Network Working Group
Internet-Draft
Intended status: Informational
Expires: April 26, 2019

D. Kaiser

C. Huitema
Private Octopus Inc.

October 23, 2018

Device Pairing Design Issues
draft-ietf-dnssd-pairing-info-02

Abstract

This document discusses issues and problems occuring in the design of device pairing mechanism. It presents experience with existing pairing systems and general user interaction requirements to make the case for "short authentication strings". It then reviews the design of cryptographic algorithms designed to maximise the robustness of the short authentication string mechanisms, as well as implementation considerations such as integration with TLS.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 26, 2019.

Table of Contents

1. Introduction

To engage in secure and privacy preserving communication, hosts need to differentiate between authorized peers, which must both know about the host's presence and be able to decrypt messages sent by the host, and other peers, which must not be able to decrypt the host's messages and ideally should not be aware of the host's presence. The necessary relationship between host and peer can be established by a centralized service, e.g. a certificate authority, by a web of trust, e.g. PGP, or -- without using global identities -- by device pairing.

The general pairing requirement is easy to state: establish a trust relation between two entities in a secure manner. But details matter, and in this section we explore the detailed requirements that will guide the design of a pairing protocol.

This document does not specify an actual pairing protocol, but it served as the basis for the design of the pairing protocol developed for DNS-SD privacy [I-D.ietf-dnssd-pairing]. The requirement of a

pairing system for private discovery are analyzed in part in [I-D.ietf-dnssd-prireq].

1.1. Document Organization

NOTE TO RFC EDITOR: remove or rewrite this section before publication.

This document results from a split of an earlier pairing draft that contained two parts. The first part, presented the pairing need, and the list of requirements that shall be met. The second part presented the design is the actual specification of the protocol.

In his early review, Steve Kent observed that the style of the first part seems inappropriate for a standards track document, and suggested that the two parts should be split into two documents, the first part becoming an informational document, and the second focusing on standard track specification of the protocol, making reference to the informational document as appropriate.

The working group approved this split.

2. Protocol Independent Secure Pairing

Many pairing protocols have already been developed, in particular for the pairing of devices over specific wireless networks. For example, the current Bluetooth specifications include a pairing protocol that has evolved over several revisions towards better security and usability [BTLEPairing]. The Wi-Fi Alliance defined the Wi-Fi Protected Setup process to ease the setup of security-enabled Wi-Fi networks in home and small office environments [WPS]. Other wireless standards have defined or are defining similar protocols, tailored to specific technologies.

In this document we provide background and discuss the design of a manually authenticated pairing protocol that is independent of the underlying network protocol stack. We discuss (1) means allowing the two parties engaged in the pairing to discover each other in an existing unsecured network -- e.g. means for learning about the network parameters of the respective other device -- which allows them to establish a connection; (2) agreeing on a shared secret via this connection; and (3) manually authenticating this secret. For our discussion and our secure pairing protocol specification [I-D.ietf-dnssd-pairing], we assume an IP based unsecured network. With little adaption, this pairing mechanism can be used on other protocol stacks as well.

We limit the goal of the protocol to the establishment of a shared secret between two parties. Once that secret has been established, it can trivially be used to secure the exchange of other informations, such as for example public keys and certificates.

3. Identity Assurance

The parties in the pairing must be able to identify each other. To put it simply, if Alice believes that she is establishing a pairing with Bob, she must somehow ensure that the pairing is actually established with Bob, and not with some interloper like Eve or Nessie. Providing this assurance requires designing both the protocol and the user interface (UI) with care.

Consider for example an attack in which Eve tricks Alice into engaging in a pairing process while pretending to be Bob. Alice must be able to discover that something is wrong, and refuse to establish the pairing. The parties engaged in the pairing must at least be able to verify their identities, respectively.

4. Manual Authentication

Because the pairing protocol is executed without prior knowledge, it is typically vulnerable to "Man-in-the-Middle" attacks. While Alice is trying to establish a pairing with Bob, Eve positions herself in the middle. Instead of getting a pairing between Alice and Bob, both Alice and Bob get paired with Eve. Because of this, the protocol requires specific features to detect Man-in-the-Middle attacks, and if possible resist them.

