Peltonen, Aleksi; Sethi, Mohit; Aura, Tuomas

Formal verification of misbinding attacks on secure device pairing and bootstrapping

Published in:
Journal of Information Security and Applications

DOI:
10.1016/j.jisa.2020.102461

Published: 01/04/2020

Document Version
Publisher's PDF, also known as Version of record

Journal of Information Security and Applications 51 (2020) 102461

Contents lists available at ScienceDirect

Journal of Information Security and Applications

journal homepage: www.elsevier.com/locate/jisa

Formal verification of misbinding attacks on secure device pairing and bootstrapping☆

Aleksi Peltonen[a,*], Mohit Sethi[a,b], Tuomas Aura[a]

[a] Aalto University, Finland
[b] NomadicLab, Ericsson Research, Finland

ARTICLE INFO

Article history:
Available online 13 February 2020

Keywords:
Device pairing
IoT Security
Misbinding attack
Bluetooth
EAP-NOOB
DPP
ProVerif
Formal modelling

ABSTRACT

In identity misbinding attacks against authenticated key-exchange protocols, a legitimate but compromised participant manipulates the honest parties so that the victim becomes unknowingly associated with a third party. These attacks are well known, and resistance to misbinding is considered a critical requirement for security protocols on the Internet. In the context of device pairing, on the other hand, the attack has received little attention outside the trusted-computing community. This paper points out that most device pairing protocols are vulnerable to misbinding. Device pairing protocols are characterized by lack of a-priory information, such as identifiers and cryptographic roots of trust, about the other endpoint. Therefore, the devices in pairing protocols need to be identified by the user's physical access to them. As case studies for demonstrating the misbinding vulnerability, we use Bluetooth and protocols that register new Internet of Things (IoT) devices to authentication servers on wireless networks. We have implemented the attacks. We also show how the attacks can be found in formal models of the protocols with carefully formulated correspondence assertions. The formal analysis yields a new type of double misbinding attack. While pairing protocols have been extensively modelled and analyzed, misbinding seems to be an aspect that has not previously received sufficient attention. Finally, we discuss potential ways to mitigate the threat and its significance to security of pairing protocols.

1. Introduction

Secure device pairing is a process that bootstraps secure communication between two physical devices. It is a type of authenticated key-exchange, but with the special characteristic that the endpoints are physical devices which the user can see or touch directly. Unlike most security protocols, secure device pairing does not require pre-established cryptographic credentials or security infrastructure. Instead, the user acts as an out-of-band communications channel or as a trusted party that provides the initial security.

The focus of this paper is on *identity-misbinding* [2] or *unknown-key-share* attacks [3] where the wrong endpoints are paired with each other. These attacks depend on one of the user's devices being compromised, and they do not violate the basic secrecy goals. Nevertheless, such vulnerabilities have been considered unacceptable in network security protocols. Our main message is that most device-pairing protocols are vulnerable to the misbinding attacks, and they may not always be avoidable. As we will argue, the vulnerability is not caused by technical errors in the protocol design; rather, it arises from the lack of verifiable identifiers in situations where the endpoint identity is defined by the user's physical access to the device.

This paper is not intended to sound alarm but rather to bring clarity and understanding to a previously ignored question about device authentication. Our contributions are the following: (i) bringing attention to identity-misbinding vulnerabilities in device-pairing and bootstrapping protocols, (ii) detailed analysis and characterization of the vulnerabilities, (ii) examples of concrete, implemented attacks against Bluetooth Secure Simple Pairing and the proposed EAP-NOOB and DPP protocols for registering new devices to a network, (iii) formal specification of the violated security property as a correspondence assertion that takes into account

☆ This article is an extended version of a paper that appeared in ASIACCS 2019 [1]. It includes an extended background section, rewritten EAP-NOOB protocol example, a section describing misbinding attacks on the Wi-Fi Alliance DPP protocol, and a full formal model of misbinding in Bluetooth pairing.

* Corresponding author.
E-mail addresses: aleksi.peltonen@aalto.fi (A. Peltonen), mohit.sethi@aalto.fi (M. Sethi), tuomas.aura@aalto.fi (T. Aura).

https://doi.org/10.1016/j.jisa.2020.102461

the user intention, and (iv) balanced discussion of the impact of the attacks and potential countermeasures. The significance of our work arises from the wide deployment of the vulnerable pairing protocols in everyday applications.

The rest of the paper is structured as follows. Section 2 discusses the relevant state of the art in security protocols and attacks. Section 3 explains the misbinding attack against device-pairing protocols, and Section 4 describes a similar attack when registering new IoT devices to an authentication server or wireless network. In Section 5, we show how to model the attacks and the related security properties. We also discover a new variant of the misbinding attack. Section 6 considers the potential solutions. Section 7 discusses the significance of the results, and Section 8 concludes the paper.

2. Background

2.1. Security protocol attacks and correspondence assertions

The goal of authenticated key exchange is to establish a shared cryptographic key between two or more communication endpoints, which then use the shared key for protecting communication integrity and confidentiality. Authenticated key-exchange protocols should be secure against the so-called *Dolev-Yao attacker* [4], which is able to spoof, intercept and modify messages in the network in arbitrary ways, except when it lacks the necessary cryptographic keys. The attacker may impersonate one of the communication endpoints or set itself as a *man in the middle* (MitM) between them. Even carefully designed protocols have been found to be vulnerable to *forwarding and interleaving attacks* [5,6], in which the attacker itself is a legitimate participant in the protocol but can mislead others by cleverly replaying messages. In closed systems, such *insider attacks* could sometimes be tolerated, but in large systems and open networks such as the Internet and the Internet of Things, there always are some malicious "insiders". Thus, modern security protocols are required to be immune to these attacks.

The authentication goals of key-exchange protocols can be defined in terms of matching or agreement between the records made by different endpoints on the protocol execution [7,8]. The same goals can be stated as *correspondence assertions* [9]. These assertions define relations between later and earlier events in the protocol execution. For example, a common assertion is that, if Alice accepts a session key to be used with Bob, both Alice and Bob must have previously declared an intent to create such a session key. This way, we can make global assertions about the events that should or should not take place in a distributed system. *Injective* correspondence further requires that each such declaration of intent can result in at most one accepted session key. When formalized, the correspondence assertions are typically parameterized with all the knowledge of protocol inputs and parameters which should match between the events and endpoints.

An advantage of specifying security properties as correspondence assertions is that, in addition to basic authentication properties, the assertions capture the protocol designer's implicit expectations about its execution and, thus, can help to detect subtle flaws that might otherwise go unnoticed.

2.2. Identity misbinding

In this paper, we are interested in failures of authentication protocols where the following two conditions hold:

1. One of the protocol endpoints is confused about the identity of the other communication endpoint.
2. The confusion is caused by malicious behavior by one of the intended communication endpoints.

Consider the high-level scenario of Fig. 1(a). *A* is a client computer that wants to connect to the server *E* and, therefore, initiates the cryptographic authentication protocol. In security-protocol terminology, *A* is the *initiator* and *E* is the *responder*. Unfortunately, the responder *E* is malicious and tricks *A* to connect to another responder *B* instead. The result is that *A* believes it has a secure connection to *E* while, in reality, the connection is with *B*. The responder *B* in this scenario can be entirely honest, and *B* correctly believes it is talking with *A*.

This attack can work only if there is a weakness in the authentication protocol. As we will explain below, most modern authentication protocols are designed to prevent such attacks. Nevertheless, let us persist on exploring the potential failures.

In the above scenario, the initiator is confused about the identity of the responder. It is equally possible that the responder is confused about the identity if the initiator. This second scenario is shown in Fig. 1(b). *B* is a server that believes it is accepting a connection from client *E*, but the malicious *E* exploits a weakness in the authentication protocol and tricks *B* into accepting a connection from *A* instead. As the result, *B* believes that it has a secure connection with *E* while, in reality, the connection is with *A*.

Next, we take a look at a concrete protocol that is vulnerable to the above attacks. Fig. 2 shows two well-known attacks against a badly-authenticated Diffie-Hellman (DH) key exchange. The attacks correspond to the two scenarios discussed above. In Fig. 2(a), the malicious responder *E* forwards messages from the initiator *A* to another responder *B*. The endpoints *A* and *B* establish a Diffie-Hellman shared secret. However, *A* thinks that it has created a shared secret with *E*. In Fig. 2(b), on the other hand, *E* acts as a kind of man-in-the-middle attacker that modifies messages between the initiator *A* and responder *B*. The Diffie-Hellman shared secret is established between *A* and *B*, but the responder *B* mistakenly thinks it shares the secret with *E*.

The above attacks were identified by Diffie et al. [7], and they have later been given many names including *unknown-key-share* [3] and *identity misbinding* [2]. We will use the name *misbinding* in this paper.

The impact of misbinding attacks is somewhat difficult to understand because it does not compromise secrecy of data. First, the malicious entity *E* does not learn the shared secret and, thus, cannot intercept the data sent over the established secure connection. Indeed, *E* could learn and leak more secrets by completing the protocol normally and becoming a endpoint of the secure channel. Second, one could argue that *B* has correctly authenticated *A*, and more controversially, that *A* has correctly authenticated *E* because *E* is entitled to choose any key share it likes. Nevertheless, something clearly is amiss about the authentication. *A* and *B* have different understanding of who they are communicating with, and one of them has the wrong idea about who shares the session key. This violates correspondence properties that an authenticated key exchange intuitively should have.

2.3. Standard defenses against misbinding

Diffie et al. [7] initially presented the misbinding attack to motivate the station-to-station (STS) protocol. In basic STS, the signatures are encrypted with the Diffie-Hellman session key, and the paper also suggests another variant where a message authentication code (MAC) replaces the encryption. The function of the encryption or MAC is to bind the session key to the signatures, which prevents the attacker, who does not know the session key, from replacing the signatures with its own in the way it does in Fig. 2.

The STS protocol, including both the encryption and MAC variants, is still vulnerable to misbinding attacks if the attacker *E* manages to register *A*'s or *B*'s public signature key as its own. This vulnerability is well known and caused by failure of the certification

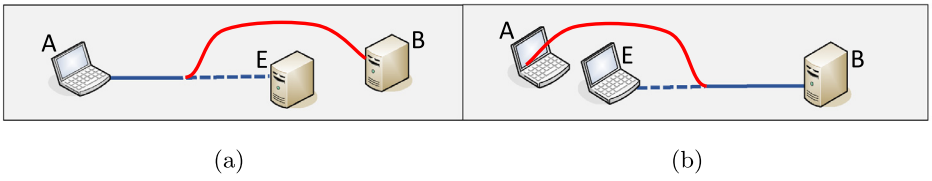

Fig. 1. Misbinding of (a) initiator identity at responder and (b) responder identity at initiator.

