applied sciences

MDPI

Article

Secure Audio-Visual Data Exchange for Android In-Vehicle Ecosystems

Alfred Anistoroaei, Adriana Berdich, Patricia Iosif and Bogdan Groza *

Faculty of Automatics and Computers, Politehnica University of Timisoara, 300223 Timisoara, Romania; alfred.anistoroaei@aut.upt.ro (A.A.); adriana.berdich@aut.upt.ro (A.B.); patriciaiosif@outlook.com (P.I.)
* Correspondence: bogdan.groza@aut.upt.ro

Abstract: Mobile device pairing inside vehicles is a ubiquitous task which requires easy to use and secure solutions. In this work we exploit the audio-video domain for pairing devices inside vehicles. In principle, we rely on the widely used elliptical curve version of the Diffie-Hellman key-exchange protocol and extract the session keys from the acoustic domain as well as from the visual domain by using the head unit display. The need for merging the audio-visual domains first stems from the fact that in-vehicle head units generally do not have a camera so they cannot use visual data from smartphones, however, they are equipped with microphones and can use them to collect audio data. Acoustic channels are less reliable as they are more prone to errors due to environmental noise. However, this noise can be also exploited in a positive way to extract secure seeds from the environment and audio channels are harder to intercept from the outside. On the other hand, visual channels are more reliable but can be more easily spotted by outsiders, so they are more vulnerable for security applications. Fortunately, mixing these two types of channels results in a solution that is both more reliable and secure for performing a key exchange.

Keywords: mobile device pairing; audio-visual domain; in-vehicle ecosystem

Citation: Anistoroaei, A.; Berdich, A.; Iosif, P.; Groza, B. Secure Audio-Visual Data Exchange for Android In-Vehicle Ecosystems. *Appl. Sci.* **2021**, *11*, 9276. https://doi.org/10.3390/app11199276

Academic Editors: Enrico Vezzetti, Subhas Mukhopadhyay and Yoshiyasu Takefuji

Received: 8 September 2021
Accepted: 3 October 2021
Published: 6 October 2021

1. Introduction and Related Work

Pairing smart mobile devices inside modern vehicles has become a common task. Due to the increased number of mobile devices that a user carries, e.g., smartphones, smartwatches, etc., finding a fast and efficient solution for secure associations is an immediate goal. In this work we explore the use of the audio-video domain in order to securely pair devices that are located in the same vehicle.

Using audio channels is particularly attractive since modern vehicles can be well isolated from the outside, assuming that doors and windows are closed, and thus devices that do not belong to the in-vehicle environment cannot eavesdrop on the communication channel. Also, when ultrasounds are used, i.e., frequencies above the 16–18 kHz range, they will not impede humans which have reduced hearing above this range. However, as also proven by our experiments, audio channels are less reliable due to existing environmental noise and microphone calibration issues. On the contrary, visual channels are far more reliable and allow devices to easily exchange data with modern technologies such as QR codes [1,2]. However, the visual channel may be easier to intercept from the outside and for this reason merging it with the audio channel will result in a more secure and reliable solution. This mixture of the audio and visual domains is the main idea behind our approach. Security wise, we use the elliptical curve version of the Diffie-Hellman [3] key-exchange protocol. The use of elliptical curves allows many more compact keys to be used, i.e., of 160 bits. The regular key shares of the Diffie-Hellman protocol are further exchanged partly by using the audio channel and partly by using the visual channel as we later discuss.

For a crisper image of the exact capabilities of the audio channels we now recall some technical data from an existing study. The work in [4] obtains a bit error rate (BER) for audio

channels that is in the range of 6.44% to 14.59%. We have observed a similar behavior in our experiments and obtained similar bit error rates as shown later. Naturally, the error rate increases with the distance, and it also depends on the frequency. Higher frequencies will tend to be more directional, and they do not have the same power as mid-range frequencies, i.e., smartphone speakers will not behave so well above the 15 kHz range. Since we are interested in exchanging keys that are at a minimum of 160 bits in length, i.e., this is the smallest size for an elliptical curve that still offers sufficient security, we choose to keep the interaction on the audio channel as low as possible. Thus, we add to the audio channel a visual channel by which the vehicle head unit can further expose its key share.

