SN Computer Science (2020) 1:18
https://doi.org/10.1007/s42979-019-0018-8

ORIGINAL RESEARCH

Evaluation of Out-of-Band Channels for IoT Security

Sampsa Latvala[1] · **Mohit Sethi**[1,2] · **Tuomas Aura**[1]

Received: 9 August 2019 / Accepted: 21 August 2019 / Published online: 3 September 2019

Abstract
Secure bootstrapping is the process by which a device gets the necessary configuration information and security credentials to become operational. In many pervasive computing and Internet-of-Things scenarios, it is often not possible to rely on the existence of a trusted third party or other network infrastructure for bootstrapping. Therefore, several device bootstrapping protocols rely on an out-of-band (OOB) channel for initial device authentication and configuration. We begin this paper by understanding the need for OOB channels and performing a literature survey of existing standards and devices that rely on OOB channels. We then look at one candidate bootstrapping protocol: Nimble out-of-band authentication for EAP (EAP-NOOB). We provide a brief overview of the EAP-NOOB protocol and describe its unique OOB channel requirements. Thereafter, we implement three OOB channels for EAP-NOOB using near-field communication, quick response codes, and sound. Using our implementation, we evaluate the usability, security, benefits, and limitations of each of the OOB channels.

Keywords IoT security · Out-of-band · Authentication · EAP-NOOB · QR code · NFC

Introduction

The number of internet-of-things (IoT) devices is growing rapidly. According to a forecast [12], by 2022, the amount of Internet connected devices will reach about 29 billion, of which 18 billion will be IoT-related devices. IoT devices include connected cards, machines, instruments, wearables, and other consumer electronics. Internet connectivity for these devices aims to improve user experience by automating various processes and exchanging information without user involvement. While IoT provides great benefits and opportunities, there exist severe security risks [28]. Insecure IoT devices may act as gateways for attacks against the entire Internet infrastructure. According to Oracevic et al. [28], the biggest challenges in IoT are presently related to security and privacy protection. Therefore, it is important that IoT devices have robust security mechanisms.

Secure bootstrapping is the process by which an IoT device gets the necessary configuration information and security credentials to become an operational part of the network and an IoT ecosystem. IoT device manufacturers currently employ a variety of often proprietary secure bootstrapping techniques for their IoT devices [44]. This contradicts the idea of seamless interoperability of the devices in the IoT while also adding costs and complexity to the manufacturing process. Furthermore, users are often confused when different devices require different processes for initial bootstrapping and device configuration.

Extensible authentication protocol (EAP) is an authentication framework that supports numerous methods for authentication [42]. Nimble out-of-band authentication for EAP (EAP-NOOB) is a new EAP method which provides a nimble approach for IoT device bootstrapping. EAP-NOOB is intended specifically for bootstrapping various IoT devices with limited input and output capabilities [6]. Compared to all other EAP methods, EAP-NOOB does not require any pre-configured credentials on the device. Instead, the device registration in a server database, ownership of the device, and the authentication credentials for both network access

This article is part of the topical collection "Advances in Internet Research and Engineering" guest edited by Mohit Sethi, Debabrata Das, P. V. Ananda Mohan and Balaji Rajendran.

✉ Sampsa Latvala
sampsa.latvala@aalto.fi

Mohit Sethi
mohit.sethi@aalto.fi

Tuomas Aura
tuomas.aura@aalto.fi

[1] Aalto University, Espoo, Finland

[2] NomadicLab, Ericsson Research, Kirkkonummi, Finland

and application-level security are all be established at the time of the device deployment. EAP-NOOB relies on a user-assisted out-of-band (OOB) channel for ensuring security of the bootstrapping and configuration process.

We begin this paper by discussing the reasons for which many secure IoT bootstrapping protocols such as EAP-NOOB rely on an OOB channel. We then perform a literature survey of existing security protocols and devices that use OOB channels ("Background"). Thereafter, we briefly summarize the EAP-NOOB protocol and its unique OOB channel requirements ("EAP-NOOB"). We implement Near-Field Communication (NFC), Quick Response (QR) codes, and audio as OOB channels for the EAP-NOOB protocol ("OOB channel implementation"). We also develop a custom Android application for better understanding the security and user experience of the three OOB channels. Using our implementation, we evaluate the benefits and limitations of each of the OOB channel against the unique requirements of the EAP-NOOB protocol ("Evaluation"). Finally, we provide some concluding remarks and discuss potential future research work ("Discussion and future work").

To summarize, the contributions of this paper are as follows:

1. A thorough literature survey of the use of OOB channels in security protocols.
2. Summarize the EAP-NOOB protocol and its unique OOB requirements.
3. Implement three different OOB channels for the EAP-NOOB protocol.
4. Using EAP-NOOB as a candidate bootstrapping protocol, analyze the benefits and limitations of the three OOB channels implemented.

Background

Many network protocols rely on an authenticated key exchange (AKE) for setting up secure communication and thwarting any network attackers. Indeed, AKE has become one of the cornerstones for many widely deployed Internet protocols such as Transport Layer Security (TLS). Typical AKE schemes require a trusted third party (TTP). This TTP may need to be always online, as is the case for Kerberos. Alternatively, a somewhat offline TTP with certificate authorities (CAs) and the public key infrastructure (PKI) is used by protocols such as TLS.

Relying on a trusted third party is, however, not always possible or desirable. For example, a protocol that uses AKE for network access authentication and security cannot rely on a scheme that requires connectivity to a remote TTP. Similarly, the operational costs and network overhead of deploying certificates maybe prohibitive. The certificate issuance process itself can also have shortcomings leading to vulnerabilities in an AKE scheme. Mayrhofer et al. [24] also note that many pervasive computing and IoT scenarios cannot rely on the existence of a globally trusted third party.

Therefore, several network protocols have taken the approach of involving the user to act as the trusted third party for achieving AKE. These protocols often involve human interaction through an out-of-band (OOB) communication channel. Out-of-band (OOB) refers to a separate communication channel severed from the primary in-band channel over which the actual network communication occurs [11, 19]. OOB channel provides robustness against attacks by introducing a second, independent communication channel. To eavesdrop or perform a man-in-the-middle attack (MITM) on the primary channel, the attacker is also required to gain access to the OOB channel during the protocol execution. This secondary channel is often either used to transmit an authenticated shared secret or to verify information exchanged over the primary communication channel. Protocols that rely on OOB channels can be roughly divided into four different classes based on the characteristics of the OOB channel and the information communicated over it:

1. *Direct key-provisioning* In this method, the cryptographic keys are provisioned directly over the OOB channel. Currently, a security strength of 128 bits is necessary for acceptable cryptographic protection. Therefore, this approach requires a relatively long OOB message for satisfying the key length requirements. For stronger keys, such as 256 or 368 bits, the message becomes even longer. Thus, the OOB channel bandwidth plays an important role. In addition, each key update requires repeated communication over the OOB channel. Furthermore, when the cryptographic keys are provisioned directly over the OOB channel, the confidentiality of the channel becomes extremely important.
2. *Confirming key exchange* In this method, the OOB channel is utilized for only verifying the Diffie–Hellman (DH) key exchange performed over the in-band channel. Therefore, the OOB messages are often short.
3. *Fuzzy OOB channels* In this method, the nodes authenticating each other use a lossy analogue OOB channel. The OOB channel may be used directly for transferring a secret to both the nodes. For example, keys sent directly over a human-perceptible audio channel. Alternatively, the fuzzy OOB channel may be used to confirm the key exchange performed over the primary in-band channel. For example, Sethi et al. [33] use synchronized user drawn patterns as the OOB channel.
4. *OOB channel for login services* This approach provides the user with additional secret information over the OOB channel. For example, the user may receive a PIN code

via the Short Message Service (SMS) for logging into an online service.

While OOB channels provide an efficient mechanism for bootstrapping security in ad-hoc IoT deployments, their benefits can be questioned, since they typically require some level of user interaction and involvement. Requiring users to perform complex operations can not only have a negative effect on the usability, but also the security of the system. Kainda et al. [19] remind us that secure systems have a tendency of being broken by their users. Security is determined by the weakest link, and most often, the user is the weakest link [10, 19]. Therefore, protocols using OOB channels need to also consider the added complexity for the user and should attempt to reduce the number of steps or operations required from the user.

Many modern smart home devices require the user to have a smartphone or a tablet as a companion device for performing the OOB steps. This companion device might be used for both the initial setup process of the devices and for remote control of the devices once they are functional. We will now look at some existing protocols, standards, and devices that make use of OOB channels for security.

