OpenUAT: The Open Source Ubiquitous Authentication Toolkit

Rene Mayrhofer
University of Vienna
Dr.-Karl-Renner-Ring 1, A-1010 Wien, AT
rene@mayrhofer.eu.org

Iulia Ion
ETH Zurich
Haldeneggsteig 4, 8092 Zurich, Switzerland
iulia.ion@inf.ethz.ch

Abstract

Authenticating spontaneous interactions between devices and users is challenging for several reasons: the wireless (and therefore invisible) nature of device communication, the heterogeneous nature of devices and lack of appropriate user interfaces in mobile devices, and the requirement for unobtrusive user interaction. The most promising approach that has been proposed in literature involves the exploitation of so-called auxiliary channels for authentication to bridge the gap between usability and security. This concept has spawned the independent development of various authentication methods and research prototypes, that, unfortunately, remain hard to compare and interchange and are rarely available to potential application developers. We built a system which implements and unifies these approaches. In this paper, we present OpenUAT, an open source toolkit that implements our novel, unified cryptographic authentication protocol (UACAP), and a comprehensive range of specific auxiliary channels. We evaluated OpenUAT based on a user study in which we compared four authentication methods implemented by the toolkit. The user study showed that users tend to prefer the visual channel in spite of its comparatively poor performance.

1   Introduction

Security in Ubiquitous Computing is currently a hot topic; as many research projects mature and their core findings start to influence real-world applications, non-functional requirements become increasingly more important. Security is one of the most important of these non-functional requirements and is a prerequisite to wide deployment.[1] Using standard cryptographic approaches, many security requirements – for example confidentiality, integrity, non-repudiability, auditability, or access control – can be fulfilled once all involved parties have been successfully authenticated. *Authentication* is therefore required to secure any interaction. Within the vision of ubiquitous computing, this is a particularly challenging task mostly due to three main reasons: (1) wireless communication channels are insecure, (2) many devices lack sufficiently capable user interfaces, and (3) user attention does not scale.

These problems have, over the past years, spawned the development of different authentication methods for specific application areas (e.g. [23, 10, 28, 20, 29, 33, 21, 22, 24]). However, most of these efforts are still separate and thus difficult to compare and not interchangeable. Actual implementations are often unavailable or otherwise restricted to specific, prototypical demonstration applications. This hinders both additional research on authentication methods for ubiquitous computing and application developers using those that have already been suggested. We therefore need a unified basis for comparable and interchangeable protocols and a library of ready-made implementations aimed at in-production deployment. To this

---

[1]Unfortunately, this is often neglected and product manufacturers as well as standardization bodies often try to retrofit security measures onto otherwise fixed projects. The recent track record shows that this procedure is clearly unsuccessful in producing secure systems.

end, we previously suggested to develop an open source toolkit implementing some of these methods with a common structure [19].

We use the following application scenarios as motivating examples that illustrate different potential use-cases for our system. In all these cases, selecting the intended communication partner is a major issue, with potentially tens of different wireless networks and hundreds of unknown devices in these networks. Depending on the lifetime of the keys exchanged, we distinguish between *short-lived* associations (ephemeral "one-shot" keys, used just once) and *long-lived* pairings (keys are stored and reused for future communication between the devices).

1. *Exchanging vCards and PGP keys:* Alice and Bob meet at a conference and wish to exchange contact information (vCards) and PGP keys for future remote communication. This exchange is short-lived, as their mobile phones are unlikely to directly communicate again. A specific issue is spontaneous authentication with severely limited user interfaces and highly personal devices that users might not wish to hand over.

2. *Printing a confidential document on a Bluetooth printer:* Alice is in the airport and wishes to use the (partially) trusted printer in her waiting lounge to print multiple parts of a confidential report saved on her phone. Keys may be reused for printing separate documents after initial establishment. A specific issue is selecting the "correct" printer in a list of similar ones.

3. *Connecting a mobile phone to a Wi-Fi router:* While visiting his friend's house, Bob wishes to check his email. He connects his smart phone to his friend's wireless LAN. Keys may be long-lived to be reused on future visits, but authorization may be limited to the actual visits. A specific issue is that the wireless access point might not be directly accessible.

4. *Temporarily pairing a mobile phone and Bluetooth headset:* Alice wishes to talk on the phone while driving and borrows a Bluetooth headset to make a single call. Keys are short-term, because she returns the headset after use. A specific issues is that a headset typically has no user interface besides a single button (and audio).

5. *Permanently pairing a mobile phone and Bluetooth headset:* Alice wishes to listen to music using her mobile phone and a stereo Bluetooth headset.

In this paper, we present a system that can be used to securely pair heterogeneous devices in situations like the ones presented above. The main contribution of the paper is *OpenUAT*, an open source toolkit that implements multiple auxiliary channels. By supporting a comprehensive set of different auxiliary channels, OpenUAT can adapt to different device capabilities (e.g. no display, no audio interface, limited input only), pairing context and environment (e.g. bright daylight, night, noisy environment, public place) and user preferences and capacities (e.g. users with handicaps). We present the OpenUAT implementation and its modular, extensible design in Section 4. The implemented auxiliary channels run on top of *UACAP*, a novel unified cryptographic protocol for device authentication, which brings an important amount of novelty and stands as a contribution of this paper. We introduce UACAP in Section 3. Finally, in Section 5 we conduct a dual evaluation of the system. Firstly, we analyze the security of UACAP, its underlying protocol. Secondly, we prove through an user study how OpenUAT can be used to easily select different auxiliary channels for pairing and act as a framework for usability studies.

2 Related Work

Over the last few years, different cryptographic protocols for multi-channel authentication have been proposed (e.g. [9, 17, 2, 11, 34, 5, 4, 2, 36]. Most of them have in common that they assume a main wireless

communication channel and a more restricted, so-called *auxiliary* or *out-of-band* channel. While the main channel has – in terms of cryptographic key exchange – practically unlimited bandwidth, the auxiliary channel is often limited to either short messages or slow and/or obtrusive transfer. Consequently, different cryptographic protocols have been developed to exploit these diverse characteristics for the purpose of secure authentication between devices, users, and services. However, this diversification of protocols means that they are not easily interchangeable and that security analysis need to be done for each of them. In the present paper, for the first time, we contribute a unified protocol that can exploit any combination of security guarantees from arbitrary auxiliary channels. UACAP is a unification of some of the recently proposed protocols, retaining their security properties with a minimal number of messages.

A considerable amount of prior work on using auxiliary channels to establish shared secret keys between two (or multiple) devices has been presented. The "resurrecting duckling" as a pairing model suggested direct electrical contact [30], while "constrained channels" [13] and "location-limited channels" [2] were proposed as more general models of auxiliary channels for authentication purposes. Specific auxiliary channels are *video* by using mobile phone cameras and 2D barcodes [23], blinking patterns [27], or laser channels [22], *audio* by comparing spoken sentences [10] or MIDI tunes [28], *ultrasound* [21], *motion* by common movement [20], gestures [24], or synchronised button presses [29], or *radio frequency* by measuring common environment [33].

