Towards an Open Source Toolkit for Ubiquitous Device Authentication

Rene Mayrhofer
Lancaster University
Computing Department
Infolab21, Lancaster LA1 4WA, UK
rene@comp.lancs.ac.uk

Abstract

Most authentication protocols designed for ubiquitous computing environments try to solve the problem of intuitive, scalable, secure authentication of wireless communication. Due to the diversity of requirements, protocols tend to be implemented within specific research prototypes and can not be used easily in other applications. We propose to develop a common toolkit for ubiquitous device authentication to foster wide usability of research results. This paper outlines design goals and presents a first, freely available implementation.

1. Introduction

Needs for authentication in ubiquitous computing environments are as diverse as the applications. They may need to authenticate users or other devices, either after an identification within some system-wide naming scheme, pseudonomously, or sometimes even anonymously, with the communication partners being identified by no more than their network addresses. Anticipating the growing importance of personal devices like mobile phones in interacting with the environment, we focus on *device-to-device* communication during spontaneous interaction. This kind of authentication, either with identified subjects or anonymously, provides a good compromise between security, scalability, and privacy. A user can authenticate to her/his personal device the moment it is activated, e.g. with passwords or biometric schemes, and then use this trusted device to interact with others. As such ad-hoc encounters are expected to be numerous throughout the day, authentication needs to be handled in an efficient manner, and be as unobtrusive as possible.

Context based authentication, that is, authentication based on certain properties of the user or device context, promises efficient, secure, and most importantly intuitive authentication methods (see e.g. [4, 5, 6, 9, 10, 11, 12, 13, 14]). The last point is especially important, because security measures are frequently disabled if they are not usable enough but get in the way of user's daily jobs. One example for using context is authentication based on spatial reference: by using a positioning system, devices can measure their spatial positions relative to each other. A visualised map can be used select devices to interact with. This is intuitive, and users can directly verify the spatial relationship, in contrast to purely wireless communication. We can construct secure device-to-device authentication out of this intuitive user interaction by coupling wireless communication with spatial sensing, using well-known cryptographic primitives.

However, context based authentication is a new research topic, and implementations of such protocols thus tend to be very application-specific. Currently, authenticating wireless communication in ubiquitous computing applications needs to be designed and implemented for each application. More often than not, it is therefore left out of research prototypes, to be "added later", because security is hardly the research focus of many of these projects. But practical experience shows, and handbooks insistently suggest [7], that security needs to be a requirement from the start; fitting it later on top of an existing system does not work in most instances. Re-usable hard- and software components are required that can be treated as black boxes from the developer's point of view to make it easier for applications to benefit from the advantages of context based authentication.

In this paper, we define our design goals for a ubiquitous authentication software toolkit and present a first implementation. The main contribution of this toolkit is the combination of cryptographic protocols with sensor data to create context based authentication. It provides lower-level primitives and higher-level context authentication protocols in the same way as the OpenSSL toolkit [1] provides cryptographic algorithms and an SSL/TLS implementation. Our approach is to rely on simple, standardised, off-the-shelf and therefore cheap sensors and provide software components to use them for authentication purposes.

This toolkit is part of ongoing research and will be extended to include new developments. Three projects for device-to-device authentication with different sensors already make use of it, and we expect more projects to follow.

Section 2 analyses requirements for such a toolkit, followed by a brief comparison of different software platforms for its implementation in section 3. Section 4 gives an overview of the current implementation and its availability, and in section 5 we provide a brief overview of projects making use of it.