Bluetooth

Bluetooth [9] is a widely deployed short-range wireless communication standard. Bluetooth is used as the communication channel in wireless accessories, such as keyboards, computer mice, headphones, and speakers. It takes advantage of OOB channels for pairing devices before they can securely communicate.

The Bluetooth Secure Simple Pairing process consists of three steps [37]. In the first step, the devices being paired perform a Diffie–Hellman (DH) key exchange. This is followed by an authentication step, which is performed over an OOB channel. The Bluetooth standard supports three OOB authentication methods that verify the performed key exchange [9, 37]:

1. *Numeric comparison* In this method, the user is required to compare and confirm that the short six-digit codes are identical on the displays of the two devices. In addition to verifying the key exchange, it provides confirmation that the user has paired the correct devices. This pairing method is suitable for pairing devices with displays and some input capabilities for the user to confirm the pairing.
2. *Passkey entry* In this method, the first device displays a six-digit passkey, which the user is required to enter into the second device. This pairing method is suitable for scenarios, where one device only has input capabilities and lacks a display, e.g., a keyboard.
3. *Out of band* This method relies on an OOB channel, e.g., near-field communication (NFC). The channel is utilized for discovering other devices and exchanging or transferring cryptographic information needed for completing the device-pairing process.

Finally, in the third step, both devices confirm the DH key exchange with message authentication codes (MAC) over the in-band channel [37].

Bluetooth also supports an unauthenticated Diffie–Hellman key exchange for devices that have no input/output (I/O) capabilities. This just-works method only protects against passive eavesdropping while remaining vulnerable to man-in-the-middle attacks.

Nest

Nest Labs[1] is a smart home appliances manufacturer. These smart appliances range from cameras, thermostats, doorbells, and locks. To function correctly, Nest devices require Internet access, a companion Nest smartphone application, and a registered user account. Nest devices must be paired over Bluetooth before they can be configured. The pairing process is completed with the help of QR codes (or serial number and PIN codes) which are affixed to the device or are available in the retail packaging. Configuration information such as the Service Set IDentifier (SSID) and passphrase of the Wi-Fi network that the device should use for Internet connectivity is then sent over the secure Bluetooth connection established.

Chromecast

Google Chromecast[2] is a digital media player which allows users to cast media from their smartphone or tablet onto a television. Chromecast is attached to the television over a high-definition multimedia interface (HDMI) port. Chromecast creates its own local wireless network over which it receives management commands issued by the user through the companion Google Cast application.

To pair Chromecast with the corresponding companion application, a user needs to connect to the wireless network created by Chromecast. Information about the local wireless network of Chromecast, i.e., SSID and passphrase is displayed on the television to which the Chromecast is attached. After a user connects to the network created by Chromecast, the application and the television display OOB verification messages in the form of four-digit codes. This step requires the user to compare and confirm that the codes are indeed

[1] Nest Labs. https://nest.com.

[2] Chromecast. https://support.google.com/chromecast/.

the same. Thereafter, the application asks the user for details of the Wi-Fi network which provides Internet-access for Chromecast.

Newer versions of Google Chromecast also use ultrasound as an additional OOB channel. This is used for pairing Chromecast with guest devices.[3] Instead of manually entering or verifying any PIN codes, the Chromecast sends PIN codes via the speakers of the television to which it is attached. Chromecast uses high-frequency sound to send ultrasonic tokens which are picked by the guest user's smartphone. The pairing process then completes and the guest user can cast digital media to the TV. PIN codes sent to guest users are only valid for a day.

Apple

Apple manufactures various wireless accessories, such as, keyboards and headphones for their smartphones, tablets, and computers. As with other wireless devices, Apple devices require pairing before they can securely communicate. Most Apple devices communicate with Bluetooth and follow the pairing process defined by the Bluetooth specification. Devices with limited I/O capabilities, such as AirPod[4] headphones, rely on the just-works pairing method. The initial pairing process is triggered by opening the charging case of the headphones near another Apple device, e.g., iPhone. User is then required to accept the UI prompt displayed on the smartphone, which finalizes the pairing process.

For pairing Apple devices with displays, Apple employs a custom moving image [4]. This method of pairing requires users to already have a registered Apple product with a functional camera. An example of this would be the pairing process between an Apple Watch[5] and an iPhone. After turning on the watch, nearby iPhones with Bluetooth enabled are prompted with a UI notification to pair with the watch. Accepting the prompt opens the camera application on the phone and the watch simultaneously displays a swirling image. The watch and the iPhone are successfully paired once the swirling image has been captured by the camera application. For scenarios, where an iPhone camera is not available, Apple Watch supports manual pairing. In this method, the user identifies the watch on the phone with a five-digit identifier. After choosing the watch from the available Bluetooth devices list, the watch displays a six-digit code which the user is required to input into the phone. This approach follows the passkey entry method of Bluetooth Secure Simple Pairing.

Apple Home[6] is an application that allows users to pair and control many Apple Homekit certified smart home devices. Although the pairing process may vary slightly for each Homekit certified device, all of them rely on transferring static codes over an OOB channel. The codes are either attached to the device itself or are found inside the retail packaging. As an illustrative example, we describe the pairing process between a Homekit FIBARO sensor[7] and an iPhone. Once a FIBARO sensor device is powered up by removing the battery blocker, it shows a blue light indicating that it is ready to be paired. The user then selects the sensor device from the list of nearby Bluetooth devices shown on the iPhone. To complete the pairing process, the user enters an eight-digit PIN code or scans a QR code when prompted. This PIN code or QR code is found inside the retail packaging.

Group Messaging

Popular group messaging applications, such as Telegram,[8] WhatsApp[9] and Signal,[10] support end-to-end encryption with OOB verification. The verification requires users to compare information shown on each other's devices.

In Telegram, encrypted communication is established with a Diffie–Hellman key exchange [39]. Based on the key exchange, a picture is generated with additional textual representation of the keys [40]. To confirm that the end-to-end connection is secure, users compare the pictures. If the images are identical on the participating devices, then users can have a strong guarantee that the connection is secure.

To confirm encrypted end-to-end communication in WhatsApp, users visually compare a 60-digit string [43] shown on each of their devices. The 60-digit string is constructed by hashing each user's public identity key to a 30-digit string and concatenating the two strings. A study conducted by Naor et al. [26] showed that the approach of numerical comparison in WhatsApp remains vulnerable to man-in-the-middle attacks when the users are lazy and only compare half (or less) of the 60-digit string. WhatsApp also provides the option of using QR codes instead of visually comparing strings. In this method, users scan a QR code shown on the other user's device. After scanning the QR

[3] Chromecast Guest mode. https://developers.google.com/cast/docs/guest_mode.

[4] AirPods. https://www.apple.com/airpods/.

[5] Apple Watch. https://www.apple.com/watch/.

[6] Apple Home. https://www.apple.com/ios/home/.

[7] FIBARO Sensor. https://manuals.fibaro.com/hk-door-window-sensor/.

[8] Telegram. https://telegram.org, and Secret chat. https://telegram.org/faq.

[9] WhatsApp. https://whatsapp.com.

[10] Signal. https://signal.org.

code, WhatsApp displays a green check mark if the public identity key embedded in the QR code matches with public identity key originally registered with WhatsApp.

One-Time Passwords

One-time password (OTP) sent via the short message service (SMS) is one of the most used multi-factor authentication and authorization schemes [32]. Users must have access to a registered smartphone in addition to the typical user ID and password information for logging into an online service that uses OTPs. SMS is used by many banks, online stores, and social networks [36] for sending OTPs. However, in some cases, the way these methods are implemented does not protect against phishing and man-in-the-middle (MITM) attacks. Some service providers, including various Internet forums, even use access to an email account as an OOB channel to verify users. However, Grassi et al. [16] suggest that email should no longer be considered a secure out-of-band authentication channel.

Most banking systems in Finland take advantage of key lists in online banking and authentication. These are small paper sheets with printed random numbers which are asked during login process and during confirmation of payments and wire transfers. Furthermore, these numbers are used for verifying the identity of the user when logging into public services. However, most banks now provide an application to generate OTPs instead.

EAP-NOOB

In this section we provide a brief overview of the Nimble Out-of-Band authentication for EAP (EAP-NOOB) protocol [6]. We also document the unique OOB channel requirements of EAP-NOOB.