Part of this research is slowly moving into products. The Bluetooth Simple Secure Pairing (SSP) [3] and the Wi-Fi Protected Setup (WPS) [35] already use some of the results on pairing protocols, although initial implementations will be limited to standard display and keypad entry methods. Apple intends to "make users lives easier by letting them pair wireless devices just by bringing them together".[2] Envisaged applications are pairing Bluetooth mice or keyboards, which currently use empty (and therefore insecure) passwords, with laptops or desktop PCs. However, these approaches are so far completely separate with different cryptographic protocols and implementations. OpenUAT aims to implement as many auxiliary channels as possible in a common structure so that they are comparable and interchangeable.

Usability is a major issue in the design of any secure system. Secure systems protocols may be used insecurely or not at all if users feel that they do not understand the underlying principles or if they find security measures obtrusive. Effective security therefore can only be achieved by giving special consideration to end users. In practice, mock-ups are rarely sufficient. Interesting user behavior and many more additional issues surface when user studies are done with functional (but maybe prototypical) applications. In [14], Kostiainen and Uzun propose a framework for comparative usability testing of distributed applications and in [32], Uzun et al. analyze usability of different pairing methods. Suomalainen et al. present a comparative analysis of security associations in personal networks [31]. However, such comparative studies are currently hindered by the lack of available implementations and remain, therefore, often limited to mock-ups or Wizard-of-Oz studies that can hardly discover issues in real-world deployment. In contrast, OpenUAT aims to provide usable implementations to support rapid application development and is, therefore, more specific than high-level frameworks concerned with usability.

Balfanz et al. [1] propose a system called "Network-in-a-box" that effectively reduces the time and task complexity of connecting a laptop or mobile phone to a wireless router, using a location limited auxiliary channel. This is a perfect example of applying research results in secure device pairing to improve the security and usability of real world systems. We take such research further and aim to target a broader application context. By implementing more auxiliary channels, our system aims to support a wider variety of devices and application scenarios.

In the area of peer-to-peer (P2P) networks, two platforms stand out: JXTA [15] for connecting and communicating with networked devices, and MyNet [12] for personal and social networks. Both JXTA

---

[2] http://www.appleinsider.com/articles/08/09/27/apple_seeks_distance_based_pairing_auto_contact_data_patents.html.

and MyNet assume unique, secure device identifiers and pre-established public or secret keys, and provide support for device authorization and fine-grained resource access control. Therefore, they remain complementary to the aims of OpenUAT, which could be used to bootstrap device pairing for both systems as a basis for further communication, thus coping with the dynamicity of the pervasive computing environment.

We designed OpenUAT to be a toolkit instead of a framework [19]; by providing a library of interconnected methods instead of forcing applications to conform to a specific pattern, toolkits may be easier to use in different scenarios. Other toolkits in the area of ubiquitous computing include the widely used "Context Toolkit" [26], the "Subtle" toolkit for determining interruptibility [8], and "Place Lab" for determining location [16]. The "OpenSSL" toolkit[3] is currently one of the most widely used cryptographic toolkits for applications written in various languages while the "Bouncy Castle" toolkit[4] implements these primitives and protocols specifically for Java. OpenUAT builds on top of these layers to provide authentication methods and secure channels, which in turn can be used by applications or application frameworks.

3 Unified Auxiliary Channel Authentication Protocol (UACAP)

UACAP, our *Unified Auxiliary Channel Authentication Protocol*, is based on the recent proposal by Laur and Nyberg called MA-DH [17], but adopts aspects of the MANA III variant described by Wong and Stajano [36] and an option for pre-authentication as suggested, among others, by Balfanz et al. [2]. Its main part relies on the well-known (Merkle-) Diffie-Hellman (DH) key agreement [6] with a prior one-way commitment to prevent the most basic man-in-the-middle (MITM) attacks. The result of this main phase is a secret session key shared between two devices. Then, to ascertain that the intended devices share this key and there really is no other device involved (by accidentally pairing with the wrong device or malicious MITM attack), the obtained key must to be authenticated using the properties of an auxiliary channel.

Depending on the application scenario and properties of the auxiliary channel, UACAP supports different modes of operation from a user point of view:

- **Input** channels allow the user to provide common input to all involved devices, for example by explicit PIN code entry (cf. [9]), synchronous button presses (cf. [29]), or shaking them together (cf. [20]). We need to further distinguish if the user input can be shielded from others or not:

    - **IN:** *Non-confidential* input must happen interactively during the protocol run.
    - **IC:** *Confidential* "pre-authentication" is possible before the main protocol part and with no further interaction on the auxiliary channel. This is the only case in which the auxiliary channel must be confidential; in all other protocol cases it is sufficient to be authentic.

- **Transfer** channels support direct, user-mediated (and ideally human-verifiable) transmission of messages, for example by capturing a 2D barcode displayed on one device with the camera of another (cf. [23]) or audible MIDI sequences (cf. [28]). We further distinguish according to the auxiliary channel bandwidth:

    - **TS:** *Short* transfer (10–60 Bits) must happen interactively during the protocol run.
    - **TL:** *Long* "pre-authentication" ($\geq$128 Bits) is possible before the main protocol part and with no further interaction on the auxiliary channel (non-interactive with respect to the auxiliary channel). This has the advantage that, by taking place before any communication on the main wireless channel, required addresses (for example MAC or IP addresses) may also be transmitted in the same pre-authentication message to support easy-to-use device selection methods.

---

[3] http://www.openssl.org
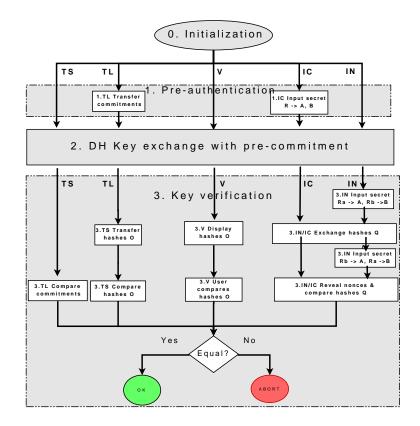[4] http://www.bouncycastle.org

Figure 1: UACAP overview: different options for key verification

- **Verification** channels allow the user to compare data from different devices, for example by reading non-sensical English sentences (cf. [10]), comparing random visual art (cf. [25]), or MIDI tunes (cf. [28]). We denote this option:

  - **V:** Explicit user verification can always use "short" bit strings in the range of 10–20 Bits over a public, non-confidential medium and must be done after the main protocol part.

These options depend on the choice of authentication mode (transfer, verify or input) and channel type (short or long, confidential or non-confidential) and may be predetermined by the respective application scenario (e.g. direct input is not possible when interacting with non-accessible infrastructure devices like large, distant screens). The main part of the protocol that is responsible for the actual key agreement and its cryptographic security is universal among all cases of authentication.

In the following, we describe UACAP on three levels: First, the overview in Fig. 1 depicts the logical flow of UACAP, pointing out the decision points and different options at different steps, until reaching the final decision; second, Fig. 2 presents a detailed specification on the level of cryptographic operations and variables exchanged in messages; third, we describe the current reference implementation in terms of on-the-wire message transfer and protocol commands in section 4.2.