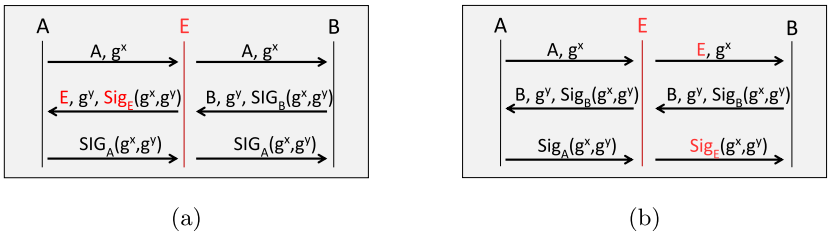

Fig. 2. Misbinding of (a) initiator and (b) responder in badly-authenticated Diffie-Hellman.

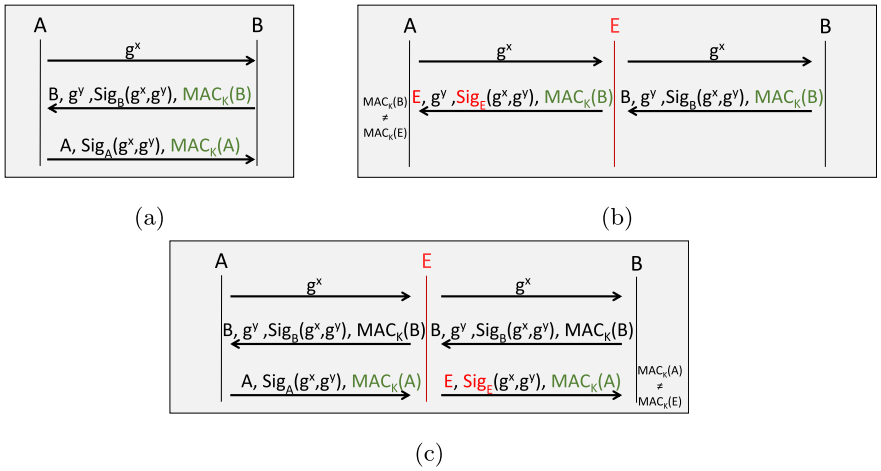

Fig. 3. SIGMA protocols (a) bind identity to the session key with a MAC to detect misbinding against (b) initiator and (c) responder.

authority to verify that the subject possesses the private key. Nevertheless, the dependence on the CA following best practices can and should be avoided, as in the protocols that we discuss next.

The SIGMA protocol family by Krawczyk [2] computes the MAC explicitly on the message sender's identifier, rather than its signature, as seen in Fig. 3(a). The SIGMA protocols are highly influential because they include the IKEv2 key exchange [10] and its predecessors in the IPsec protocol suite. As a consequence, resistance to the misbinding attacks is considered one of the critical requirements for key-exchange protocols designed for the Internet.

The SIGMA defense to misbinding is easy to understand by considering how the potential attacks are detected, as shown in Figs. 3(b) and 3(c). When verifying the MAC, the receiver verifies that the identity claimed by the other holder of the Diffie-Hellman shared secret matches its own expectation. This guarantees the correspondence between the two endpoints' beliefs.

A slightly different approach was taken in the ISO 9798-3 protocol [11], where each endpoint includes the identity of the other endpoint in its signature. This allows he receiver to compare its own understanding of the two identities with that of the sender. All the known defenses against misbinding follow this general pattern where *each endpoint communicates its view of the initiator and responder identities in the protocol messages, and each side compares its own view with that of the other*. Consequently, lack of correspondence between the initiator or the responder views will be detected.

2.4. Device pairing and relay attack

Secure *device pairing* is a bootstrapping process that establishes a secure channel between two previously unassociated devices. These devices often communicate over a short-range wireless channel such as Bluetooth [12], Wi-Fi [13], or Zigbee [14]. While the goals of device pairing are similar to those of any authenticated key-exchange protocol, there is one major difference: the devices typically have no prior security context, such as knowledge of each other's public keys or certificates and identifiers. They may not even have identifiers or an assigned owner before the pairing establishes those. Additionally, the devices may not be able to rely on the availability of trusted infrastructure due to the ad-hoc and local nature of the short-range wireless communication.

Typical device pairing protocols perform a Diffie-Hellman (DH) or an Elliptic Curve Diffie-Hellman (ECDH) key exchange over the in-band wireless channel and then use a *human-assisted out-of-band (OOB) channel* to thwart potential impersonation and man-in-the-middle attackers in the in-band channel. Several researchers have studied the security and usability of device pairing protocols in considerable detail [15–18]. The existing literature assumes a powerful Dolev-Yao type attacker on the in-band wireless channel and an OOB channel that provides some inherent protection for the confidentiality and/or integrity of the data exchanged over it.

Bluetooth (see Section 3.1) is one of the most widely deployed and analyzed wireless technologies. Modern Bluetooth devices use

the *Simple Secure Pairing* (SSP) [12] protocols, although some may be backward compatible with the less secure Legacy Pairing methods. Wireless devices have different input and output capabilities, which is why SSP supports multiple different user interactions and is actually a family of key-exchange protocols. In the *numeric-comparison* mode, the user is asked to compare six-digit codes on two device displays while, in the *out-of-band* mode, the user delivers similar verification information securely from one device to another. Either way, the out-of-band communication by the user prevents man-in-the-middle attacks on the ECDH key exchange that takes place over the in-band wireless channel. There is also a *just-works* mode for devices that support neither output nor input of six-digit codes. Obviously, this mode lacks secure authentication.

Research literature on Bluetooth security discusses several attacks that are relevant to pairing protocols in general. It may be possible to spy on the OOB channel or to misrepresent the device capabilities so that the devices negotiate the insecure just-works mode [19]. The attacker can trick remote devices into believing that they are in direct communication by *relaying* unmodified protocol messages between their locations [20]. In the legacy version of Bluetooth where session encryption was not mandatory, relaying of the authentication messages could result in pairing of the wrong devices. In modern protocols, this attack is relevant when the primary goal is the device authentication and not protection of the following communication, for example, when a Bluetooth device is used as a door key or as a location beacon. Moreover, the Bluetooth just-works mode can lead to accidental or maliciously induced association with a wrong peer device, as noted among others by Suomalainen et al. [21]. If the device supports multiple simultaneous key exchanges, there can be confusion between the resulting sessions [22]. The end result in these attacks is akin to identity misbinding because the reality of the created security associations does not correspond to the device's or user's perception.

Poorly designed internal architecture of a Bluetooth endpoint, such as a mobile phone, may also lead to attacks. Naveed et al. [23] describe how malicious applications on an Android smartphone can hijack connections from attached Bluetooth (medical) devices in order to steal data. The problem arises from the fact that the Android permission and security model allows any application with the Bluetooth permission to communicate with all external Bluetooth-paired devices. A more general lesson that we can draw from the paper is that it is important to pay attention to malicious insiders, such as untrusted apps, residing in the endpoint devices, which may be able to interfere with the communication without fully compromising the device.

The pairing protocols critically depend on user actions, such as comparing or delivering codes. Ellison [24] introduced the concept of security *ceremonies* where the users are participants to the protocol and their actions are specified, modelled and analyzed just like those of the communicating endpoints. Carlos et al. [25] use Bluetooth as an example for reasoning about basic security properties of a security ceremony. We will continue this line of investigation by including the user and user actions in our models of pairing protocols including Bluetooth SSP.

2.5. Trusted computing and cuckoo attack

The published work closest to ours comes from the trusted-computing community. In trusted computing, a computer or a mobile device incorporates a secure hardware component that is certified by the manufacturer and acts as a trusted entity inside the device. The most common secure hardware component is a *trusted platform module* (TPM) [26], which supervises the boot process of the device and either enforces secure boot or measures (as a cumulative hash value) the loaded software. The latter case is also called dynamic root of trust for measurement (DRTM). The latest microprocessors have more advanced trusted execution environments (TEE), such as ARM TrustZone[1] and Intel SGX[2], which allow trusted software to be isolated and launched after the device has booted. A common feature in these technologies is that, in addition to enforcing some security policies inside the computer, they can *attest* the integrity of the device and its software configuration to an external verifier. This could allow, for example, the user to cryptographically verify the integrity of a cryptocurrency wallet before storing high-value secrets to it. The attestation naturally needs to be cryptographically linked to a secure communication channel [27] with the verifier.

Parno et al. [28] first pointed out the problem that, while users may be able to cryptographically verify that they are communicating with a trusted hardware module and measured software, it is difficult to be certain that they are physically accessing the very device where that module is embedded. In the *cuckoo attack*, the device in the verifier's physical proximity is not actually trusted but tricks the verifier into believing so. The cuckoo device achieves this by forwarding the communication to another device which has the correct configuration and a DRTM for attesting it.

Fink at al. [29] suggest measuring the round-trip times of requests to the trusted device to detect if it is in the proximity of the verifier. Zhang et al. [30] also investigate the problem of a human user distinguishing genuine secure hardware from adversarial devices. They divide the presence attestation into two phases: first, existence checking, which uses the standard remote attestation protocols, and second, residence checking, which provides assurance that the attesting hardware module is, in fact, in the specific physical device. We will return to the suggested mechanisms for residence checking in Section 6. Ding et al. [31] further argue that presence attestation with DRTM differs significantly from device pairing where both devices are trusted. The current paper sets out to investigate whether this is always the case.

2.6. Formal modelling

Formal modelling and model checking are standard methodology in the development and analysis of key-exchange protocols [32–34]. Various protocol flaws have been found with these methods but, perhaps more significantly, formal models are a way to lift the security-protocol design to a higher abstraction level than message formats and state machines, and to define precisely the security properties that the protocol is expected to have.

The model checkers for security protocols are special compared to other formal modelling tools in that, in addition to taking the system design as input, they typically have a built-in model of the Dolev-Yao type powerful attacker, which the researcher does not need to explicitly define. Instead, the researcher has to specify the desired security properties. The model checker then determines whether the attacker is able to play a game against the honest parties and trick them into violating these properties. There is, however one type of attack that the researchers need to explicitly consider: corrupt insiders. The corruption of an insider is often modelled as a previously honest party handing out its secrets and capabilities to the attacker, after which it is subsumed into the attacker.

