingSet(profileId, kCncParamName,
              &myConnectionName, StrLen(myConnectionName)+1);
// Weitere benoetigte Parameter setzen
  err = CncProfileSettingSet(profileId, kCncParamPort, &port, kCncParamPortSize);
  err = CncProfileSettingSet(profileId, kCncParamBaud, &baud, kCncParamBaudSize);
  err = CncProfileSettingSet(profileId, kCncParamDeviceKind,
&deviceKind,kCncParamDeviceKindSize);
[..]
```

Anmerkungen: Nach demselben Prinzip funktioniert auch die Funktion *UpdateBtConnectionEntry*, die während des Programmablaufs die Adresse des aktuellen AP in diesem Verbindungeintrag aktualisiert.

Netzwerkeinstellungen

In den Palm OS Netzwerkeinstellungen (Network Settings) können verschiedene Profile für den Zugriff in verschiedene Netzwerke definiert werden. In diesen Profilen werden Parameter wie Verbindungstyp (PPP, SLIP), Benutzername und Kennwort gespeichert. Aber auch die zu verwendende Verbindung (Connection Setting) wird dort festgelegt.
Für den Palm Client ist es nun erforderlich, auch eine Netzwerkeinstellung während des Programmstarts einzurichten. Diese muss alle Parameter für die LAN-Verbindung enthalten und weiters als Verbindung die vorher erstellte *„BlueKey_Connection“* eingetragen haben.
Bekannte Probleme: Das erstellte Network Setting ist im Gegensatz zu den vorher beschriebenen Connection Setting in den Palm OS Netzwerk-Einstellungen nicht sichtbar. Es kann daher nicht manuell bearbeitet werden und wird außerdem überschrieben, sobald man in den Palm-Einstellungen eine andere Netzwerkverbindung auswählt. Daher muss die Netzwerkeinstellung bei jedem Programmstart von *BlueKey* erneut erstellt werden.
Die wesentlichen Ausschnitte der Funktion *„CreateNetworkSettings“*, sehen folgendermaßen aus (gekürzt):

```
Funktion: Err CreateNetworkSettings()

// Definitionen
  UInt32 ifCreator = netIFCreatorPPP;  // Definiert eine PPP Verbindung
  UInt16 ifInstance = 0;               // Interface 0
  UInt16 IntTimeOut = NETWORK_ESTABLISH_TIMEOUT;   // Timeout
[..]
// Network-Setting "BlueKey" erstellen (netIFSettingServiceName)
// (Achtung: dieses ist in den Palm OS-Settings *nicht* sichtbar)
  StrCopy(pValue, "BlueKey");  valueLen = StrLen(pValue) + 1;
  err = NetLibIFSettingSet(AppNetRefnum, ifCreator, ifInstance,
                           netIFSettingServiceName, pValue, valueLen);

// Zu verwendendes Connection-Setting eintragen: "BlueKey_Connection"
  StrCopy(pValue,"BlueKey_Connection");  valueLen = StrLen(pValue) + 1;
  err = NetLibIFSettingSet(AppNetRefnum, ifCreator, ifInstance,
                         netIFSettingConnectionName, pValue, valueLen);
[..]
// Network-Setting "BlueKey" speichern
  err = NetLibConfigSaveAs(AppNetRefnum, pConfigName);
[..]
```

Erklärungen: Die Abkürzung if* (z.B. ifCreator) steht für (Network) Interface

4.1.2 Aktivierung des Clients

Aktivieren des Clients (OBEX Empfang) und ermitteln des aktuellen APs

Wird der Client vom AP durch die ankommende OBEX Datei aktiviert („aufgeweckt“), wird vom Palm OS in der Funktion *PilotMain* zuerst das Event *sysAppLaunchCmdExgAskUser* erzeugt. Im Normalfall würde dem Benutzer eine Dialogbox angezeigt werden. Diese muss aber automatisch ausgeblendet und das Empfangen der Datei eingeleitet werden.

Wird das Empfangen der Datei eingeleitet, dann wird vom OS das Event *sysAppLaunchCmdExg-ReceiveData* erzeugt. Erst in diesem Moment kann die Bluetooth-Adresse des Sender abgefragt werden. Nach dieser Abfrage muss die *BlueKey* Applikation gestartet werden (da auf dem Palm Device vor dem OBEX Event eine beliebige andere Anwendung laufen könnte), und die eben erhaltene Adresse wird dem Programm als Parameter übergeben.
Das OBEX Event Handling Funktion wird im folgenden Beispiel gezeigt (gekürzt):

