Personal Servers as Digital Keys

Allan Beaufour, Philippe Bonnet
Department of Computer Science
University of Copenhagen
beaufour@diku.dk, bonnet@diku.dk

Abstract

Personal servers are an attractive concept: People carry around a device that takes care of computing, storage and communication on their behalf in a pervasive computing environment. So far personal servers have mainly been considered for accessing personal information. In this paper, we consider personal servers in the context of a digital key system. Digital keys are an interesting alternative to physical keys for mail or good delivery companies whose employees access tens of private buildings every day. In this paper, we present a digital key system tailored for the current incarnation of personal servers, i.e., a Bluetooth-enabled mobile phone. We describe how to use Bluetooth for this application, we present a simple authentication protocol and we provide a detailled analysis of response time and energy consumption on the mobile phone.

1 Introduction

Intel and HP are promoting the concept of *personal server* [13, 22]. People carry around their personal server, a small device that takes care of storage, computing, and communication on their behalf in a pervasive computing environment. Mobile phones with extended functionalities or PDAs enhanced with communication capabilities are the current incarnation of the personal server. Ideally, a personal server should be small-scale so that people can carry it easily; it should avoid consuming too much energy so that it can run for several days; it should be efficient to ensure good response time and thus a nice user experience.

So far personal servers have been essentially considered as a means to access personal information in an ubiquitous manner. In this paper, we consider a natural extension of the personal server functionalities: *the personal server as a digital key*.

First, why consider digital keys? What is wrong with the good old physical keys? We use them every day to unlock doors. It is a well-known and well-tested technology. This being said, we have all experienced problems like carrying too many keys while forgetting the one key we need, or maybe having a lock replaced because we lost a single key.

The problems with physical keys are strategic for companies in the business of mail or goods delivery. These companies need access to many different private buildings – doors spread over a wide geographical area and governed by many different owners. Personnel need to carry keys for each single door on the delivery route. Carrying all these keys (literally kilograms) is a hassle

in the daily use. But just as important, routes, personnel, and locks change over time, making it a resource consuming operation to assure that all personnel have the right keys at the right time.
Consider the following scenario. In the morning, a mailman downloads the route she is going to follow for the day (a mailman can be assigned various routes depending on the deliveries and on the staff available). The digital keys she needs are uploaded to her personal server. Whenever she approaches a door during her tour, the appropriate key is transfered to the lock automatically, without physical contact and without user interaction. The automatic unlocking makes it possible for the mailman to unlock the door while carrying goods, making the system very user-friendly.
Replacing the physical keys with digital keys managed by a personal server has the following advantages: A digital key can easily be distributed to users, it can be specialized for each user so each key will be unique, and it can be restricted to a given time or date range to increase security. Moreover, many such digital keys can be contained in a personal server. Several companies have expressed commercial interests in such digital key systems and it is not unreasonable to expect that most private buildings in large cities will be equipped with such systems in a near future.
We propose to use a mobile phone equipped with Bluetooth as a personal server. Mobile phones are getting very common and bringing your mobile phone with you is becoming as natural as bringing your keys. Combined with the ability to run custom applications and to perform short-range wireless communications via Bluetooth, the mobile phone is a good candidate to become a personal server. There are some unknowns however: Can the functionality of a digital key system be implemented on a Bluetooth-enabled mobile phone? If yes, is the platform efficient in terms of response time and resource usage (both in terms of storage and energy)?
In this paper, we describe how a Bluetooth-enabled mobile phone can be used as a personal server in the context of a digital key system. Specifically, we are making the following contributions:

- We present how Bluetooth can be used to establish connections in a digital key system. The major issues are (i) to devise an efficient way to use Bluetooth's device discovery protocol and (ii) to establish connections efficiently when the user enters a building.
- We present a simple, specialized, and secure authentication protocol. Existing general purpose authentication protocols, such as SSH [2], are too complex for our purpose.
- We implemented a prototype digital key system on a Sony Ericsson P800 mobile phone. We present the results of experiments to evaluate response time and energy consumption when running our prototype digital key system. Note that we calibrated energy consumption on the mobile phone under different regimes. Such performance numbers are not available from the constructor or in the litterature. They provide a reference for further studies on personal servers.

