Bridging the UI Gap for Authentication in Smart Environments

Sebastian Unger and Dirk Timmermann
Institute of Applied Microelectronics and Computer Engineering
University of Rostock, Germany
Email: sebastian.unger@uni-rostock.de, dirk.timmermann@uni-rostock.de

***Abstract*—In this paper we describe a common problem when it comes to mutual authentication of devices in smart environments. Basically, there exist two approaches. In the brokered approach, two parties authenticate with the help of a common trusted party. However, this often introduces a single point of failure. The alternative approach lets the devices authenticate with each other directly, usually by exchanging a shared secret out of band. But this is only possible if both devices have the necessary user interface peripherals to establish an out-of-band channel.**

We will describe an abstract basic protocol to bridge the possible peripherals' gaps by authenticating indirectly with the help of nowadays basically omnipresent multimedia devices such as smart phones. After extending the proposed protocol for more complex use cases, we refine it to a practical authenticated key establishment protocol for indirect authentication. Finally we describe our freely available prototype implementation consisting of an internet-of-things-enabled light bulb and light switch and an Android app that comprise our proof of concept.

***Keywords*—Applied Cryptography; Authentication; DPWS; Intelligent Environments; Internet of Things; Usability**

I. Introduction

More than twenty years after ideas such as Pervasive Computing and Smart Environments shifted into focus of computer scientists ([1]), their security aspects are still often neglected when developing new technologies. Despite the consensus, that these visions will never become reality without providing adequate security that protects our privacy, developing secure solutions is considered expensive and not user-friendly ([2]). To provide confidentiality for resource-constrained devices – the atomic building blocks of smart environments and the Internet of Things – lightweight cryptography such as elliptic curve algorithms performs well. However, most unsolved problems concern bootstrapping. That is, how to set up an ensemble of previously unknown devices in a secure and user-friendly way. Once the setup is accomplished, data can be transmitted securely by encrypting it and sophisticated authorization frameworks manage access to certain resources.

One approach to set up a secure connection between two parties is to exchange a shared secret over a separated secure channel, a so-called out-of-band (OOB) exchange. However, this simple approach is limited to device pairs that have matching peripherals to establish such an OOB channel such as visual light, audio or NFC. For example, a light switch whose only peripheral is the switch itself and a light bulb whose only peripheral is the bulb itself have no way to exchange a shared secret without supplying them with additional peripherals.

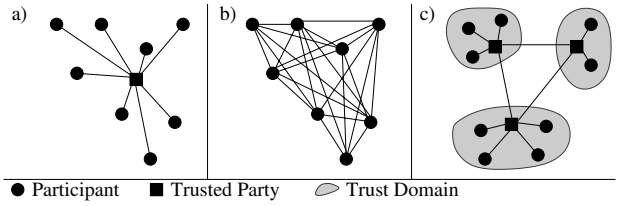

Fig. 1. Trust relationships a) brokered, b) direct c) brokered in trust domains

In this paper we present a solution to bridge these peripheral user interface (UI) gaps by employing additional devices already present in future smart spaces: multimedia devices. There are up to dozens of TV sets, digital picture frames or smart phones already present in nowadays homes and offices and each of them owns plenty of peripherals such as displays, keypads, cameras, NFC transceivers, sensors and so forth.

The remainder of this paper is structured as follows. In section II we describe the basic principles and our motivation in more detail. In section III we describe a simple abstract protocol to bridge a UI gap between two devices that we subsequently extend to more complex use cases and finally refine to a concrete authenticated key exchange protocol. After describing our proof of concept prototype in section IV, we sum up this paper in section V.

II. Basic Principles and Motivation

In this section, we will examine authentication in smart environments and derive the motivation of our work. As the Devices Profile for Web Services (DPWS, [3]) is our underlying communication middleware its properties and features are briefly discussed in this section as well.

A. Authentication in Smart Environments

Stajano and Anderson stated in [4] that authentication is the most interesting problem when it comes to embedded device security. Once this is done, confidentiality can be easily achieved. In existing communication middlewares, two approaches can be identified: brokered and direct authentication.