```
Funktion: UInt32 PilotMain( UInt16 cmd, MemPtr cmdPBP, UInt16 launchFlags)
[..]
case sysAppLaunchCmdExgAskUser:
  // "Aufwecken" des Clients durch Bluetooth-OBEX Empfang
  askInfo=(ExgAskParamType *) cmdPBP;   // Pointer auf Parameterblock
  askInfo->result=exgAskOk;             // Accept (OK) der Ask-Box vorgeben
  break;

case sysAppLaunchCmdExgReceiveData:
  // Nach autom. accept (sysAppLaunchCmdExgAskUser) kommt dieses Event
  // Verbinndung accepten, danach OBEX-Sender-Adr ausfindig machen

  // Als erstes: Verbindung (OBEX-Empfang) akzeptieren
  exgSocketP = (ExgSocketType *) cmdPBP;
  error = ExgAccept(exgSocketP); if (error) ErrAlert(error);
[..]
  // URL (OBEX-Sender-Adr) abfragen, als Parameter die Groesse ubergeben
  getUrl.URLP = MemPtrNew(getUrl.URLSize);
  ExgControl(exgSocketP, exgLibCtlGetURL, &getUrl, &getUrlLen);

  // Nach Abfrage der URL Verbindung (Empfang) canceln
  error = ExgDisconnect(exgSocketP, errNone); if (error) ErrAlert(error);

  // Anwendung neu starten mit Obex-Sender als Parameter (getUrl.URLP)
  MemPtrSetOwner(getUrl.URLP, 0);  // Parameterblock MemPointer freigeben
                                   // und OS-System als Owner eintragen
  AppLaunchWithCommand(appFileCreator, sysAppLaunchCmdNormalLaunch,
                                                 (void *)getUrl.URLP);
```

4.1.3 Kommunikationsablauf

Nach dem Aktivieren des Clients durch den AP wird dieser für die nächste Verbindung eingetragen und die eigentliche Kommunikation mit dem Server initiiert. Der Client schickt zuerst sämtliche Authentifizierungsdaten zum Server und stellt anschließend die Antworten (z.B. „Tür geöffnet“) am Display dar.

Die Hauptfunktion, in der die Kommunikation stattfindet ist, ist *BlueKeyOpen*. Diese besteht aus drei Teilen:

- Initialisierung und Verbindungsaufbau

- Kommunikations - Haupt-Schleife
- Verbindungsabbau

Initialisierung und Verbindungsaufbau

Bevor die Netzwerkverbindung aufgebaut werden kann, muss die Bluetooth-Adresse des aktuellen AP im Verbindungseintrag mittels *UpdateBtConnectionEntry* aktualisiert werden (siehe Kapitel 4.1.1). Erst dann kann versucht werden, über diesen eine Verbindung aufzubauen. Fall dies auf Anhieb nicht gelingt (z.B. aufgrund einer schlechten Verbindungsqualität), wird dies automatisch wiederholt (*MAX_CONNECTION_RETRIES*). Der Ablauf der Initialisierung sieht also so aus:

```
Funktion: Err BlueKeyOpen(), Initialisierungsteil

// Akt.BT-Device eintragen (ascii aus prefs lesen und als BT-Adr übergeben)
  BtLibAddrAToBtd(gBtLibRefNum, prefs.LastBtAddr, &lastBtAddr);
  UpdateBtConnectionEntry(lastBtAddr,theCncProfileId);

// Verbindung zu Server aufbauen, mehrere Versuche in for-schleife
  for(connCount = 0; connCount < MAX_CONNECTION_RETRIES; connCount++) {
      err = NetLibOpen(AppNetRefnum, &ifErrs); if (err) ErrAlert(err);
      err = ConnectSocket(&socket);
            if(!err) break;
      NetLibClose(AppNetRefnum, true);
  }
```

Die Verbindung zum Server stellt oben verwendete Funktion *ConnectSocket* her und liefert einen *Socket* bzw. eventuelle Fehlercodes zurück. Die Funktion sieht wie folgt aus (gekürzt):

```
Funktion: Err ConnectSocket(NetSocketRef *socket)
[..]
// IP Adresse und  Port (aus den Preferences prefs.) eintragen
  inetAddrP = (NetSocketAddrINType*)&addr;
  inetAddrP->family = netSocketAddrINET;    // socket type
  inetAddrP->port = NetHToNS( prefs.ServerPort ); // Port
  inetAddrP->addr = NetHToNL( prefs.ServerIP );   // IP