Before discussing these contributions in Section 3, Section 4 and Section 5 respectively, we precise the application context. We discuss related work in Section 6.

2 Digital Key System

Ideally, a user carrying mail or goods as her personal server running the digital key application should not have to worry about anything but pushing the door open. More specifically, the digital key system should be:

- *secure*: access to a door should be limited to authorized persons
- *easy-to-use*: access to a door should be granted without user interaction
- *efficient*: a user should not have to wait more than 5 seconds for a door to open (an arbitrary yet reasonable target)

In terms of architecture, the digital key system we consider has three components (see Figure 1):

Lock Device A fixed device, attached to the physical lock unit. The lock device is an actuator that can open the physical lock. It is equipped with storage and computation capabilities as well as a Bluetooth radio. It can thus establish local connections. The lock device is not connected to a LAN or a WAN, to simplify installation and management and to eliminate running costs. The lock device is externally powered as it is fixed and located in a building connected to the power grid. The design of a lock device is beyond the scope of this paper (BTNodes developed at ETH Zurich[12] are a good approximation of what lock devices could be).

Personal Server The mobile phone acting as a personal server can communicate with the lock device via Bluetooth.

External Authority A third party responsible for generating the digital keys. It can communicate with the personal server and is trusted by the lock device. No direct communication with the lock device is possible. Key distribution from the external authority to the personal servers is beyond the scope of this paper.

Figure 1: Digital Key System Architecture

3 Bluetooth Connections

Bluetooth requires devices to establish a connection before they can exchange data. This is a constraint from a performance point of view: devices first need to discover each other before they can establish a connection and then exchange data. This is time and energy consuming.

Bluetooth device discovery is a powerful mechanism. It allows connections to be established between devices that have no a priori knowledge of each other. In the discovery phase, one of the devices transmits search requests, *inquiry*, while the other listens for these requests, *inquiry scan*. Inquiry is an expensive operation in terms of energy consumption: an inquiring device broadcasts inquiry packets continuously during the entire inquiry period (i.e., 1.28 seconds). In contrast, a device doing inquiry scan with default parameters only listens for inquiry messages for 11.25 ms every period. This qualitative analysis is confirmed by our calibration of the mobile phone (see Section 5.2.4).

The Bluetooth specification recommends performing inquiry for 10.24 seconds to assure that all devices present in the vicinity are found [5, Part B, 10.7.3]. Practical experiments show that 99 percent of all discoveries are successful after approximately 6 seconds (2.3 seconds on average) [12, 14]. These numbers are large compared to our goal of five seconds for opening a door.

A first issue for our digital key system is when and how to perform inquiries. The first approach is to let the lock device perform inquiry scan. This seems like the most natural approach: the personal server moves around and discovers the lock device it needs to communicate with to unlock a door. One problem though is that performing inquiry is more energy consuming than performing inquiry scan. Another problem is that doing constant inquiries ties up the Bluetooth chip on the personal server, making it unavailable for other purposes. Letting the lock device perform inquiries solves these problems (there is no power limitation on the lock device and it does not need to serve any other purpose than connecting to personal servers). This is the design we adopt. It does however create a new problem: how does a lock device connect to the right personal server?

This is the second issue for our digital key system: when and how should the lock device stop inquiry and establish a connection. When the lock device performs an inquiry, it detects the Bluetooth devices in its vicinity. There could be plenty (given that Bluetooth radios have a range of tens of meters). The first question is when to stop inquiry: after a fixed period or as soon as one device is detected. Our goal is not to find all the devices in the vicinity of the lock device but to establish a connection with a personal server as quickly as possible. We therefore choose to stop inquiry as soon as a device is detected.