As stated in "Introduction", EAP-NOOB is a new EAP method that is specified as an open standard. It is intended as a generic protocol for bootstrapping IoT devices. As is the case with many IoT devices, EAP-NOOB specifically supports devices that have limited input and output capabilities. EAP-NOOB does not require the devices to have any pre-configured authentication credentials. Instead, device configuration and registration to a server database along with ownership information and authentication of newly created credentials are all performed during the initial device deployment. In this regard, EAP-NOOB is unique compared to most other EAP methods that assume some credentials have been provisioned.

Since EAP-NOOB does not require any pre-configured credentials, it follows the common device-pairing approach of performing a Diffie–Hellman (DH) key exchange over insecure network. However, the key exchange alone does not provide authentication, and therefore, the key exchange is authenticated with a message sent over an OOB channel. This prevents impersonation and man-in-the-middle attacks on the in-band channel.

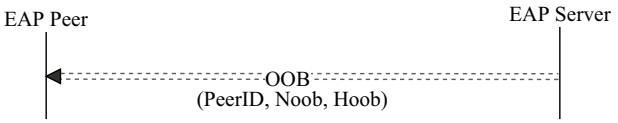


Fig. 1 OOB message is generated at the EAP server and is delivered to peer over the OOB channel [6]

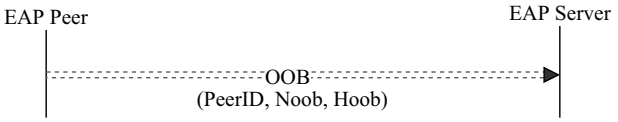


Fig. 2 OOB message is generated at the EAP peer and is delivered to server over the OOB channel [6]

EAP-NOOB contains two security features used in the OOB message. The protocol relies on the secret nonce (Noob), which is used as the first authentication feature. The secret nonce is utilized for mutually authenticating the session key established between the peer IoT device and the server. The second authentication feature is the cryptographic fingerprint (Hoob), which is used to verify the integrity of the key exchange. The end point that receives the OOB message uses this fingerprint to detect impersonation and man-in-the-middle attacks on the in-band channel.

Another feature of EAP-NOOB protocol is that it allows peer devices to scan for possible wireless networks and, based on the results, generate multiple dynamic OOB messages. These messages are relatively long and require partly automated means of transfer. The main purpose of an OOB message is to mutually authenticate the peer devices and the server. OOB messages can be sent from the peer device to the server (shown in Fig. 2) or from the server to the peer device (shown in Fig. 1). If the IoT device being bootstrapped has an output interface, such as a smart TV, then the OOB message is generated and shown to the user by the peer device. If on the other hand, the IoT device being bootstrapped has only an input interface, such as a web camera, then the OOB message is generated and shown to the user by the server. Lastly, if a peer device has both input and output interfaces, the protocol allows both the server and the peer to show an OOB message and let the user deliver any one of them to the other endpoint. However, in this paper, we only focus on scenarios, where the OOB message is sent from the peer to the server.

EAP-NOOB requires the side generating the OOB message to show additional metadata so that a user can correctly deliver the OOB message between the peer IoT devices and the server. If the OOB message is generated on the peer device, then information such as the server name and the SSID of the wireless network is shown along with each of the OOB messages. If, on the other hand, the OOB message is generated on the server, then information such as the peer IoT device make, model and serial number are shown by the server.

OOB messages in the EAP-NOOB protocol have a limited lifetime within which they must be delivered from the peer to the server or vice-versa. The protocol recommends a fairly generous timeout value of 3600 s but leaves the actual choice to the deployment scenario. During this time, the OOB message is valid and should be accepted for completing the EAP-NOOB authentication. The side generating the OOB message is expected to generate new OOB messages containing fresh nonces (Noob) at regular intervals. The protocol specification recommends a refresh cycle that is half of the OOB message lifetime.

Although EAP-NOOB requires a user-assisted OOB channel, it is up to the specific deployment to choose from options such as audio, QR code, NFC etc. The protocol specification suggests that it may be convenient to encode the OOB message as a Universal Resource Locator (URL). This method is suitable for scenarios, where the OOB message is sent from the peer device to the server (Fig. 2). The OOB message can be delivered to the server by simply visiting the URL in any standard web browser. The URL consists of the server domain name and additionally carries the PeerId, secret nonce (Noob) and fingerprint (Hoob) as query string parameters.

The EAP-NOOB protocol specification sets some restrictions and suggestions for the URL. The server domain name has a maximum length of 60 characters. The PeerId provided by the server and carried in the OOB message has a maximum length of 60 bytes and should not include the '+' sign. To shorten the query parameters, the specification suggests a PeerId length of 22 characters resulting from base64url encoding. The secret nonce (Noob) and the fingerprint (Hoob) are both specified as 16-byte values, which are encoded into character strings with base64url encoding. After encoding the length of both strings is 22 characters. This results in approximately a 70–130 character string that needs to be transferred over the OOB channel. The length may vary depending on the varying length of the used data fields, i.e., server domain name and PeerId.

OOB messages encoded as URLs can easily be opened by users with a smartphone if they are embedded in QR codes or NFC tags. This is ideal for IoT devices with limited input or output. For instance, a simple printer may output a printed QR code on paper, or it may have an NFC module that can

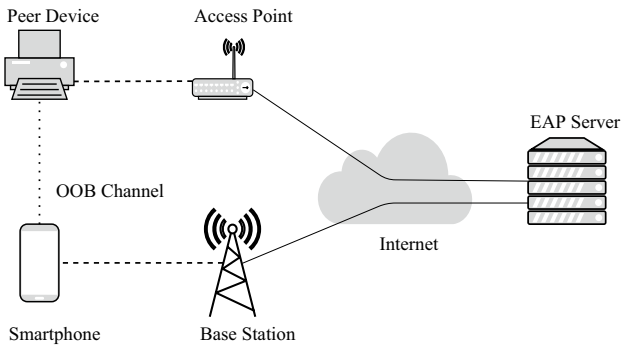


Fig. 3 EAP-NOOB setup

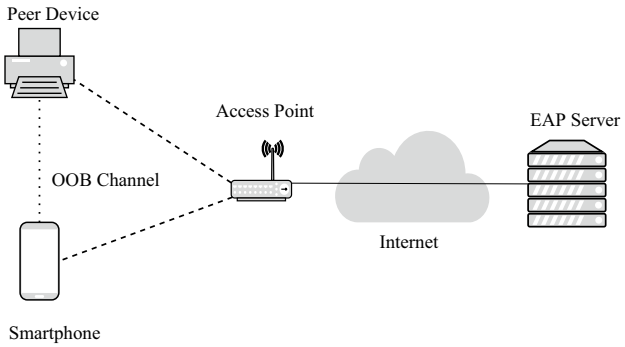


Fig. 4 EAP-NOOB setup, where peer device and smartphone share the same network

be tapped with a smartphone. Another example could be a digital signage display that during the setup process can display the OOB message embedded in a QR code. Both scenarios rely on the user for transferring the OOB message and finalizing the registration process by visiting the URL in the web browser of the smartphone. Such a deployment of EAP-NOOB requires the user's smartphone to have an Internet connection. Users will typically have access to the Internet on their smartphones through the traditional mobile network (3G, 4G, and 5G) as shown in Fig. 3. However, there may be scenarios, where both the peer IoT device and the user's smartphone are accessing the Internet through the same wireless network (shown in Fig. 4). The later scenario requires greater user alertness. This is because if the access point (AP) itself is compromised, the user might be exposed to phishing attacks and may inadvertently register the peer IoT device to the wrong server.

Example data fields in the OOB message when encoded as URL are shown in Table 1. ?P= indicates the PeerId which the server has allocated for the peer device.&N= indicates the secret nonce (Noob) and&H= indicates the cryptographic fingerprint (Hoob).

The EAP-NOOB specification [6] has undergone multiple iterations. Experimental implementations have been

Table 1 OOB message format as URL

Data field	Prefix	Example data
ServerURL	https://	https://example.com/Noob
PeerId	?P	?P=ZrD7qkczNoHGbGcN2bN0
Nonce	&N	&N=rMinS0-F4EfCU8D9ljxXA
Fingerprint	&H	&H=QvnMp4UGxuQVFaXPW_14UW

developed to examine the protocol [25, 41]. Furthermore, the EAP-NOOB protocol has been modeled with mCRL2[11] formal modeling language [30] to simulate protocol behavior and with ProVerif[12] tool for verifying its security characteristics [34].

OOB Channel Implementation

In this section we describe the implementation of three different OOB channels. We begin by describing the tools used in this process, followed by the implementation of NFC, QR code and audio as OOB channels. Finally, we describe the development process of the Android application.

Tools and Setup

Android Studio[13] is an Integrated Development Environment (IDE) that provides various software development tools including an Android emulator. This tool was used in developing and testing the Android application during the implementation process.