// Verbindung zu socket aufzubauen oder ggf. err zurueckliefern
  if(NetLibSocketConnect(AppNetRefnum, *socket, &addr, sizeof(addr),
                         5*SysTicksPerSecond(), &err) == -1)
                return err;
[..]
```

Anmerkungen: Für den Aufbau der Netzwerkverbindung (TCP/IP Socket) werden vom Palm OS automatisch die vorher definierten Netzwerk- und Verbindungseinstellungen verwendet. Ob die Verbindung über Bluetooth, Infrarot, WLAN o.ä. aufgebaut wird, kann an dieser Stelle von der Applikation nicht ausgewählt werden und ist für die Applikation in dieser Abstraktionsebene transparent. Es ist wird daher zur eigentlichen Kommunikation nur die

Standard-Netzwerkbibliothek (NetLib) benötigt, nicht die Bluetooth-Programmbibliothek.

Kommunikations - Haupt-Schleife

Nach erfolgreichem Verbindungsaufbau beginnt die Kommunikation mit dem Server gemäß dem *BlueKey* Übertragungsprotokoll (siehe Kapitel 3.4). Die Implementierung dieses Teils wird hier kurz beschrieben, ohne auf den Sourcode im Detail einzugehen:

Nach Verbindungsaufbau wird das Kommando *HELO* mit sämtlichen Fähigkeiten des Clients an den Server geschickt.
Danach beginnt die Haupt-Schleife, die sich im folgenden Ablauf wiederholt:

- Warten auf Antwort vom Server (*receiveBuffer*) bzw. auf ein TimeOut, sowie Zerlegen eventueller mehrzeiliger Antworten (*lineBuffer*)
- Zerlegen der Zeile in Kommando (*cmd*) und restliche Parameter (*paramsP*)
- Vergleichen des Kommandos mit den vordefinierten *BlueKey* Kommandos (Liste *blk_commands*)
- Ausführen der in der Liste *blk_commands* zugeordneten Antwort-Funktion (*blk_cmd_***)
- Senden der Antwort

Je nach Kommando, das vom Server geschickt wird, wird eine nach diesem Kommando benannte Antwort-Funktion ausgeführt. Wenn der Server also z.B. *„BLUEKEY_DISP Tür 35 geöffnet.“* schickt, wird die Antwort-Funktion *blk_cmd_disp* mit dem Parameter *„Tür 35 geöffnet.“* aufgerufen.

Die für den *BlueKey* Client wichtigste Antwort-Funktion *blk_cmd_send* wird an dieser Stelle genauer vorgestellt:

```
Funktion: void blk_cmd_send(Char* params)
[..]
// Wird POSN verlangt, die BT-Adr des verbundenen AP schicken (prefs.LastBtAddr)
  if(StrStr(params, "POSN")) {
      StrCat(sendBuffer, CLIENT_CMD_POSN);
      StrCat(sendBuffer, CryptTxt(prefs.LastBtAddr, recentCryp, recentCrypParam));
      StrCat(sendBuffer, "\n");
  }

[..] // Senden von Username und Password gek\"urzt (analog wie POSN)

// Wird PINC verlangt, entweder Pin schicken, wenn dieser 0 ist, "Fail" schicken
  if(StrStr(params, "PINC")) {
      if(StrLen(Pin)>0) {
          StrCat(sendBuffer, CLIENT_CMD_PINC);
          StrCat(sendBuffer, CryptTxt(Pin, recentCryp, recentCrypParam));
```