Protocol Specification

In the protocol description in Fig. 2, the different options are indicated in the heading of specific steps and are only executed for the respective channel type. Mandatory parts are indicated with a grey background. Channels over which messages are transmitted are either the main wireless channel (RF), an authentic but

"short" (AS), authentic and "long" (AL), or an authentic and confidential but "short" (ACS) auxiliary channel. Note that all auxiliary channels must always provide authenticity, which is the primary reason for their use in authenticating device interaction.

When the (short) user input values R_a and R_b can be guaranteed to remain confidential until the protocol run finishes, then they may be provided to both devices at the same time and before even starting the protocol, and R_a and R_b may be equal. If R_a and R_b are only authentic but not confidential (and if they are "short" in terms of brute-force attacks), then they *must not* be made public until Step 3 in any case. At this time, they *must* be provided to the respective other device, but may be made fully public.

For the formal description in Figure 2, the following notation is used: $H(m)$ describes the hashing of message m with some secure hash, and $m|n$ the concatenation of strings m and n. Subscripts denote the different sides (a or b for an authentication between A and B). The notation $\widetilde{X}$ is used to point out that a variable X has been sent over an insecure channel and may therefore have been modified (by transmission error or malicious behavior). $\text{SHA}_{\text{DBL}}\text{-256}$ is used as a secure hash for H, which is a double execution of the standard SHA-256 message digest to safeguard against length extension and partial-message collision attacks [7] and is defined as $\text{SHA}_{\text{DBL}}\text{-256}\,(m) = \text{SHA-256}\,((\text{SHA-256}\,(m))\,|m)$. $Com(x)$ describes a cryptographic commitment and is also implemented as $Com(x) := \text{SHA}_{\text{DBL}}\text{-256}(x)$.

The protocol execution consists of several steps:

0. Initialize   This phase is common to all protocol options (TL, TS, V, IN, IC) and initializes the DH parameters (the base g is assumed to be publicly known) and identities of the involved parties. The (potentially ephemeral) identity of a party, for example their network address, is represented by I_a and I_b for parties A and B, respectively. In the current form, nonces (in combination with network addresses) are used to prevent replay attacks even when DH keys are not ephemeral.

1. Pre-authentication   In this phase, pre-authentication data is used before starting the actual key exchange protocol. This has the advantage that the actual protocol run can be non-interactive and thus even more unobtrusive to the user. It also means that the part requiring user involvement (transfer or input) can happen at any time before the associated devices start their wireless interaction. Pre-authentication is only supported in the following two cases:

- **TL**: Commitments are exchanged over the authentic channel. As these commitments must be at least 128 bits long to ensure adequate levels of security, the same is not applicable to the **TS** case, where only short messages can be exchanged. This step is depicted as *1.TL Transfer commitments*.

- **IC**: A common, short secret R is input to both devices on a confidential channel (*1.NC Input secret*). R *must* remain confidential until the protocol finishes. Such confidential user input can be sensor data such as the accelerometer time series resulting from shaking devices together, or the same password/PIN entered on all devices (and shielded from others).

2. Diffie-Hellman key exchange with pre-commitment   This part is mandatory and common to all protocol options, as indicated by the grey background in Figure 2. Optional parameters for specific protocol instances (for example the number of rounds of an interlock scheme as in the ultrasonic spatial authentication protocol [21]) can be transmitted from the initiator A to the responder B in plain text using the optional variable P_a. If not required, it may simply be omitted in the first message (indicated with the notation $[P_a]$). By waiting to initialize the responder's ephemeral DH key until after the initiator's commitment has been received, denial-of-service attacks become (marginally) harder.

Channel	A (initiator)	Message	B (responder)

0. INITIALIZE

	$I_a \in \{0 \ldots 2^{128}-1\}$		$I_b \in \{0 \ldots 2^{128}-1\}$
	$x \in \{1 \ldots q-1\}$		$y \in \{1 \ldots q-1\}$
	$X := g^x$		$Y := g^y$

[1.TL: TRANSFER COMMITMENTS]

| AL | $OC_X := Com(X)$ (≥ 128 Bits) | $\mathbf{M}_{0,a} := (I_a, OC_X)$ $\longrightarrow$ | |
| AL | | $\mathbf{M}_{0,b} := (I_b, OC_Y)$ $\longleftarrow$ | $OC_Y := Com(Y)$ (≥ 128 Bits) |

[1.IC INPUT SECRET]

| ACS | INPUT $R_a = R$ | | INPUT $R_b = R$ |

2. KEY EXCHANGE (DH) WITH PRE-COMMITMENT

RF	$C_X := Com(X)$	$\mathbf{M}_1 := (I_a, C_X[, P_a])$ $\longrightarrow$	
RF	$K_a := H(\widetilde{Y}^x)$	$\mathbf{M}_2 := (I_b, Y)$ $\longleftarrow$	
RF		$\mathbf{M}_3 := (X)$ $\longrightarrow$	if $Com(\widetilde{X}) \neq \widetilde{C}_X$ then ABORT else OK else $K_b := H(\widetilde{X}^y)$

3. OOB KEY VERIFICATION

	$OI_a := (I_a|\widetilde{I}_b)$		$OI_b := (\widetilde{I}_a|I_b)$
	$OK_a := (X|\widetilde{Y})$		$OK_b := (\widetilde{X}|Y)$
	$O_a := \mathrm{HMAC}_{OK_a}(OI_a)$		$O_b := \mathrm{HMAC}_{OK_b}(OI_b)$
	(ca. $10 \ldots 60$ Bits)		(ca. $10 \ldots 60$ Bits)

[3.TL: COMPARE COMMITMENTS]

| | $Com(\widetilde{Y}) \neq \widetilde{OC}_Y$ then ABORT else OK | | if $Com(\widetilde{X}) \neq \widetilde{OC}_X$ then ABORT else OK |

[3.V: DISPLAY HASHES O, USER VERIFY]

| AS | OUTPUT O_a | | OUTPUT O_b |
| AS | INPUT OK iff $O_a = O_b$ | | INPUT OK iff $O_a = O_b$ |

[3.TS: TRANSFER HASHES]

AS		$\mathbf{M}_{4,a} := O_a$ $\longrightarrow$	
			if $\widetilde{O}_a\, ! = O_b$ then ABORT else OUTPUT OK
AS	INPUT OK iff B did not ABORT (without transfer from B to A)		
AS		$\mathbf{M}_{4,b} := O_b$ $\longleftarrow$	
	if $\widetilde{O}_b\, ! = O_a$ then ABORT else OUTPUT OK		
AS			INPUT OK iff B did not ABORT (without transfer from A to B)