Brokered authentication means that few trusted parties inside a network delegate trust relationships between most other participants (fig. 1 a)). This way, most participants need to maintain only very few trust relationships. Examples for brokered authentication are implemented in the Kerberos protocol

or public key infrastructures based on X.509 certificates and certificate authorities (CA). Many middlewares (e.g. [5]), [6], [7]) rely on this principle.

Instead of relying on brokered trust, participants can authenticate each other directly. For that purpose, a one-time shared secret is exchanged over a secure out of band channel which is then used for an authenticated key establishment or exchange protocol (e.g. a challenge-response-based exchange as we proposed earlier in [8] or an authenticated Diffie-Hellman-based establishment as in [9]). Exchanging the shared secret OOB can happen by simply displaying and entering a pin. If one participant lacks the necessary UI components alternative channels can be used and an excessive amount of research has been conducted on this. [10] gives a comprehensive overview over possible mechanisms such as visual light channel ([11]), (2D) bar codes ([12]) and audio ([13], [14]). Even IrDA, NFC or acceleration sensors ([15]) could be used.

Both approaches have drawbacks. Brokered authentication relies on an existing infrastructure and requires every participant to be deployed with its own credentials as well as the credentials of chosen trusted parties. This appears feasible in scenarios where all participants come from one manufacturer but less likely for manufacturer-independent infrastructures. However, when authenticating devices directly, every device must maintain the credentials of every other party it needs to communicate with (see fig. 1 b)). This potentially sums up to (ten) thousands of credentials in smart home or office scenarios. Direct authentication is also unlikely to work when connecting different trust domains that can also be spatially divided (e.g. different homes or satellite offices), while brokered authentication works (fig. 1 c)).

Combining brokered and direct authentication leads to hybrid approaches that can overcome the drawbacks of both. If there exists a brokered trust relationship between two devices, it can be used completely transparent to the user. If however no such link exists, it can be established directly. Using hybrid authentication approaches is common and they are employed in Mobile Gaia ([16]) and Amigo ([17]) for example.

A problem arises where direct authentication is necessary but the two devices to be authenticated cannot establish an OOB channel due to the lack of matching UI capabilities. Considering the simplest-to-use example we defined earlier employing light bulbs and switches ([18]) demonstrates the problem. There is no way for a light bulb and a switch to establish an OOB channel to exchange a shared secret.

It is the declared goal of this work to bridge these UI gaps between different devices without supplying additional peripherals. We will describe the developed protocols in section III and present a working prototype in section IV.

B. The Devices Profile for Web Services

Planned as the successor of Universal Plug and Play, DPWS became an independent technology ([3]). It provides a communication middleware for embedded devices on the basis of Web Services, allowing devices such as printers or scanners to be easily deployed in existing network infrastructures. Besides, DPWS increasingly gets employed in medical appliances ([19]), wireless sensor networks ([20]) and automation industry ([21]).

In DPWS, *Devices* can host different *Services* that can offer *Operations*. Latter can be invoked by *Clients*. DPWS is based on SOAP Web Services and provides classic request response messaging pattern, as well as asynchronous communication. It supports dynamic discovery without a central registry and allows Clients to retrieve metadata about the Device itself as well as its hosted Services.

The security concept of DPWS relies on profiles that can be understood as a set of rules two parties agree on before communicating. The DPWS specification document describes such an optional profile ([3]) which bases on establishing TLS-secured channels between devices. Where this is not applicable (the discovery messages are sent via UDP multicast), traffic gets signed by a compact signature mechanisms described in WS Dynamic Discovery [22]. The necessary credentials are provided by X.509 certificates and their matching private keys. The process of authenticating or exchanging certificates is not specified.

III. Bridging the Gap

In this section we describe our approach on bridging UI gaps. We discuss device classes we assume to exist in every smart environment and show how they can cooperate for user-friendly, secure bootstrapping.

A. Device Classes

In [23] Ben Saied et al. classify M2M-communication participants in three groups: resource constrained nodes, nodes, and high-power nodes. We adopted this classification with a minor adjustment. We assume the following three categories:

Resource constrained nodes are e.g. sensors and actuators in a smart environment. They have limited computational power and memory and are likely to be battery driven. Since they often are specialized on fulfilling a single task (e.g. 'measure temperature' or 'open window'), they also usually have very few and limited control elements such as buttons, LEDs, etc. and therefore support only few authentication mechanisms.