Jia and Hsu [35] develop a formal model of the Bluetooth SSP for the Murphi model checker [34]. They discuss two potential vulnerabilities in the numeric-comparison authentication mode. First, an *impersonator* device can pretend to be a good one and trick the user into pairing an honest initiator device with it. The example given in the paper is one where the entertainment system in a

[1] https://developer.arm.com/technologies/trustzone.
[2] https://software.intel.com/en-us/sgx.

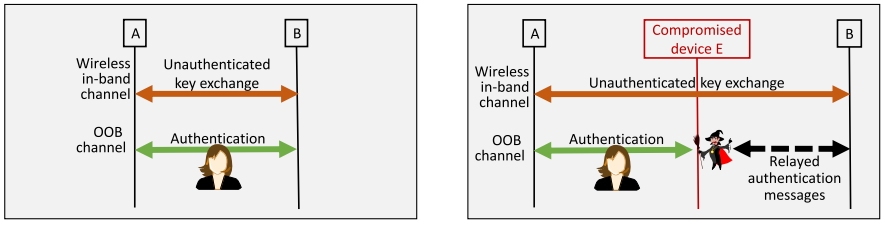

Fig. 4. Device pairing (a) in the normal case and (b) with identity misbinding.

rental car has been replaced with one that is under the adversary's control. Once the unsuspecting user has paired her phone with it, the system can steal confidential data. Second, a *proxy MitM* device can forward the unmodified connection to another device (similar to [20]). While these threats might be considered obvious and unavoidable, the formal analysis focuses our attention to them and enables systematic consideration of the threats.

The most interesting idea of Jia and Hsu for us is the notion of *intention preservation*. It means that the initiating device is paired with the device with which the user originally intended to pair it, even if the non-initiating device belongs to an intruder. They show that Bluetooth pairing with numeric comparison has this property. We develop further the idea of modelling user intention, which we state as a correspondence assertion. Because of subtly different security definitions, we end with a different result regarding Bluetooth pairing.

3. Misbinding in device pairing

We will now look at identity misbinding attacks against wireless device pairing where user authenticates the key exchange between two physical devices. Fig. 4(a) shows a common structure for many such pairing protocols. The unauthenticated key exchange takes place over an insecure in-band channel, and the user with physical access to the devices authenticates the in-band exchange over a secure out-of-band channel. The two phases may not always be distinguishable by time, but they are distinguishable by the channel.

The authentication in user-assisted pairing protocols is typically based on physical access to the device. That is, the user must see or touch the devices directly. The devices could have serial numbers, public keys, or other unique identifiers, but it is the physical access that defines which devices need to be paired.

We consider a scenario where one of the devices selected by the user for the pairing is compromised. (Recall that identity misbinding is an insider attack where one of the intended communication endpoints is corrupt.) The device has to be compromised at least to the extent that the user can control the device's inputs and outputs on the OOB channel. In Fig. 4(b), the user wants to pair devices A and E. However, device E is malicious and relays the authentication messages to another device B. Devices A and B end up paired, which does not correspond to the user's intention. Device B does not need to collude with E and may be entirely honest, except that the attacker can put it into the pairing mode and interact with it.

Let's try to understand why this attack is not easy to prevent. If we take guide from other authenticated key-exchange protocols, such as SIGMA, we might try to prevent the attack by checking that the two endpoints agree on the identifiers A and B. This comparison can be done either on the in-band or on the OOB channel, as long as the identifiers are cryptographically bound to the created session. Sadly, that does not help in device pairing. The attack by E will cause A and B to be paired, but if the user is not aware of the identifiers communicated in band, the user still thinks A is paired with E. As the next step towards a solution, we would need to check that the device identifiers A and B correspond to the user's expectations. For example, if device A shows the peer identifier to the user, the user sees that it is B and not E as intended. However, the typical user in device pairing does not have any expectations about the device identifiers: the user just sees two physical devices and wants them to be paired.

Many pairing protocols are like this: the user's physical access to the device defines its identity. Since the physical device identity cannot be communicated in bits and bytes, it cannot be included into the messages sent over the in-band or out-of-band channel, and it cannot be used as input to a cryptographic function. Cryptographic protocol vulnerabilities of the early days could often be fixed by adding a missing identifier to the right message, but that is not the case with device pairing where the endpoints either have no identifiers or, if identifiers exist, user intentions are not expressed in terms of them.

So far, our discussion of misbinding may appear as rehashing of the relay attack in the context of device pairing. This perception is partly true, but the misbinding attack is easier to implement. As hinted in Fig. 4(b), if all three devices are within the wireless range from each other, E does not actually need to relay the wireless in-band traffic. It can let A and B communicate directly over the wireless channel and focus on relaying the authentication messages between the two OOB channels. E can then pull out after the authentication is complete, which leaves A and B communicating directly.

Comparing with the cuckoo attack against trusted computing hardware, there are also similarities. The problem there was the lack of secure binding between the physical device and the long-term public key of the DRTM inside it. Our problem is the lack of secure binding between the physical devices and the ephemeral session key. The similarity extends to the lack of definite solutions by the means of traditional security protocol design. However, there are ways of mitigating the threats, as we will see in Section 6.

Next, we will look at some examples of the attack in actual pairing protocols. That will help us assess the impact of the vulnerability in a more concrete way.

3.1. Bluetooth case study

We use the widely-studied Bluetooth SSP as a case study of misbinding in pairing protocols. The attack is shown in Fig. 5. The human user Alice is trying to pair the computer A with the phone E. She is unaware that the phone E has a malicious app that is controlled by the attacker Mallory. The malicious app is able to spoof the pairing user interface on the phone at Mallory's command. The attacker also has a third device B, which is hidden from the user's view. The attacker's goal is to pair Alice's device A with the third device B while Alice believes that A is paired with E. For a successful misbinding attack, A and B must be within Bluetooth radio

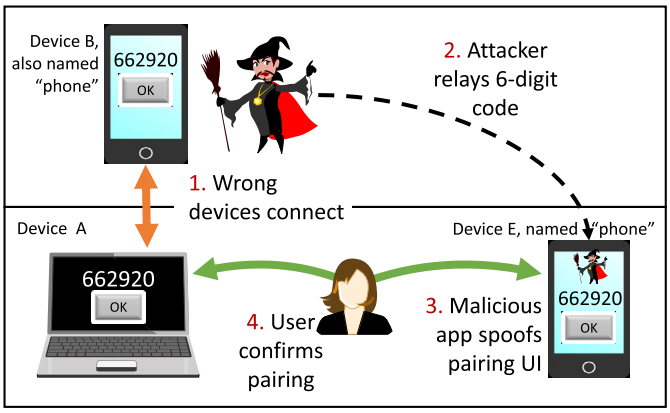

Fig. 5. Misbinding attack against Bluetooth SSP numeric comparison.

range from each other. For example, Mallory and device *B* could be in the next room from where Alice performs the pairing process.

A brief explanation of the notation is required here: For the purposes of telling the story, we denote the three devices *A*, *E* and *B*. Two of these match the notation of the Bluetooth specification, where the *initiating device* is *A* and the *non-initiating device* is *B*. The user intends device *E* to play the non-initiating role, but the attacker prevents it from participating in the protocol. Here, these symbols only denote the physical device, and they are not names or identifiers that could be communicated in the protocol. For that purpose, each device has a unique 48-bit Bluetooth address (BD_ADDR) and a name, which is non-unique and often can be modified by the user.

From the user's and the attacker's points of view, the following steps occur in the misbinding attack of Fig. 5:

1. Alice makes device *E* discoverable and starts a search for other devices on device *A*. Mallory makes device *B* discoverable. Device *A* presents Alice with a list of the names of discoverable devices in its vicinity. Alice chooses the one she thinks is *E*. At this point, Mallory needs to arrange things so that Alice mistakenly chooses *B* from the list. To achieve this, the malicious app in device *E* should keep that device non-discoverable, even though Alice thinks otherwise. Mallory should also ensure that the name of device *B* matches the name that Alice expects to see for device *E*. (We will discuss the naming in more detail below.) As the result of the attacker's meddling, the wrong devices *A* and *B* start the cryptographic pairing protocol with each other.
2. During the pairing, devices *A* and *E* display six-digit codes and expect the user to compare them. Mallory reads the six-digit code from the screen of device *B* and forwards it to the malicious app in device *E*.
3. The malicious app in device *E* displays the replayed six-digit code to Alice.
4. Seeing the same six-digit verification code on the screens of devices *A* and *E*, Alice confirms the pairing on both devices. The action on the compromised device *E* has no real effect; instead, Mallory confirms the pairing on device *B*. This allows the pairing of *A* and *B* to complete. In the end, Alice believes *A* and *E* have successfully paired when, in fact, device *A* is paired with *B*.

To understand why the Bluetooth SSP protocol does not prevent the attack above, we need to look at the protocol in more detail. The hardest practical obstacle for the attacker is, in fact, not the actual SSP protocol but the device naming and selection that takes place before the actual pairing. Bluetooth core specification [12] defines Inquiry and Paging procedures for discovering

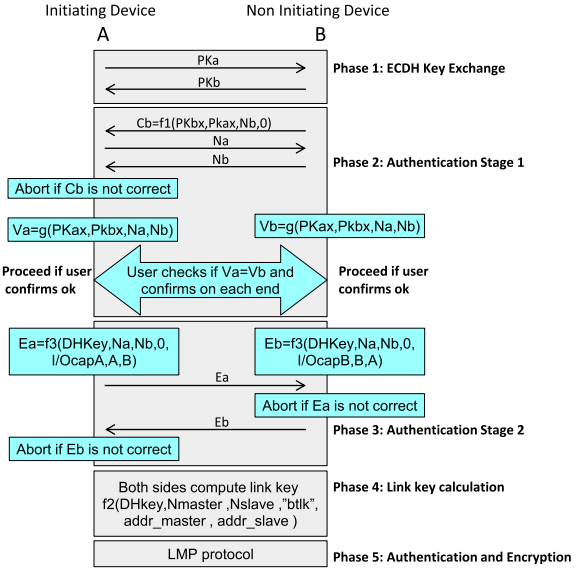

Fig. 6. Bluetooth Secure Simple Pairing with numeric comparison [12].

nearby devices and subsequently connecting to one of them. The user typically selects the name of the non-initiating device from a list of nearby devices on the initiating device. The device names are strings that aid the user in identifying the correct peer device. Each device has a default name that often indicates its make and model, for example "Nokia8" or "Alice's iPhone". Depending on the device, the name may be user configurable. In the attack, Mallory needs to trick Alice into choosing device *B* from the list by its name. Thus, Mallory should rename *B* to have the same name as *E*.