2. Design goals

The main functional requirement for the authentication toolkit is to provide methods for creating shared secrets between two (or multiple) devices. These secrets should be authenticated to prevent man-in-the-middle (MITM) attacks. Authentication between personal devices and the environment is difficult, because such devices are small, mobile, and typically have limited resources. The absence of large screens and efficient input devices makes authentication based on sensor information even more attractive. Different applications require different sensor modalities for interaction as well as different levels of security. Therefore, a *toolkit*, i.e. a collection of loosely coupled components, is better suited to fulfil those different needs than a *framework* that implements the complete program flow and offers only defined hooks for application behaviour. For adding authentication to applications, it is easier to select and combine provided components than to make the application fit a pre-defined structure. Following this general design choice, we identify more detailed non-functional requirements. The toolkit should be:

- *lightweight*: Resources on mobile, battery-powered devices are generally sparse. This includes storage and run-time memory, CPU, communication bandwidth, but also battery lifetime, input/output devices, and user attention. A toolkit should be as small as reasonably possible, use static memory buffers when possible, and minimise communication. These aims are conflicting, and when no generally acceptable compromise can be found in some case, the respective components should be parameterisable for application developers.

- *self-contained*: Devices and platforms where the toolkit might be used are expected to be extremely diverse. Therefore, we can not depend on specific libraries to be available. Any dependencies that are not included in the default platforms should be included in the toolkit.

- *simple to use*: An authentication toolkit is most useful if it can be used without great care on the side of application developers. This has two reasons: if it is too complex to learn, developers will not use it for simple applications, and if it is complex to use, it is likely that it will be used erroneously and thus insecurely. Ideally, the various components of the toolkit can be used as black boxes with simple interfaces, and can be combined with each other and with application-specific hooks to build secure context authentication protocols without knowing about the internals. We therefore explicitly minimise the number of exported options, and focus on reasonable default values and automatic parameterisation whenever possible.

- *extensible*: It is obvious that a toolkit should be easily extensible by additional components. When designing it as a collection of related and compatible, but separable components, this goal should be automatically fulfilled, in contrast to framework-type design structures where extensibility must be explicitly considered.

- *vertical*: As context based authentication concerns all layers from sensing hardware, input/output devices, networking, application context, up to user interaction, a toolkit should provide components that span the layers. High-level components that relate to complete use-cases can make use of primitives from various lower-level layers.

- *interoperable*: Ubiquitous computing environments are inherently heterogeneous. Authentication protocols therefore need to be interoperable between different platforms. Thus, network communication should either be based on standardised protocols (e.g. IETF RFCs) or use simple ASCII line based protocols in the spirit of SMTP, HTTP, and others.

Implementation issues are mainly that the toolkit must be secure and that it needs to work asynchronously. Making a system secure is hard to achieve in the general case, because the security of a system depends on all of its components. The weakest parts will most likely be outside the toolkit. Nonetheless, the toolkit itself should be written carefully to protect against known and mitigate future attacks. This includes systematic protection against overflow attacks, pre- and post-condition checks for all methods, sanity checks for internal consistency, and wiping cryptographic key material from memory as soon as it is no longer required (see e.g. [7] for a more detailed introduction into the topic of secure programming). Especially the last point can be tricky to implement with run-time platforms like a JVM (Java virtual machine) or a .NET CLR (common language runtime). In addition to these standard best practises, “defensive” programming techniques demand checking every input value syntactically, semantically, and for the context/state in which it is read. This includes validating received network packets and

direct user input, but, in contrast to typical desktop applications, also sensor values, which might be tampered with as well.

The second issue is to deal with asynchronous program flows. An event based structure has two advantages over the standard blocking procedure calls: First, authentication methods are likely to cause noticeable delays while engaging in wireless communication or waiting for user or sensor input. Instead of forcing all applications to deal with this issue, it is simpler to provide asynchronous callbacks for all actions that delay a program flow. Second, reacting to events and switching hardware components to standby or sleeping modes in between can be used as an effective way of saving battery power (see e.g. the design of TinyOS [3]).

Finally, a toolkit is most useful when it is released under a license liberal enough to allow linking in all cases, both for other open source and for proprietary, closed source applications. The GNU General Public License (GPL) is problematic in this respect, because it does not allow linking with components that are not released under the same license.

3. Platform choices

As already mentioned, we expect a very heterogeneous range of devices to make use of context authentication. Consequently, no single software platform can be the base for supporting all of these systems. We briefly evaluate the most common platforms that promise to be supported on a large range of devices.