Furthermore, a laptop computer is used for simulating the peer IoT device behavior, i.e., for generating and sending OOB messages. The laptop is equipped with an NFC card reader, speakers and a display. The messages are encoded as URLs and transmitted over NFC, QR codes and sound. While NFC and QR codes are natively supported by Android devices, we developed a custom Android application to interpret OOB messages sent over the audio channel. We added support for NFC and QR codes to our own Android application so that all the three OOB channels can be used within the same application. Our Android application was written in the Java programming language. For simulating the peer device behavior, we used Java for generating OOB messages sent via NFC. At the same time, we used Python for generating OOB messages sent via QR codes and audio channel.

[11] mCRL2. https://www.mcrl2.org/web/user_manual/index.html.

[12] ProVerif. https://prosecco.gforge.inria.fr/personal/bblanche/proverif/.

[13] Android Studio. https://developer.android.com/studio.

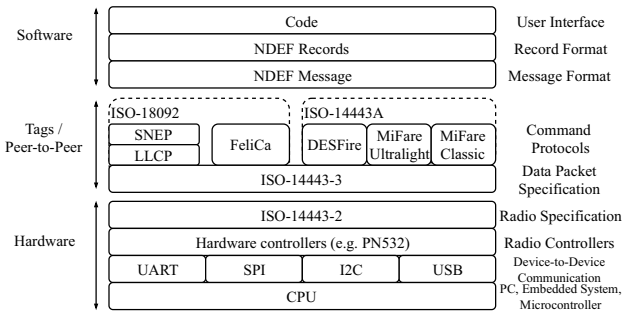


Fig. 5 The NFC protocol stack [17]

NFC

ACR122U-A9 is a contactless smart card reader manufactured by Advanced Card Systems Ltd. It is compliant with both the ISO 14443 and ISO 18092 standards. ACR122U-A9 supports MIFARE and ISO 14443 Type A/B cards, as well as FeliCa and NFC tags [1]. It is equipped with NXP Semiconductor's PN531 NFC controller chip. In addition to the basic read-write and peer-to-peer modes, this smart card reader can also enter a card emulation mode via the TgInitAsTarget command. The card emulation mode allows the reader to act as an NFC target to which contents can be written [27]. However, the device does not have its own memory. This means that all operations and commands need to be issued from the computer to which the device is connected. To send NFC signals via the NFC card reader, the device needs its own native Application Protocol Data Unit (APDU) commands. Fortunately, there exists a collection of Java libraries[14] for issuing these commands to the card reader.

NFC Data Exchange Format (NDEF) is a standard data format which can be used for exchanging information between NFC devices and tags. For delivering the OOB message, i.e., URL from the peer IoT device to the user's mobile device, we use the standard NDEF message format. The URL is, therefore, encoded as a well-known type (TNF value of 0x01) with full URI reference urn:nfc:wkt:U. Although we developed our own native Android application, we also wanted to investigate the default Android behavior for smartphones which do not have our application installed. Therefore, we chose to use the Simple NDEF Exchange Protocol (SNEP) for communicating the NDEF records (Fig. 5). SNEP is a stateless request–response protocol and it is supported in Android since version 4.0.

As we recall from "EAP-NOOB", the EAP-NOOB protocol allows the peer to scan for available networks and,

[14] NFC tools. https://github.com/grundid/nfctools.

based on the scan, generate multiple OOB messages. When multiple OOB messages encoded as URLs need to be sent over NFC, there are the following options available:

1. Encoding each URL as a separate record, i.e., each scanned network is encoded as a separate record within the NDEF message. However, most Android Internet browsers declare an NFC intent filter for URLs and as such, only process the first record by opening the link while discarding rest of the information.
2. Encoding the first URL as an NDEF well-known text record type and the remaining URLs as NDEF well-known full URI records. The first text record includes information of the peer device such as its make and model.
3. Encoding all URLs as NDEF well-known text records. This allows additional information to be included with each URL. However, some Android devices may not identify included URLs as links, and therefore, they may not be clickable.

Android devices always use the first record in an NDEF message to determine how the rest of the message should be interpreted. We can take advantage of this and encode the first record as a well-known text record (urn:nfc:wkt:T). This method circumvents the automatic behavior of opening URLs on most Android devices. The user is thus able to access all the following NDEF records in the NDEF message on default reader and choose which URL to open. Moreover, the text record can include identifying information regarding both the peer device as well as the available networks.

Opening the URL contained in an OOB message authorizes the peer device to access the network and registers it to the user's account on the server. Therefore, it is essential for the user to identify the correct server. In the presence of multiple networks, the user needs to carefully choose the familiar server URL from the list. This may be a difficult task on Android devices that do not display the full server address. Therefore, in some scenarios it could be beneficial to include an Android Application Record (AAR) within the NDEF message to open a custom application. AAR is encoded as External NDEF type (TNF = 0x04) with URI reference android.com:pkg and the application name as the payload. AAR can either prompt the smartphone user to the Google Play Store or launch the application if it is already installed. This can be used to guide users to a custom application designed for the OOB channel. However, including an AAR within the NDEF message may be infuriating for users who do not wish to use designated applications or do not have access to the application store.

Table 2 Supported acoustic transmission methods of proprietary Chirp protocol

Mode	Max. message length	Transmission time	Data rate
Standard	32 bytes	4.52 s	56.6 bps
16 kHz	90 bytes	8.16 s	88.2 bps
16 kHz-mono	32 bytes	4.48 s	57.1 bps
Ultrasonic	8 bytes	4.08 s	15.7 bps

QR Codes

QR codes are generated with the Python qrcode[15] library package. In addition, Python Imaging Library (PIL)[16] is used for adding metadata such as the server name into the QR code image above the quiet zone. This metadata helps the user to identify the correct OOB message in scenarios, where the peer device has scanned multiple available networks. The added human-readable information also gives the user some indication of the QR code content. The QR codes follow the standard URI scheme and begin with prefix https://.

There are two ways for encoding multiple URLs containing different OOB messages. Since most of the default QR code readers do not support multiple URLs in a single QR code, each URL can naturally be encoded as a separate QR code. Alternatively, each included URL is partitioned with a null character and encoded into a single QR code. However, as more characters are encoded into a single QR code, the pattern becomes more complex. Dense and complex patterns require more time to be successfully decoded. We also cycle through the first URL when multiple URLs are encoded into the same QR code. This allows the default readers to scan a different entry on each scan. Advanced QR code readers can read the complete QR code with all the URLs in one scan.

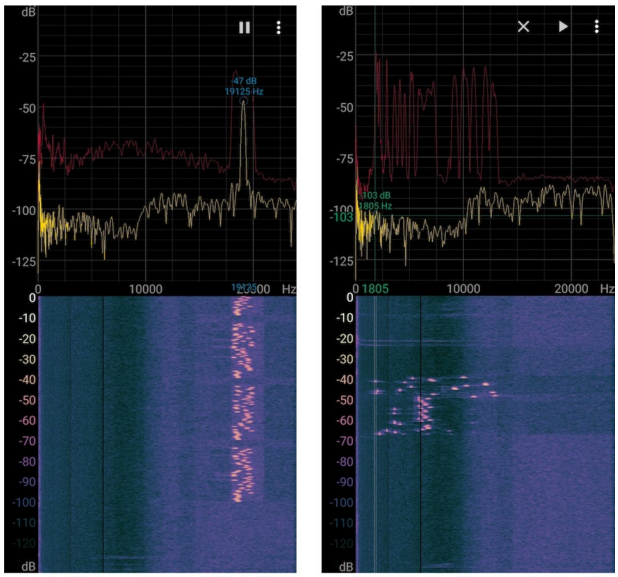
Audio Channel

Chirp.io[17] provides a Software Development Kit (SDK) for multiple platforms and programming languages. In this paper, we use the Chirp.io SDK for transmitting the OOB message from a peer device (Python Chirp SDK) to a smartphone (Android Chirp SDK). Chirp SDK uses a proprietary protocol and supports four acoustic transmission methods shown in Table 2. These are labeled as standard, 16 kHz, 16 kHz-mono and ultrasonic. Each method has a maximum message length. The main differences between these

[15] qrcode 6.1. https://pypi.org/project/qrcode/.

[16] Pillow 6.0.0. https://pypi.org/project/Pillow/.

[17] Chirp. https://chirp.io.



(a) Chirp in ultrasonic mode. **(b)** Chirp in standard mode.