This section discusses existing techniques that are used in practice for manually authenticating a Diffie-Hellman key exchange, and Section 5 provides a layman description of the MiTM problem and countermeasures. A more in depth exploration of manually authenticated pairing protocols may be found in [NR11] and [K17].

4.1. Short PIN Proved Inadequate

The initial Bluetooth pairing protocol relied on a four digit PIN, displayed by one of the devices to be paired. The user read that PIN and provided it to the other device. The PIN was then used in a Password Authenticated Key Exchange. Wi-Fi Protected Setup [WPS] offered a similar option. There were various attacks against the actual protocol; some of the problems were caused by issues in the protocol, but most were tied to the usage of short PINs.

In the reference implementation, the PIN is picked at random by the paired device before the beginning of the exchange. But this

requires that the paired device is capable of generating and displaying a four digit number. It turns out that many devices cannot do that. For example, an audio headset does not have any display capability. These limited devices ended up using static PINs, with fixed values like "0000" or "0001".

Even when the paired device could display a random PIN, that PIN had to be copied by the user on the pairing device. It turns out that users do not like copying long series of numbers, and the usability thus dictated that the PINs be short -- four digits in practice. But there is only so much assurance as can be derived from a four digit key.

The latest revisions of the Bluetooth Pairing protocol [BTLEPairing] do not include the short PIN option anymore. The PIN entry methods have been superseded by the simple "just works" method for devices without displays, and by a procedure based on an SAS (short authentication string) when displays are available.

A further problem with these PIN based approaches is that -- in contrast to SASes -- the PIN is a secret instrumental in the security algorithm. To guarantee security, this PIN would have to be transmitted via a secure out-of-band channel.

4.2. Push Buttons Just Work, But Are Insecure

Some devices are unable to input or display any code. The industry more or less converged on a "push button" solution. When the button is pushed, devices enter a "pairing" mode, during which they will accept a pairing request from whatever other device connects to them.

The Bluetooth Pairing protocol [BTLEPairing] denotes that as the "just works" method. It does indeed work, and if the pairing succeeds the devices will later be able to use the pairing keys to authenticate connections. However, the procedure does not provide any protection against MitM attacks during the pairing process. The only protection is that pushing the button will only allow pairing for a limited time, thus limiting the opportunities of attacks.

As we set up to define a pairing protocol with a broad set of applications, we cannot limit ourselves to an insecure "push button" method. But we probably need to allow for a mode of operation that works for input-limited and display limited devices.

4.3. Short Range Communication

Many pairing protocols that use out-of-band channels have been defined. Most of them are based on short range communication systems, where the short range limits the feasibility for attackers to access the channels. Example of such limited systems include for example:

- QR codes, displayed on the screen of one device, and read by the camera of the other device.
- Near Field Communication (NFC) systems, which provides wireless communication with a very short range.
- Sound systems, in which one systems emits a sequence of sounds or ultrasounds that is picked by the microphone of the other system.

A common problem with these solutions is that they require special capabilities that may not be present in every device. Another problem is that they are often one-way channels.

The pairing protocols should not rely on the secrecy of the out-of-band channels; most of these out-of-band channels do not provide confidentiality. QR codes could be read by third parties. Powerful radio antennas might be able to interfere with NFC. Sensitive microphones might pick the sounds. However, a property that all of these channels share is authenticity, i.e. an assurance that the data obtained over the out-of-band channel actually comes from the other party. This is because these out-of-band channels involve the user transmitting information from one device to the other. We will discuss the specific case of QR codes in Section 8.

4.4. Short Authentication Strings

The evolving pairing protocols seem to converge towards using Short Authentication Strings and verifying them via the "compare and confirm" method. This is in line with academic studies, such as [KFR09] or [USK11], and, from the users' perspective, results in a very simple interaction:

1. Alice and Bob compare displayed strings that represent a fingerprint of the afore exchanged pairing key.
2. If the strings match, Alice and Bob accept the pairing.