The rare tricky case for Mallory is if she wants to use a device *B* that does not have a configurable name, or if Mallory does not have the permission to change the device name. In that case, Mallory may be able to choose a device *B* that has the same make and model as device *E* and thus the same default name. If Mallory absolutely needs to use a device *B* with a Bluetooth name that is not configurable and does not match device *E*, there is still a way forward. The Inquiry and Paging procedure is not authenticated, and the attacker can manipulate the device names on the in-band wireless channel. While that requires more skill and tools than changing the name of device *B* on its user interface, message modification on a wireless channel is within the expected capabilities of a Dolev-Yao attacker.

Once Alice has been fooled into choosing the wrong device, the SSP security protocol starts between devices *A* and *B*. We will review the pairing protocol to be certain that it does not present obstacles to the attack. The numeric-comparison mode of SSP, shown in Fig. 6, has several phases that must be completed before an initiating device *A* and a non-initiating device *B* are paired securely. In phase 1, the devices perform an ECDH key exchange. In phase 2, the non-initiating device *B* commits to a random nonce *Nb*, which it reveals after the initiating device *A* has sent its own nonce *Na*. Device *A* checks the commitment to ensure that the nonces have been fairly chosen. The user-assisted authentication then takes place. Each of the devices displays to the human user a six-digit verification code, which it computes from the ECDH key shares and nonces. If the codes match, the user confirms successful pairing on both devices, which allows them to continue. In phase 3, the devices confirm cryptographically the derived ECDH secret and their input and output capabilities, which were used to select the authentication mode in the beginning. In phase 4, the devices derive a link key, i.e. a shared session key. Finally, in phase 5, they use the link key for encryption in the Link Manager Protocol.

The critical thing to observe about the SSP protocol is that it does not even try to verify the device names (or other device properties like make, model and serial number). This is understandable because Bluetooth device names do not uniquely identify a device. The protocol does bind the link key to the link-layer addresses of the two devices but, during the pairing, each device will accept any peer address.

Note that only the software in device *E* needs to be compromised for the misbinding attack, while devices *A* and *B* can be entirely normal. The only access the attacker needs on device *B* is to make it discoverable, to change its name if necessary, and to confirm the code comparison. Moreover, the attack requires device *E* to be compromised only to the extent that the attacker can spoof the pairing user interface. We implemented the attacker in device *E* as a full-screen app that receives the six-digit code over the 4G data connection and emulates the pairing process without actually participating in the pairing protocol on the in-band channel. Thus, the vulnerability occurs relatively often in practice, even though we do not know of attack implementations outside our laboratory.

The above attack against Bluetooth pairing will work for any version of SSP or Legacy Pairing. Indeed, we believe it will work for all device-pairing protocols where the device identity is determined only by physical access to the device.

4. Misbinding in device bootstrapping

4.1. EAP-NOOB Case study

We will now look at a protocol for security-bootstrapping and registration of Internet-of-Things (IoT) devices to an online server. Although the protocol differs considerably from device pairing, they are similar in the sense that the identity of the correct device is defined by physical access to it. This makes the protocol vulnerable to identity misbinding attacks.

Extensible Authentication Protocol (EAP) [36] is an authentication framework used, for example, in enterprise wireless networks. It normally assumes that the wireless devices are pre-registered at a back-end authentication server. This means that the deployment of new wireless devices is a multi-step process that includes device registration and credential provisioning.

Nimble out-of-band authentication for EAP (EAP-NOOB) [37] is an authentication method for EAP that also supports user-assisted bootstrapping and registration of new devices. It is intended for off-the-shelf IoT devices that initially have no known identifiers, no credentials, and no knowledge of their intended owner and network. EAP-NOOB registers the new devices to the authentication server and associates them with the user's account on the server. The device, called *peer*, first performs an ECDH key exchange with the server. The authentication takes place when the user delivers a single out-of-band (OOB) message from the peer device to the server, or in case of peer devices with only input capability such as cameras, from the server to the peer device. Information delivered in the OOB message enables mutual authentication of the peer and server, and it authorizes, on one hand, the server and user to take control of the device and, on the other, the device to be registered to the server and user account. The protocol does not limit the ways in which the OOB message is transferred; the implemented ways include a QR code, an NFC message, LED light, and an audio clip. After the OOB message has been delivered, the device registration completes in-band between the peer and the server.

The misbinding attack (shown in Fig. 7) arises when the peer device is compromised. Alice has a new device *E*, which she wants to register to the network and to the authentication server. In this case, the device has an NFC interface from which the OOB message can be read with a mobile phone app. Unknown to Alice, the attacker Mallory has compromised the device *E* to the extent that Mallory can control the NFC output. By mounting the misbinding attack, the attacker can trick the user into registering a different peer device *B* to the user's account in the server. From the user's and the attacker's points of view, the following steps take place in the attack:

1. Alice initiates the registration of device *E* to the wireless network and authentication server *A*. Device *E* starts (or pretends to start) the EAP-NOOB protocol with the server. At the same time, Mallory initiates the registration of another new device *B* to the same network and authentication server *A*. Device *B* starts the EAP-NOOB protocol with the server.
2. Unknown to Alice, the attacker reads an OOB message from the NFC output of device *B* and relays the message to the compromised device *E*. Device *E* is now ready to output the relayed message.
3. Alice logs into her user account on server *A* with the mobile phone app. She then taps the NFC output on device *E* to read the OOB message. The compromised device *E* outputs the relayed message.
4. The app on Alice's phone delivers the OOB message to the server. Since the message originated from device *B*, this action registers device *B* to Alice's account in the server and establishes credentials for future authentication and wireless network access of device *B*. Alice mistakenly believes that device *E* has been registered.

The above attack will work regardless of the direction and the number or the messages sent over the OOB channel. The compromised device *E* simply relays all OOB messages between the user's phone and device *B*, until device *B* is authorized to register to Alice's account ton the server.

A slight complication arises when the OOB communication is initiated by the server. In that case, there could be multiple devices attempting to register at the same time to the same server. Since the OOB messages are specific to the device, the user has to choose the correct device on the server. To trick the user into choosing the wrong device *B*, the attacker must match the make, model and any other metadata of device *E* by which the user selects the correct device from those available for registration. It is easiest for the attacker to clone the metadata of device *E* by creating a virtual device *B* whose behavior is fully under the attacker's control. On the other hand, if the attacker wants to register an actual physical device *B*, it may have to choose one of the same make and model as device *E*.

Unlike in device pairing with Bluetooth, Mallory's device *B* does not need to be in close proximity to Alice or to *A*. Mallory can run the EAP-NOOB protocol on her device *B* from anywhere in the coverage area of the wireless networks served by the same authentication server. She only needs the capability of sending or receiving the OOB message to or from the compromised device *B*.

Device bootstrapping and registration with EAP-NOOB is designed to be efficient for deploying large numbers of devices. Thus, the person installing the devices might not be the eventual user, and the failure of device *E* to associate with the server might go unnoticed for some time. In comparison, device pairing with Bluetooth is often followed by another user action such as transfer of media, which may lead to the user detecting the failure of device *E* to pair.

4.2. DPP Case study

Device Provisioning Protocol (DPP) [38] is a bootstrapping mechanism recently standardized by the Wi-Fi Alliance for configuring Wi-Fi network information on devices with limited user

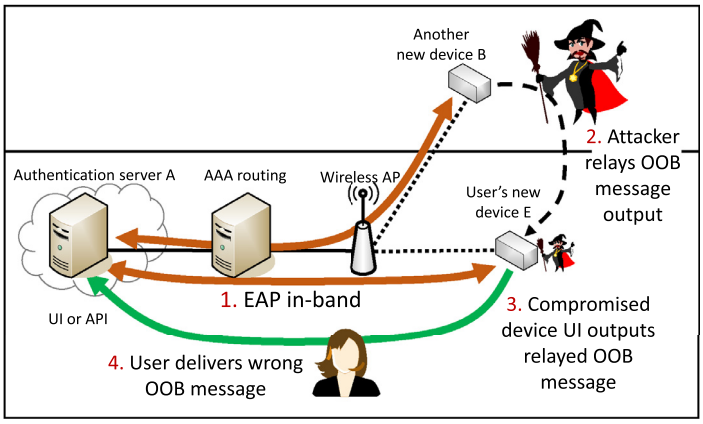

Fig. 7. Misbinding attack against EAP-NOOB.

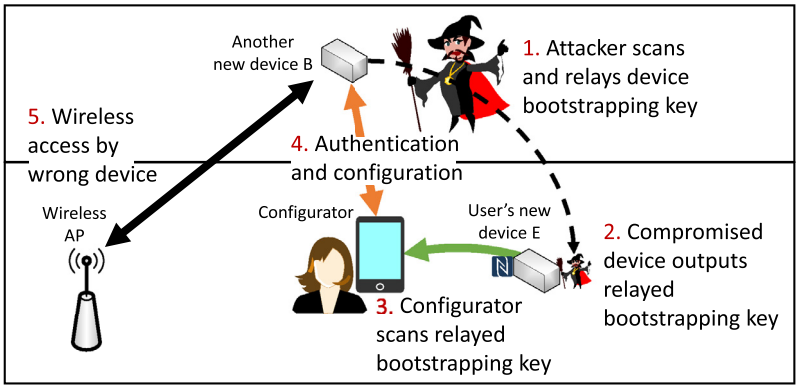

Fig. 8. Misbinding attack against DPP.

interfaces. We discuss the misbinding threat separately for DPP because of its expected wide deployment.

DPP relies on a *configurator*, e.g. a smartphone application, for bootstrapping all other devices, called *enrollees*, in the network. In the most typical use case, every enrollee has a public bootstrapping key, which is communicated to the configurator in the bootstrapping phase over an out-of-band channel. The OOB data includes communication metadata such as the radio channel on which the enrollee device is listening. The configurator authenticates the enrollee using the bootstrapping key and then configures it for Wi-Fi access.

The misbinding attack against DPP (Fig. 8) is almost trivial: when the user is configuring a compromised device *E*, the attacker replaces the public key and communication metadata output from *E* with those of another device *B*. This requires the attacker to compromise the interface, such as NFC, which outputs the OOB message. In one variant of DPP, the public bootstrapping key is printed as a QR code, and in that case, the device compromise is equal to replacing this piece of paper in the retail packaging. In other variants, the configurator and device exchange one or more dynamic messages, and the attacker has to relay them in real time between the two devices.