Fig. 6 Screen capture of monitored audio graph of Chirp transmission with ultrasonic and standard setting

methods are the sound frequency, message length and data transfer rates. In the standard audio transmission mode, the length of the message is limited to 32 bytes, which is sent in a 4.52-second time frame. This results in a data rate of 56.6 bps. In the 16 kHz mode, the maximum message length is 90 bytes, which is sent in 8.16 s. This provides the fastest option with a data rate of 88.2 bps. For 16 kHz-mono option the data rate is 57.1 bps, while the ultrasonic option has the lowest transfer speed of 15.7 bps. To achieve faster data rates, the protocol supports multiple channels within the modes; however, this option requires special developer privileges.

Chirp SDK expects the payload as an array of bytes and the SDK transforms it into audible sound. In the standard mode, the audible sound follows a melodic pattern. Each message follows a recognizable pattern consisting of an initial chirp pattern which is followed by the encoded data and the end-of-message pattern. We found from our experiments that the standard and ultrasonic options proved to be the most robust transfer methods in the presence of noise (music or chatter did not disrupt message), while the 16 kHz option did not work on laptop speakers and required high-end speakers for the smartphone to interpret the signal. Furthermore, closer examination of the sound profile confirmed that the ultrasonic mode is not technically ultrasound, since the operating frequencies are between 18 and 20 kHz, as shown in Fig. 6a. In the standard mode the frequency range is between 1.8 and 13 kHz, shown in Fig. 6b. Both figures display the same message that contains a URL. In addition, it displays the spectrogram visualization of the transmission of the 32-byte message. In the ultrasonic mode, the message is required to be split into 8-byte parts, while the standard mode was able to transmit the whole message in one burst.

The transmission was recorded with a Samsung Galaxy S8 with the Spectroid[18] application. Since the standard mode provided reasonable data rates, good robustness against noise, and was audible; we chose it as the transfer method for the final implementation of the sound-based channel between the computer and the smartphone. To combat data corruption and modification attacks, the audio channel is listened to only when the user presses a button, i.e., is prepared to perform the OOB step.

As described in "EAP-NOOB", the OOB messages of the EAP-NOOB protocol are relatively long when encoded as URLs. Therefore, most of the URLs need to be split into multiple messages. To indicate the end of a URL, the message is terminated with '+' sign, which is then removed from the final URL. The Chirp protocol could be used for two-way communication to take advantage of acknowledgements; however, this would significantly lengthen the duration of transferring the OOB message. It would also increase the time window for any possible attacks. To indicate that the message is being received, the user is displayed a status bar. This also makes the wait time more pleasant for the user and keeps the user's attention [23]. Furthermore, each part of the message received is shown to the user as a visual confirmation that the message parts are being received correctly.

Android Application for EAP-NOOB

To avoid the identified problems with default NFC and QR code reader applications, we developed our own Android application for investigating if the security and usability can be improved. Our application supports NFC, QR codes and audio OOB channels.

Android includes NFC data handlers in their Android framework API [14]. We take advantage of Android NFC tag dispatch system. Our application is declared to filter for both HTTPS as well as plain-text NDEF tags. Therefore, each time a NDEF tag is scanned, the Android operating system launches our application and passes the NDEF data without requiring any further actions. If there are multiple applications that filter for this, the user needs to choose the application to handle the data. This is less intrusive approach compared to the AAR approach discussed in "NFC".

Our application is capable of decoding QR codes. There were two options available for decoding QR codes: taking a picture and then processing the image or decoding the QR code from the camera feed. We implement the latter

[18] Spectroid. https://play.google.com/store/apps/details?id=org.intoorbit.spectrum.

method, because it decodes the QR codes immediately without requiring access to the device's storage. This way, the user is only required to click the register button once the correct server is identified. We take advantage of Google's Barcode API,[19] which is a part of Google's Mobile Vision API. While supporting numerous barcode formats, the object detection is restricted to only QR codes. Initially, the scanning process was slow due to the camera being out of focus most of the time. However, this was solved by utilizing the camera's auto focus capabilities, which expedited the scanning process significantly. To indicate a successful scan, the device vibrates. This notifies the user if the QR code is successfully decoded.

The initial view of our application displays short instructions for navigating the application. The main view also functions as an NFC reader and features a row of navigational buttons. These buttons allow the user to navigate to the QR code reader, audio channel decoder or access application options. Android features a default navigation bar, which includes navigation options for back, home, and overview. Since most of the Android users are familiar with the navigation bar, we use the back button for navigating back to the main activity.

Our application contains an important additional security feature: a list of trusted servers. Users can add custom entries of trusted servers to this list using the application options button. Whenever users tap an NFC tag, read a QR code, or record Chirp audio; our application notifies if the URL in the OOB message is from a trusted server. Furthermore, the application supports optional HTTP basic authentication as specified in RFC 7617 [31] and RFC 7235 [13]. HTTP basic authentication is a simple challenge and response method with which servers can request user authentication information from the client. To take advantage of this feature, users need to manually enter the authorization credentials (username and password) when adding a new trusted server. User credentials are automatically sent over secure HTTPS if the URL in the OOB message belongs to a trusted server.

Adding a trusted server and entering login credentials a priori allows the user to register new peer devices with a single tap or scan without requiring any further action. The requirements for singe-tap or scan authentication are that the peer device does not find several trusted networks, the server is added to the list of trusted servers, and the user login credentials are configured. User login credentials are stored within the application's persistent storage using the Android SharedPreferences.[20]

[19] Google Barcode API. https://developers.google.com/vision/android/barcodes-overview.

[20] SharedPreferences. https://developer.android.com/guide/topics/data/data-storage.html#pref.

Both authorization credentials and the list of trusted servers take advantage of Android's SharedPreferences. They are stored as key-value pairs in XML files located in the application's data directory. User credentials is a string and stored with the key token, while server list is stored as a string with the key servers. They are secured by Android's file permission system. However, anyone with root access or the same application UID can access and modify them. Furthermore, our application does not allow cleartext and thus web pages without HTTPS will intentionally not work.

Android features Security-Enhanced Linux (SELinux) to define boundaries for application sandboxing [3, 5]. Therefore, each application runs in a limited-access sandbox. If the application requires resources outside its own sandbox, it is required to request for permissions [15]. To capture QR codes or sound bursts for audio channel, the application requests user's permissions to camera and microphone. These permissions are classified as "dangerous" and, therefore, explicitly require the user's permission [15]. Furthermore, the application requests permissions for accessing the Internet, NFC, and vibration engine. However, these permissions are granted by the system automatically during installation, because they are classified as "normal" [15].

Evaluation

In this section, we evaluate the advantages, limitations, and vulnerabilities of each of the three OOB channels implemented. We also discuss the broader usability aspects of the three OOB channels. Where relevant, we also highlight how each of the three OOB channels perform against the special requirements of EAP-NOOB. In the following subsections, the '−' sign indicates a negative feature and '+' sign indicates a positive feature.

NFC

Android supports NFC by default and it provides an intuitive way of bootstrapping a device by a simple tap. The NDEF data format provides multiple ways for conveying URL encoded OOB messages. However, a potential weakness in the Android default NFC reader is that it does not display the full URLs when the NDEF record is encoded as well-known URI record. In most cases the information left out is the query parameters, such as the nonce and cryptographic fingerprint. Therefore, OOB messages should be encoded as text records with additional information. This method circumvents the behavior of automatically opening URLs and provides information about the networks to the user so that the user can choose the correct network. NFC also provides a viable OOB channel for transferring multiple OOB messages at the same time.

Usability

+ NFC follows the intuitive method of tapping the device. This method of device interaction is becoming well-known and it can be seen, for example, from the growing trend of contactless payment [38].
+ NDEF enables multiple options for encoding the message. Applications can declare intent filters for associating with specific formats, and developers may include an AAR within the NDEF message for the Android system to either launch a specific application or to prompt for installation from the Google Play Store.
+ NFC provides a high-bandwidth channel, which can quickly transfer even relatively long OOB messages without saturating the channel capacity.
– NFC requires specific hardware and is not supported by all smartphones. Apple iPhones have been equipped with NFC modules limited only to contactless payment since iPhone 6; however, the latest iPhones are capable of reading NDEF tags without requiring third-party applications.
– The NFC channel is imperceivable to the human user. Peer devices need to have some indicators, such as a LED light or a buzzer, to notify the user that it is ready to transmit an OOB message.