Now the problem is that the first Bluetooth device detected by the lock device might not be a personal server, even if there is one in its vicinity. The lock device must determine if the detected device is a personal server or not. The lock device keeps a history of the Bluetooth devices it has encountered and classifies those that are personal servers and those that are not. If the first detected device is already known as something else than a personal server then the lock device returns to its inquiry mode. If the first detected device is not known, then a connection is established. If the device is not a personal server then the lock device goes back to the inquiry mode.

Once a connection is established between lock device and personal server, the actual authentica-

tion can start. This is the topic of the next section.

4 Secure Access Control

The main problem is to design a secure protocol that allows a personal server with a valid key to unlock a door. Recall that keys are created by an external authority that cannot communicate with the lock device that handles the unlocking of the door. The solution we propose here relies on well-known and well-tested security mechanisms whenever possible and keeps the system as simple as possible (note that simplicity is also important because of ressource constraints on the personal server).

4.1 Secure Protocol

In order to unlock a door, a personal server needs (i) an *identity*, i.e., a unique identifier and (ii) a valid *access key* for the door, i.e., a simple data structure associating lock device identifiers with the personal server identity – the list of doors the personal server is authorized to open – in our digital key system, access keys are created and distributed by the external authority. The secure protocol ensures that the personal server proves its identity (*authentication*) and that the data it sends are actually received unaltered by the lock device and cannot be reused later (*integrity*) [1]. Existing protocols (e.g., TLS [7], IPSec [21] and SSH [2]) support secure and authenticated communication but do not include access control. Their complexity make them poor candidates for the constrained environment of a personal server. We thus implement our own, simple access control protocol:

1. The lock device establishes a connection and sends its identifier to the personal server.
2. The personal server looks up an access key for the door whose identifier it just received and sends that access key to the lock device together with a proof of its identity.
3. The lock device unlocks the door if the proof of the identity is correct and if the access key is valid. Success or failure is reported back to the personal server (for feedback and logging purposes).

We rely on asymmetric cryptography for authentication [2]. A personal server proves its identity by (i) sending its public key to the lock device, (ii) proving that this public key is actually associated to its identity, and (iii) proving that it possesses the corresponding private key. The personal server uses a X.509 certificate, provided (and signed) by a certificate authority trusted by all components of the digital key system, to associate its public key with its identity. In order to prove that it possesses the private key, it signs the packet containing the certificate and the access key. This signature also guarantees the integrity of the data transferred from the personal server to the lock device. We skip details of the protocol because of space limitation (see [3] for details).

[1] Note that denial-of-service or relay attacks cannot be avoided. Their effects can be limited but at the cost of increased software complexity.

[2] Symmetric cryptography requires personal servers and lock devices to share secret information. This is neither desirable nor practical.

Note that in this protocol, the only secret information is the information needed to prove the personal server's identity, i.e., its private key. If an attacker accesses the private key (e.g., by stealing the personal server) then the associated identity should become invalid. Using traditional *Certificate Revocation Lists* on the lock devices is unpractical because there is no direct connection between lock devices and the certificate authority. Instead we propose to limit the validity of the access keys to short time ranges. In the worst case the attacker can use the private key for the length of the access key validity period. Once a theft or a loss of a personal server is registered, the external authority stops producing access key for the invalid identity.

4.2 Cryptographic Algorithms

RSA [18] is a generally trusted and well-tested algorithm that has been used for years. Recently, ECDSA [16] has gained ground, and is now regarded as at least as secure as RSA (see [19]). We evaluate these protocols by comparing key size and speed of signature generation/verification.
The size of the asymmetric keys influences both the size of data packets exchanged between personal server and lock device as well as the speed of both signing and verification. The current NIST recommendations [17] recommends using 1024 bit keys for RSA/DSA, and 160 bits for ECDSA while existing certificate authorities (e.g., VeriSign) use 2048 bits RSA/DSA and 224 bits for ECDSA.