Java The Java platform, i.e. the programming language and the virtual machine (JVM), is becoming increasingly well supported by off-the-shelf devices. This includes laptop/desktop systems, PDAs, many newer mobile phones, industrial and embedded devices, and some consumer devices like upcoming Blue-ray players. Most of the resource limited devices do not provide the complete runtime library and language features, but only a subset called J2ME. From a security point of view, there are advantages and disadvantages: on the one hand, memory management embedded into the runtime makes buffer overflows harder to trigger, but at the same time it allows less control in terms of managing key material in memory.

.NET With similar goals to Java, .NET provides its own equivalent to the virtual machine (the CLR) and runtime components with memory management, but with the benefit of supporting multiple languages. It is supported on laptop/desktop systems and less commonly on PDAs and mobile phones. The .NET runtime seems to have neither particular advantages nor disadvantages over Java in terms of security.

C++ This is not a platform in the same sense as Java and .NET, but libraries like Boost allow cross-platform development. The main use of C++ in our target range of devices is Symbian OS, which is currently used on the majority of reasonably powerful mobile phones. C++ allows in-depth control of memory management and supports automatic wiping of key material by destructors and stack unwinding. It is also regarded as more lightweight than both Java and .NET, especially in terms of run-time memory consumption. However, experience shows that buffer overflows are a major issue with C and C++.

TinyOS Finally, many sensor nodes used in research projects run TinyOS [3] as their platform. It uses a dialect of C as its language, and compiles the whole firmware into one statically linked binary. Owing to the restricted CPUs that most sensor nodes use, TinyOS does not support dynamic memory management, and is thus expected to have slightly fewer vectors for buffer overflow attacks. On the other hand, cryptographic operations are expensive in terms of CPU consumption and battery lifetime, so programming on TinyOS poses different challenges than the other platforms.

The toolkit as a collection of algorithms is necessarily dependent on the target platform. Ideally, the toolkit should be ported to all above platforms, because all are widely used for devices we envisage to make use of context authentication. Authentication protocols should be interoperable between the different platforms, even if implementations differ. The key point is that the components should look the same to application developers, abstracting from possibly significant platform differences.

At the moment, we concentrate on Java with J2ME compatibility to support many mobile platforms in the first release. A port to .NET should include the complete functionality, while implementations for Symbian OS and TinyOS might use only selected components due to resource constraints.

4. Current implementation

Our current implementation contains components on the layers of cryptographic primitives, key agreement and authentication protocols, secure channels, and dealing with sensor data. Fig. 1 gives an overview of the dependencies between the components in different layers. These layers include the following specific components at the time of this writing:

Cryptographic primitives Implementations of ciphers, secure hashes, etc. are widely available, even as part

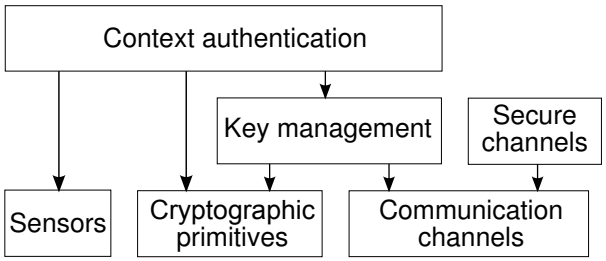

Figure 1. Interactions between components of different layers of the toolkit

of newer Java 2 runtimes (generally starting with version 1.4). However, this so-called Java cryptography extension (JCE) has not yet been included in the J2ME standard that is supported by most Java implementations on small, mobile devices. To fulfil the goal of being self-contained, the toolkit therefore uses wrappers around the JCE algorithms where necessary, and includes alternative implementations from the Bouncycastle cryptographic library [2]. When JCE is not supported on a target platform, the toolkit can use these as a fallback, albeit typically with slightly worse performance due to missing native implementations. Classes from Bouncycastle are also used to augment JCE primitives where they lack higher-level support.

On top of these primitives, we add small wrapper and utility classes that make the underlying algorithms as simple to use as possible. One notable example is a class to create X.509 certificates on-the-fly, for the purpose of using standard protocols like TLS.