Most existing pairing protocols display the fingerprint of the key as a 6 or 7 digit number. Usability studies show that this method gives good results, with little risk that users mistakenly accept two

different numbers as matching. However, the authors of [USK11] found that people had more success comparing computer generated sentences than comparing numbers. This is in line with the argument in [XKCD936] to use sequences of randomly chosen common words as passwords. On the other hand, standardizing strings is more complicated than standardizing numbers. We would need to specify a list of common words, and the process to go from a binary fingerprint to a set of words. We would need to be concerned with internationalization issues, such as using different lists of words in German and in English. This could require the negotiation of word lists or languages inside the pairing protocols.

In contrast, numbers are easy to specify, as in "take a 20 bit number and display it as an integer using decimal notation".

4.5. Revisiting the PIN versus SAS discussion

In section Section 4.1 we presented the drawbacks of using short pins. One could object that many of the technical issues could be overcome by use of better PAKE algorithms, or by supporting longer PIN. And one could also argue that if PIN based pairing algorithms suffer from failure modes such as static PIN configuration, SAS based protocols are vulnerable to SAS bypass.

The SAS bypass argument is rooted in the psychology of users. In practice, pairing processes can be stressful. The user has to discover on each device the proper combination of key entries that brings up the required pairing UI, will be anxious and eager to complete the procedure, and may well be predisposed to click "OK" in the final stage of the algorithm without actually verifying the SAS. Some users may bypass the required comparison step, because they just want to be done with the pairing.

An advantage of PIN based processes is that they cannot be bypassed. The user must enter the PIN before continuing. Also, once the PIN is entered, everything is automatic. The user does not need to input more data, or press any additional button. PIN based protocols would be a great fit for the QR-code based interaction. One device would display a QR code that contains the PIN. Once the QR code is scanned by the other device, the process is automated.

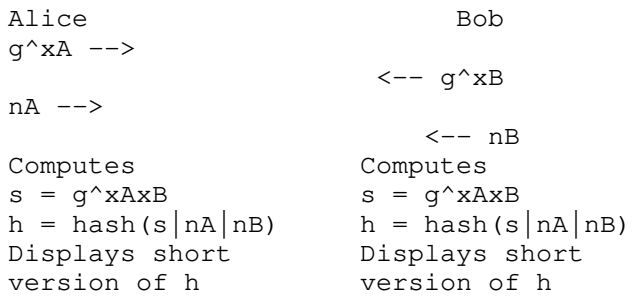
QR based PIN entry may be user friendly, but one of the arguments developed in Section 4.1 still holds. Let’s assume that an adversary somehow obtains the PIN, maybe by scanning the QR code at a distance. That adversary could mount MITM or impersonation attacks, and compromise the pairing process. It is thus very important to ensure that the PIN is only readable by the user doing the pairing.

We could also argue that the SAS bypass failure mode may be mitigated by specific user designs. For example, instead of just clicking OK, the user could be required to enter the SAS displayed by the other device. This requires about the same interactions as a PIN based process, and it would be slightly safer because the SAS does not have to be kept secret once the keys have been exchanged.

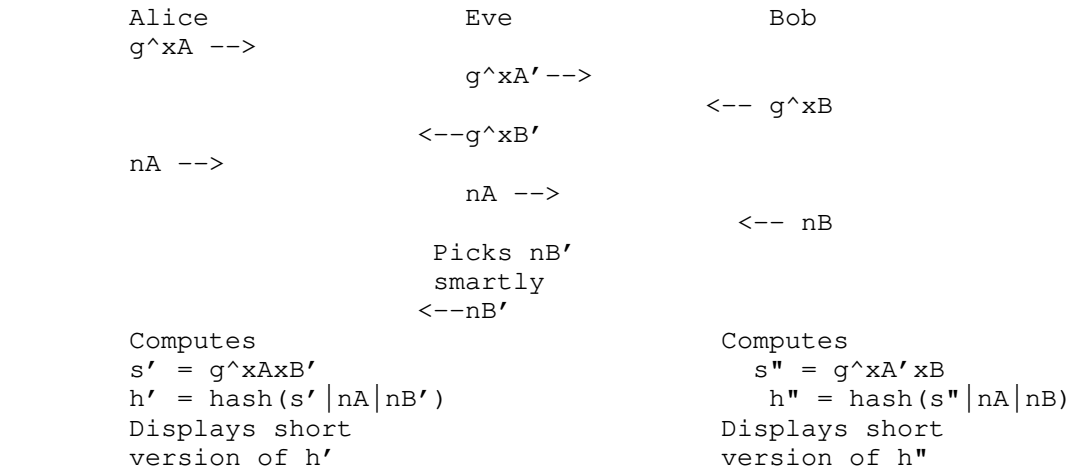
If we summarize the debate, we see that both SAS and PIN based solutions have failure modes depending on implementations. In the SAS mode, the failure happens when the UI does not force the user to copy the PIN and relies on a simple "OK to continue" dialog. In the PIN mode, the failure happens when the device fails to generate a random PIN for each session, and comes pre-programmed with a simple static PIN of "0000" or "0001".