Type	System	RSA (ms)	DSA (ms)	ECDSA (m)
Level 1				
Sign	Pager	15889	9529	1011
Verify	Pager	1008	18566	1826
Sign	PC	67	24	2
Verify	PC	4	47	4
Level 2				
Sign	Pager	111956	*n/a*	1910
Verify	Pager	3608	*n/a*	3701
Sign	PC	441	*n/a*	4
Verify	PC	13	*n/a*	8

Table 1: Algorithm timings on the Pager and the PC. From [6].

The speed of signature generation/verification is hardware dependent. There is no benchmark yet focusing on cryptographic algorithms on personal servers. The closest benchmarks we have found is in [6], where the authors implement PGP [11] on a pager (10 MHz custom Intel 386) and a PC (400 MHz Intel Pentium II). Based on the results summarized in Table 1, ECDSA appears as the best candidate. It is orders of magnitude faster for signatures, and is only a little slower than RSA for verification. Assuming that the personal server is somewhere in between the pager and the PC, we can expect that it will need well under one second to generate a signature.
In terms of storage, access keys occupy 136 bytes with ECDSA signatures and 222 bytes with RSA. This different is significant as access keys obtained with ECDSA can be sent via 1 SMS (the limit for SMS size is 140 bytes, see [9, 10]).
In conclusion, ECDSA seems preferable to RSA, however actual implementation needs to take into

account the requirements of the certificate authority and the algorithms supported by the personal server.

5 Performance Evaluation

5.1 Experiment Environment

We implemented a prototype of our digital key system using a Sony Ericsson P800 mobile phone as personal server. The phone has a 156 MHz ARM9 CPU, 16 MB RAM, and 12 MB flash. It uses Symbian OS, and is programmable in both Java and C++ (for full specifications see [8]). The prototype application was implemented on the mobile phone using standard Symbian OS libraries as well as the *RSAREF2* cryptography toolkit (implementing RSA). There was no suitable ECDSA implementation available. The lock device is simply implemented on a PC equipped with a USB Bluetooth unit.

The metrics we are interested in are response time and energy consumption. Figure 2 shows our measurement set-up. We measure response time using wall clock time. In order to measure energy consumption, we measure the voltage drop over a resistor inserted in series between the battery and the mobile phone. The voltage drop divided by the resistance gives us the current draw and thus the energy consumption (assuming a constant battery voltage). We cannot simply measure the current by inserting a multimeter in series between the batteries and the phone: the phone refuses to boot in this configuration.

Figure 2: Setup for the measurement: We measure the voltage drop over a resistor inserted in series between the battery and the mobile phone

The precision of the measurements depends on the precision of both the resistors and the multimeter, the stability of the voltage on the battery, and the resistance of the wires connected to the resistor. The resistors have an accuracy of $\pm 5\%$ and the multimeter one of $\pm 0.25\%$. The stability of the voltage is important because the voltage is used to calculate the energy consumption using Ohm's law. Ideally the voltage should be measured continuously, but a second multimeter for

this purpose was not available during our experiments. We did some initial measurements and found that the battery deviated less than 1% from 3.6V for the first hour at least. We also tried to measure the resistance in the wires connected to the resistor and found it was negligible, so the wires will not influence measurements either. All in all, the measurements will have an inaccuracy of maximum 6–7%. All measurements were performed a minimum of five times and for at least ten minutes.

5.2 Device Calibration

There are no public energy specifications for the mobile phone. The only available information is the voltage of the battery (3.6V), the capacity of the battery (1000 mAh), best case life time during conversations (13 hours) and standby (400 hours). From this we can deduce that the battery should have 12960 Joule, standby should consume ∼9 mW and conversations ∼277 mW. There is no further information available. We therefore calibrate the mobile phone in its different operating modes.