5. Formal analysis of misbinding

We modelled the case-study protocols and their security requirements with ProVerif [32,39]. First, we wanted to enhance previous models of device pairing and especially Bluetooth SSP to capture the misbinding attack. It was not clear to us why the existing models missed the attack when so many other, even more subtle issues have been detected. We also wondered if the attack and the security goals it violates can be reduced to previously known ones. As a result, we learned that the formal models can be made more complete so that they discover the misbinding attack, and that the violated security properties are different from what has previously been analyzed. Another goal of our modelling work was to understand how pairing protocols differ from each other in relation to the misbinding vulnerability, and whether registering a physical device to an online service is fundamentally different from pairing two physical devices. We found that misbinding occurs in a wide range of protocols where endpoints are defined by physical access. We also found that the attacks can be classified into a small number of variants, and not all protocols are vulnerable to all of them.

5.1. Modelling device pairing

We will mainly discuss Bluetooth SSP with numeric comparison because of its familiarity to many readers. The full model is included in Appendix Appendix A. However, we also modelled the SSP OOB mode and Wi-Fi Direct [40] with similar results.

In addition to the protocol messages and the device state machines, we model the *security ceremony that includes user intentions, choices and actions*. We follow the example of Carlos [41] and model the user as a separate process in ProVerif. However, while Carlos considers pairing between two devices belonging to different users, we consider pairing where a single user has physical access to both intended endpoints. Thus, our model consists of three kinds of processes: user, initiating device *A*, and non-initiating device *B*. Both types of devices can become compromised and, thus, take the role of *E*, which corresponds to the two scenarios of Fig. 1.

The challenging part of the model was capturing the user intention, i.e. decision to pair specific two devices, when the *devices*

are identified by physical access and do not have names or other identifiers. In the end, the solution is fairly simple and intuitive: the users and devices have identifiers in the model (see below), but the identifiers can never be communicated over a channel or used as input to a cryptographic function. Instead, they are used for marking local events and for checking correspondence properties between the events, such as whether the user intended the devices to be paired. This inability to communicate the identifiers goes a long way towards explaining why the traditional solutions of adding explicitly or implicitly communicated identifiers are not applicable to device pairing.

Similar to Chang et al. [22], we use private channels in ProVerif to model the *physical access* by the user to the devices. These channels protect both secrecy and integrity of the communication. In the case of Bluetooth, the private channels are used both for reading the numeric codes and, if the values match, for confirming the match to the devices. To initiate pairing, user needs to have access to two private channels, `PhysicalChannelA` to an initiator device and `PhysicalChannelB` to a non-initiator device. We use these physical channels as the device identifiers, which is both practical and semantically correct. For the users, on the other hand, we simply create new identifiers.

Compromised endpoints are commonly modelled by leaking their secrets, such as private keys, to a public channel. Consequently, the built-in attacker model of the model-checking tool can emulate any honest or malicious behavior by that endpoint. In the Bluetooth model, however, the devices do not have any master secrets. Instead, we *model the compromise of a device by leaking its private channel to the network*. This allows the attacker to take control of that channel.

In addition to modelling the compromise of devices, we also model the compromise of a user. This is done to conceptually distinguish between a tampered device and a malicious user having physical access to an intact device. There is no real difference between the two in the Bluetooth case. However, the distinction becomes significant when we compare mitigation techniques and different levels of user access to the device.

The user model is shown below. The user (i) selects two devices and logs her decision to pair them as an event, (ii) compares the six-digit verification codes displayed by the devices, and (iii) confirms a match to the devices. The user may be compromised any time, yielding control of the physical access channels to the attacker.

```
let UserProcess(User:User_t, PhysicalChannelA:channel,
    PhysicalChannelB:channel) =
(
  event HasAccess(User, PhysicalChannelA);
  event HasAccess(User, PhysicalChannelB);
  (* Decide to pair A and B with A as initiator *)
  event IntendToPair(User, PhysicalChannelA, PhysicalChannelB);
  (* Receive Va and Vb *)
  in(PhysicalChannelA, (=CodeTag, Va:Hash_t));
  in(PhysicalChannelB, (=CodeTag, Vb:Hash_t));
  (* Numeric comparison *)
  if Va = Vb then
    out(PhysicalChannelA, (OkTag, Va)); (* Confirm to A *)
    out(PhysicalChannelB, (OkTag, Vb))  (* Confirm to B *)
) | (
  event CompromiseUser(User);
  out(c, PhysicalChannelA);
  out(c, PhysicalChannelB)).
```

Intuitively, misbinding is a violation of the following security property: *two devices are paired only if their user intended them to be*. When formalizing the absence of misbinding as a correspondence property in ProVerif, we need to be more precise: *If two devices complete the pairing with the same link key and a user has physical control of at least one of them, then either the user previously intended the two devices to be paired, the user is compromised, or both devices are compromised.* In ProVerif, this correspondence property can be defined as follows:

```
query PhysicalChannelA:channel, PhysicalChannelB:channel, K:Key_t,
    User:User_t;
  (event(HasAccess(User, PhysicalChannelA)) && (*or B*)
   event(InitiatorComplete(PhysicalChannelA, K)) &&
   event(NoninitiatorComplete(PhysicalChannelB, K)))
     ==> (event(IntendToPair(User, PhysicalChannelA,
          PhysicalChannelB)) ||
           event(CompromiseUser(UserId)) ||
          (event(CompromiseDevice(PhysicalChannelA)) &&
           event(CompromiseDevice(PhysicalChannelB)))).
```

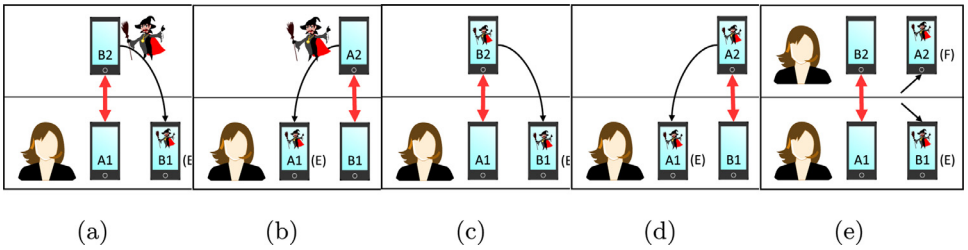

Fig. 9. Five variants of misbinding found with ProVerif.

As expected, ProVerif returned *false* for the query and produced a counterexample, i.e. an execution trace that violates the security property. There are two versions of the query, one with `PhysicalChannelA` and another with `PhysicalChannelB` on the second line. The queries can be refined to exclude already analyzed attacks or to focus on specific cases.

Investigating further, we found five different types of misbinding attacks with ProVerif, which are summarized in Fig. 9. Each sub-figure shows two rooms. The honest user tries to pair two devices, initiator *A*1 and non-initiator *B*1, in her room. One of these devices is compromised, and the other ends up being paired (indicated by the thick red arrow) with a device in the room above. The sub-figures show the locations of the honest users, compromised users, and compromised devices. The black one-directional arrow is specific to Bluetooth SSP with numeric comparison. It shows how the attacker forwards the six-digit code from one device to another.

The first one of the attack variants, seen in Fig. 9(a), is the basic misbinding attack described in Section 3. In that attack, the compromised device is the non-initiator *B*1, and there is a compromised user with physical access to the third device *B*2. This corresponds to Figs. 4(b) and 5, where *E* corresponds to *B*1. Other attacks arise as variations of the first one. On one hand, the compromised device in the user's physical possession can be the initiator *A*1 or the non-initiator *B*1. On the other hand, the third may be a compromised one or an uncompromised device accessed by a compromised user. These choices make the four different variants of the misbinding attack in Figs. 9(a)-(d).

It came as a surprise to us that there is a fifth type of misbinding attack, which we call *double misbinding*. In this attack, seen in Fig. 9(e) and more clearly illustrated in Fig. 10, there are two honest users. Each one of them is trying to pair two devices, one of which is compromised. The compromised devices collude so that, as the result, the two uncompromised devices are paired.

Double misbinding is easiest to understand in the out-of-band mode of Bluetooth SSP, where the user transfers some information out-of-band from one device to another. In that case, one compromised device receives the OOB message from the first honest user and forwards it secretly to the second compromised device, which outputs it to the second honest user. The attack is also possible in SSP with numeric comparison because all the values needed for computing the verification codes *Va* and *Vb* are transmitted on the wireless link (see Fig. 6). The attacker can sniff these values, compute *Va* and *Vb*, and show them on the displays of the two compromised devices.

Afterward finding the five attack variants by formal verification, we systematically enumerated the different combinations of initiator and non-initiator devices, compromised and uncompromised users and devices, and user physical access in a setting of maximum two users and four devices. This analysis confirmed that, after removing impossible and equivalent cases, the five attack variants remain. Increasing the number of users and devices does not seem to give raise to any new types of attacks because there is maximum that can be involved in a single pairing.

5.2. Modelling device bootstrapping

Although the ProVerif models of EAP-NOOB and Bluetooth differ greatly, the parts relevant to detecting misbinding are similar. The main difference is that, in EAP-NOOB, only the peer device is identified by the physical access channel. The EAP-NOOB server has a strong cryptographically verifiable identity (HTTPS URL and web certificates), and we assume that the server cannot be compromised. The query for the absence of misbinding attacks is as follows:

```
query OobChannelS:channel, OobChannelP:channel, K:Key_t, User:
    User_t;
  (event(HttpsAccess(User, OobChannelS)) &&
   event(ServerComplete(OobChannelS, K)) &&
   event(PeerComplete(OobChannelP, K)))
    ==> (event(CompromiseUser(User)) ||
         event(IntendToPair(User, OobChannelS, OobChannelP)) ||
        (event(CompromisePeer(OobChannelP)) &&
         event(CompromiseServer(OobChannelS)))).
```

Again, ProVerif finds a counterexample to this query. Because only the peer side can be compromised, there are only two possible variants of misbinding. They correspond to Fig. 9(a) and (c). In the first one, the server is *A*1, the compromised peer device *B*1, and the uncompromised peer device *B*2. In the other attack variant, both peer devices are compromised and there is no need for a user to operate device *B*2. The first of these two variants matches the attack discussed earlier and shown in Fig. 7.

We also modelled DPP with ProVerif and verified its vulnerability to misbinding of the enrollee, as explained in Section 4.2.