The other end of the scale are *high-power nodes*. Examples are desktop PCs, laptops or control servers in maintenance rooms. They most likely have a constant power supply and sheer unlimited resources such as computational power, bandwidth and memory. While they are extremely extensible in terms of resources as well as control- and UI elements, they usually are immobile or at least bulky.

What Ben Saied et al. consider in between high-power and resource-constrained and what they call *nodes*, is what we call *UI Devices*. Examples are e.g. smart phones. Although they indeed have moderate resources and they may be battery-driven, this is not critical. What distinguishes them from the remaining devices is their plethora of different UI capabilities. They offer for example a keypad, LEDs, speakers, a microphone, NFC, a camera, visible light, an HD display and acceleration sensors - often all in a single device. Besides, they often are highly mobile and usually carried with their user.

B. Basic Protocol

Given that we assume resource-constrained nodes, high-power nodes and UI Devices in a network, we now describe how they can provide user-friendly security bootstrapping.

1) Bridging with a UI Device: To explain the abstract protocol, we consider an example consisting of a light bulb, a switch and a smart phone. The light bulb is a DPWS Device, offering a Service for light bulb status monitoring and control and a Service for authentication. Its only possibility for exchanging a one-time pin OOB is to flicker it. The wall switch is a DPWS Client invoking the Operations of the Services described above. Its only control element is the switch itself, which can be on or off. This way, a key can be 'tapped in' bitwise. The phone is the intermediary UI device and hosts Device- and Client-functionality. For this example, it is at least equipped with a display to show a pin to be tapped and a digital camera to catch a pin flickered by the bulb in its binary form.

The abstract protocol is depicted in figure 2. After discover-

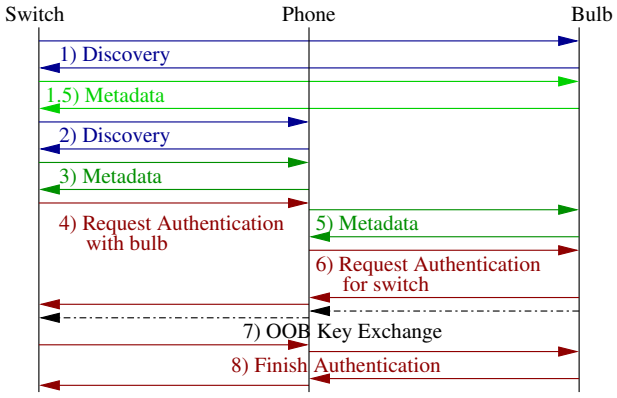

Fig. 2. Indirect Authentication – Basic Protocol

ing the bulb it wants to switch (1), the switch optionally can retrieve the bulb's authentication service's metadata to find out if it offers an authentication method the switch supports (1.5)). If this fails or to minimize its effort, the switch will look for existing authentication devices and discovers the phone (2)). By accessing the phone's metadata (3), the switch makes sure that they share at least one authentication mechanism, which is 'tapping' in this example. If this fails, the switch needs to repeat steps 2) and 3) until it finds a suitable UI Device. We assume, that at least one exists.

In step 4) the switch requests authentication with the bulb from the phone as the mediator. It is the phone's responsibility to access and understand the bulb's metadata to find an authentication mechanism it can handle (5)). Now, the phone can request authentication with the bulb on behalf of the switch (6)). The reply containing the first credentials for authentication is bypassed to the switch. After the bulb replied to step 6), it flickers a one-time pin in its binary form which can be read by the phone's camera. At the same time, when the switch received its reply to step 4) it waits for a key to be tapped. So, once the phone read the flickered key, it can display the pin in a way that a user can tap it in the switch.

After the PIN has been exchanged over the OOB channel(s), the remainder of the authenticating handshake can take place. The phone again bypasses messages from the switch to the bulb and vice versa. For the switch and especially for the bulb the use of the phone is completely transparent. There is no difference to direct authentication. It should be noted, that the phone is (and stays) an untrusted device (although it gets authenticated implicitly). This does however not affect security as it is in control of a legitimate user. Otherwise it would not have access to at least one of the OOB channels.