5.2.1 Idle Modes

As a first step, we must find out how the phone behaves in idle mode. The radio can be turned off by setting the mobile device in a *flight mode*, that makes it usable in a plane or any other area where mobile phones are disallowed. The display can have the back-light turned on or off, and the phone has power-save modes that saves power by turning the display off after a specific amount of time. The power usage of the CPU can not be influenced by user-available settings.
To reveal the different power modes of the phone, measurements were made for all the possible combinations of the radio and back-light modes. It takes 150 seconds before the phone reaches its final idle mode after being booted. The lowest power consumption is ∼9 mW, in flight mode with the screen turned off. The screen has four modes: On (∼320 mW), dimmed back-light (∼140 mW), no back light (∼20 mW), and off.
To obtain a stable baseline, we want to use the mode where (i) Bluetooth is enabled and (ii) where the power consumption is as stable as possible. We use a mode where back-light and power save are off. It consumes ∼33 mW. The problem is that power consumption is not really constant: it fluctuates because of the GSM radio. Fortunately, the GSM power usage comes in peaks, so as long as the functionality being tested has a stable power consumption it will be possible to disregard the GSM peaks. We have tried to find a way to have Bluetooth turned on with GSM turned off, but it is not supported on the phone.

5.2.2 Usual Functionalities

In order to be able to relate the power usage of our application to the power usage of the normal functions of the phone, we have tested the following: using the GUI, having a conversation, using the camera, and playing a game.
Having a phone conversation takes around 250–300 mW, not far from the specifications. Using the GUI or the camera costs 500–600 mW, while playing a game ranges from 800 mW to 1000

mW. Compared to these numbers it was a bit surprising to see the cost of just touching the screen. Holding the pointer still at the same position on the screen costs 500 mW, no matter whether the program has a GUI or not (a console mode program). This is probably caused by the endless processing of pointer events from the screen. For a GUI application this is unavoidable, but for a console program this is not necessary, and must be considered a design flaw. Lastly, we tried to stress the phone maximally by playing a game with back-light on, full sound, a GSM connection and Bluetooth connection on at the same time – this peaked at around 1570 mW.

5.2.3 Cryptography

For cryptography, the cost of one iteration of each operation is (with idle mode cost subtracted):

Operation (bits)	**Time (s)**	**Cost (mJ)**
Sign, 1024	0.48	241.00
Verify, 1024	0.02	10.08
Sign, 512	0.08	40.32
Verify, 512	< 0.01	0.01

The device performs well. Generating a signature takes 0.5 seconds – only 7.5 times slower than the PC used in [6]. Extrapolating these results, an ECDSA signature would take 0.015 seconds for 160 bits keys and 0.030 for 224 bits keys. This would practically eliminate cryptography as a time factor for the application.

5.2.4 Bluetooth

Experiments showed that Bluetooth costs (with idle mode cost subtracted):

Action	**Cost**
Being discoverable	4 mW
Being discoverable and connectable	7 mW
Answering an inquiry	< 1 mW
Doing inquiry	127 mW
Maintaining a connection or transferring data	557 mW / 532 mW[3]
Transferring data, > 180 seconds	291 mW / 331 mW[3]
Slack	408–720 mJ

One of our first assumptions was that inquiry was expensive compared to inquiry scan, and it certainly is. Doing inquiry scans cost 7 mW while doing inquiry costs 127 mW. The cost of answering an inquiry was so low that it disappeared in the GSM background noise. Doing inquiries constantly would drain the battery 14 times faster than doing inquiry scans, so the choice of the lock device as the inquiring party is correct. Compared to having a conversation (250–300 mW), transferring data with Bluetooth is quite expensive – especially when adding the slack (extra energy usage after each Bluetooth activity on the phone for which we do not have an explanation – our best guess is that the Bluetooth chip power down is delayed).

[3] When phone initiates or sends.

5.3 The Application

Based on the measurements in the previous sections we can deduce what discovery and connection from the lock device will cost. We do not know how much time the actual data transfer takes, but with a conservative assumption of around 0.5 seconds (1 KB at 5 KB/sec plus overhead), the cost calculation is the following:

Function	**Cost (mJ)**
Answering inquiry	0
Signature (1024 bit)	241
Bluetooth connection (0.5 s)	278.5
Bluetooth slack	408–720
Total	928–1240

We obtain 1000 mJ for the cost of running the application, to which the 40 mW should be added to get the actual power usage (the idle mode baseline cost plus Bluetooth cost).