[3.IN: NON-CONF. INPUT]

| AS | INPUT R_a | | INPUT R_b |

3.IN/IC: EXCHANGE HASHES Q

	$J_a \in \{0 \ldots 2^{128}-1\}$		$J_b \in \{0 \ldots 2^{128}-1\}$
RF	$Q_a := \mathrm{HMAC}_{OK_a|J_a}(OI_a|R_a)$	$\mathbf{M}_{5,a} := (Q_a)$ $\longrightarrow$	$Q_b := \mathrm{HMAC}_{OK_b|J_b}(OI_b|R_b)$
RF		$\mathbf{M}_{5,b} := (Q_b)$ $\longleftarrow$	

[3.IN: NON-CONF. INPUT EXCHANGE]

| AS | INPUT R_b | | INPUT R_a |

[3.N/IC: REVEAL NONCES AND COMPARE]

RF		$\mathbf{M}_{6,a} := (J_a)$ $\longrightarrow$	
RF		$\mathbf{M}_{6,b} := (J_b)$ $\longleftarrow$	
	if $\mathrm{HMAC}_{OK_a|\widetilde{J}_b}(OI_a|R_b) \neq \widetilde{Q}_b$ then ABORT else OK		if $\mathrm{HMAC}_{OK_b|\widetilde{J}_a}(OI_b|R_a) \neq \widetilde{Q}_a$ then ABORT else OK

Figure 2: Unified Auxiliary Channel Authentication Protocol (UACAP) specification

3. Out of Band (OOB) key verification This first part initializes the components for the key verification while the actual OOB verification depends on the chosen protocol option. A double line indicates where the protocol part ends for the respective options.

- **TL**: Commitments of the DH parameters have been exchanged in the pre-authentication phase. For verification, it therefore suffices to compare those commitments with the locally computed ones.

- **V**: Verification hashes O_a and O_b are displayed to the user for comparison. This can be done either by showing MADLib sentences, displaying hexadecimal hash representation, or playing MIDI tunes, etc. The user inputs OK if hashes match, otherwise the protocol fails.

- **TS**: Hashes O_a and O_b are transferred over the authentic but short channel and subsequently compared with the local ones. If they match on both devices, the key exchange completes successfully.

- **IN/IC**: Only in the *IN* case, the user must first input R_a to device A and a different R_b to device B. Next, devices compute and exchange hashes Q_a and Q_b. Nonces J_a and J_b are used to protect against replay attacks. Steps *3.IN/IC* in Figure 2 are common to the input case, both for confidential and non-confidential subcases, and must be executed in order. For the confidential case *IC*, the common secret $R = R_a = R_b$ was input to both devices during the pre-authentication phase and no further input is required. Only in the *IN* case, the user now inputs R_b to device A and R_a to device B. Finally, nonces J_a and J_b are made public, and the hashes Q_a and Q_b are verified by A and B.

4   OpenUAT

OpenUAT aims to be an open-source, ready-to-use toolkit for authentication in ubiquitous computing applications. The methods and protocols it implements are selected and designed to be intuitive and usable for the end user, and the overall toolkit aims to be modular and compact for the developer. Most parts are implemented in Java and verified to work on most Java virtual machines (JVMs) including Java 2 Micro Edition (J2ME) as available on many off-the-shelf mobile devices. However, protocol implementations use ASCII commands whenever possible to ease interoperability with other platforms (as specified below in section 4.2). A central design pattern is to use asynchronous, background processing and event notification. This has the advantage that applications and their user interfaces are not blocked by potentially lengthy protocol runs and that events provide a general "hooking" mechanism to react to various stages of authentication. All core parts are documented using Javadoc and covered by JUnit tests to ensure API stability during code changes. In the following, we describe the main components, pointing out how the UACAP specification is currently implemented.

4.1   Structure

Figure 3 depicts the architecture and components of the OpenUAT toolkit from an application point of view. Three central components are especially noteworthy:

- `RemoteConnection` is an interface for in-band (main, RF) communication between devices and abstracts arbitrary communication channels. Currently implemented are TCP sockets and Bluetooth, and prototypes for communication via Jabber chat and HTTP servers already exist to avoid NAT and other communication issues. Other RF channels can be easily added by extending the abstract class `RemoteConnection`. For Bluetooth communication, additional helper classes provide peer device search, service registration, and service search (`BluetoothPeerManager`) as well as complete management of opportunistic key agreement to better deal with slow Bluetooth communication
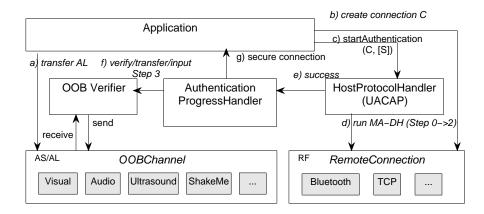
Figure 3: OpenUAT toolkit: Components and interactions

(`BluetoothOpportunisticConnector`). UDP multicast communication is supported separately for non-DH protocols (e.g. CKP [18]).

- `OOBChannel` is the corresponding interface for auxiliary channels besides the main, in-band channel. It is described in more detail in section 4.3.

- Steps 0 and 2 in UACAP are implemented by the central `HostProtocolHandler` class. Upon successful completion of the DH key exchange, it raises a corresponding event with the registered `AuthenticationProgressHandler`. The different modes explained above in section 3 can be pre-set before starting the protocol run: for "long" pre-authentication transfer, one side can fetch the respective message and transmit it over an auxiliary channel so that the other side can provide it to its `HostProtocolHandler` instance; for "short" confidential pre-authentication input, both sides can provide the same byte string. All other modes start in step 3, i.e. the out-of-band verification using different `OOBChannel` implementations depending on the verification mode (verify, input, or transfer) and on the type of auxiliary channel (see table 1). This step often involves user interaction/assistance (e.g. taking a picture of the barcode, shaking devices together, manual input) or decision making (e.g. comparing melodies, sentences, images).

Cryptographic primitives are used either from Java JSSE/JCE cryptographic extensions or, when not available (as is the case in J2ME), provided by the Bouncy Castle toolkit. OpenUAT currently wraps Diffie-Hellman key agreement (`SimpleKeyAgreement`), AES, SHA-256, and SHA_{DBL}-256 (`Hash`) and provides implementations of HMAC (`Hash`), interlock* (`InterlockProtocol`), and CKP (`Candidate-KeyProtocol` and `CKPoverUDP`). Sensor data acquisition is supported by standard statistical features such as moving average or windowed variance for floating point and integer computation (`TimeSeries` and `TimeSeries_Int`, respectively). Simple activity detection of multi-dimensional sensors based on variance thresholds (`TimeSeriesBundle`), quantization (`Quantizer`), floating and fixed point Fast Fourier Transform (`FFT` and `FPIntFFT`), and the coherence function (`Coherence`) build upon the basic time series for more complex feature extraction. Most of these are currently used for determining if devices are shaken together [20] but are applicable to arbitrary sensor data.

A key manager (`KeyManager`) assists in securely keeping track of pairwise keys with other devices and can be registered as `AuthenticationProgressHandler` to automatically receive keys created and authenticated by `HostProtocolHandler` and higher level protocols.

9

Additionally, logging is supported both on J2SE (based on *log4j*) and J2ME (using either *microlog* or *openbandy*) by using an abstract `Log` interface for all utility classes and applications.