2) Bridging larger UI gaps: The client is not aware of a possible UI gap between the intermediary it chose and the device it wants to authenticate with. Instead, it picks the first UI Device that matches the client's mechanisms and requests authentication with the target (cmp. fig 2, steps 1)-4)). The

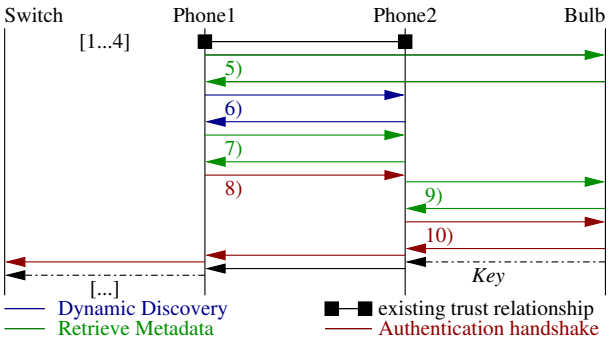

Fig. 3. Bridging larger gaps

purpose of this behavior is to offload resource-intense tasks such as discovery and metadata processing from the weakest to stronger participants. If the intermediary determines that it can handle none of the mechanisms offered by the target (fig. 3, step 5)), it will conduct a discovery process for other UI Devices (step 6) it already trusts and will examine their capabilities (step 7)). It will repeat these steps until it finds a matching, trusted UI Device. If this fails, it will discover untrusted devices and authenticate with a matching one.

In steps 8) and ongoing authentication is conducted as in the previous section with the only difference that there are two intermediaries and handshake messages are bypassed through both of them. This does not affect security since both intermediaries are authenticated and thus can share all information over a confidential channel. Again, this is completely transparent for Switch and Bulb. Also, Phone2 and Phone1 behave exactly identical, as for Phone2, Phone1 simply appears as a Client requesting authentication.

3) Connecting Trust Domains: The extension of the basic protocol described in the preceding section can scale further to a (theoretically) arbitrary number of intermediaries if indirect and brokered authentication are combined as depicted in figure 4. The only prerequisite is that the participating high-power nodes (hpn1,2) offer mechanisms of their trusted devices as their own. In the example, this means that hpn1 offers its own as well as hpn2's mechanisms, while hpn2 also offers hpn1's and uid2's. This way, authentication among several

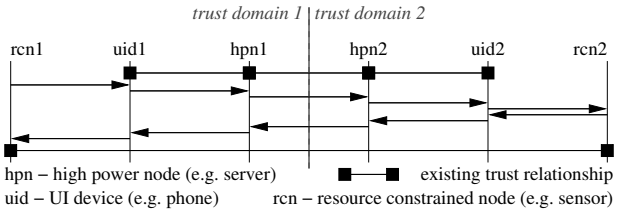

Fig. 4. Combining brokered and indirect authentication

intermediaries is still completely transparent to the resource-constrained nodes and for the UI Devices. Also, employing several intermediaries does not affect security since there is a secure channel for every hop.

4) Discussion of the proposed protocols: The approaches described above bring several advantages. They can bridge UI gaps between devices without the necessity of brokered authentication and enable direct authentication where this has not been possible before. Regarding the device (e.g. the bulb) this is completely transparent. The protocols are less transparent for the client (e.g. the switch) but by letting the UI device download and parse existing metadata from the device (bulb) and deciding on suitable mechanisms, it takes over very heavy-weight tasks from a critically resource-constrained device. It should be stressed here, that no explicit trust relationships between switch and phone and between phone and bulb are formed and trust is simply brokered subsequently. Instead, the phone remains unauthenticated. It cannot be an attacker though, because the phone must be controlled by the user. The approach dramatically gains in momentum with every additional authentication mechanism implemented (NFC, audio, QR codes, ...), as more and more device pairs with very different UI peripherals can be indirectly authenticated with each other.

C. Authenticated Key Exchange

After explaining the abstract protocols to bridge UI gaps to connect trust domains, we can now derive a practical authenticated key exchange protocol from them. Our proposed solution bases on an authenticated Diffie Hellman key establishment presented by Ho in [9].