5. Resist Cryptographic Attacks

It is tempting to believe that once two peers are connected, they could create a secret with a few simple steps, such as for example (1) exchange two nonces, (2) hash the concatenation of these nonces with the shared secret that is about to be established, (3) display a short authentication string composed of a short version of that hash on each device, and (4) verify that the two values match. This naive approach might yield the following sequence of messages:



If the two short hashes match, Alice and Bob are supposedly assured that they have computed the same secret, but there is a problem. Let’s redraw the same message flow, this time involving the attacker Eve:



In order to pick a nonce nB' that circumvents this naive security measure, Eve runs the following algorithm:

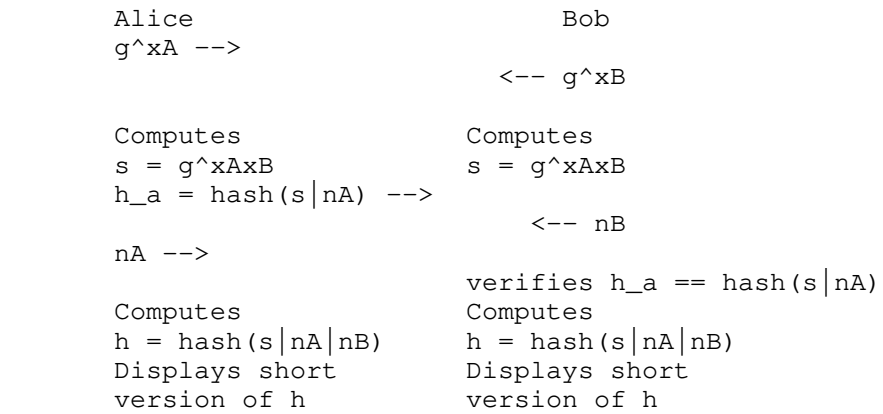
```
s' = g^xAxB'
s" = g^xA'xB
repeat
   pick a new version of nB'
   h' = hash(s'|nA|nB')
   h" = hash(s"|nA|nB)
until the short version of h'
matches the short version of h"
```

Running this algorithm will take O(2^b) iterations on average (assuming a uniform distribution), where b is the bit length of the SAS. Since hash algorithms are fast, it is possible to try millions of values in less than a second. If the short string is made up of fewer than 6 digits, Eve will find a matching nonce quickly, and Alice and Bob will hardly notice the delay. Even if the matching string is as long as 8 letters, Eve will probably find a value where the short versions of h' and h" are close enough, e.g. start and end with the same two or three letters. Alice and Bob may well be fooled.

Eve could also utilize the fact that she may freely choose the whole input for the hash function and thus choose g^xA' and g^xB' so that an arbitrary collision (birthday attack) instead of a second preimage is sufficient for fooling Alice and Bob.

The classic solution to such problems is to "commit" a possible attacker to a nonce before sending it. This commitment can be

realized by a hash. In the modified exchange, Alice sends a secure hash of her nonce before sending the actual value:



Alice will only disclose nA after having confirmation from Bob that hash(nA) has been received. At that point, Eve has a problem. She can still forge the values of the nonces, but she needs to pick the nonce nA' before the actual value of nA has been disclosed. Eve would still have a random chance of fooling Alice and Bob, but it will be a very small chance: one in a million if the short authentication string is made of 6 digits, even fewer if that string is longer.