4.2 UACAP Reference Implementation

In the current reference implementation of UACAP in OpenUAT, the following protocol commands are assumed to be executed in order. A client (or *initiator*) A connects to the server (or *responder*) B to start a protocol run. Then, following common Internet protocols, the server starts by sending its greeting:

1. "`HELO OpenUAT Authentication`", sent by B, indicates that the client may start its authentication request, i.e. that the (insecure, in-band) channel the client has connected to is connected to an OpenUAT instance.

2. "`AUTHREQ UACAP-1.0` I_a C_a `[PARAM` P...`]`" (M_1) is sent from A to B to transmit the client identity (e.g. its IP or Bluetooth MAC address or a nonce acting as an ephemeral identifier for a single interaction), its commitment and an optional parameter, which may be free-form and of arbitrary length. Note that the client requests key exchange and verification using a named protocol (currently `UACAP-1.0`) for interoperability with future versions.

3. "`AUTHACK` I_b Y" (M_2) is sent from B to A to respond with the server identity and the server (long-lived or ephemeral) public DH key part.

4. "`AUTHACK2` X" (M_3) is sent from A to B to finish the DH key agreement with the client public key part.

All variables are encoded depending on the chosen (in-band) RF channel. If it supports 8-bit data transmission (such as TCP sockets or Bluetooth RFCOMM), the variables are transmitted efficiently as byte arrays in standard network byte order (i.e. big-endian) with a prepended single byte specifying the array length. If the channel only supports character (e.g. ASCII) transmission (such as Jabber chat-messages as transport medium), all variables are Hex-encoded and sent as ASCII strings with space as delimiter.

The following protocol messages are dependent on the chosen authentication option. For the *input* case:

1. "`AUTHINPCOM` $O_{a/b}$" (M_5) provides the second commitment and is symmetrically sent by both A and B. After correctly receiving these messages, both devices should *exactly at this stage* (not earlier and not later) query for the user input that was previously provided to the other side.

2. "`AUTHINPOPEN` $J_{a/b}$" (M_6) is also sent symmetrically by both A and B to open the previous commitment and allow comparison of the user inputs.

For *transfer* and *verification*, other messages are transmitted over auxiliary channels and therefore with different format. The encoding and specific protocol message depends on the respective channel.

4.3 Auxiliary Channels Implementation

Currently, OpenUAT implements several auxiliary channels:

- **Ultrasound** is used for a verification of *Spatial References* using the interlock* protocol both on RF and ultrasonic channels [21].

- **Motion** is used to detect when two devices are shaken together, either in verification mode by exchanging accelerometer time series with an interlock* protocol or in input mode by creating keys directly out of sensor time series [20]. Accelerometer data acquisition has been implemented for

10

Table 1: OpenUAT: Currently implemented authentication methods

Channel/Mode	Input	Transfer	Verify
Visual	-	Barcode transfer	Compare sentences
			Manual string comparison
Audio		Audio transfer	Compare melodies
			Ultrasound
Motion	ShakeMe		ShakeMe
Keypad	BEDA		
	Manual keypad entry		

Spark Fun Electronics WiTilt sensors over Bluetooth, some on-main-board sensors (e.g. in Thinkpad and Macbook laptops), and for Symbian S60 (through a Python module which communicates with the Java MIDlet via a TCP socket) and some Windows Mobile phones (through a native C# implementation that communicates over TCP using the same format).

- **2D barcodes** [23, 27] are used to display the out-of-band messages on any screen and capture them with a mobile phone camera. QR codes are generated with an adapted embedded implementation, while we make use of the Google ZXing library [5] for decoding. The QR code includes 7 bits of O_a or O_b in hexadecimal notation.

- **Audio** as used in HAPADEP [28] is integrated into the OpenUAT code base. It was initially developed only for desktop systems, i.e. for Java 2 Standard Edition (J2SE) but subsequently ported to J2ME/MIDP and restructured to integrate with UACAP. In transfer mode, O_a or O_b is encoded with a "fast" codec and played as a wave file. The other device records the sound, decodes it, and verifies that $O_a = O_b$. In verify mode, applying the "slow" codec results in a piano-like melody, which should be more pleasant and easy to recognize by users.

- **Manual keypad entry** is used for "short" (confidential or non-confidential) input of R_a and R_b.

- **Manual string comparison** uses MADlib to create non-sensical sentences from the short messages O_a or O_b which are then compared by the user [10].

- **Synchronised button presses** are used in an implementation of BEDA [29] as another form of common input.

In addition to the basic MIDP libraries, we have used several optional APIs: JSR 82 for Bluetooth communication, JSR 75 for logging data to a file, JSR 135 (the Mobile Media API) for capturing audio and video input and JSR 234 (the Advanced Multimedia Supplements) which adds basic playback functionality for audio and video. We have successfully tested these channels on Nokia Series60 devices that implement all these optional JSRs (e.g. N95, N82, 5500) and on appropriately equipped laptops.

To extend OpenUAT with a new auxiliary (OOB) channel, developers should implement the `OOBChannel` interface and its public methods `receive` and `transmit` as well as the `OOBMessageHandler` interface with its method `handleOOBMessage`. When a message is received, the `OOBChannel` notifies the `OOBMessageHandler`, which then processes the message and completes the key verification step.

Full source code is available under the terms of the GNU Lesser General Public License (LGPL) at `http://www.openuat.org`.

---

[5]ZXing http://code.google.com/p/zxing/

5 Evaluation

We perform a dual evaluation of our system. First, we present a security analysis of UACAP to show that it prevents known attacks. Second, we conduct a user study using OpenUAT and several of its implemented out-of-band channels to show in practice that the toolkit can be used with different, easily interchangeable auxiliary channels in the same framework. Users' perceptions of the toolkit and their preference for using visual rather than audio channels provides interesting insights for application designers.

5.1 UACAP Security Analysis

In the following section, we analyze the security properties of UACAP. We prove its resistance to MITM, online attacks. Brute-force, offline attacks carried out in a passive manner are mitigated through the properties of the Diffie-Hellman key exchange parameters (the attacker cannot compute K if it knows X and Y but is not in possession of any of the secrets x and y. Furthermore, being based on previously verified protocols, UACAP carries over their security properties.

Transfer Long The protocol starts with pre-authentication of both parties, through the exchange of commitments on an authentic channel. To impersonate Alice (A), an attacker Eve would then have to find a variable z, such that $Com(X) = Com(Z)$, where $Z =: g^z$. Only then could Eve pretend to be Alice on the wireless, insecure channel and deliver $M_3 := (Z)$ to Bob (B) with successful comparison of commitments in step *3.TL*. However, due to the properties of the commitment function, which is implemented as $Com(x) := \text{SHA}_{\text{DBL}}\text{-}256(x)$ this is infeasible (the $\text{SHA}_{\text{DBL}}\text{-}256$ family of hash functions is currently considered to be resistant to pre-image collisions). The same holds true for the other case (impersonating the responder B).