Communication channels Java already offers good support for working with TCP or UDP connections. The toolkit again adds utility classes for simplifying setup and use of these protocols for standard cases, e.g. a threaded TCP server that listens in the background and starts key agreement protocols upon connection, or UDP multicast sockets for point-to-multipoint authentication protocols. These classes generally take care of low level details like binding to every network interface that has been found on the device without showing this complexity, to fulfil the goal of being simple to use.

Key management protocols This layer uses cryptographic primitives and communication channels to provide simple key agreement protocols. One example is a standard, unauthenticated Diffie-Hellman (DH) key agreement over TCP connections with an ASCII based protocol. Another protocol is just being created, and will interactively create cryptographic key material from sensor data streams, with either UDP multicast or Bluetooth communication. Additional candidates for this layer that have recently been proposed are the MANA family of protocols [8], a variant proposed by Wong and Stajano [20], and SAS [18]. Key management protocols for multi-party settings like the one proposed by Wacker et al. [19] and trust delegation protocols like the token based approach by Steffen and Knorr [17] also belong to this layer of components. We expect some of these and other protocols to be added to the toolkit in the future, with a common basic structure to make them simple to combine or exchange.

Sensors and feature extractors Dealing with sensor data is an important part for context authentication; this includes interfacing to the hardware sensors for data acquisition, handling of time series, and extracting appropriate features. The toolkit focuses on ease of use and automatic parametrisation, but exposes the parameters to applications when reasonable defaults can not be set. Currently, we provide base classes for reading data from ASCII based sources, simple Bluetooth RFCOMM channel access via the JSR82 API, computing time series statistics, time series aggregation, and segmenting time series based on a simple activity detection. An FFT implementation and a quantizer support feature extraction. All of these classes are optimised for real-time processing on resource limited devices.

Context authentication protocols Components on this layer tie together key agreement and authentication based on sensor data to create context authentication protocols. The result of a successful execution of one of these protocols is an authenticated secret shared key that can be used by applications. Protocols for authentication based on spatial reference and on common motion patterns are already available in the toolkit.

Secure channels The last layer implements secure communication channels, preferably based on standard protocols. These protocols generally depend on either trusted third parties, which can not be realistically assumed for ubiquitous computing environments, or shared secrets, as e.g. generated by components of the context authentication protocols layer. Currently, the toolkit provides wrappers for managing IPSec channels with operating system support. This has been implemented for Linux (with FreeS/WAN, Openswan, strongSwan, or racoon), Windows 2000/XP, and Mac OS X (with racoon), either using X.509 certificates or PSKs (pre shared keys). We also intend to provide a secure channel implementation based on TLS-PSK as an alternative to IPSec.

All key agreement and authentication protocols are event based and executed in the background. Three types of events can be generated: success (with the resulting shared secret key embedded into the event), failure (with a message giving reason for the failure), and progress (optionally with indication of how many steps have been finished and how many are left). These events can be used to provide user feedback in applications.

The Log4j framework is used for run-time configurable logging and JUnit for an extensive set of unit tests. Test cases cover single components as well as combinations spanning multiple layers, and special tests including real-world data samples for the context authentication protocols. Additional utility classes are used for defensive and fail-safe programming, like a "safety belt" timer used to terminate authentication protocols after timeouts.

Developers can use components simply by adding the single JAR file (or only the required components if program size is an issue) and using the provided classes. Applications only need to implement a single interface to process standard authentication events as described above. Although applications are free to use components from all layers, the two top layers are expected to be the most useful: *context authentication protocols* provide secret, authenticated key material which can then be used by *secure channels* components for securing the actual communication between devices.

Extending the toolkit with new components is similarly simple: there is no central structure that needs to be followed for every part, nor are there main interfaces that must be implemented. The toolkit provides some basic infrastructure, and extensions are free to use it. Its design loosely follows the principle of UNIX command line tools: to combine components with simple interfaces into more complex parts. When no suitable context authentication protocol is available for a specific applications, then the more basic layers should help in constructing it, ideally also adding it as a new component to the open source toolkit.