6. Mitigation

6.1. Authentication solutions

As explained in Section 2.2, the STS and SIGMA protocols and their variants [2,3,7] tackle misbinding by binding endpoint identities cryptographically to the created session. These solutions are

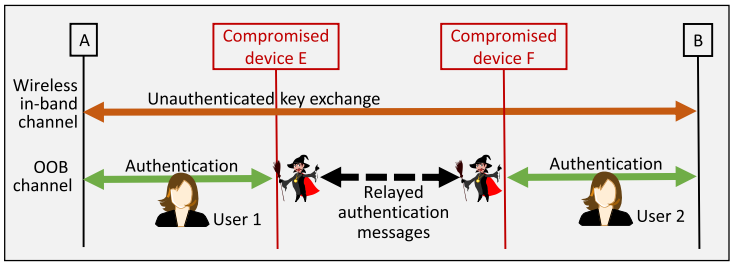

Fig. 10. Double misbinding.

suitable for situations where the devices have certificates, public keys for authentication, and unique names. This is typically not the case in device pairing. Moreover, as we explained in Section 3, the endpoints in device pairing have no a-priory knowledge of each other's identifiers, and neither does the typical user who is assisting the key exchange.

The common way to communicate the device identifier, such as model and serial number, to the user is printing them on an identification plate attached to the device. Together with a certificate issued by the manufacturer, this information can be used for authenticating the device. Another possibility is to print a fingerprint of the device's public-key onto the device, e.g. as a hexadecimal value. If a metal plate, sticker or printing on the device is not considered tamper-proof enough, the identifiers could be etched to the device enclosure. While such physical indicators can ultimately be counterfeited, the burden on the attacker is increased significantly. The disadvantage of these solutions is that the user needs to compare the authenticated device identifiers with the serial-number plates or key fingerprints, which complicates the pairing process.

6.2. Presence checking

As noted in Section 2.5, trusted-computing research has not put much faith in the printed serial numbers or public-key fingerprints. Instead, the researchers have tried to find more secure ways of checking the presence of a DRTM inside a physical device. We can generalize these approaches from DRTM to any device with a trusted computing base (TCB) that is surrounded by potentially compromised layers of software. The techniques for DRTM presence checking could be applied to checking the physical presence of the pairing endpoint for a given device, which could prevent the misbinding attacks.

The round-trip time measurement suggested by Fink [29] depends on the latency caused by the cuckoo in the communication chain. In our attacks against device pairing, the in-band communication takes place directly with the third device, and timing measurement is unlikely to be able distinguish between two devices within the Bluetooth radio range. This issue of *distance bounding* has been widely studied in relation to RFIDs and wireless keys [42,43].

Ding et al. [31] provide a summary of several other solutions. One is a hardware-based secure channel, i.e. a *trusted path*, that allows the user to communicate directly with the DRTM or TCB inside the device. This could, for example, be an LED indicator light or a special-purpose USB port. The need for such a feature in smart devices is well known, but the idea has never been widely adopted by device manufacturers. The great variety of manufacturers and form factors in smart devices would also make it difficult for the user to know which feature can be truly trusted. Another solution is to enclose the devices into a Faraday cage to prevent them from communicating with external entities during the key-presence checking. This approach was previously suggested for bootstrapping sensor nodes wirelessly [44]. Zhang et al. [30] propose several presence checking methods based on analog channels, which do not provide strong security guarantees but make the attacks impractical. One method is based on comparing the GPS location measurements by the two endpoints, and another on comparing images captured by co-located devices of their immediate environment. They also propose measuring the timing of a screen-to-camera video channel, which would be difficult to forward to a remote device without causing a detectable delay.

In one practical form of presence checking, which is already widely deployed, the user can ask a peer device to blink an LED indicator light. The primary purpose for this is to help the user to identify a specific physical device among many similar ones. For example, the pairing process of Apple Homekit devices relies primarily on a static code that is attached to the device or inside the retail packaging. Before scanning the static code during the pairing process, the user can optionally ask the selected device to blink its LED. If we consider this a security feature, the assumption must be that the attacker cannot make the LED blink, at least not at the right time. Naturally, if the device is compromised, the attacker might be able to make the LED blink at the right time. Nevertheless, even such a weak device identification mechanism increases the burden of the attacker compared to not having one; without it, the attacker could achieve misbinding simply by replacing the static code on the user's new device.

6.3. Asset tracking

We believe the practical approach to detecting misrepresented device identities might be *asset tracking*, i.e. bookkeeping of the physical assets that belongs to an organization or an individual. This requires each device to have a unique identifier, which is registered into a database when the user purchases a device. In the simplest case, the database is accessed only by human users, in which case any existing asset tracking system or database can be used.

When the organization knows the models and serial numbers of its devices and the purpose assigned to each one, the information can be used for cross-checking during device pairing. For example, if there is only one new display device allocated for Alice, Alice can compare the device information from the database with the identifier authenticated in the device pairing process when she deploys the device.

For this to work, each device needs to know its own identifier and learn the peer identifier during the key exchange. The identifiers should be bound to the cryptographic key exchange in such a way that agreement on session key cannot be reached without also agreeing on the identifiers. Each device should show the iden-

tifier of its peer to the user, e.g. when initiating the pairing protocol or when confirming the numeric comparison. In Bluetooth SSP protocol, this would require changes to the input of the verification codes, while EAP-NOOB already has a built-in authenticated message field (`PeerInfo`) for communicating such auxiliary peer information. Of course, the software of an uncompromised device should not allow the users to modify the device identifier. As the result of these measures, device *A* in the scenario of Fig. 4(b) would show the identifier of the unknown device *B* to the user and the attacker cannot replace it with the expected identifier of device *E*.

Manufacturer-issued device certificates [45,46] can further help the process by providing secure information about the types and models of the devices. This will reduce the reliance on the asset database because all other information except correctness of the device identifier can be communicated in the certificate. One downside of certificates when compared to purely ad-hoc OOB based pairing mechanisms is that the certificates can reveal the identities of the participating devices to both passive and active observers. The identities can be encrypted with the SIGMA 3-round protocol by encrypting the identity and certificate payloads [47]. However, in these protocols, one side must reveal its identity first, thereby putting it at disadvantage. Wu et al. [48] propose a policy mechanism for mitigating this weakness with identity-based encryption and hierarchical human-readable names. Such mechanisms could be used for protecting the manufacturer certificates in any pairing or device bootstrapping protocol including those where the main authentication method is an OOB channel.

For the average consumer, it is difficult to keep track of purchased devices over any longer span of time. However, this obstacle may be disappearing as smart devices are increasingly cloud connected and their ownership is therefore often registered by the manufacturer or some other cloud service. The same online service can replace the corporate asset-tracking system for an individual user. Furthermore, there are proposals for logging Internet-of-Things devices to a blockchain [49,50], which could also be used for asset tracking.

Above, we have mostly discussed device pairing and Bluetooth, but the same solutions also work for EAP-NOOB and device registration to the cloud. The main difference is that only one endpoint of the key exchange is a physical asset that needs to be tracked. The fact that the authentication server is online and provided to the user as a service means that it could help with ownership tracking or connect directly to the manufacturers on the user's behalf.

6.4. On bluetooth SSP and double misbinding

As noted in Section 5.1, SSP with numeric comparison is vulnerable to double misbinding because all the inputs for computing the verification codes *Va* and *Vb* are transmitted on the wireless link and can be sniffed. If *Va* and *Vb* were computed as function of the ECDH shared secret, the two compromised devices could not show the value on their displays. This would prevent double misbinding, although not the simpler misbinding attacks. Similar protocols in the future might consider taking advantage of the secrecy to limit the space that the attacker has for maneuvering. A possible disadvantage is that the devices would have to compute the ECDH shared secret before displaying the verification codes, which could impact the user experience on devices with slow processors. The current SSP protocol also has a clean design where the six-digit verification codes are not at all expected to be secret.

7. Discussion

It remains to be discussed how serious the misbinding vulnerability is and whether we should be worried about it. We do not want to be alarmist but instead try to provide balanced arguments for thinking about the issue.

First, the vulnerability exists in a wide range of protocols and systems. Any pairing or bootstrapping protocol that relies solely on the user's physical identification of the endpoints will be equally vulnerable regardless of the protocol design. In fact, even strong authentication of the endpoint identifiers does not prevent misbinding unless each endpoint knows what the other's identifier should be.

The risk of misbinding attacks against out-of-band authentication is increased by the fact that the attacks are easy to implement. The compromised device only needs to forward authentication messages on the user-interface level. This is considerably easier for the typical attacker than, for example, relaying communication at the radio or logical link layer. It is also easier than continuously forwarding application-layer messages to and from a compromised device. This makes misbinding an attractive attack for technically less competent attackers. In our attack implementations against Bluetooth or EAP-NOOB, device *E* was compromised by simply installing a malicious app that emulates the pairing user interface at the attacker's command.

Misbinding depends on the user trying to pair with or register a device *E* that is compromised. Thus, there must be a (partially) corrupt insider involved. The user is misled because the user makes a bad decision and trusts the corrupt device. Some protocol designers might dismiss the problem at this point, thinking that it is outside their threat model. One counterargument to such dismissal is that the corrupt device is not the one that ends up being paired, and thus the honest device *B* also suffers. Another counterargument is that the Internet of Things will be full of corrupt insiders, just like the regular Internet. Also, we should protect the users from their own mistakes whenever possible.

The practical impact of misbinding attacks is somewhat difficult to grasp. It has been demonstrated with the help of two example scenarios, one presented by Diffie et al. in the original STS paper and the other by Krawczyk in a lecture:

- *A* connects to bank *B*, over a supposedly secure session, to deposit an electronic coin. Since *E* mounted a misbinding attack, bank *B* thinks the coin was deposited by *E*. This is the scenario of Fig. 1(b).
- *E* and *B* are fighter jets, and *A* is their commander. *E* has been compromised by the enemy. *A* tells *E* to self-destruct, but because *E* mounted a misbinding attack, the command goes to *B*. This is the scenario of Fig. 1(a).

The banking scenario does not seem to have obvious equivalents in the world of physical devices. The fighter jets, on the other hand, are devices, and we can construct a related IoT example:

- *E* and *B* are IoT devices, and *A* is the user's computer. *E* has been infected by malware. User wants to connect from *A* to *E* and wipe *E*'s memory. Because *E* mounts a misbinding attack, the user wipes device *B* instead.

Note that all these scenarios require some prior relation between the endpoints, and the misbinding attack leads to a failure of correspondence between that prior relation and the newly-established secure connection. In pairing and bootstrapping, there often is no such common history. Either one of the endpoints is a new, fresh device, or the history is not significant because the endpoints have no secure way of knowing that they have reconnected

to the same peer. This may be one reason why the practical impact of misbinding for IoT devices remains somewhat elusive.