An actual application run with 1024 bit keys is shown in Figure 3. Average inquiry time was found to be 3.6 seconds (minimum 2.8 seconds, maximum 4.4 seconds), which is stable but more than one second slower than previously reported. The reason for inquiry being slower may be found in the interoperability between the specific Bluetooth devices we used, or the specific noise conditions in our office environment. It is beyond our scope to investigate this further, but the results confirm that the Bluetooth discovery process is influenced by the specific equipment and environment.

The Bluetooth connection only consumes energy after it has been established. In our application, the connection is established for around 0.7 seconds (see Figure 4). This too fast to be captured on the graph with the sampling rate of our multimeter (1 sample per second). Note that our graph shows a peak of energy consumption (at 550 mW) for 3.5 seconds after the connection is closed. This is due to a time-out in the Bluetooth stack we used on the PC, which should be optional. A way to avoid this time-out is to close down the entire Bluetooth stack on the PC when closing a connection. We thus ignore this cost in the rest of our discussion.

The average time needed to establish a connection was 1.2 seconds which was a lot faster than we expected. Even though most connects took around 1 second a few connects took over 10 seconds, the maximum being 18 seconds. The same phenomenon was experienced in [4], it is unclear why. The standard *page timeout* for a Bluetooth device is 5.12 seconds and the Bluetooth device we used was also set to this (see [5, Part H, 4.7.16]). So the connection establishment should timeout after this period. Unfortunately we do not have an explanation for this. Our conclusion is that there must be a flaw in the implementation of the connection establishment procedure.

Figure 4 shows the average time spent in different parts of the application. Apart from discovery and connection establishment the rest of the application takes 0.7 seconds on average, where 0.48 seconds are from the signature generation, so the rest of the application including communication takes around 0.2 seconds. All in all, the total average unlocking time is 5.5 seconds: Bluetooth discovery and connection establishment accounts for 87% of the total response time.

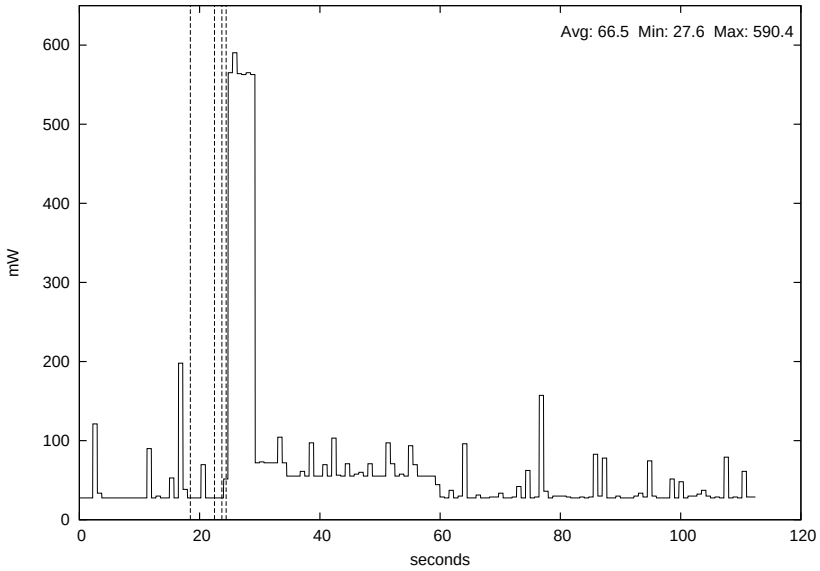

Figure 3: This graph shows the lock device connection to the mobile device, with mobile device key being 1024 bits. The vertical lines denote: Inquiry start, connection start, connection succeeded, and connection closed.