5. Initial projects using the toolkit

We currently use the toolkit in three applications that make use of context authentication:

- Our first application authenticates WLAN clients by spatial reference to set up IPSec channels [14]. The protocol for authenticating relative device positions with ultrasonic pulses [16] was the first complete context authentication protocol to be implemented within the toolkit. Additionally, this application creates X.509 certificates on the fly and configures the operating system IPSec support by using components of the lower layers of the toolkit.

- The second application authenticates devices based on common movement by comparing accelerometer time series [15]. It uses the same building blocks of the key management layer as the first application, namely a standard DH key agreement over TCP with an interlock protocol to prevent MITM attacks while exchanging the time series. This application triggered improvements in the sensor layer, as it requires real-time computation of features on the accelerometer data streams in time and frequency domain. As a more efficient alternative to the DH/interlock protocol combination, we are currently working on a novel protocol to create key material from sensor data streams using only symmetric cryptographic primitives.

- In the third project, visible light pulses are used to transmit authentic messages between devices with direct line of sight. This application again uses the DH and interlock primitives, but due to one-way "transmission" over this out-of-band channel, we are currently designing an alternative combination of these key management components. This is easily possible due to the loose coupling of the components.

These applications are mostly concerned with the "user interface" parts, while the protocols, hardware access, and internal management functions are provided by the toolkit. The first application is finished and available as an example with the current release, and new components developed for the other two applications will be added after they have received sufficient testing.

All components used in the first application (with the exception of the operating system IPSec channels, which rely on native libraries) run and are tested on an Asus MyPAL PocketPC with an IBM J9 JVM in addition to the standard desktop JVMs. In fact, the application scenario explicitly includes PDAs as devices that participate in authentication by spatial reference. First experiments show that most primitives and protocols also run on mobile phones with J2ME implementations. One of the current challenges is to consistently support sensor data access and wireless communication on different mobile phone platforms in the respective layers.

Our current experiences show that implementation of the cryptographic protocols was among the easier parts and, implemented for one application, it was in practise simple to reuse for others. It was more difficult and time consuming to interface with and use sensor data, e.g. to find appropriate feature extraction algorithms. This shows an important difference to the typical (and better understood) usage of sensor data: for context authentication, we do not need good separation between different classes, but high entropy from an attacker's point of view. Therefore, we expect the toolkit to be of particular value in this area, as well as in up-

per layers that tie together sensor data with cryptographic protocols.

6. Conclusions and future outlook

The toolkit is a collection of cryptographic primitives, key management protocols, wrappers for dealing with sensor data, and high-level context authentication protocols. Because of its design, it is easy to use and extend. Work on this toolkit has been ongoing for over a year. Some of its components have initially been developed for specific applications that make use of context authentication. After splitting them out and generalising them, they were integrated into this toolkit. The version available at the time of this writing is an alpha release that lays groundwork for implementing a growing set of protocols. Its lightweight, event based structure has proven useful in the projects that triggered the development of this toolkit as well as in further research projects currently under development.

Our choice of Java as the first implementation platform and a liberal open source license should allow the toolkit to be used on a large range of devices. We invite researchers working on security in ubiquitous computing to contribute their proposals to a common collection, so that application developers can easily evaluate and use them.

A first alpha release of the toolkit is available at `http://www.openuat.org` under the terms of the GNU Lesser General Public License (LGPL). This allows linking with proprietary and closed source applications, but guarantees that changes and additions to the toolkit itself will remain open source.

7. Acknowledgements

We gratefully acknowledge support by the Commission of the European Union under contract 013790 "RELATE" and the FP6 Marie Curie Intra-European Fellowship program contract MEIF-CT-2006-042194 "CAPER", and by the Engineering and Physical Sciences Research Council in the UK under grant GR/S77097/01. We thank Gerd Kortuem and two anonymous reviewers for comments on an earlier draft.