1) Diffie Hellman revisited: The Diffie Hellman (DH) key establishment protocol bases on asymmetric cryptography and allows two parties to agree on a shared secret without ever transmitting it. When using the lightweight elliptic curve cryptography (EC) as underlying asymmetric cryptography, the protocols works as follows. The two parties Alice and Bob publicly agree on a selected elliptic curve G. Each party selects its own secret key SK and both calculate their own public key $PK = SK \times G$. Alice and Bob exchange their public keys and finally calculate the common shared secret, the so called session key $S = S_A = SK_A \times PK_B = S_B = SK_B \times PK_A$.

2) Authenticated Diffie Hellman by Ho: The Diffie Hellman protocol is easily breakable by means of a man-in-the-middle attack (MITM), because the key-agreeing parties are not authenticated ([24, cpt. 9.3.6]). In [9] Ho presents a suite of authentication protocols for resource-constrained devices that later became part of the IEEE 805.15.6 specification ([25]) defining Body Area Networks. It defines an unauthenticated key agreement protocol based on Elliptic Curve Diffie Hellman (ECDH) and describes ways to authenticate it. We focus on his variant exchanging shared secrets via OOB channels, since our protocols rely on this.

The solution presented by Ho relies on ECDH as described above and assumes an already OOB-exchanged shared secret PW (see fig. 5). To authenticate the key exchange, Ho

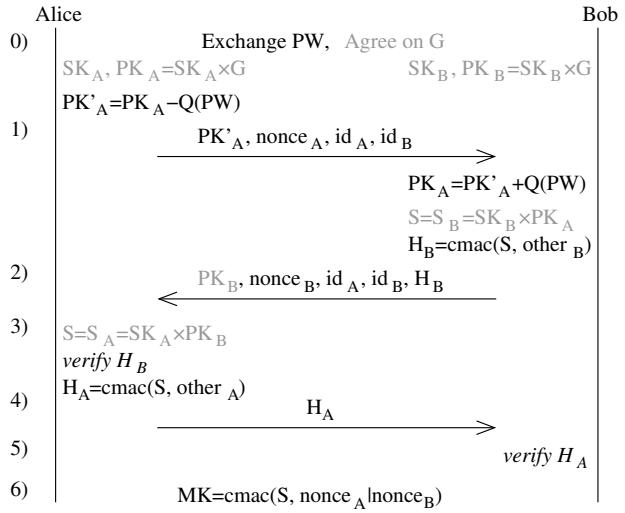

Fig. 5. Authenticated ECDH by Ho ([9]), classic ECDH colored gray

scrambles the public key of one participant using PW and by deriving an additional elliptic curve point $Q(PW)$ in step 0. Subsequently $PK' = PK - Q(PW)$ is transmitted instead of PK and due to its knowledge of PW, the recipient can restore $PK = PK' + Q(PW)$ before step 2. In addition, Ho splits the session key S. The first half is used to cryptographically hash the relevant parameters $other_{A/B}$(including target and source ids) using the CMAC algorithm. Computing, exchanging and verifying these hashes H_A and H_B is a sufficient amount of authentication by itself. The second half is used to derive the master key MK from the random values $nonce_A$ and $nonce_B$.

3) Proposed Solution: The authenticated key establishment protocol proposed in this paper is based on the authenticated key exchange by Ho described above and differs in only two minor changes (see fig. 6). For one, the intermediaries

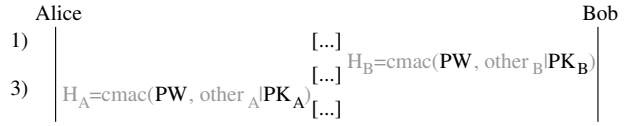

Fig. 6. Difference in proposed solution, base by Ho ([9]) colored gray

described in the previous section need the ability to alter the exchanged parameters (esp. used authentication mechanisms) to be able to preserve transparency for Devices. This means, that the intermediaries also need the ability to recompute the

cryptographic hashes H_A and H_B which is not possible when the Diffie Hellman Session Key is required. Thus, the shared secret PW is employed instead to compute the hashes.