The "pre-authentication" case was also previously described for SiB [23] and by Balfanz et al. [2]. Basically, because pre-authentication requires transfer of "long" messages that are effectively secure hashes of public DH key parts, authentication is secure as long as those hashes remain so (that is, no second pre-image can be found online during the protocol run).

Transfer Short Because the message transfer through the authentic channel is done at the end of the protocol, to be successful, an attacker Eve would have to find a variable z and identity I_c, such that $\text{HMAC}_{OK_a}(OI_a) = \text{HMAC}_{OK_c}(OI_c)$, where $OK_c := (X\|Z)$. Note that the session key K_a is not used during the verification phase. Ensuring that the correct variables X and Y have been transmitted is sufficient to ensure device authentication. Our design is consistent with the MANA family of protocols, which provide formal security proofs.

Basically, in comparison to the SAS [34] and DH-SC [4] protocols, MA-DH [17] and in extension UACAP only differ by their message order. However, SAS and MA-DH have already been shown to belong to the MANA IV family of protocols and therefore share their security proofs, while MA-DH is the optimal variant in terms of number of messages [17]. The main feature that UACAP adapts from the MANA IV family is the commitment prior to Diffie-Hellman key agreement to prevent *brute-force* attacks on short authentication strings. This one-way commitment to (ephemeral or static) public Diffie-Hellman key parts X and Y together with random nonces I_a and I_b used as session identifiers ensures that any MITM attack must be performed *online*. An adversary is reduced to either attacking DH keys once the commitment and key exchange has been completed (passive attack, which is currently assumed to be infeasible) or a single, one-off chance for fabricating the DH key parts in such a way that the (short) auxiliary messages will still match (active attack). Because of this protocol design element, all auxiliary messages besides the pre-authentication case may be "short" in terms of brute-force key search. With only 20 Bits, an adversary
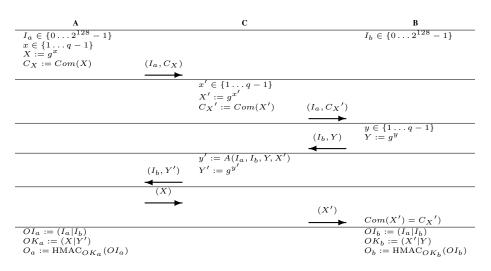
$I_a \in \{0 \dots 2^{128} - 1\}$ $I_b \in \{0 \dots 2^{128} - 1\}$

$x \in \{1 \dots q - 1\}$

$X := g^x$

$C_X := Com(X)$ $(I_a, C_X) \longrightarrow$

          $x' \in \{1 \dots q - 1\}$

          $X' := g^{x'}$

          $C_X' := Com(X')$ $(I_a, C_X') \longrightarrow$

                $(I_b, Y) \longleftarrow$ $y \in \{1 \dots q - 1\}$ $Y := g^y$

          $y' := A(I_a, I_b, Y, X')$

$(I_b, Y') \longleftarrow$ $Y' := g^{y'}$

$(X) \longrightarrow$

                $(X') \longrightarrow$ $Com(X') = C_X'$

$OI_a := (I_a | I_b)$ $OI_b := (I_a | I_b)$

$OK_a := (X | Y')$ $OK_b := (X' | Y)$

$O_a := \mathrm{HMAC}_{OK_a}(OI_a)$ $O_b := \mathrm{HMAC}_{OK_b}(OI_b)$

Figure 4: Wong-Stajano attack on MANA III is prevented in UACAP

is left with a single 2^{-20} chance to remain undetected during an online attack, which seems acceptable for most scenarios.

Verify The Wong-Stajano attack on MANA III assumes that an attacker, after running a standard MITM attack on the DH key exchange, can find a collision for the verification function $m_K(...)$ so that the random values K_1 and K_2, which the attacker can choose freely, mask the differences in DH keys and lead to the same verification codes [36]. When those verification functions are implemented using cryptographic hashes, e.g. HMAC keyed by X, then this attack translates to finding an l-Bit collision in the hash function, which can always (even assuming perfect hash functions) be performed as a brute-force search in $O(2^l)$ (which is a much simpler attack than the one-off chance we would like to remain as the only online attack vector). When transmitting the full hash output (as defined in MANA III), this seems infeasible.

However this attack does not work on UACAP by inheriting the principal security properties of MA-DH. A man-in-the-middle C would, in the general UACAP protocol run – independently of how the auxiliary message is transmitted –, perform the steps listed in Figure 4. The adversary function A must then generate an x' so that the l-Bit message *oob* suffers from a collision, i.e. $H(I_a | I_b, X | Y') = H(I_a | I_b, X' | Y)$. At this time, the adversary has access to the components I_a, I_b, X', Y, and attempts to generate Y' so that the collision occurs. However, X has not yet been made public, and under the assumption that ephemeral DH keys are used and thus X is random, this translates to a guessing game with a one-off chance of 2^{-l} of succeeding. The difference between MANA III (and the subsequently proposed Wong-Stajano variant) and MA-DH (and MANA IV) is the initial commitment message, which prevents this attack.

Input Confidential and Non-confidential From the "VIC" protocol proposal that combines MANA with SiB [27], BEDA [29] and Wong and Stajano's MANA III variant [36], we adopt the second round of mutual commitments (3.IN/IC Exchange hashes Q) for the "short input" case. Again, security arguments presented for these protocols remain valid for UACAP because equivalent messages are used. In principle, the same argument as for the first commitment (described above in the *TL* case) holds: an attacker would need to guess the short inputs before they are revealed to successfully masquerade as A or B when the long nonces J_a and J_b are revealed for comparison of Q_a and Q_b. By adding these nonces to the exchanged hashes, a brute-force guessing game has the order of 2^{128} and only a one-off chance for guessing R_a and R_b (equal for the IC case) remains when the attacker fabricates own nonces as an active MITM.

An interesting case arises when auxiliary channels are one-way, discussed in more detail in [27], [36],

and [22]. Authentication can not be fully mutual in the sense of human verifiability, but when a remote device can be trusted to correctly perform comparison of received auxiliary messages and report results of this comparison, then mutual authentication on the protocol level is possible.
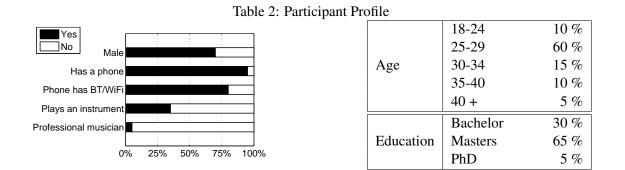
5.2 User Study

We used OpenUAT to conduct a first experiment towards comparing the usability of different authentication methods. For this user study, we selected four different authentication methods: using the *audio* and *visual* channels in *transfer* and *verification* modes. As a result, in transfer cases, the devices automatically decide whether authentication was successful, while in the verify mode, this responsibility lies with the user. We call the device initiating the communication C (client) and the device responding S (server).