We also need to compare misbinding to alternative attacks. In Fig. 4(b), the attacker in device *E* can achieve almost the same results by accepting the connection from *A*, establishing another connection to *B*, and then forwarding the application-layer messages between *A* and *B*. The main difference between misbinding and such relaying of communication is that the misbinding attacker can remove itself from the communication chain after the pairing. Thus, in misbinding, the continuation of the attack does not depend on the compromised device *E* being online or within radio range. Furthermore, the user is in physical control of the compromised device *E* but not of *B*. In misbinding, if the user disables device *E*, e.g. by disconnecting it from the network or even by physically destroying it, device *B* and its connection with *A* will nevertheless persists — unknown to the user.

Since the design of STS and IKE, there has been consensus among security protocol designers that misbinding vulnerabilities are not acceptable in authenticated key-exchange protocols for computer networks and for the Internet. In device pairing, there is no similar consensus, and the attack has been mostly ignored with the exception of the trusted-computing community. We do not expect this paper to stop people from using protocols like Bluetooth SSP. The misbinding attacks and impact scenarios are relatively marginal compared to the advantage of encrypting wireless communication and having basic authentication in place, and the value of these is not nullified by misbinding. The attacks should, however, not be ignored because they are so widely applicable to device-pairing and IoT bootstrapping protocols. Our message is that protocol and system designers should understand the misbinding vulnerability for physical devices, keep eyes open for unexpected consequences in new situations, and make an informed judgment about whether additional countermeasures are needed.

8. Conclusion

We studied identity-misbinding (or unknown-key-share) attacks in device pairing protocols where the devices are identified by physical access rather than cryptographic credentials. We showed that Bluetooth and other similar device-pairing protocols are vulnerable to this attack regardless of their cryptographic details. The same vulnerability also exists in protocols for security-bootstrapping IoT devices. We confirmed the attacks by implementing them. Formal modelling allowed us to discuss the precise definition of misbinding, which led to the discovery of a new attack variant, double misbinding. We also discussed potential mitigation mechanisms, arguing in favor of solutions based on asset tracking. While the vulnerability to identity misbinding does not make the existing device pairing protocols completely insecure, it is a threat that needs to be fully understood also in device pairing, and this paper is a step towards that goal.

Author Statement

The entire article is teamwork by the authors.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

We would like to thank Eric Rescorla for his inspiring comments on EAP-NOOB and Kaisa Nyberg for insightful discussion on Bluetooth SSP. This work was supported by Academy of Finland (grant number 296693).

Appendix A. Bluetooth Model

```
(* ===== CHANNELS ===== *)
free c:channel. (* In-band channel *)
free PhysicalChannelA1:channel [private]. free PhysicalChannelA2:channel [private]. (* OOB channel *)
free PhysicalChannelB1:channel [private]. free PhysicalChannelB2:channel [private]. (* OOB channel *)

(* ===== DATA TYPES ===== *)
type IOcap_t. (* IO capability *)    type Hash_t. (* Hash *)    type K_t.   (* Key *) type N_t. (* Nonce *)
type Addr_t. (* Bluetooth address *) type User_t. (* User ID *) type Tag_t. (* Tag *)

(* ===== CONSTANTS ===== *)
const btlk:bitstring.  const EN_RAND:bitstring.      const COF:bitstring.
(* User tags and IDs *)
const OkTag:Tag_t. const CodeTag:Tag_t. const UID1:User_t. const UID2:User_t.
(* Device-specific values *)
const A:Addr_t. (* Bluetooth address of A *) const IOcapA:IOcap_t.  (* IO capabilities of device A *)
const B:Addr_t. (* Bluetooth address of B *) const IOcapB:IOcap_t.  (* IO capabilities of device B *)

(* ===== QUERIES & EVENTS ===== *)
(* Completion events *)
event InitiatorComplete(channel, K_t).
event NonInitiatorComplete(channel, K_t).
event IntendToPair(User_t, channel, channel).

(* Compromisation events *)
event CompromisedUser(User_t).
event CompromisedDevice(channel).

(* A user has access to a channel *)
event HasAccess(User_t, channel).

(* A successful connection must be initiated by a user *)
query PhysicalChannelA:channel, PhysicalChannelB:channel, K:K_t, User:User_t;
  (event(HasAccess(User,PhysicalChannelA)) &&
   event(InitiatorComplete(PhysicalChannelA, K)) &&
   event(NonInitiatorComplete(PhysicalChannelB, K)))
     ==> (event(IntendToPair(User, PhysicalChannelA, PhysicalChannelB)) ||
          event(CompromisedUser(User)) ||
         (event(CompromisedDevice(PhysicalChannelA)) &&
          event(CompromisedDevice(PhysicalChannelB)))).

(* ===== FUNCTIONS ===== *)
(* P256 scalar multiplication *)
const G:K_t [data].
fun P256(K_t, K_t):K_t.
equation forall k1:K_t, k2:K_t;
```

```
46      P256(k1,P256(k2,G)) = P256(k2,P256(k1,G)).
47
48  (* Public key encryption/decryption *)
49  fun encrypt(bitstring, K_t): bitstring.
50  reduc forall x:bitstring,y:K_t;
51      decrypt(encrypt(x,y),y) = x.
52
53  (* Cryptographic functions *)
54  fun g(K_t,K_t,N_t,N_t):Hash_t.
55  fun f1(K_t,K_t,N_t):Hash_t.
56  fun f2(K_t,N_t,N_t,bitstring,Addr_t,Addr_t):K_t.
57  fun f3(K_t,N_t,N_t,IOcap_t,Addr_t, Addr_t):Hash_t.
58  fun E3(K_t,bitstring,bitstring):K_t.
59
60  (* ===== PROCESSES ===== *)
61  let UserProcess(User:User_t,PhysicalChannelA:channel,PhysicalChannelB:channel) =
62    (
63      event HasAccess(User, PhysicalChannelA);
64      event HasAccess(User, PhysicalChannelB);
65      (* Decide to pair devices A and B with A as initiator *)
66      event IntendToPair(User, PhysicalChannelA, PhysicalChannelB);
67      (* Receive hashes Va and Vb *)
68      in(PhysicalChannelA, (=CodeTag, Va:Hash_t));
69      in(PhysicalChannelB, (=CodeTag, Vb:Hash_t));
70      (* Compare the hashes *)
71      if Va = Vb then
72        out(PhysicalChannelA, (OkTag, Va)); (* Confirm to A *)
73        out(PhysicalChannelB, (OkTag, Vb))  (* Confirm to B *)
74    ) | (
75      event CompromisedUser(User);
76      out(c, PhysicalChannelA);
77      out(c, PhysicalChannelB)
78    ).
79
80  (* Initiating device *)
81  let DeviceA(PhysicalChannelA:channel) =
82    (
83      (* Create a public/secret key pair *)
84      new SKa:K_t;
85      let PKa = P256(SKa,G) in
86      (* PHASE 1: PUBLIC KEY EXCHANGE *)
87      out(c, (PKa,A,IOcapA));                    (* 1a. A->B: PKa,A,IOcapA *)
88      in(c, (PKb:K_t,B:Addr_t,IOcapB:IOcap_t)); (* 1b. B->A: PKb,B,IOcapB *)
89      let DHKey = P256(SKa,PKb) in
90      (* PHASE 2: AUTHENTICATION STAGE 1 - NUMERIC COMPARISON *)
91      new Na:N_t;
92      in(c, Cb:Hash_t);                          (* 4. B->A: Cb *)
93      out(c, Na);                                (* 5. A->B: Na *)
94      in(c, Nb:N_t);                             (* 6. B->A: Nb *)
95      if Cb = f1(PKb,PKa,Nb) then
96      let Va = g(PKa,PKb,Na,Nb) in
97      out(PhysicalChannelA, (CodeTag,Va));       (* 7a. A->U: Va *)
98      in (PhysicalChannelA, (=OkTag,=Va));
99      (* PHASE 3: AUTHENTICATION STAGE 2 *)
100     let Ea = f3(DHKey,Na,Nb,IOcapA,A,B) in     (* 9a. compute Ea *)
101     out(c, Ea);                                (* 10. A->B: Ea *)
102     let Eb = f3(DHKey,Nb,Na,IOcapB,B,A) in
103     in(c, =Eb);                                (* 11. B->A: Eb *)
104     (* PHASE 4: LINK KEY CALCULATION *)
105     let LK = f2(DHKey,Na,Nb,btlk,A,B) in
106     (* PHASE 5: LMP AUTHENTICATION AND ENCRYPTION *)
```

```
    let KC = E3(LK,EN_RAND,COF) in
    (* Key-derivation successful *)
    event InitiatorComplete(PhysicalChannelA, KC)
  ) | (
    event CompromisedDevice(PhysicalChannelA);
    out(c, PhysicalChannelA)
  ).

(* Non-Initiating device *)
let DeviceB(PhysicalChannelB:channel) =
  (
    (* Create a public/secret key pair *)
    new SKb:K_t;
    let PKb = P256(SKb,G) in
    (* PHASE 1: PUBLIC KEY EXCHANGE *)
    in(c, (PKa:K_t,A:Addr_t,IOcapA:IOcap_t)); (* 1a. A->B: PKa,A,IOcapA *)
    out(c, (PKb,B,IOcapB));                   (* 1b. B->A: PKb,B,IOcapB *)
    let DHKey = P256(SKb,PKa) in
    (* PHASE 2: AUTHENTICATION STAGE 1 - NUMERIC COMPARISON *)
    new Nb:N_t;
    let Cb = f1(PKb,PKa,Nb) in
    out(c, Cb);                               (* 4. B->A: Cb *)
    in(c, Na:N_t);                            (* 5. A->B: Na *)
    out(c, Nb);                               (* 6. B->A: Nb *)
    let Vb = g(PKa,PKb,Na,Nb) in
    out(PhysicalChannelB, (CodeTag,Vb));      (* 7b. B->U: Vb *)
    in (PhysicalChannelB, (=OkTag,=Vb));
    (* PHASE 3: AUTHENTICATION STAGE 2 *)
    let Ea = f3(DHKey,Na,Nb,IOcapA,A,B) in
    in(c, =Ea);                               (* 10. A->B: Ea *)
    let Eb = f3(DHKey,Nb,Na,IOcapB,B,A) in    (* 9b. compute Eb *)
    out(c, Eb);                               (* 11. B->A: Eb *)
    (* PHASE 4: LINK KEY CALCULATION *)
    let LK = f2(DHKey,Na,Nb,btlk,A,B) in
    (* PHASE 5: LMP AUTHENTICATION AND ENCRYPTION *)
    let KC = E3(LK,EN_RAND,COF) in
    (* The key-derivation was successful *)
    event NonInitiatorComplete(PhysicalChannelB, KC)
  ) | (
    event CompromisedDevice(PhysicalChannelB);
    out(c, PhysicalChannelB)
  ).

process
  (UserProcess(UID1,PhysicalChannelA1,PhysicalChannelB1)   |
   UserProcess(UID2,PhysicalChannelA2,PhysicalChannelB2)   |
   DeviceA(PhysicalChannelA1) | DeviceA(PhysicalChannelA2) |
   DeviceB(PhysicalChannelB1) | DeviceB(PhysicalChannelB2))
```

References

[1] Sethi M, Peltonen A, Aura T. Misbinding attacks on secure device pairing and bootstrapping. In: Proceedings of the 2019 ACM Asia conference on computer and communications security. New York, NY, USA: ACM; 2019. p. 453–64. ISBN 978-1-4503-6752-3. doi:10.1145/3321705.3329813.