Although an MITM attack is still prevented by the encryption of the public key in the step 0, Alice and Bob will not detect the attack but will simply be unable xto communicate. For this reason both parties include their public key in the cryptographic hash. In case of an MITM attack, hash verification will fail before the master key MK is computed. Both these adjustment do not increase the complexity of the protocol. The only difference is the key used for the hashes and an additional digest to consider when calculating the hash and the security analysis Ho conducts in [9, sct.3] is still valid.

The resulting protocol is depicted in figure 7. First, Client

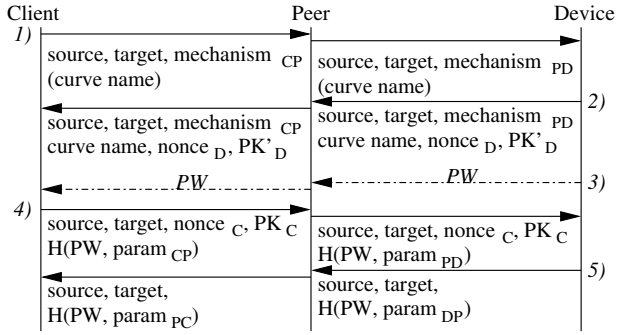

Fig. 7. Authenticated Key Exchange

directs a request to authenticate with Device at Peer. This request contains the source's id, the destination's id and the authentication mechanism Client wants to use with Peer. Optionally, Client may request an elliptic curve by submitting the curve's name. Peer bypasses this request by only replacing the authentication mechanism with one that is suitable for both Peer and Device. Device responds with its encrypted ECDH public key, its nonce and the authentication mechanism and elliptic curve it eventually uses (step 2)). Again Peer simply bypasses by replacing the authentication mechanism. After the shared secret has been transmitted over OOB channels in step 3), Client and Device both can calculate the ECDH session key S that relies on the elliptic curve G, public and private keys $PK/SK_{C/D}$ and the OOB-exchanged shared secret PW.

Client now triggers the remainder of the handshake in step 4). Therefore, it transmits its public key, its own nonce and the cryptographic hash of the exchanged parameters (source, destination, sent and received nonce, curve name, authentication mechanism and public key) with the OOB shared secret PW as the key. Peer needs to verify the received and calculate a new cryptographic hash due to the replacement of the authentication mechanism. Finally, Device responds with its own hash in step 5, which Peer needs to verify and recalculate as well. Both, Device and Client can verify the received hashes. If verification succeeds, the key establishment is successfully authenticated. Now, both Device and Client can calculate the master key $MK = H(S, nonce_D|nonce_C)$.

Eventually, Device and Client can derive keys for providing confidentiality, integrity and authenticity from the master key MK.

IV. Prototype implementation

We implemented a proof of concept that consists of two resource-constrained nodes (switch, light bulb) and a UI Device that supports the protocol described in section III-B1. Documentation as well as all source code and schematics are freely available as open source at [26].

A. Materials

The following material, tools and hardware were used.

1) JMEDS: The Java Multi Edition DPWS Stack (JMEDS, [27]) is an open source Java DPWS implementation ideal for rapid prototyping. We used the latest release, as the time of writing version 2beta8.

2) Bouncy Castle: The *Legion of the Bouncy Castle* provides ([28]) a crypto library for C# and Java. For the Java applications implementing switch and light bulb (see below), the library was used in version 1.49 to provide crypto functionality, especially elliptic curve routines and the CMAC algorithm. For the UI authenticator (see below) the Android derivative spongycastle ([29] was used in version 1.47.

3) Raspberry Pi: The resource-constrained nodes were implemented on Raspberry Pi Model B computers. These are embedded Linux platforms with a 700MHz ARM11 CPU and 512MB RAM and an average power dissipation of around 3.5 Watts. Such a platform is not exactly resource constrained, however, it is extremely inexpensive and very flexible as it even runs Java Virtual Machines. Thus, it allows rapid prototyping and implementing proof of concepts.