Nguyen et al. [NR11] survey these protocols and compare them with respect to the amount of necessary user interaction and the computation time needed on the devices. The authors state that such a protocol is optimal with respect to user interaction if it suffices for users to verify a single b-bit SAS while having a one-shot attack success probability of 2^-b. Further, n consecutive attacks on the protocol must not have a better success probability then n one-shot attacks.

There is still a theoretical problem, if Eve has somehow managed to "crack" the hash function. We can build "defense in depth" by some simple measures. In the design presented above, the hash "h_a" depends on the shared secret "s", which acts as a "salt" and reduces the effectiveness of potential attacks based on pre-computed catalogs. The simplest design uses a concatenation mechanism, but we could instead use a keyed-hash message authentication code (HMAC [RFC2104], [RFC6151]), using the shared secret as a key, since the HMAC construct has proven very robust over time. Then, we can constrain the size of the random numbers to be exactly the same as the output of the hash function. Hash attacks often require padding

the input string with arbitrary data; restraining the size limits the likelyhood of such padding.

6. Privacy Requirements

Pairing exposes a relation between several devices and their owners. Adversaries may attempt to collect this information, for example in an attempt to track devices, their owners, or their social graph. It is often argued that pairing could be performed in a safe place, from which adversaries are assumed absent, but experience shows that such assumptions are often misguided. It is much safer to acknowledge the privacy issues and design the pairing process accordingly.

In order to start the pairing process, devices must first discover each other. We do not have the option of using the private discovery protocol [I-D.ietf-dnssd-privacy] since the privacy of that protocol depends on a pre-existing pairing. In the simplest design, one of the devices will announce a user-friendly name using DNS-SD. Adversaries could monitor the discovery protocol, and record that name. An alternative would be for one device to announce a random name, and communicate it to the other device via some private channel. There is an obvious tradeoff here: friendly names are easier to use but less private than random names. We anticipate that different users will choose different tradeoffs, for example using friendly names if they assume that the environment is safe, and using random names in public places.

During the pairing process, the two devices establish a connection and validate a pairing secret. As discussed in Section 4, we have to assume that adversaries can mount MitM attacks. The pairing protocol can detect such attacks and resist them, but the attackers will have access to all messages exchanged before the validation is performed. It is important to not exchange any privacy sensitive information before that validation. This includes, for example, the identities of the parties or their public keys.

7. Using TLS

The pairing algorithms typically combine the establishment of a shared secret through an [EC]DH exchange with the verification of that secret through displaying and comparing a "short authentication string" (SAS). As explained in Section 5, the secure comparison requires a "commit before disclose" mechanism.

We have three possible designs: (1) create a pairing algorithm from scratch, specifying our own cryptographic protocol; (2) use an [EC]DH version of TLS to negotiate a shared secret, export the key to the application as specified in [RFC5705], and implement the "commit

before disclose" and SAS verification as part of the pairing application; or, (3) use TLS, integrate the "commit before disclose" and SAS verification as TLS extensions, and export the verified key to the application as specified in [RFC5705].

When faced with the same choice, the designers of ZRTP [RFC6189] chose to design a new protocol integrated in the general framework of real time communications. We don't want to follow that path, and would rather not create yet another protocol. We would need to reinvent a lot of the negotiation capabilities that are part of TLS, not to mention algorithm agility, post quantum, and all that sort of things. It is thus pretty clear that we should use TLS.

It turns out that there was already an attempt to define SAS extensions for TLS ([I-D.miers-tls-sas]). It is a very close match to our third design option, full integration of SAS in TLS, but the draft has expired, and there does not seem to be any support for the SAS options in the common TLS packages.

In our design, we will choose the middle ground option -- use TLS for [EC]DH, and implement the SAS verification as part of the pairing application. This minimizes dependencies on TLS packages to the availability of a key export API following [RFC5705]. We will need to specify the hash algorithm used for the SAS computation and validation, which carries some of the issues associated with "designing our own crypto". One solution would be to use the same hash algorithm negotiated by the TLS connection, but common TLS packages do not always make this algorithm identifier available through standard APIs. A fallback solution is to specify a state of the art keyed MAC algorithm.