User Study Setup

From the user perspective, the verification step following automatic key agreement (step 4 in UACAP) proceeds as follows for the four different authentication methods. For **Barcode transfer**, S displays a 2D barcode. The user takes a picture of the barcode using C. C confirms whether the key was correctly exchanged. In **Audio transfer**, S plays a (fast) tune. C records the tune and confirms whether the key was correctly exchanged. To **Compare melodies**, C plays a melody. Subsequently, S plays a melody. The user compares the melodies and, if they match, acknowledges that the pairing was successful. Finally, to **Compare sentences**, both devices concurrently display a sentence. The user compares these sentences and, if they match, acknowledges that the pairing was successful.

We recruited 20 participants for our user study, mainly a group of fairly young, well-educated and technology-savvy participants (19 university students and researchers and one secretary). The demographics and related background information of the participants are summarized in Table 2.

Table 2: Participant Profile

	18-24	10 %
	25-29	60 %
Age	30-34	15 %
	35-40	10 %
	40 +	5 %
	Bachelor	30 %
Education	Masters	65 %
	PhD	5 %

Test Procedure

All experiments were conducted in normal office conditions, namely good lighting conditions and relatively low noise level. Before starting, participants were asked to fill in the background questionnaire, which served to learn about their experience with mobile devices and music related background and is summarized in Table 2. Afterwards, participants were given a brief overview of the problem. We motivated the need for secure device pairing and briefly presented the goals of our study. Although most of the users were already familiar with the functionality of the devices used, we gave a brief overview of the basic operations needed to interact with the device (e.g. how to navigate through the application menu, how to select options and to take pictures).

We have chosen Scenario 1 and 2 described in section 1, namely exchanging vCards and printing a document, to motivate the participants to perform the tasks. To capture the QR code when pairing two phones (a Nokia N82 and a Nokia N95 8GB), a focus lens was attached to the camera of the phone, compensating for the lack of focus control in J2ME and consequently in the ZXing QR decoder. The printer was simulated by a laptop. For each scenario, participants were asked to run the pairing application with all four authentication methods. No attack was simulated, i.e. verification sequences always matched. To reduce the learning bias on test results, half of the users were first presented with Scenario 1 and the other half with Scenario 2. User actions and their durations were automatically logged. Afterwards, each participant filled in a post-test questionnaire form and was given some minutes of free discussion.

Results

Table 3 summarizes the logged data with average completion times between 13.2 seconds (comparing sentences) and 37.7 s (barcode transfer from a phone display) and average number of tries until successful pairing between 1 (comparing sentences) and 2.3 (barcode transfer). Figure 5 presents user's perceived ease of use and level of security, split further between short-lived and long-lived keys. Especially our direct comparison of four authentication methods produced some interesting findings:

Table 3: Summary of the logged data for pairing 1) two phones(Ph-Ph) and 2) phone and laptop (Ph-Lap)

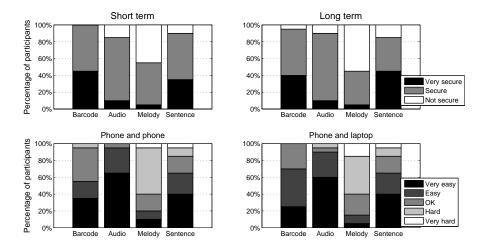| Method	Average completion time (sec.)		Average number of tries		Percentage of failures	
	Ph-Ph	Ph-Lap	Ph-Ph	Ph-Lap	Ph-Ph	Ph-Lap
Barcode transfer	37.7 *(sd*=14.0)*	34.1 *(sd=15.3)*	2.1	2.3	5%	15%
Audio transfer	30.7*(sd=12.6)*	31.9 *(sd=14.2)*	1.2	1.6	0%	5%
Compare melodies	19.6 *(sd=8.3)*	20.3 *(sd=10.8)*	1.3	1.3	20%	10%
Compare sentences	15.6 *(sd=7.8)*	13.2 *(sd=4.5)*	1.0	1.0	0%	0%

*sd = Estimated standard deviation

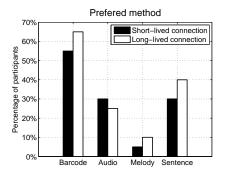Figure 5: Usability and security estimation by users

15

Figure 6: User preference

Barcode transfer Results show that, even if using the visual channel in transfer mode resulted in the longest completion time, it was by far the most preferred and most highly trusted pairing method. In fact, users showed high acceptance even when decoding the QR code failed and they had to retry. After filling in the post-test questionnaire, participants were presented with the live QR decoder application that is pre-installed on some Nokia devices (but which could not be integrated with OpenUAT due do unavailability of open APIs). Seeing how smooth and fast decoding could be made, this method appealed even more to users. These results are in accordance with the background questionnaire which revealed that taking pictures is the most widely performed task on mobile phones (by over 80% of the participants).

Audio transfer Using the audio channel in transfer mode is the only authentication method in which the user did not have to assist the devices in any way. However, while overall study results acknowledged the method as the easiest to perform (see Figure 5), other factors such as the social context made users prefer the less obtrusive, seemingly more secure visual channel. Several users suggested replacing audio by ultrasound transfer (already implemented in OpenUAT, but inherently requiring additional hardware and visualization). Another interesting result is the perceived duration time of the pairing process. Even though the method using the audio channel takes less time to complete – in average 15 s less than the visual channel – some users were bothered by the long time needed to decode the audio message (during which they did not have to conduct any task). On average, decoding the audio message took 16 s for phone to phone and 13 s for laptop to phone (due to better sound quality on the laptop).

Comparing melodies This is the method that users found the most difficult and which they least trusted. 35% of the participants have been playing at least one instrument for 3 to 27 years, with an average of 12 years. Unexpectedly, people with advanced music experience did not find the melodies easier to compare than people that do not play any instrument. On the contrary, these people were more sensitive to sound differences between devices (even the same tune will sound different when played by different devices) and trusted this authentication method less. On average, the tunes were replayed 1.3 times. Although there was no attacker, in 10% and 20% of the trials, respectively, participants failed to recognize the tune as being the same. The general impression was that the sequences were too long. A melody lasted 4 to 5 s and played 7 to 9 notes.

Comparing sentences This was generally considered a secure method. It had the fastest completion time (13 and 15 s), but required significant user attention. Some participants were bothered by the lack of semantics of the sentences (e.g. *"DURWARD FOOLHARDILY DISTORT-ed to BRANCH on a COMMITTEE"*). Because sentences are automatically constructed from a cryptographic token, they do not have any meaning.

6 Conclusions

The problem of authenticating spontaneous interactions has seen significant interest in recent years due to its wide applicability in current and future application scenarios. Many approaches have been suggested independently, and only rarely distributed with an open, reproducible implementation. In this paper, we contribute UACAP as a new, unified cryptographic protocol for device authentication to use with arbitrary auxiliary channels. We also contribute OpenUAT as an open source, publicly available toolkit for authentication and implement some intuitive authentication methods in a common library based on UACAP. Video, audio, ultrasound, motion, keypad input, and sentence comparison are already available for application developers and are easily comparable and interchangeable. Further auxiliary channel implementations are currently being integrated.