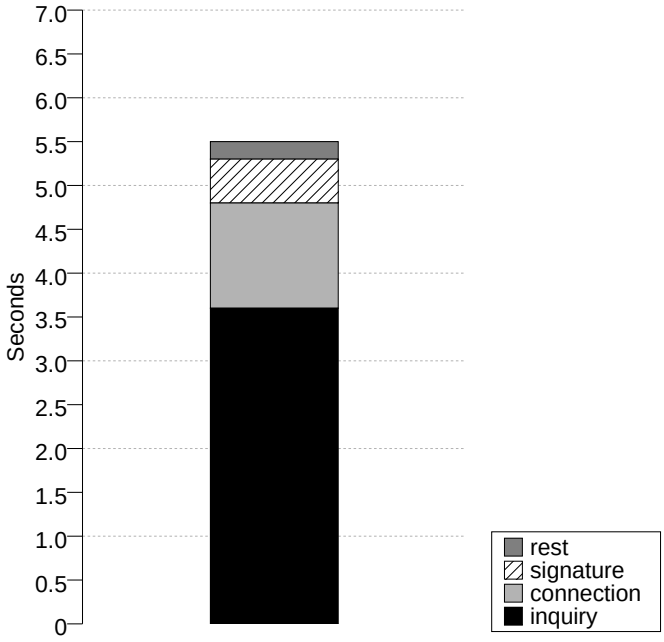

Figure 4: The average time usage of the different parts of the application.

5.4 Summary

The tests confirmed our assumptions about the time and energy performance of Bluetooth and showed that our prototype implementation performed well and allows a door to be unlocked in 5.5 seconds. From an energy consumption point of view, the program cost is around 1000 mJ. This is the same cost as holding the pointer still at the screen for 2 seconds or having a phone conversation for 4 seconds. Or put differently, with a battery of 12960 J and the phone in idle mode with Bluetooth on (using 40 mW on average) it can do more than 10000 consecutive unlockings. These figures show that the application can run on the mobile phone with little influence on the lifetime of the battery.

6 Related Work

The different types of access control systems that exist to replace physical keys do not match the requirements of our scenario. Existing systems either rely on a wireless transceiver or a card. KEELOQ [15] is a solution based on a wireless transceiver. A remote keyless entry solution using KEELOQ is presented in [20]. Most cars have the possibility of remote unlocking. Some car manufactures even allow *keyless start and go* where the user simply needs to carry the key device to be able not just to open, but also start the car[4]. The problem with these kinds of solutions is that they are designed for a one-to-one situation, where one transceiver fits exactly one door.

Card-based access control systems are commonly used in large corporations and use a card equipped either with a magnetic stripe or a microchip. The latter can either be operated so that the users needs to insert the card into a reader, or *contactless* where the user just has hold the card near a reader (see [1] for an overview of the technology). Card-based microchip systems are normally passive in the sense that the card itself is activated by and receives power from the reader. It is not capable of functioning autonomously. These systems are typically meant to be used in the domain of one company, and are normally designed so that each lock device is connected to a central access control server.

We have only found one product that use Bluetooth-based mobile devices for access control purposes. XyLoc[5] is a commercial key-less authentication product that enables automatic authorization, designed for Windows-based PCs and requires connection to a central access control server. Bluetooth is an optional communication technology that can replace their normal proprietary system. We have also found a case study using Bluetooth-enabled mobile phones to control access for *gates*, which is not specified further[6]. Unfortunately only marketing information is available.

7 Conclusion

This paper presented a digital key system that allows a Bluetooth-enabled personal server to unlock a door without user interaction. The design fulfills our requirements in terms of function-

[4] F.x. the Nissan Micra (`http://www.nissan-micra.com/`) or the Mercedes S-class (`http://www.mercedes-benz.com/`).

[5] `http://www.ensuretech.com/`.