8. QR codes

In Section 4.3, we reviewed a number of short range communication systems that can be used to facilitate pairing. Out of these, QR codes stand aside because most devices that can display a short string can also display the image of a QR code, and because many pairing scenarios involve cell phones equipped with cameras capable of reading a QR code.

QR codes are displayed as images. An adversary equipped with powerful cameras could read the QR code just as well as the pairing parties. If the pairing protocol design embedded passwords or pins in the QR code, adversaries could access these data and compromise the protocol. On the other hand, there are ways to use QR codes even without assuming secrecy.

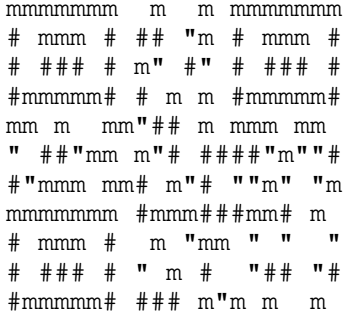
QR codes could be used at two of the three stages of pairing: Discovering the peer device, and authenticating the shared secret. Using QR codes provides advantages in both phases:

- Typical network based discovery involves interaction with two devices. The device to be discovered is placed in "server" mode, and waits for requests from the network. The device performing the discovery retrieves a list of candidates from the network. When there is more than one such candidate, the device user is expected to select the desired target from a list. In QR code mode, the discovered device will display a QR code, which the user will scan using the second device. The QR code will embed the device's name, its IP address, and the port number of the pairing service. The connection will be automatic, without relying on the network discovery. This is arguably less error-prone and safer than selecting from a network provided list.

- SAS based agreement involves displaying a short string on each device's display, and asking the user to verify that both devices display the same string. In QR code mode, one device could display a QR code containing this short string. The other device could scan it and compare it to the locally computed version. Because the procedure is automated, there is no dependency on the user diligence at comparing the short strings.

Offering QR codes as an alternative to discovery and agreement is straightforward. If QR codes are used, the pairing program on the server side might display something like:

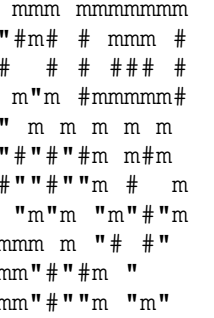
Please connect to "Bob's phone 359"
or scan the following QR code:



If Alice's device is capable of reading the QR code, it will just scan it, establishes a connection, and run the pairing protocol. After the protocol messages have been exchanged, Bob's device will

display a new QR code, encoding the hash code that should be matched. The UI might look like this:

Please scan the following QR code,
or verify that your device displays
the number: 388125



Did the number match (Yes/No)?

With the use of QR code, the pairing is established with little reliance on user judgment, which is arguably safer.

9. Intra User Pairing and Transitive Pairing

There are two usage modes for pairing: inter-user, and intra-user. Users have multiple devices. The simplest design is to not distinguish between pairing devices belonging to two users, e.g., Alice's phone and Bob's phone, and devices belonging to the same user, e.g., Alice's phone and her laptop. This will most certainly work, but it raises the problem of transitivity. If Bob needs to interact with Alice, should he install just one pairing for "Alice and Bob", or should he install four pairings between Alice phone and laptop and Bob phone and laptop? Also, what happens if Alice gets a new phone?

One tempting response is to devise a synchronization mechanism that will let devices belonging to the same user share their pairings with other users. But it is fairly obvious that such service will have to be designed cautiously. The pairing system relies on shared secrets. It is much easier to understand how to manage secrets shared between exactly two parties than secrets shared with an unspecified set of devices.

Transitive pairing raises similar issues. Suppose that a group of users wants to collaborate. Will they need to set up a fully connected graph of pairings using the simple peer-to-peer mechanism,

or could they use some transitive set, so that if Alice is connected with Bob and Bob with Carol, Alice automatically gets connected with Carol? Such transitive mechanisms could be designed, e.g. using a variation of Needham-Scroeder symmetric key protocol [NS1978], but it will require some extensive work. Groups can of course use simpler solution, e.g., build some star topology.

Given the time required, intra-user pairing synchronization mechanisms and transitive pairing mechanisms are left for further study.