B. Light bulb and switch

The light bulb and the switch are both implemented in Java with JMEDS and are executed on a Raspberry Pi Model B. While the switch can be attached to a Pi's GPIO, the bulb needs to be interfaced using a relay and a driver circuit. The bulb hosts an Authentication Service supporting the ECDH handshake described above. It is also capable of flickering a PIN in its binary form. The switch forms the DPWS Client and provides the capability that a PIN can be 'tapped in' in its binary form for authentication purposes.

C. UI Authenticator

The UI authenticator is implemented as an Android App incorporating JMEDS. The app's core is a collection of plug-ins providing support for different authentication mechanism. Besides, the app offers DPWS Device capabilities to expose the Authentication Service even if the app itself is not active.

1) User interaction flow: At this point, we describe the functional behavior as presented to the user. Further documentation about the internals, communication and the app's architecture can be found at the project page ([26]). Pressing the switch for 5 seconds triggers the protocol. A bulb and a smart phone are discovered and authentication with the former via the latter is requested. The smart phone plays an alarm

sound and shows a notification to the user. When the user clicks on the notification, a live camera view is opened with a cross hair overlay. The user is directed to aim at the bulb and to start authentication by pressing a button. This makes the bulb flicker a one-time pin which is read by the phone's camera. When finished, the user is presented with directions on how to press and release the switch several times to tap in the binary key into the switch. Subsequently, the OOB transfer of the one time pin between bulb and switch was successful and the remainder of the authenticated key establishment can take place. Subsequently, the switch is able to direct authenticated, and potentially confidential requests to the bulb.

2) Implemented Authentication methods: For the prototype to work, we implemented flickering and tapping as authentication mechanisms. Former employs the phone's camera and determines the brightness of a small area depicted by a cross hair to be aimed at a bulb. This way, the plugin can distinguish the two states high (bright) and low (dark) and can thus read a key in binary form. The tapping plugin presents a pin as a sequence of 'press'/'don't press' states. This way, a pin can be supplied to a switch. Furthermore, a plugin was written to display and enter PINs in their numerical form, as this is a very common use case and was helpful for debugging purposes.

The described protocols show their strengths for multiple combinations of incompatible authentication mechanisms. In future, we plan to incorporate further mechanisms depending on emerging use cases. Some of them could be reading and displaying QR-Codes, flickering using the phones flash LED or exchange via NFC.

V. Conclusion and Outlook

In this paper we describe the problem of existing UI peripheral gaps when directly authenticating devices with each other in smart environments. To overcome this problem, we propose a simple abstract protocol that incorporates additional devices with plenty of UI peripherals, e.g. smart phones. We extend the protocol for more complex use cases and refine it to a concrete authenticated key establishment protocol based on Elliptic Curve Diffie Hellman. We end this paper with presenting our freely available open source prototype implementation which serves as proof of concept of our proposed protocols.

In near future, we plan to integrate additional authentication mechanisms into the android application. NFC and QR codes have top priority here. Furthermore, we plan to integrate our WS Compact Security Scheme described in previous work ([30]) into our implementation so it can be used with keys derived from the authentication. Finally, all steps will be conducted to implement the scenario depicted in fig 4 employing high-power nodes and brokered authentication.

References

[1] M. Weiser, "The computer for the 21st century," *Scientific american*, vol. 265, no. 3, pp. 94–104, 1991.

[2] D. Masak, *Digitale Ökosysteme - Serviceorientierung bei dynamisch vernetzten Unternehmen*. Springer, Berlin, 2009.

[3] OASIS, "Devices profile for web services version 1.1," Juli 2009.

[4] F. Stajano and R. Anderson, "The resurrecting duckling: Security issues for ad-hoc wireless networks," in *Security Protocols, 7th International Workshop Proceedings*. Springer Verlag, 1999.

[5] D. Conzon *et al.*, "The virtus middleware: An xmpp based architecture for secure iot communications," in *Computer Communications and Networks (ICCCN), 2012 21st International Conference on*, 2012.

[6] M. Handte *et al.*, "D4.1 secure middleware specification - version 1.4," Peces - Pervasive computing in embedded systems, Tech. Rep., feb 2010.

[7] R. Baldoni *et al.*, "An embedded middleware platform for pervasive and immersive environments for-all," in *Sensor, Mesh and Ad Hoc Communications and Networks Workshops, 6th Annual IEEE Communications Society Conference on*, june 2009.