```
            StrCat(sendBuffer, "\n");
        } else {
            StrCat(sendBuffer, CLIENT_CMD_FAILPARAM);
            StrCat(sendBuffer, "\n");
        }
    }
```

Anmerkungen: Jedes Datenfeld (Username, PIN, etc) wird vor dem Verschicken mittels *CryptTxt* verschlüsselt. Die aktuelle Verschlüsselungsmethode (*recentCryp*) kann vorher vom Server mittels Kommando *„CRYP..“* bestimmt werden und wird in der dazugehörigen Antwort-Funktion *blk_cmd_cryp* gesetzt.
In den Verschlüsselungsfunktionen *CryptTxt* bzw. *DeCryptTxt* sind derzeit die Methoden *CRYP_PLAIN* (plain text) und *CRYP_ROT* (rotate x) als Demonstrationsbeispiel implementiert.

4.2 Server

Java

Die Wahl für die Programmiersprache, in der der Server implementiert werden sollte, fiel auf Java. Dies soll vor allem das Ziel haben, den Server auf unterschiedlichen Plattformen einsetzen zu können. Neben den im Serverbereich üblichen Betriebssystemen ist es so auch möglich, den *BlueKey*-Server dezentral auf einem Java-fähigen AP zu betreiben. Ein solcher wird im Zuge eines anderen Projektes am ITI entwickelt [SK04]. Ausserdem ist die Implementierung einer multithreaded Applikation in Java äußerst einfach.

Klassen

Für die unterschiedlichen Bearbeitungsschritte der *BlueKey*-Kommunikation wurde der Server in drei Klassen aufgeteilt:

BlueKeyServer.java
Diese Klasse stellt die eigentliche Serverapplikation dar. Sie wartet auf TCP/IP-Connects (Klasse *ServerSocket*) und startet für jede eingehende Verbindung einen eigenen Thread.

BlueKeyConnection.java
Diese von *Thread* abgeleitete Klasse dient zur Bearbeitung einer Client-Server-Verbindung. Hier wird die eigentliche Kommunikation gemäß dem Übertragunsprotokoll durchgeführt. Die Klasse fragt die benötigten Daten von Client und Datenbank ab und entscheidet dann, welche Tür(en) geöffnet werden soll(en).
Es wurde der Einfachheit halber darauf verzichtet, nur jene Daten anzufordern, die tatsächlich benötigt werden (siehe Kapitel 3.3). Dies hätte die Implementierung deutlich komplizierter gemacht, verringert jedoch den Kommunikationsaufwand beim derzeitigen Stand der Arbeit nur unwesentlich. Daher werden derzeit einfach alle vom Client angebotenen Capabilities abgefragt. Lediglich beim Fehlen der Capability `POSN` (Tabelle 3.8) wird die Verbindung sofort abgebrochen, da eine Positionsbestimmung und damit das Ermitteln der zu öffnenden Türen in diesem Fall nicht möglich wäre.

Das Ereignis der Türöffnung wird abschließend in die LOMOT-Datenbank eingetragen (Abbildung 4.1).

BlueKeySQL.java
Diese Klasse dient zur Kommunikation mit der LOMOT-Datenbank. In ihr sind der Verbindungsaufbau und die nötigen SQL-Querys gekapselt. Der Verbindungsaufbau zur Datenbank erfolgt über die Java-Bibliothek „mysql-connector-java“ [MyS03].

Ermitteln der zu öffnenden Türen: Die Hauptfunktionalität ist in der Methode `GetDoors(user, pass, accesspoint)` implementiert. Dort werden zunächst die zum gegebenen Username/Passwort-Paar passenden Services[1] ermittelt. Dann werden die zu APs und Services passenden Türen gesucht und schließlich deren IDs sowie die zugeordneten Pincodes zurückgeliefert. All dies wurde in einem einzigen SQL-Query realisiert:

```
SELECT LOMOT.user_rights.serviceid, LOMOT_LOCATION.gates_control.gateid,
       LOMOT_LOCATION.gates.pincode
FROM LOMOT.user_rights
INNER JOIN LOMOT.users ON
      LOMOT.users.id = LOMOT.user_rights.userid
INNER JOIN LOMOT_LOCATION.gates_control ON
      LOMOT_LOCATION.gates_control.serviceid = LOMOT.user_rights.serviceid
INNER JOIN LOMOT_LOCATION.gates ON
      LOMOT_LOCATION.gates.id = LOMOT_LOCATION.gates_control.gateid
WHERE ( LOMOT.users.username=<user>
        AND LOMOT.user_rights.pass=PASSWORD(<password>)
        AND LOMOT_LOCATION.gates_control.apid=<accesspoint> )
LIMIT 0,100;
```

Die Datenbank enthält auch Gruppen-Rechte, deren Überprüfung jedoch noch nicht implementiert wurde.

Loggen der Ereignisse: Die Methode `DoTracking(userid, accesspoint, clientaddr, gateid, message)` implementiert das Eintragen des Ereignisses in die Tabelle *LOMOT_TRACKING.log_gates* der Datenbank (Abbildung 4.1). Dies wird über folgendes SQL-Statement erreicht:

```
INSERT INTO LOMOT_TRACKING.log_gates
       (deviceid, devicetypeid, gateid, apid, aptypeid, userid,
        gateaction, gatedate, gatetime)
VALUES (<clientaddr>, 2, <gateid>, accesspoint, 2, <userid>,
        <message>, CURDATE(), CURTIME() );
```

[1] In der LOMOT-Datenbank werden Rechte über „Services“ verwaltet. Ein Benutzer ist genau dann zutrittsberechtigt, wenn ihm und der Tür dasselbe Service zugewiesen wurde sowie das Passwort und/oder der Pincode korrekt sind (siehe Kapitel 3.3).

Tracking | Gateaccesslist | Objectlist | New Object

Gate: Labor-Flur AP: -- Select AP -- Filter
Device: -- Select device -- User: -- Select user --

ID	Device	Gate	AccessPoint-Address	User	Event	Contacttime
11	7E007F278 (Bluetooth)	34	476E1BA1E (Bluetooth)	marsch	door opened	2004-01-13,17:15:07
12	7E007F278 (Bluetooth)	34	476E1BA1E (Bluetooth)	marsch	door opened	2004-01-13,17:17:03
13	7E007F278 (Bluetooth)	34	476E1BA1E (Bluetooth)	marsch	door opened	2004-01-13,17:17:49
15	7E007F278 (Bluetooth)	34	476E1BA1E (Bluetooth)	marsch	door opened	2004-01-13,17:18:08
16	7E007F278 (Bluetooth)	34	476E1BA1E (Bluetooth)	marsch	door opened	2004-01-13,17:20:17
17	7E007F278 (Bluetooth)	34	476E1BA1E (Bluetooth)	marsch	door opened	2004-01-13,17:24:10
18	7E007F278 (Bluetooth)	34	476E1BA1E (Bluetooth)	marsch	door opened	2004-01-13,17:28:43
19	7E007F278 (Bluetooth)	34	476E1BA1E (Bluetooth)	marsch	door opened	2004-01-14,09:58:00
20	7E007F278 (Bluetooth)	34	476E1BA1E (Bluetooth)	marsch	door opened	2004-01-14,10:23:03
21	7E007F278 (Bluetooth)	34	476E1BA1E (Bluetooth)	marsch	door opened	2004-01-14,10:23:38
23	7E007F278 (Bluetooth)	34	476E1BA1E (Bluetooth)	marsch	door opened	2004-01-14,10:35:09
24	7E007F278 (Bluetooth)	34	476E1BA1E (Bluetooth)	marsch	door opened	2004-01-14,10:54:22

Abbildung 4.1: Beispiel für die mitgeloggten Türöffnungsereignisse in der LOMOT-Datenbank (Tabelle LOMOT_TRACKING.log_gates)

Kapitel 5

Testaufbau, Ergebnisse

5.1 Testaufbau

Server

Der *BlueKey* Server wurde auf demselben Linux-PC, auf dem die LOMOT-Datenbank läuft installiert, damit dieser einfach auf diese MySQL-Datenbank zugreifen kann. Wenn der Server auf einem anderen PC installiert werden soll, wäre ein ssh-tunnel zur verschlüsselten Datenübertragung (für MySQL derzeit TCP-Port 3306) empfehlenswert.
Beim Start des Servers muss darauf geachtet werden, dass in dem Java-Classpath der eigene Pfad und auch die erforderliche Java-Bibliothek „mysql-connector-java“ angegeben wird. In unserem Testfall wurde der Server mit folgendem Befehl gestartet:

```
java -cp "./:./mysql-connector-java-3.0.9-stable-bin.jar" BlueKeyServer
```

Nach dem Start wartet der Server auf ankommende Verbindungen (derzeit auf TCP-Port 12345).

Client

Der *BlueKey* Client wurde auf einem Palm Tungsten T installiert (Palm OS Device mit integrierter Bluetooth-Hardware). Nach der Installation der Anwendung „BlueKey.prc“ müssen noch folgende Initialisierungen vorgenommen werden: Zuerst wird die Applikaton gestartet und im Menüpunkt „Settings“ werden die Programmparameter eingegeben (LOMOT Username, Password, *BlueKey* Server-IP und -Port). Anschliessend muss noch der Verbindungseintrag *„BlueKey_Connection“* in den Palm Systemeinstellungen (unter Verbindung) initialisiert werden (siehe Kapitel 4.1.1) durch einmaliges Suchen und Zuweisen irgendeines Bluetooth-Devices.
Nach diesen Vorbereitungen ist der Client einsatzbereit und muss als nächstes von einem AP aktiviert werden.

Bluetooth Access Point

Da *BlueKey* von einem aktiven AP ausgeht und das entsprechende Projekt (Smart Access Point, [SK04]) derzeit noch nicht implementiert ist, wurde die Funktionsweise des APs auf einem Windows-PC simuliert. Auf diesem PC waren folgende Vorbereitungen nötig:

Installation eines 3Com Bluetooth USB-Sticks und der Software „Bluetooth Connection Manager“. Weiters ist die Einrichtung eines Gateways für eingehende PPP-Verbindung über den virtuellen Seriellen Bluetooth-Port erforderlich. Dafür muss entweder im Betriebssystem ein PPP-Daemon (unter Windows ein entsprechender RAS-Dienst) eingerichtet werden, oder man verwendet dafür eine spezielle Software. Eine sehr einfache und stabile Verbindungssoftware ist z.B. „Mocha W32 PPP“ [Moc00], die in unserer Testumgebung verwendet wurde.
Als letzes muss noch eine Datei zum Aktivieren des Clients vorbereitet werden. Dafür wird einfach eine leere Datei mit der Dateinamenserweiterung „.blk“ erstellt, also z.B. „Aktiviere_Client.blk“.

5.2 Testablauf

Als erstes wird nun die Aufgabe des AP auf dem PC simuliert. Eine Testperson startet im „Bluetooth Connection Manager“ zyklisch ein Bluetooth-Inquery, während sich die zweite Testperson mit dem Palm OS Client dem Access Point nähert. Sobald der Client vom AP gefunden wird, muss die Datei „Aktiviere_Client.blk“ händisch per OBEX an diesen geschickt werden.
Der Client verbindet sich nun automatisch über diesen AP zum laufenden *BlueKey* Server, die *BlueKey* Authentifizierung beginnt und entsprechende Türen werden geöffnet (derzeit durch eine Meldung am Server angedeutet) und ein Tracking-Eintrag wird in die LOMOT-Datenbank geschrieben.

5.3 Ergebnisse

Funktionsnachweis

Das Öffnen von Türen ist prinzipiell mit dieser Systemarchitektur realisierbar. Könnte anstelle des simulierten AP ein Smart Access Point verwendet werden, würde der Authentifizierungsprozess automatisiert ablaufen.

Erfahrungsberichte mit Bluetooth

Der größte Vorteil beim Einsatz von Bluetooth für die drahtlose Kommunikation ist mit Sicherheit die große Verbreitung und Verfügbarkeit in verschiedenen mobilen Endgeräten.
Ein erheblicher Nachteil ist aber die langsame bzw. unsichere Erkennungsgeschwindigkeit beim Inquery. Laut Spezifikation kann es bis zu 10 Sekunden dauern, bis ein Device erkannt wird. Ausserdem ergaben unsere Test eine große Streuung dieser Werte (manchmal wurde ein Gerät schon nach 3 Sekunden gefunden, manchmal erst nach 10), die auf die zufällige Frequenzwahl beim Frequency-Hopping zurückzuführen ist. Weiters haben die Tests gezeigt, dass es vorkommen kann, dass sich ein Device zwar schon im Empfangsbereich des AP befindet (wird also mittels Inquery gefunden), aber keine PPP-Verbindung über diesen aufbauen kann. Das dürfte daran liegen, dass der Aufbau einer stabilen Netzwerkverbindung erst bei einer besseren Verbindungsqualität akzeptiert wird, auch wenn diese schon für ein Inquery oder kurzes OBEX-Senden ausreicht. Ob dies nur für Verbindungen mit der verwendeten Palm-Hardware oder für Bluetooth generell gilt, konnten wir nicht testen. Jedenfalls konnten wir auch Schwankungen der benötigten Zeit für den Verbindungsaufbau feststellen, gelegentlich musste der Palm Client den Verbindungsaufbau 1-2 mal wiederholen. Möglicherweise kann das Problem durch Adaptierung verschiedener TimeOut-Werte verbessert werden.

Da zumindest beide Zeiten (Inquery und Verbindungsaufbau) nicht genau vorherzusehen sind, kann Bluetooth für die „Echtzeit“-Anwendung „Türen öffnen“ noch nicht optimal eingesetzt werden.

Erfahrungsberichte mit Palm OS