10. Security Considerations

This document lists a set of security issues that have to be met by pairing protocols, but does not specify any protocol.

11. IANA Considerations

This draft does not require any IANA action.

12. Acknowledgments

We would like to thank Steve Kent for a detailed early review of an early draft of this document. Both him and Ted Lemon were influential in the decision to separate the analysis of pairing requirements from the specification of pairing protocol in [I-D.ietf-dnssd-pairing]

13. Informative References

[BTLEPairing]
Bluetooth SIG, "Bluetooth Low Energy Security Overview", 2016, <https://developer.bluetooth.org/TechnologyOverview/Pages/LE-Security.aspx>.

[I-D.ietf-dnssd-pairing]
Huitema, C. and D. Kaiser, "Device Pairing Using Short Authentication Strings", draft-ietf-dnssd-pairing-04 (work in progress), April 2018.

[I-D.ietf-dnssd-prireq]
Huitema, C., "DNS-SD Privacy and Security Requirements", draft-ietf-dnssd-prireq-00 (work in progress), September 2018.

[I-D.ietf-dnssd-privacy]
Huitema, C. and D. Kaiser, "Privacy Extensions for DNS-SD", draft-ietf-dnssd-privacy-04 (work in progress), April 2018.

[I-D.miers-tls-sas]
Miers, I., Green, M., and E. Rescorla, "Short Authentication Strings for TLS", draft-miers-tls-sas-00 (work in progress), February 2014.

[K17] Kaiser, D., "Efficient Privacy-Preserving Configurationless Service Discovery Supporting Multi-Link Networks", 2017, <http://nbn-resolving.de/urn:nbn:de:bsz:352-0-422757>.

[KFR09] Kainda, R., Flechais, I., and A. Roscoe, "Usability and Security of Out-Of-Band Channels in Secure Device Pairing Protocols", DOI: 10.1145/1572532.1572547, SOUPS 09, Proceedings of the 5th Symposium on Usable Privacy and Security, Mountain View, CA, January 2009.

[NR11] Nguyen, L. and A. Roscoe, "Authentication protocols based on low-bandwidth unspoofable channels: a comparative survey", DOI: 10.3233/JCS-2010-0403, Journal of Computer Security, Volume 19 Issue 1, Pages 139-201, January 2011.

[NS1978] Needham, R. and M. Schroeder, ". Using encryption for authentication in large networks of computers", Communications of the ACM 21 (12): 993-999, DOI: 10.1145/359657.359659, December 1978.

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, DOI 10.17487/RFC2104, February 1997, <https://www.rfc-editor.org/info/rfc2104>.

[RFC5705] Rescorla, E., "Keying Material Exporters for Transport Layer Security (TLS)", RFC 5705, DOI 10.17487/RFC5705, March 2010, <https://www.rfc-editor.org/info/rfc5705>.

[RFC6151] Turner, S. and L. Chen, "Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms", RFC 6151, DOI 10.17487/RFC6151, March 2011, <https://www.rfc-editor.org/info/rfc6151>.

[RFC6189] Zimmermann, P., Johnston, A., Ed., and J. Callas, "ZRTP: Media Path Key Agreement for Unicast Secure RTP", RFC 6189, DOI 10.17487/RFC6189, April 2011, <https://www.rfc-editor.org/info/rfc6189>.

[USK11] Uzun, E., Saxena, N., and A. Kumar, "Pairing devices for social interactions: a comparative usability evaluation", DOI: 10.1145/1978942.1979282, Proceedings of the International Conference on Human Factors in Computing Systems, CHI 2011, Vancouver, BC, Canada, May 2011.

[WPS] Wi-Fi Alliance, "Wi-Fi Protected Setup", 2016, <http://www.wi-fi.org/discover-wi-fi/wi-fi-protected-setup>.

[XKCD936] Munroe, R., "XKCD: Password Strength", 2011, <https://www.xkcd.com/936/>.

Authors' Addresses

Daniel Kaiser
Esch-sur-Alzette 4360
Luxembourg

Email: daniel@kais3r.de

Christian Huitema
Private Octopus Inc.
Friday Harbor, WA 98250
U.S.A.

Email: huitema@huitema.net