Security

– The card reader was found to show unexpected behavior when a contactless smart card was already present on the card reader. If an NFC tag (NXP MIFARE Classic 1k, NXP MIFARE DESFire EV1 or NXP MiFARE Plus) was already present on the ACR122U card reader, the smartphone would not detect the tag. After a SNEP command was issued by the reader, the smartphone received the NDEF message of the NFC tag instead of the SNEP command from the reader. If the NFC tag was removed, the reader would successfully transmit the OOB message. In some rare cases, the card reader would malfunction and the only way to get it working again was to remove the card reader and re-attach it. At other times, the ACR122U card reader would get stuck in a loop, where it tries to issue a SNEP command; however, the smartphone would detect the NFC tag instead. This behavior of the reader can potentially allow phishing and misbinding attacks if the attacker has physical access to the peer device before the honest user begins the bootstrapping process. For a phishing attack, the attacker needs to format an NFC tag with a malicious URL that would mimic the appearance of a legitimate server. Thereafter, the attacker needs to insert the NFC tag over the NFC reader of the victim's device. This way, the victim could be fooled into entering credentials or other sensitive information during the device registration process. A prudent user would likely notice the inserted tag, but it could go unnoticed from an inexperienced user.
To perform a misbinding attack, the attacker would first have to initiate the EAP-NOOB protocol on another device. Then, the attacker must scan and copy the OOB message into an NFC tag. Thereafter, the attacker has to insert the tag over the NFC terminal of the honest user's device. After the victim user taps the device, the scan would reveal the OOB message of the attacker's device and thus the user could be fooled into associating it with their account. Now the attacker is in possession of a device that may have access to various user resources. However, this type of attack might be quickly exposed, since the victim would notice that its own device has not completed the EAP-NOOB protocol and is still unregistered on the server. A prudent user would then revoke the unintended device association.
These types of attacks are targeted attacks and they require physical access to the peer device before the bootstrapping begins. As stated by Sethi et al. [34], most device-pairing protocols, where authentication is established by physical access are vulnerable to misbinding. The paper suggests a trusted path as one mitigation mechanism. The authors mention that a LED light indicating direct communication with the peer device hardware could be one such example. The ACR122U card reader includes programmable led lights and a buzzer [2]. The LED light is red if there are no available NFC terminals and green if there is an NFC-capable device within range. Since these indicators are programmable they cannot be considered as a trusted path [22]. However, they can expose the presence of a false NFC tag or sticker on top of the reader. Furthermore, since SNEP is a request–response protocol, the card reader can detect failed transmissions and report them to the user. Nonetheless, these mechanisms only help the user to detect the attack and do not prevent the user from scanning the malicious tag in the first place.
– NFC is vulnerable to eavesdropping. The maximum range for a successful tag read was observed to be 3 cm in passive mode and 8 cm in active mode.

QR Code

QR codes provide a viable option for displaying OOB messages. In scenarios, where there are multiple networks, the peer device's display plays a significant role. It limits how many QR codes the device can output in a size that can be scanned reliably. If the device has a small display, it might

Table 3 QR code readability

PPI	QR code with 6 URLs (cm)	Single-URL QR code (cm)
326	2.3	1.1
227	2.8	2.1
109	4.4	2.5

have to cycle through the QR codes. Both the length of the QR code and the error correction level contribute to the size of the QR code.

Multiple varying 126-character URLs were generated to evaluate the performance of QR codes. Single-URL QR codes smaller than 1.1 cm on a 326 Pixel-Per-Inch (PPI), 2.1 cm on a 227 PPI display, and 2.5 cm on a 109 PPI display were observed to be difficult for the default readers and third-party readers without digital zoom. However, iPhone's QR code reader with the manual digital zoom capability was able to decode even the smaller QR codes. Furthermore, it was noted that the error correction level does not affect the overall read speed of the QR code on test devices. However, it might be a factor for smartphones with less powerful camera sensors. With multiple URLs, the QR code becomes more complex and requires more area. This is shown in Table 3.

If multiple QR codes were displayed close to each other, all the readers displayed the contents of the first detected QR code and ignored the rest. This issue was less prominent in iPhone, since unlike the tested third-party readers, the iPhone default QR reader features a digital zoom, which narrows down the area of detection. Therefore, for peer devices that display the OOB messages as QR codes, it may be more reliable to display one QR code at a time.

Tampering a QR code on a display is unlikely to occur. To produce a tampered QR code on a display, the peer device needs to be compromised beforehand. In this scenario, the user has no way of knowing that the device is compromised. Therefore, it should be recommended to reset the peer device before initiating the EAP-NOOB registration process. Tampering of the QR code is also possible on printed surfaces. This would require the legitimate user to leave the device unsupervised during the initial device deployment. To combat tampering, Krombholz et al. [21] suggest using complex color schemes. This causes the attack on QR codes to be costlier and makes it difficult to modify the QR code in an undetectable way.

Usability

+ QR codes are widely supported by smartphones due to large application stores. Furthermore, numerous URI schemes are supported by the Android and iOS mobile operating systems.
+ QR codes are easy to use by simply focusing the camera towards the code. However, various studies have shown the scanning of QR codes to be a difficult task for users who are not familiar with them [34, 35].
– The QR code standard does not support multiple URLs and, therefore, displaying multiple URLs requires a separate QR code for each URL. This can result in a cumbersome scanning task in environments with numerous networks. In custom solutions, a single QR code can be encoded with multiple URLs with delimiters. However, this eliminates the compatibility with most readers. To maintain some support for default readers, it is possible to separate the URLs with the null character, which most readers interpret as the terminator. Default readers would be able to read the first URL, while custom solutions would be able to read the complete QR code. It is also important to note that encoding multiple URLs in the same QR code naturally makes it denser.
– Closely placed QR codes proved to be difficult to scan reliably. Neither default readers nor third-party solutions allowed the user to choose which QR code to decode. Only the iPhone default QR code reader displayed a marker over the QR code which it had decoded.
– It is possible to generate QR codes that contain regular text together with a URL. The URL in such cases is not clickable on default readers. Only half of the tested third-party readers identified the URL link accompanying the text.

Security

– Since QR codes are only machine readable, the human user cannot determine their content without a mobile device. This is problematic with devices that process the QR code content without user action. Therefore, it is advisable to include the URL in plain-text form next to the QR code to show the content to the user.
– QR codes provide a visual OOB channel, and therefore, the implementation is vulnerable to opportunistic snooping attacks, such as shoulder surfing with a camera in public environments. Furthermore, QR codes can be spied over greater distances, e.g., by taking advantage of optical telescopes.

– Printed QR codes are vulnerable to tampering. This may occur in scenarios, where the peer device outputs a printed QR code and the user leaves the printed QR code unsupervised. However, using color schemes increases the difficulty of unobtrusive tampering.

Sound

The sound channel provides a viable option in scenarios, where there are only a few OOB messages to be sent. This is mainly due to the low bandwidth of the audio channel. Furthermore, since the relatively long OOB messages of EAP-NOOB must be split into multiple shorter messages, the initial and end-of-message patterns add to the duration of the process. On average, a 126-character URL is successfully transferred in around 20 s. If the transfer is disrupted, the whole process must be started over again.

A long waiting period of over 15 s is often seen as detrimental to productivity and lowers user satisfaction [23]. However, the animated loading bar in addition to the updated status of the OOB message should significantly reduce the user's time estimation and, therefore, restlessness. Nonetheless, after the first OOB message, the user experience for utilizing the sound-based channel may become annoying due to the relatively long wait time.

While the transmitted data is not encrypted and can be easily eavesdropped, the channel is resilient against man-in-the-middle attacks. This is because the transmitted data is authentic and users can correctly verify that the data originates from their peer device. Attempts to transmit audio data over the original signal can be easily observed by listening, which exposes spoofing attacks. The implemented audio channel is also resilient against data modification attacks. This is due to the fixed length of messages. Data insertion attacks results in corrupted data, which also highlights the main vulnerability of the channel: the channel is easily disrupted. This can be achieved by playing louder noise at the same frequency. A gust of wind reaching the device's microphone during transmission will similarly disrupt the process.

Usability

+ Sound signal strength is easily controlled with the device's speaker volume.
+ Audio channel is suitable for smart devices with only speakers or microphone, in environments with only one or two networks.
– The audio channel has very low bandwidth. Therefore, transmitting relatively long OOB messages introduces a lengthy waiting period for the user. Waiting for 20 s to receive an OOB message and then requiring user action can be extremely unpleasant.
– Humans have varying thresholds for which frequencies are perceived as unpleasant. However, for close-range transmissions the audible signal does not need to be loud.
– The audio channel is unreliable in environments, where the microphone picks up a lot of noise. Even a small breeze of wind to the microphone is enough to corrupt a message. Therefore, the audio channel is not viable for outdoor scenarios in windy conditions.
– In our implementation with Chirp, users must initiate the audio transmission on the peer device once the smartphone has started recording. This is because the Chirp SDK is not capable of sorting out-of-order messages.