Ein Palm OS PDA ist prinipiell für den Einsatz als *BlueKey* Client gut geeignet. Da das aktuelle Palm OS 5 ein Single-Task Betriebssystem ist, ist es für die automatisierte Authentifizierung gut geeignet. Es kann im Gegensatz zu einem PocketPC-PDA (Windows) nicht von einer anderen Anwendung gestört bzw. blokiert werden. Auf dem Palm PDA können weiterhin problemlos alle andere Anwendungen benutzt werden. Die *BlueKey* Applikation wird verlässlich gestartet, sobald ein neuer AP die OBEX Datei schickt. Ein Nachteil dabei ist aber, dass die vorher benutzte Anwendung (z.B. Kalender) möglicherweise genau während einer Eingabe beendet/unterbrochen wird. Weiters ist während Netzerk-Kommunikaton der Palm OS PDA etwa 10 Sekunden für alle Eingaben blockiert. Im normalen Anwendungsfall sollen die Smart Access Points aber einen *BlueKey* Client nur dann automatisch aktivieren, wenn sich dieser in dem Gebäude bewegt, also nur beim neuen Eintreten in den Empfangsbereich.

Leider bereitete das auf dem Palm Tungsten T verwendete Betriebsystem Palm OS 5.0 einige Probleme. Es traten immer wieder (nicht reproduzierbare) Abstürze bei der Verwendung von Bluetooth auf. Manchmal stürzte er beim Empfang beliebiger OBEX Dateien ab. Am häufigsten traten die Abstürze beim Aufbau von Netzwerkverbindungen über Bluetooth auf, egal ob diese vom Web-Browser, Emailprogramm oder von der *BlueKey* Applikation initiiert wurden.
Da das verwendete Testgerät aber schon etwas älter ist, und keine OS Updates installiert wurden, ist es durchaus möglicht, dass diese Probleme bei neueren Palm OS Geräten schon behoben sind.

Ein weiteres Spezifikum des Palm Tungsten T ist, dass das Gerät im „Standby“-Zustand zwar durch direktes Senden einer Datei über OBEX aktiviert werden kann (und dabei auch die *BlueKey* Applikation gestartet wird), er aber in diesem Betriebszustand leider nicht per Inquery auffindbar ist. Das bedeutet, dass der PDA ständig eingeschaltet sein muss, um *BlueKey* nutzen zu können, wodurch der Vorteil des geringeren Energieverbrauchs, der durch den Einsatz des Smart Access Points entsteht, teilweise wieder zunichte gemacht wird. Allerdings ist der Energieverbrauch immer noch geringer als wenn der Client ständig aktiv nach APs suchen müsste. Als Richtwert konnten etwa 4-5 Stunden Laufzeit beim passiven Warten auf die Aktivierung gegenüber weniger als 2 Stunden bei ständiger Bluetooth-Aktivität ermittelt werden.

Kapitel 6

Zusammenfassung

6.1 Schlussfolgerungen

Wenngleich *BlueKey* prinzipiell funktionstüchtig ist, zeigen sich doch einige Einschränkungen durch die Bluetooth-Technologie und den verwendeten PDA.
Die u.U. lange Zeit bis zum Auffinden eines Gerätes beim BT-Inquery (lt. BT-Spezifikation 10.24 Sekunden, bis sicher alle Geräte gefunden sind – bei unbewegten Geräten) sowie die unvorhersehbare Reichweite (in den Tests schwankte sie zwischen 5 und knapp 30 Metern) lassen jedoch Zweifel an der tatsächlichen Praxistauglichkeit für diesen Anwendungsfall aufkommen.

Hinzu kommt, dass sich das verwendete Palm OS 5.0 in den Tests als recht instabil erwies. Da diese nicht nur beim von uns entwickelten Programm, sondern auch bei anderen (zum Teil mit dem Gerät standardmäßig mitgelieferten) Applikationen auftreten, ist davon auszugehen, dass es sich um Fehler im BT-Stack oder der Netzwerk-Library von Palm OS handelt.

6.2 Weiterführende Arbeit

Java-Client

Durch die immer größere Verbreitung von Java-fähigen Geräten (Mobiltelefone, PDAs,...) erscheint eine Implementierung der Client-Software in Java sinnvoll. Insbesondere werden Mobiltelefone ständig mitgetragen, was sie für *BlueKey* besonders geeignet macht.
Diese Weiterentwicklung setzt jedoch eine einheitliche netzwerk- und OBEX-fähige Java-VM auf den verschiedenen Geräten voraus.

Verschlüsselung

Einen weiteren wichtigen Punkt stellt die Erweiterung von *BlueKey* um die Möglichkeit einer starken Verschlüsselung dar. Hier scheint uns die Verwendung eines Public Key oder Public Key/Private Key Verfahrens (vgl. [HJP03]) sinnvoll. Für letzteres könnte das Passwort des Benutzers als Private Key dienen, welcher vom Client unter keinen Umständen verschickt werden darf. Alternativ könnte die Username/Passwort-Authentifizierung um einen zusätzlichen – dem Benutzer zugeordneten – Private Key ergänzt werden. Hierfür wäre jedoch auch eine Erweiterung der LOMOT-Datenbank erforderlich.

Server auf Smart Access Point