[8] S. Unger *et al.*, "Extending the devices profile for web services for secure mobile device communication," in *Internet of Things Conference - TIoPTS Workshop*, 2010.

[9] J.-M. HO, "A versatile suite of strong authenticated key agreement protocols for body area networks," in *8th International Conference on Wireless Communication and Mobile Computing*. IEEE, 2012.

[10] A. Kumar *et al.*, "A comparative study of secure device pairing methods," *Pervasive and Mobile Computing*, vol. 5, no. 6, 2009.

[11] N. Saxena *et al.*, "Secure device pairing based on a visual channel," in *IEEE Symposium on Security and Privacy*, 2006.

[12] J. McCune *et al.*, "Seeing-is-believing: using camera phones for human-verifiable authentication," in *IEEE Symposium on Security and Privacy*, 2005.

[13] M. Goodrich *et al.*, "Loud and clear: Human-verifiable authentication based on audio," in *26th IEEE International Conference on Distributed Computing Systems*. IEEE, 2006.

[14] C. Soriente, G. Tsudik, and E. Uzun, "Hapadep: Human-assisted pure audio device pairing," in *Information Security*. Springer Berlin Heidelberg, 2008, vol. 5222, pp. 385–400.

[15] R. Mayrhofer and H. Gellersen, "Shake well before use: Authentication based on accelerometer data," in *Pervasive Computing*. Springer Berlin Heidelberg, 2007, vol. 4480, pp. 144–161.

[16] S. Chetan *et al.*, "Mobile gaia: a middleware for ad-hoc pervasive computing," in *IEEE Consumer Communications and Networking Conference (CCNC)*, 2005.

[17] M. Ahler *et al.*, "Detailed design of the amigo middleware core security & privacy , content distribution, data storage," IST Amigo Project, Tech. Rep., September 2005.

[18] S. Unger, S. Pfeiffer, and D. Timmermann, "How much security for switching a light bulb – the soa way," in *Wireless Communications and Mobile Computing Conference (IWCMC), 2012 8th International*, 2012.

[19] S. Pöhlsen *et al.*, "A dpws-based architecture for medical device interoperability," in *World Congress on Medical Physics and Biomedical Engineering*, 2009.

[20] G. Moritz *et al.*, "Devices profile for web services in wireless sensor networks: Adaptations and enhancements," in *Emerging Technologies Factory Automation, 2009. ETFA 2009. IEEE Conference on*, 2009.

[21] ——, "Web services on deeply embedded devices with real-time processing," in *Emerging Technologies and Factory Automation, 2008. ETFA 2008. IEEE International Conference on*, 2008.

[22] OASIS, "Web services dynamic discovery version 1.1," July 2009.

[23] Y. Ben Saied, A. Olivereau, and M. Laurent, "A distributed approach for secure m2m communications," in *The Fifth IFIP International Conference on New Technologies and Security (NTMS 2012)*, Mai 2012.

[24] C. Eckert, *IT-Sicherheit, Konzepte - Verfahren - Protokolle*, 5th ed. Oldenbourg Wissenschaftsverlag GmbH, 2008.

[25] I. C. Society, "Ieee standard for local and metropolitan are networks – part 15.6: Wireless body area networks," 2012.

[26] S. Unger. (2013) Sebastian unger / ws4d mobile authenticator | gitlab. [Online]. Available: http://gitlab.amd.e-technik.uni-rostock.de/sebastian.unger/ws4d-mobile-authenticator/wikis/home

[27] Materna GmbH. (2013) Jmeds (java multi edition dpws stack) — free development software downloads at sourceforge.net. [Online]. Available: http://sourceforge.net/projects/ws4d-javame/

[28] The Legion of the Bouncy Castle. (2013) bouncycastle.org. [Online]. Available: http://www.bouncycastle.org/

[29] GitHub, Inc. (2013) rtyley/spongycastle. [Online]. Available: https://github.com/rtyley/spongycastle

[30] S. Unger, S. Pfeiffer, and D. Timmermann, "Dethroning transport layer security in the embedded world," in *5th International Conference on New Technologies, Mobility and Security (NTMS)*, 2012.