Security

+ Sound frequencies used by the Chirp protocol are limited to rooms and the high-frequency audible signal does not travel through walls or windows.
+ Audible tune can be perceived and the user can confirm the signal is coming from the device. One of the problems in device-pairing protocols is the human imperceptibility of the wireless signal [20]. The audio channel mitigates this problem.
+ The sound protocol is immune to data modification attacks during active data transfers. Attempts to alter the data cause the messages to be corrupted.
– The channel can be easily disrupted. It is possible to disrupt the channel by playing sound at similar frequencies as the Chirp protocol.
– If ultrasound is used, the inaudible sound cannot be monitored, and the disruptive signals are hard to detect without tools.
– As with NFC, the channel is vulnerable to eavesdropping. This would require the attacker to be in the same room or have an eavesdropping device in the room during the OOB message transfer.

Android Application for EAP-NOOB

Overall, the application enhances the security and usability of the protocol. By filtering untrusted servers, it can prevent against most phishing and misbinding attacks. However, this requires the user to manually add trusted servers to the application a priori. The added security results mostly from this restraint. Usability is enhanced over default readers with the NFC intent filter, support for HTTP basic access authentication, and the list of trusted servers.

Usability

+ The application supports HTTP basic authentication. If users add their authentication credentials a priori, then each initial HTTPS request to the server will include the credentials in the authorization header. Thus, users do not need to input the credentials for each authentication attempt. This allows them to immediately register the IoT device to the server without any user action other than the initial NFC tap or QR code scan with the smartphone.
+ The trusted list can also be utilized for filtering networks. This improves usability in environments with multiple networks, since users do not have to manually browse through a list of URLs or server names.
+ The application supports one handed use, because all the UI buttons are placed within the functional area of the thumb following the model specified by Bergstrom-Lehtovirta and Oulasvirta [8] for touchscreen surfaces.
+ Due to the NFC intent filter, the application is immediately launched after scanning NDEF tags with HTTPS URLs and the data is displayed to the user. If the scan finds a trusted server, the OOB message is delivered immediately. In this way, peer devices can be registered with a simple tap on an unlocked Android device.
– The Google's Barcode API does not support null-separated data. Therefore, the application cannot decode QR codes that have several null-partitioned URLs.

Security

+ URLs are not automatically opened and the server URL is displayed to the user.
+ The application supports only HTTPS and insecure HTTP links are discarded.
+ The application protects against phishing and Internationalized Domain Name (IDN) homograph attacks. Users are informed if the URL contains Cyrillic characters that are often used in phishing and IDN attacks. This is done by parsing through the URLs and checking if potentially compromising Cyrillic characters are found. However, it is important to note that this implementation is only a proof of concept and does include thorough character set checks. For protection against phishing attacks, the application also supports a list of trusted servers. This allows the user to identify which scanned networks are trusted.
– While using third-party libraries, the application becomes vulnerable to supply chain attacks. The application developed for this paper takes advantage of a proprietary third-party Chirp SDK, which allows the application to interpret the data transmitted over sound. Furthermore, the SDK sends anonymized analytical data back to the Chirp developers. Chirp claims that the data sent is used to improve the service and no payload data is revealed.
– It is important to note that the Google Barcode API employs a powerful QR code detector as it can detect and process multiple barcodes in real time. This can lead to security problems, where an attacker might be able to insert another QR code within the camera frame near the legitimate QR code. However, this problem exists also on other Android QR code readers. Only the iOS default QR code reader displays a marker over the QR code which has been decoded.

Discussion and Future Work

From our experiments, we find that NFC is the fastest method for transferring OOB messages. This is mainly due to Android's native support for NFC and the fact that it allows applications to declare intent filters. We also examined the default NFC behavior on Android. This examination showed that NDEF records formatted as URLs are automatically opened with the smartphone's Internet browser. For avoiding this unwanted and insecure behavior, we noted that encoding the first NDEF record as plain text eliminates the intent filters set by the Internet browsers. Our implementation also shows that encoding URLs as text records allows more information to be added to each OOB message.

The EAP-NOOB bootstrapping process naturally gets more complex when there are multiple potential networks to which peer devices can connect. However, this is not a problem that is unique to EAP-NOOB. RFC5113 [7] discusses the challenges faced by a user when selecting the correct access network in significant detail. EAP-NOOB addresses this issue by providing server metadata to the peer during the initial contact on the in-band channel. This metadata is then shown to the user along with the OOB messages to aid the selection process. Naturally, this metadata information can be forged by malicious servers. However, if the user is able to identify a familiar server URL, it should be safe to follow. This is because tampered nonces or cryptographic fingerprints result in the failure of IoT device authentication and registration.

In general, the more OOB messages the user is required to browse through, the more work the user needs to perform. This can be problematic in environments with many candidate networks. Only NFC and QR code are potential options from the user experience perspective in deployments with many candidate networks. The low bandwidth of the audio-based channel can be tedious for most users and is

viable only in scenarios with one or two candidate networks to which the peer device could connect.

To alleviate the problem of displaying many QR codes each containing a different URL with an OOB message, we implemented a flexible method which supports both standard QR code readers and more advanced readers. In this method, the peer encodes a QR code with null-partitioned URLs cycling the first entry. Therefore, advanced readers can decode the complete QR code with all the URLs after only one scan, while standard readers are able to decode the first entry on each scan. When multiple OOB messages are encoded into the same QR code, the server metadata from multiple servers is shown along with a single QR code.

All the implemented OOB channels are vulnerable to eavesdropping. For example, an audio-based channel is vulnerable to undetected recording by an attacker. Similarly, QR codes can be spied over from great distances with a camera; and NFC in the active mode is vulnerable to eavesdropping from a distance of 10 m. These vulnerabilities can compromise the confidentiality of the OOB channel. However, OOB messages have a configurable expiration time which reduces the time frame for exploiting a captured OOB message. These attacks also require physical access or proximity to the device. Additionally, as we recall from "EAP-NOOB", EAP-NOOB specification protects against such attacks by adding a cryptographic fingerprint to the OOB message. The end point that receives the OOB message uses this fingerprint to detect impersonation and man-in-the-middle attacks on the in-band channel. Thus, an attacker would need to compromise both the confidentiality and integrity of any OOB channel to carry out a successful MiTM attack on the in-band channel.

As noted by Sethi et al. [34], all device pairing and bootstrapping protocols are vulnerable to misbinding attacks. These attacks require that the device being configured by the user has already been compromised. We noted that misbinding attacks are easier to carry out and harder to detect when NFC is used as an OOB channel. This is because the attacker only needs to place a tag for a different device on top of the reader. Even a device reset does not resolve this issue. To combat this, the EAP-NOOB [6] specification suggests a trusted path indicator such as a LED light or a buzzer.

In addition to implementing three OOB channels, we developed an Android application for EAP-NOOB which can protect users against most phishing attacks. The application can also help the user in selecting the correct network when multiple candidate networks are available. Our application can significantly reduce the extra work which is introduced by the user-assisted OOB channel. This is due to the application's ability to filter out unknown servers, cache user credentials, and use HTTP basic authentication.

Fig. 7 A stylized QR code with a logo featuring SMS URI scheme. Generated using the QRcode Monkey tool https://www.qrcode-monkey.com/

The performed literature survey shows that, overall, device bootstrapping is a multi-step process. In current commercial products, the process often begins with pairing the device to an auxiliary device, such as a smartphone. This is followed by entering the Wi-Fi credentials which the device should use for Internet access. In this regard, EAP-NOOB can be a competitive alternative as a secure device bootstrapping protocol. Our implementation shows that the EAP-NOOB protocol can make the device bootstrapping process as simple as an NFC tap when combined with a designated smartphone application.

This paper focused on URL-formatted OOB messages. However, NFC and QR codes can also be used for constructing OOB messages that are encoded using other URI types. For example, consider an EAP-NOOB deployment, where users deliver the OOB messages from the peer to the server using the Short Message Service (SMS). The SMS character limit can fit the PeerId, nonce and the cryptographic fingerprint. The hostname in URL-formatted OOB messages can be replaced with a telephone number. A peer device in such a scenario can generate an SMS message which can then be displayed as a QR code to the user. One such QR code containing the OOB message encoded in a SMS is shown in Fig. 7.