Mit der Implementierung des Smart Access Point [SK04] ergibt sich die Möglichkeit, den *BlueKey*-Server dezentral auf den APs laufen zu lassen. Dazu wären nur kleine Erweiterungen an der Client-Software nötig, um die Verbindung nicht zu einer statisch vergebenen IP-Adresse sondern direkt zum AP aufzubauen.
Der Vorteil dieses Systemaufbaus wäre die erhöhte Ausfallsicherheit; beim Ausfall eines Servers (eines APs) würde die Funktionalität der restlichen Bereiche nicht beeinträchtigt.

Stabilitätsverbesserung

Da sich die verwendete Version von Palm OS im Zusammenhang mit Bluetooth als recht instabil erwiesen hat, könnte versucht werden, die Fehlertoleranz des *BlueKey*-Clients gegenüber Betriebssystemabstürzen zu erhöhen. Dies könnte beispielsweise durch einen Workaround erreicht werden, bei dem die Applikation jeden ihrer Arbeitsschritte protokolliert bzw. abspeichert. Wenn nun ein Weg gefunden werden kann, die Client-Software nach dem Booten des Betriebssystems automatisch zu starten, könnte diese ihre Arbeit genau an dem Punkt wieder aufnehmen, an dem die Unterbrechung auftrat.

Literaturverzeichnis

[AIM03] AIM Inc. Radio Frequency Identification (RFID) Homepage. URL: http://www.rfid.org/, 2003.

[Blu03] Bluetooth.org. The Official Bluetooth Membership Site. URL: http://www.bluetooth.org/, 2003.

[Cha00] Y. Chansu. Designing Low-Power Embedded Systems. Technical report, Information an Communications University (ICU), 2000.

[Fos03] L. R. Foster. *Palm OS Programming Bible, Second Edition*. Wiley Publishing, Inc., 2003.

[HJP03] A. Hämäläinen, P. Jäppinen, and J. Porras. Applying Wireless Technology to Access Control Systems. In *Workshop on Applications of Wireless Communications*. Lappeenranta University of Technology, Finland, aug 2003.

[Met03] Metrowerks Inc. Metrowerks Homepage. URL: http://www.metrowerks.com/, 2003.

[Moc00] MochaSoft. PPP-Deamon: Mocha W32 PPP. URL: http://www.mochasoft.dk/f_download2.html, 2000.

[MyS03] MySQL. JAVA MySQL-Treiber: MySQL Connector/J. URL: http://www.mysql.com/products/connector-j/index.html, 2003.

[Pal03] PalmSource Inc. Palm Software, Palm OS SDK and code samples. URL: http://www.palmsource.com/, 2003.

[PBFF99] C.A. Pinto, A.C. Borim, J.M. Fernandes, and A.R. Ferreira. Wireless implementation for access control to restricted areas. In *Circuits and Systems*, pages 1078–1081 vol. 2. Dept. de Eng. Electr., Univ. Fed. de Uberlandia, okt 1999.

[Pol03] M. Polaschegg. Entwurf und Implementierung einer ”Location Awarness” Applikation unter Verwendung von Smart Tags. Technical report, ITI, 2003.

[RJK88] R.P. Rumble, D.Y. Jong, and S.M. Kluz. The CAIN II access-control system. In *Security Technology*, pages 127 – 132. Lawrence Livermore Nat. Lab., Livermore, CA, USA, okt 1988.

[Sch02] C. Schotzko. Intelligente Gebäude und Wohnungen, may 2002.

[SK04] S. Schranzhofer and D. Kopeinigg. Smart Access Point. Technical report, ITI, 2004.

[Ski02] Skidata. Ski Access, Presseaussendung Interalpin 2002. Technical report, Skidata AG, www.skidata.com, 2002.

[Tho03] M. Thonhauser. Webinterface für LOMOT. Technical report, ITI, 2003.

Anhang A

Beispiele für den Kommunikationsablauf

Hier werden einige Beispiele für den Ablauf der Kommunikation zwischen Client und Server sowie verschiedene Fehlerfälle, die dabei auftreten können, angeführt. Man beachte, dass das gewählte Übertragungsprotokoll es ermöglicht, die Kommunikation auch im Fehlerfall schon nach kurzer Zeit abzubrechen.

Normale Türöffnung

Client	*Server*
`BLUEKEY_HELO 01.00 POSN,DISP,USER,PASS,CRYP_PLAIN`	
	`BLUEKEY_CRYP CRYP_PLAIN`
	`BLUEKEY_SEND POSN`
`BLUEKEY_POSN 0A3417BE1`	
	`BLUEKEY_SEND USER,PASS`
`BLUEKEY_USER asdfghj`	
`BLUEKEY_PASS qwertz`	
	`BLUEKEY_ACCP`
	`BLUEKEY_DISP Tür 35 geöffnet.`
	`BLUEKEY_QUIT`
`BLUEKEY_QUIT`	

Fehler bei Userdaten

Client	*Server*
BLUEKEY_HELO 01.00 POSN,DISP,USER,PASS,PINC,CRYP_PLAIN	
	BLUEKEY_CRYP CRYP_PLAIN
	BLUEKEY_SEND POSN
BLUEKEY_POSN 0A3417BE1	
	BLUEKEY_SEND USER,PASS
BLUEKEY_USER asdfghj	
BLUEKEY_PASS qwerty	
	BLUEKEY_FAIL USER,PASS
	BLUEKEY_DISP Username oder Passwort falsch.
	BLUEKEY_QUIT
BLUEKEY_QUIT	

Kein Pincode bei PIN-geschützter Tür

Client	*Server*
BLUEKEY_HELO 01.00 POSN,DISP,USER,PASS,CRYP_PLAIN	
	BLUEKEY_CRYP CRYP_PLAIN
	BLUEKEY_SEND POSN
BLUEKEY_POSN 1EA4199B	
	BLUEKEY_FAIL PINC
	BLUEKEY_DISP Tür 22: Pincode erforderlich.
	BLUEKEY_QUIT
BLUEKEY_QUIT	

Client kann keine Verschlüsselung, Server fordert eine

Client	*Server*
BLUEKEY_HELO 01.00 POSN,DISP,USER,PASS,CRYP_PLAIN	
	BLUEKEY_FAIL CRYP_RSA
	BLUEKEY_DISP RSA-Verschlüsselung erforderlich.
	BLUEKEY_QUIT
BLUEKEY_QUIT	