References

[1] OpenSSL: The Open Source toolkit for SSL/TLS web page. `http://www.openssl.org`, 2006.

[2] The Legion of the Bouncy Castle web page. `http://www.bouncycastle.org`, 2006.

[3] TinyOS web page. `http://www.tinyos.net`, 2006.

[4] D. Balfanz, G. Durfee, R. E. Grinter, D. K. Smetters, and P. Stewart. Network-in-a-box: How to set up a secure wireless network in under a minute. In *Proc. 13th USENIX Security Symposium*, pages 207–222. USENIX, August 2004.

[5] J. E. Bardram, R. E. Kjær, and M. Ø. Pedersen. Context-aware user authentication - supporting proximity-based login in pervasive computing. In *Proc. UbiComp 2003: 5th Int. Conf.*, pages 107–123. Springer, October 2003.

[6] S. Creese, M. Goldsmith, R. Harrison, B. Roscoe, P. Whittaker, and I. Zakiuddin. Exploiting empirical engagement in authenticated protocol design. In *Proc. SPC 2005: 2nd Int. Conf. on Security in Pervasive Computing*, pages 119–133. Springer, April 2005.

[7] N. Ferguson and B. Schneier. *Practical Cryptography*. Wiley Publishing, 2003.

[8] C. Gehrmann, C. J. Mitchell, and K. Nyberg. Manual authentication for wireless devices. *RSA Cryptobytes*, 7(1):29–37, 2004.

[9] M. T. Goodrich, M. Sirivianos, J. Solis, G. Tsudik, and E. Uzun. Loud and clear: Human verifiable authentication based on audio. In *Proc. ICDCS 2006: 26th Conf. on Distributed Computing Systems*, page 10. IEEE, July 2006.

[10] M. K. R. Jonathan M. McCune, Adrian Perrig. Seeing-is-believing: Using camera phones for human verifiable authentication. Technical report, CMU, November 2004.

[11] T. Kindberg and K. Zhang. Validating and securing spontaneous associations between wireless devices. In *Proc. ISC'03: 6th Information Security Conf.*, pages 44–53. Springer, October 2003.

[12] T. Kindberg, K. Zhang, and S. H. Im. Evidently secure device associations. Technical Report HPL-2005-40, HP Laboratories Bristol, March 2005.

[13] T. Kindberg, K. Zhang, and N. Shankar. Context authentication using constrained channels. In *Proc. WMCSA: 4th IEEE Workshop on Mobile Computing Systems and Applications*, pages 14–21. IEEE Computer Society, June 2002.

[14] R. Mayrhofer. A context authentication proxy for IPSec using spatial reference. In *Proc. TwUC 2006: 1st Int. Workshop on Trustworthy Ubiquitous Computing*, pages 449–462. Austrian Computer Society (OCG), December 2006.

[15] R. Mayrhofer and H. Gellersen. Shake well before use: Authentication based on accelerometer data. accepted for publication at Pervasive 2007, *to appear*.

[16] R. Mayrhofer, H. Gellersen, and M. Hazas. An authentication protocol using ultrasonic ranging. Technical Report COMP-002-2006, Lancaster University, October 2006.

[17] R. Steffen and R. Knorr. A trust based delegation system for managing access control. In *Advances in Pervasive Computing: Adjunct Proc. Pervasive 2005*, volume 191, pages 1–5. Austrian Computer Society (OCG), April 2005.

[18] S. Vaudenay. Secure communications over insecure channels based on short authenticated strings. In *Advances in Cryptology - Proc. CRYPTO 2005: 25th Int. Cryptology Conf.* Springer, August 2005.

[19] A. Wacker, T. Heiber, H. Cermann, and P. Marron. A fault-tolerant key-distribution scheme for securing wireless ad-hoc networks. In *Proc. Pervasive 2004: 2nd Int. Conf. on Pervasive Computing*, pages 194–212. Springer, April 2004.

[20] F.-L. Wong and F. Stajano. Multi-channel protocols. In *Proc. Security Protocols Workshop 2005*. Springer, 2006.