While this paper examined OOB channels based on NFC, QR codes, and audio; other OOB channels can be experimented with in the future. One such promising OOB channel is Visible Light Communication (VLC) [18, 29]. Finally, additional studies on network discovery performance of the peer device and correct server identification in environments with multiple available networks could also help improve the EAP-NOOB protocol specification.

Acknowledgements Open access funding provided by Aalto University. This study was funded by Academy of Finland (Grant number 296693).

Compliance with Ethical Standards

Conflict of interest The second author is a guest editor for the special issue to which this article was accepted. The authors declare that they have no other conflicts of interest.

References

1. Advanced Card Systems Ltd. ACR122U USB NFC Reader A Product Presentation. ACS. https://www.acs.com.hk/download-manual/11/PPE-ACR122U-2.02.pdf. Accessed 8 May 2019.
2. Advanced Card Systems Ltd. ACR122U USB NFC Reader Application Programming Interface V2.04. ACS. https://www.acs.com.hk/download-manual/419/API-ACR122U-2.04.pdf. Accessed 8 May 2019.
3. Android Open Source Project. Security-Enhanced Linux in Android. 2019. https://source.android.com/security/selinux. Accessed 8 May 2019.
4. Apple: Set up your Apple Watch. 2019. https://support.apple.com/en-us/HT204505. Accessed 5 May 2019.
5. Asokan N, Davi L, Dmitrienko A, Heuser S, Kostiainen K, Reshetova E, Sadeghi AR. Mobile platform security. Synth Lect Inf Secur Priv Trust. 2014;4(3):1–108.
6. Aura T, Sethi M. Nimble out-of-band authentication for EAP (EAP-NOOB). Internet-Draft draft-aura-eap-noob-06. 2019. https://datatracker.ietf.org/doc/html/draft-aura-eap-noob-06 (**Work in Progress**).
7. Bari F, Arkko J, Aboba BD, Korhonen J. Network discovery and selection problem. RFC 5113. 2008. https://doi.org/10.17487/RFC5113.
8. Bergstrom-Lehtovirta J, Oulasvirta A. Modeling the functional area of the thumb on mobile touchscreen surfaces. In: Proceedings of the SIGCHI conference on human factors in computing systems. ACM; 2014. p. 1991–2000.
9. Bluetooth SIG. Bluetooth core specification version 5.1. 2019.
10. Böhme R, Grossklags J. The security cost of cheap user interaction. In: Proceedings of the new security paradigms workshop. ACM; 2011. p. 67–82.
11. Buttyan L, Gligor V, Westhoff D. Security and privacy in ad-hoc and sensor networks: third European workshop, ESAS 2006, Hamburg, Germany, September 20–21, 2006, revised selected papers, vol. 4357. Berlin: Springer; 2007.
12. Ericsson: Internet of things forecast. 2018. https://www.ericsson.com/en/mobility-report/internet-of-things-forecast. Accessed 5 Mar 2019.
13. Fielding RT, Reschke J. Hypertext Transfer Protocol (HTTP/1.1): Authentication. RFC 7235. 2014. https://doi.org/10.17487/RFC7235.
14. Google. NFC Basics. 2019. https://developer.android.com/guide/topics/connectivity/nfc/nfc. Accessed 14 Mar 2019.
15. Google Developers. Request App Permissions, 2019. https://developer.android.com/training/permissions/requesting. Accessed 1 Mar 2019.
16. Grassi PA, Garcia M, Fenton J. NIST special publication 800–63-3 revision 3 digital identity guidelines. Los Altos: National Institute of Standards and Technology; 2019.
17. Igoe T, Coleman D, Jepson B. Beginning NFC: near field communication with Arduino, Android, and Phonegap. Newton: O'Reilly Media Inc; 2014.
18. Jovicic A, Li J, Richardson T. Visible light communication: opportunities, challenges and the path to market. IEEE Commun Mag. 2013;51(12):26–32.
19. Kainda R, Flechais I, Roscoe A. Usability and security of out-of-band channels in secure device pairing protocols. In: Proceedings of the 5th symposium on usable privacy and security. ACM; 2009.
20. Kobsa A, Sonawalla R, Tsudik G, Uzun E, Wang Y. Serial hookups: a comparative usability study of secure device pairing methods. In: Proceedings of the 5th symposium on usable privacy and security. ACM; 2009.
21. Krombholz K, Frühwirt P, Kieseberg P, Kapsalis I, Huber M, Weippl E. QR code security: A survey of attacks and challenges for usable security. In: Proceedings of the international conference on human aspects of information security, privacy, and trust. Springer; 2014. p. 79–90.
22. Latham DC. Department of Defense Trusted Computer System Evaluation Criteria. Department of Defense Standard. 1985.
23. Li S, Chen CH. The effects of visual feedback designs on long wait time of mobile application user interface. Interact Comput. 2019;31:1–12.
24. Mayrhofer R, Gellersen H. On the security of ultrasound as out-of-band channel. In: Proceedings of the parallel and distributed processing symposium (IPDPS). IEEE; 2007. p. 1–6.
25. Mudugodu Seetarama R. Secure device bootstrapping with the nimble out of band authentication protocol. Master's thesis, Aalto University. 2017. http://urn.fi/URN:NBN:fi:aalto-201706135412.
26. Naor M, Rotem L, Segev G. The security of lazy users in out-of-band authentication. In: Proceedings of the theory of cryptography conference. Springer; 2018. p. 575–599.
27. NXP Semiconductors. PN532 application note Rev. 01.00. NXP. 2006. https://www.nxp.com/docs/en/nxp/application-notes/AN133910.pdf.
28. Oracevic A, Dilek S, Ozdemir S. Security in internet of things: A survey. In: Proceedings of the international symposium on networks, computers and communications (ISNCC). IEEE; 2017. p. 1–6.
29. Pathak PH, Feng X, Hu P, Mohapatra P. Visible light communication, networking, and sensing: a survey, potential and challenges. IEEE Commun Surv Tutor. 2015;17(4):2047–77.
30. Peltonen A. Formal Modelling and Verification of the EAP-NOOB Protocol. Master's thesis, Aalto University. 2018. http://urn.fi/URN:NBN:fi:aalto-201809034896.
31. Reschke J. The 'Basic' HTTP Authentication Scheme. RFC 7617. 2015. https://doi.org/10.17487/RFC7617.
32. Reyes ARL, Festijo ED, Medina RP. Securing one time password (otp) for multi-factor out-of-band authentication through a 128-bit blowfish algorithm. Int J Commun Netw Inf Secur. 2018;10(1):242–7.
33. Sethi M, Antikainen M, Aura T. Commitment-based device pairing with synchronized drawing. In: Proceedings of the international conference on pervasive computing and communications (PerCom). IEEE; 2014. p. 181–189.
34. 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, Asia CCS '19. ACM, New York; 2019. p. 453–464. https://doi.org/10.1145/3321705.3329813.
35. Shin DH, Jung J, Chang BH. The psychology behind QR codes: user experience perspective. Comput Hum Behav. 2012;28(4):1417–26.
36. Siadati H, Nguyen T, Gupta P, Jakobsson M, Memon N. Mind your SMSes: mitigating social engineering in second factor authentication. Comput Secur. 2017;65:14–28.

37. Suomalainen J, Valkonen J, Asokan N. Security associations in personal networks: a comparative analysis. In: Proceedings of the European workshop on security in ad-hoc and sensor networks. Springer; 2007. p. 43–57.
38. Suomen Pankki: Payments statistics. 2018. https://www.suomenpankki.fi/en/Statistics/payments-statistics/. Accessed 9 May 2019.
39. Telegram: End-to-End Encryption, Secret Chats. 2019. https://core.telegram.org/api/end-to-end. Accessed 6 May 2019.
40. Telegram: Secret Chats. 2019. https://core.telegram.org/blackberry/secretchats. Accessed 6 May 2019.
41. Thagadur Prakash, S. Enhancements to Secure Bootstrapping of Smart Appliances. Master's thesis, Aalto University, 2017. http://urn.fi/URN:NBN:fi:aalto-201709046881.
42. Vollbrecht J, Carlson JD, Blunk L, Aboba B, Levkowetz H. Extensible Authentication Protocol (EAP). RFC 3748. 2004. https://doi.org/10.17487/RFC3748.
43. WhatsApp: WhatsApp Encryption Overview, Technical white paper. 2017. https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf.
44. Wu L, Du X, Wang W, Lin B. An out-of-band authentication scheme for internet of things using blockchain technology. In: Proceedings of the international conference on computing, networking and communications (ICNC). IEEE; 2018. p. 769–773.