[2] Krawczyk H. SIGMA: the 'SIGn-and-MAc' approach to authenticated Diffie-Hellman and its use in the IKE protocols. In: Annual international cryptology conference. Berlin, Heidelberg: Springer Berlin Heidelberg; 2003. p. 400–25.

[3] Blake-Wilson S, Menezes A. Unknown key-share attacks on the station-to-station (STS) protocol. In: Proceedings of the international workshop on public key cryptography. Berlin, Heidelberg: Springer-Verlag; 1999. p. 154–70.

[4] Dolev D, Yao A. On the security of public key protocols. Trans Inf Theory 1983;29(2):198–208.

[5] Abadi M, Needham R. Prudent engineering practice for cryptographic protocols. Trans Softw Eng 1996;22(1):6–15.

[6] Lowe G. An attack on the needham-Schroeder public-key authentication protocol. Inf Process Lett 1995;56(3):131–3.

[7] Diffie W, Van Oorschot PC, Wiener MJ. Authentication and authenticated key exchanges. Design Codes Cryptogr 1992;2(2):107–25.

[8] Lowe G. A hierarchy of authentication specifications. In: Proceedings of the 10th computer security foundations workshop. Washington, DC, USA: IEEE Computer Society; 1997. p. 31–43.

[9] Woo TY, Lam SS. A semantic model for authentication protocols. In: Proceedings of the IEEE computer society symposium on research in security and privacy. Washington, DC, USA: IEEE Computer Society; 1993. p. 178–94.

[10] Kaufman C., Hoffman P.E., Nir Y., Eronen P., Kivinen T.. Internet key exchange protocol version 2 (IKEv2). http://tools.ietf.org/rfc/rfc7296.txtRFC 7296; 2014.

[11] ISO. It security techniques – entity authentication – part 3: mechanisms using digital signature techniques. International standard; 1993.

[12] SIG B. Bluetooth specification version 5.0. Core Specification. Bluetooth SIG; 2016. https://www.bluetooth.com/specifications/bluetooth-core-specification.

[13] IEEE. IEEE standard for information technology–telecommunications and information exchange between systems local and metropolitan area networks–specific requirements - part 11: wireless LAN medium access control (MAC) and physical layer (PHY) specifications. Tech. Rep.. IEEE; 2016.

[14] Alliance Z. ZigBee specification. ZigBee Alliance Document. ZigBee Alliance; 2012.

[15] Kainda R, Flechais I, Roscoe AW. Usability and security of out-of-band channels in secure device pairing protocols. In: Proceedings of the 5th symposium on usable privacy and security. New York, NY, USA: ACM; 2009. 11:1–11:12.

[16] Saxena N, Ekberg J-E, Kostiainen K, Asokan N. Secure device pairing based on a visual channel. In: Proceedings of the symposium on security and privacy. IEEE; 2006.

[17] Gajbhiye S, Sharma M, Karmkar S, Sharma S. Design, implementation and security analysis of Bluetooth pairing protocol in NS2. In: Proceedings of the international conference on advances in computing, communications and informatics (ICACCI). IEEE; 2016. p. 1711–17.

[18] Hassan SS, Bibon SD, Hossain MS, Atiquzzaman M. Security threats in bluetooth technology. Comput Secur 2018;74:308–22.

[19] Haataja K, Toivanen P. Two practical man-in-the-middle attacks on blue-

tooth secure simple pairing and countermeasures. Trans Wirel Commun 2010;9(1):384–92.
[20] Levi A, Çetintaş E, Aydos M, Koç ÇK, Çağlayan MU. Relay attacks on Bluetooth authentication and solutions. In: International symposium on computer and information sciences. Berlin, Heidelberg: Springer Berlin Heidelberg; 2004. p. 278–88.
[21] Suomalainen J, Valkonen J, Asokan N. Security associations in personal networks: acomparative analysis. In: Proceedings of the European workshop on security in Ad-hoc and sensor networks. Berlin, Heidelberg: Springer Berlin Heidelberg; 2007. p. 43–57.
[22] Chang R, Shmatikov V. Formal analysis of authentication in Bluetooth device pairing. In: Proceedings of the joint workshop on foundations of computer security and automated reasoning for security protocol analysis; 2007.
[23] Naveed M, Zhou X, Demetriou S, Wang X, Gunter CA. Inside job: Understanding and mitigating the threat of external device mis-binding on Android. In: Proceedings of the network and distributed system security symposium (NDSS); 2014.
[24] Ellison CM. Ceremony design and analysis. IACR Cryptology ePrint Archive 2007.
[25] Carlos MC, Martina JE, Price G, Custódio RF. An updated threat model for security ceremonies. In: Proceedings of the 28th annual ACM symposium on applied computing. New York, NY, USA: ACM; 2013. p. 1836–43.
[26] 11889-1:2015 I. Information technology – trusted platform module library – part 1: Architecture. Standard. International Organization for Standardization; 2015.
[27] Goldman K, Perez R, Sailer R. Linking remote attestation to secure tunnel endpoints. In: Proceedings of the first workshop on scalable trusted computing. New York, NY, USA: ACM; 2006. p. 21–4.
[28] Parno B, McCune JM, Perrig A. Bootstrapping trust in modern computers. Springer Science & Business Media; 2011.
[29] Fink RA, Sherman AT, Mitchell AO, Challener DC. Catching the cuckoo: verifying tpm proximity using a quote timing side-channel. In: Proceedings of the international conference on trust and trustworthy computing. Berlin, Heidelberg: Springer-Verlag; 2011. p. 294–301.
[30] Zhang Z, Ding X, Tsudik G, Cui J, Li Z. Presence attestation: the missing link in dynamic trust bootstrapping. In: Proceedings of the ACM SIGSAC conference on computer and communications security. New York, NY, USA: ACM; 2017. p. 89–102.
[31] Ding X, Tsudik G. Initializing trust in smart devices via presence attestation. Comput Commun 2018;131:35–8.
[32] Blanchet B. An efficient cryptographic protocol verifier based on Prolog rules. In: Proceedings of the 14th computer security foundations workshop. IEEE; 2001. p. 82–96.
[33] Armando A, Basin D, Boichut Y, Chevalier Y, Compagna L, Cuéllar J, et al. The AVISPA tool for the automated validation of internet security protocols and applications. In: Proceedings of the international conference on computer aided verification. Berlin, Heidelberg: Springer Berlin Heidelberg; 2005. p. 281–5.
[34] Dill DL. The Murphi verification system. In: Proceedings of the international conference on computer aided verification. London, UK, UK: Springer-Verlag; 1996. p. 390–3.
[35] Jia D., Hsu R.. Formal modeling and analysis of Bluetooth 4.0 pairing protocol. 2013.
[36] Aboba B., Blunk L.J., Vollbrecht J.R., Carlson J., Levkowetz H.. Extensible authentication protocol (EAP). http://tools.ietf.org/rfc/rfc3748.txtRFC 3748; 2004.
[37] Aura T, Sethi M. Nimble out-of-band authentication for EAP (EAP-NOOB). Internet-Draft. Internet Engineering Task Force; 2019.
[38] Alliance W-F. Device provisioning protocol specification version 1.0. Tech. Rep.. Wi-Fi Alliance; 2018.
[39] Blanchet B., Smyth B., Cheval V., Sylvestre M.. ProVerif 2.00: automatic cryptographic protocol verifier, user manual and tutorial. INRIA; 2018.
[40] Alliance W-F. Wi-Fi peer-to-peer (P2P) technical specification, v. 1.7. Tech. Rep.. Wi-Fi Alliance; 2016.
[41] Carlos MC. Towards a multidisciplinary framework for the design and analysis of security ceremonies. Royal Holloway, University of London; 2014.
[42] Hancke GP, Kuhn MG. An RFID distance bounding protocol. In: Security and privacy for emerging areas in communications networks, 2005. SecureComm 2005. First international conference on. Washington, DC, USA: IEEE Computer Society; 2005. p. 67–73.
[43] Rasmussen KB, Capkun S. Realization of RF distance bounding. In: Proceedings of the USENIX security symposium. Berkeley, CA, USA: USENIX Association; 2010. p. 389–402.
[44] Kuo C, Luk M, Negi R, Perrig A. Message-in-a-bottle: user-friendly and secure key deployment for sensor nodes. In: Proceedings of the 5th international conference on Embedded networked sensor systems. New York, NY, USA: ACM; 2007. p. 233–46.
[45] IEEE. IEEE standard for local and metropolitan area networks - secure device identity. Tech. Rep.. IEEE; 2009. doi:10.1109/IEEESTD.2009.5367679.
[46] DigiCert. Device certificates. https://www.digicert.com/device-certificates/ Accessed: 11.5.2019; 2019.
[47] Aiello W, Bellovin SM, Blaze M, Canetti R, Ioannidis J, Keromytis AD, et al. Just fast keying: key agreement in a hostile internet. Trans Inf Syst Secur 2004;7(2).
[48] Wu DJ, Taly A, Shankar A, Boneh D. Privacy, discovery, and authentication for the internet of things. Lect Notes Comput Sci 2016:301–19.
[49] Nuss M, Puchta A, Kunz M. Towards blockchain-based identity and access management for Internet of Things in enterprises. In: Proceedings of the international conference on trust and privacy in digital business. Cham: Springer International Publishing; 2018. p. 167–81.
[50] Kravitz DW, Cooper J. Securing user identity and transactions symbiotically: IoT meets blockchain. In: Proceedings of the global internet of things summit (GIoTS). IEEE; 2017. p. 1–6.