It seems important to provide a vast library of different methods — they should be chosen to best suit the envisaged application, and direct comparability in rapid prototyping will assist application designers in doing so. By providing OpenUAT as a toolkit for system builders, we hope to both foster future research and to shorten the gap between research prototypes and real-world applications.

References

[1] 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 Symp.*, pages 207–222. USENIX, August 2004.

[2] D. Balfanz, D. K. Smetters, P. Stewart, and H. C. Wong. Talking to strangers: Authentication in ad-hoc wireless networks. In *Proc. NDSS'02*. The Internet Society, February 2002.

[3] Bluetooth SIG. Bluetooth Special Interest Group. Simple Pairing Whitepaper (Revision V10r00), 2006.

[4] M. Čagalj, S. Čapkun, and J.-P. Hubaux. Key agreement in peer-to-peer wireless networks. *IEEE (Special Issue on Cryptography and Security)*, 94:467–478, 2006.

[5] S. Creese, M. Goldsmith, R. Harrison, B. Roscoe, P. Whittaker, and I. Zakiuddin. Exploiting empirical engagement in authenticated protocol design. In *Proc. SPC 2005*, pages 119–133. Springer-Verlag, April 2005.

[6] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Trans. on Information Theory*, IT-22(6):644–654, 1976.

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

[8] James Fogarty and Scott E. Hudson. Toolkit support for developing and deploying sensor-based statistical models of human situations. In *Proc. CHI 2007 Ubicomp tools*, pages 135–144. ACM Press, 2007.

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

[10] 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*, page 10. IEEE CS Press, July 2006.

[11] J.-H. Hoepman. The emphemeral pairing problem. In *Proc. 8th Int. Conf. Financial Cryptography*, pages 212–226. Springer-Verlag, February 2004.

[12] Dimitris N. Kalofonos, Zoe Antoniou, Franklin D. Reynolds, Max Van-Kleek, Jacob Strauss, and Paul Wisner. Mynet: A platform for secure P2P personal and social networking services. In *Proc. PerCom '08*, pages 135–146. IEEE CS Press, 2008.

[13] T. Kindberg and K. Zhang. Context authentication using constrained channels. Technical Report HPL-2001-84, HP Laboratories, April 2001.

[14] K. Kostiainen, E. Uzun, N. Asokan, and P. Ginzboorg. Framework for comparative usability of distributed applications. Technical Report NRC-TR-2007-005, Nokia Reserach Center, 2007. http://sconce.ics.uci.edu/CUF/ex_abs.pdf.

[15] Navaneeth Krishnan. JXTA and security. In Darren Govoni Daniel Brookshier, Navaneeth Krishnan and Juan Carlos Soto, editors, *JXTA: Java P2P Programming*. Sams Publishing, 2002.

[16] Anthony LaMarca, Yatin Chawathe, Sunny Consolvo, Jeffrey Hightower, Ian Smith, James Scott, Timothy Sohn, James Howard, Jeff Hughes, Fred Potter, Jason Tabert, Pauline Powledge, Gaetano Borriello, and Bill Schilit. Place Lab: Device positioning using radio beacons in the wild. In *Proc. Pervasive 2005*, pages 116–133. Springer-Verlag, May 2005.

[17] S. Laur and K. Nyberg. Efficient mutual data authentication using manually authenticated strings. In *Proc. CANS 2006*, pages 90–107. Springer-Verlag, December 2006.

[18] Rene Mayrhofer. The candidate key protocol for generating secret shared keys from similar sensor data streams. In *Proc. ESAS 2007*, volume 4572 of *LNCS*, pages 1–15. Springer-Verlag, July 2007.

[19] Rene Mayrhofer. Towards an open source toolkit for ubiquitous device authentication. In *Workshops Proc. PerCom 2007: 5th IEEE International Conference on Pervasive Computing and Communications*, pages 247–252. IEEE CS Press, March 2007. Track PerSec 2007: 4th IEEE International Workshop on Pervasive Computing and Communication Security.

[20] Rene Mayrhofer and Hans Gellersen. Shake well before use: Authentication based on accelerometer data. In *Proc. Pervasive 2007*, volume 4480 of *LNCS*, pages 144–161. Springer-Verlag, May 2007.

[21] Rene Mayrhofer, Hans Gellersen, and Mike Hazas. Security by spatial reference: Using relative positioning to authenticate devices for spontaneous interaction. In *Proc. Ubicomp 2007*, volume 4717 of *LNCS*, pages 199–216. Springer-Verlag, September 2007.

[22] Rene Mayrhofer and Martyn Welch. A human-verifiable authentication protocol using visible laser light. In *Proc. ARES 2007*, pages 1143–1147. IEEE CS Press, April 2007.

[23] J. M. McCune, A. Perrig, and M. K. Reiter. Seeing-is-believing: Using camera phones for human-verifiable authentication. In *Proc. IEEE Symp. on Security and Privacy*, pages 110–124. IEEE CS Press, 2005.

[24] S. N. Patel, J. S. Pierce, and G. D. Abowd. A gesture-based authentication scheme for untrusted public terminals. In *Proc. UIST 2004*, pages 157–160. ACM Press, October 2004.

[25] A. Perrig and D. Song. Hash visualization: a new technique to improve real-world security. In *Proc. CrypTEC'99*, pages 131–138, 1999.

[26] D. Salber, A. K. Dey, and G. D. Abowd. The context toolkit: Aiding the development of context-enabled applications. In *Proc. CHI '99*, pages 434–441, May 1999.

[27] N. Saxena, J.-E. Ekberg, K. Kostiainen, and N. Asokan. Secure device pairing based on a visual channel. Cryptology ePrint Archive, Report 2006/050, 2006.

[28] C. Soriente, G. Tsudik, and E. Uzun. HAPADEP: Human asisted pure audio device pairing. Cryptology ePrint Archive, Report 2007/093, March 2007.

[29] Claudio Soriente, Gene Tsudik, and Ersin Uzun. BEDA: Button-enabled device pairing. In *Proc. IWSSI 2007*, pages 443–449, September 2007.

[30] F. Stajano and R. Anderson. The resurrecting duckling: Security issues for ad-hoc wireless networks. In *Proc. 7th Int. Workshop on Security Protocols*, pages 172–194. Springer-Verlag, April 1999.

[31] J. Suomalainen, J. Valkonen, and N. Asokan. Security associations in personal networks: A comparative analysis. In *Proc. ESAS 2007*, pages 43–57. Springer-Verlag, 2007.

[32] E. Uzun, K. Karvonen, and N. Asokan. Usability analysis of secure pairing methods. In *Proc. USEC 2007: Usable Security*, February 2007.

[33] A. Varshavsky, A. Scannell, A. LaMarca, and E. de Lara. Amigo: Proximity-based authentication of mobile devices. In *Proc. UbiComp 2007*, pages 253–270. Springer-Verlag, September 2007.

[34] S. Vaudenay. Secure communications over insecure channels based on short authenticated strings. In *Proc. CRYPTO 2005*. Springer-Verlag, August 2005.

[35] Wi-Fi Aliance. Wi-Fi protected setup specification v1.0, January 2007.

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