[6] `http://www.codeisland.com/studies/studies.dvd.asp`.

alities (no user interaction is needed to open a door, lock devices are autonomous) and in terms of performance (a door can be opened in around 5 seconds without a major impact on battery lifetime).
The mobile phone we used as a personal server for our prototype provided most of the functionalities we needed. We missed some cryptographic capabilities (support for RSA and ECDSA). More importantly, Bluetooth is not ideal for such a digital key system. First, Bluetooth connections are expensive in terms of energy consumption (more expensive than a conversation over GSM). Second, the device discovery procedure is slow. Bluetooth might not be the prefered standard for future generations of personal servers providing a wide range of functionalities including digital keys.

References

[1] Smart Card Allicance. Contactless technology for secure physical access: Technology and standards choices. Technical report, Smart Card Alliance, 2002. URL `http://www.smartcardalliance.org/`.

[2] Daniel J. Barrett and Richard Silverman. *SSH, The Secure Shell: The Definitive Guide*. O'Reilly & Associate, 1st edition, 2001.

[3] Allan Beaufour. Secure access control using mobile bluetooth devices. Master's thesis, Department of Computer Science, University of Copenhagen, 2003.

[4] Allan Beaufour, Martin Leopold, and Philippe Bonnet. Smart-tag based data dissemination. In *First ACM International Workshop on Wireless Sensor Networks and Applications WSNA02*, June 2002.

[5] *Specification of the Bluetooth System – Core*. Bluetooth SIG, 1.1 edition, February 2001.

[6] Michael Brown, Donny Cheung, Darrel Hankerson, Julio Lopez Hernandez, Michael Kirkup, and Alfred Menezes. PGP in constrained wireless devices. In *Security Symposium*. USENIX, 2000.

[7] T. Dierks and C. Allen. RFC2246 – the TLS protocol version 1.0. Technical report, RFC Editor, Januar 1999.

[8] Sony Ericsson. P800/P802 white paper. Technical report, Sony Ericsson, 2003.

[9] ETSI. Alphabets and language-specific information. Technical report, European Telecommunications Standards Institute, 1998. ETSI TS 100 900 V7.2.0.

[10] ETSI. Technical realisation of the short message service (SMS). Technical report, European Telecommunications Standards Institute, 1998. ETSI TS 100 901 V7.2.0.

[11] Simon L. Garfinkel. *PGP: Pretty Good Privacy*. O'Reilly & Associates, Inc., 1994.

[12] O. Kasten and M. Langheinrich. First experiences with bluetooth in the smart-its distributed sensor network. In *Workshop on Ubiquitous Computing and Communications, PACT 2001*, 2001.

[13] HP Cambridge Research Lab. Personal server project. URL `http://www.crl.hpl.hp.com/projects/personalserver/`.

[14] Martin Leopold. Evaluation of bluetooth communication: Simulation and experiments. Technical Report 02/03, Institute of Computer Science, University of Copenhagen, 2002.

[15] Microchip. KEELOQ authentication products, 2002. URL `http://www.microchip.com/1010/pline/security/index.htm`.

[16] NIST. Digital sinature standard (DSS). Technical Report fips186-2, National Institute of Standards and Technology, 2000.

[17] NIST. Recommendation for key management. part 1: General guideline. Special Publication 800-57, National Institute of Standards and Technology, January 2003. URL `http://csrc.nist.gov/CryptoToolkit/tkkeymgmt.html`. Draft.

[18] R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 1978.

[19] Bruce Schneier. *Applied Cryptography*. John Wiley & Sons Ltd., 1996.

[20] Microchip Technologies. Remote keyless entry and convenience center reference design with LIN bus interface, 2002. URL `http://www.microchip.com/`.

[21] R. Thayer, N.Doraswami, and R. Glenn. RFC2411 –IP security: Document roadmap. Technical report, RFC Editor, November 1998.

[22] Roy Want. New horizons for mobile computing. Keynote speech at PerCom 2003, 2003. URL `http://www.percom.org/percom_2003/slides/`.