To obtain a secure secret key based on the elliptical curve version of the Diffie-Hellman [3] key-exchange protocol, we use the handshake that is graphically suggested in Figure 1. In this graphical depiction, by $P(\xi)$ we denote the elliptic curve point altered by the in-vehicle noise extracted as ξ in accordance with the SPEKE protocol described in [5]. A smartphone acoustically emits toward the head unit the value of aP. Further, the head unit shares the value of bP over the visual channel and the secret session key that results from the Diffie-Hellman key-exchange, i.e., $k_{ses} = abP$, is further used to encrypt a secret. In our case, we use the Wi-Fi network SSID and password as a secret which will allow the phone and head unit to connect to the same Wi-Fi network. We use the Wi-Fi connection key just as an example, clearly this can be replaced by any useful information. As an additional means to increase security, we use the in-vehicle noise to bootstrap security bits in the generation of the elliptical curve point P which is necessary for the guessing-resilient authenticated key-agreement protocol SPEKE described in [5]. This protocol was later analyzed in [6] and is known to be secure. From the application standpoint, we assume that the car owner is in charge of starting the application on the head unit and will proceed so whenever they want to grant access to any of the passengers to the car Wi-Fi environment. Nonetheless, while we specifically target in-vehicle environments where pairing is done with an in-vehicle head unit, there are no restrictions to use the application between two smartphones as the figure already suggests. We also try a second protocol version in which the visual channel is replaced by an audio channel, but due to the higher error rate this solution seems less preferable and more suitable when pairing two smartphones rather than an infotainment unit and a smartphone.

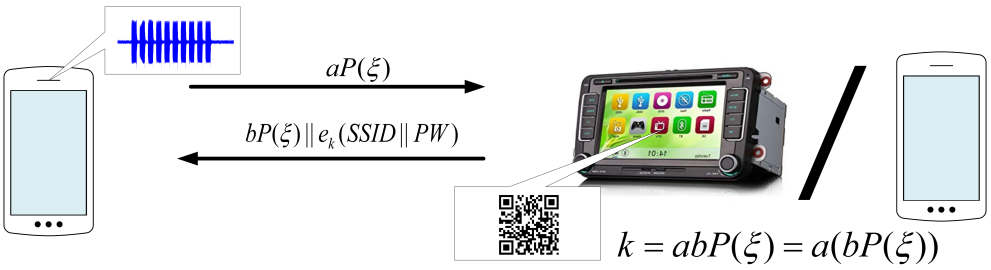


Figure 1. The addressed setting: key exchange between a phone and a head unit (or another phone).

The rest of our work is organized as follows. Next, we briefly outline some of the existing related works. Then, in Section 2 we give a full description of the devices from our setup. Section 3 presents the employed protocol and the experimental results. Finally, Section 4 holds the conclusion of our work.

Context and Related Work

First, in order to clarify the context, we provide a brief summary of commonly employed wireless communication channels in Table 1, i.e., Wi-Fi, Bluetooth and the audio channel which is within the scope of this work. The table outlines various characteristics related to bandwidth, range, operating frequency, security and the area of application. Notably, audio channels have been recently explored by various papers which address device

pairing as they have a security advantage since acoustic signals are not so easy to intercept by outsiders. Of course, in terms of bandwidth and range, sounds are much less efficient than Wi-Fi or Bluetooth. However, this can also be interpreted as a security advantage since sound has a much shorter range and thus it will be harder to eavesdrop. For the same security reason, channels with an even shorter range such as NFC are sometimes preferred. However, commercial in-vehicle head units do not always possess NFC capabilities and such a solution will have a reduced practical coverage for the in-vehicle ecosystem.

Table 1. Comparison with others wireless communication methods inside vehicular environments.

	Bandwidth	Range	Operating Frequency	Security	Area of Application
Wi-Fi	>500 Mbps (802.11 ac) 100 Mbps–1 Gbps	50 m indoors, 250 m outdoors (with stock equipment)	2.4 Ghz and 5 Ghz	WPA2 (implementation dependent attacks reported)	high-volume data transfer, e.g., notebooks, tablets, smartphones
Bluetooth	700 Kbs	<10 m	2.400–2.4835 GHz ISM band	based on upper layers	peripheral connectivity, e.g., ear-buds, sound-bars, smart weareables
Audio	4–20 characters/second	<2 m (with the devices/experiments in this work)	any frequencies between 20 Hz and 20 kHz	less leakage from closed/isolated perimeters	recently used as an intensive communication channel for: device pairing [7–9], Wi-Fi pairing [4]

The authors in [4] use audio signals to exchange a Wi-Fi network's SSID and password. Each character from the exchanged data is encoded by one frequency using the range between 2 kHz and 5 kHz. For the same purpose, in [10] the Wi-Fi password is shared between friends using a social network graph. Also, in [11] a server is used as a third-party instance to share the Wi-Fi password.

In [12] a device pairing system is proposed for smartwatches which uses audio signals. The authors use a short-time Fourier transform and the resulted signal energy to compute a matrix used for key generation. A fuzzy cryptographic scheme for communication between devices based on audio fingerprints extracted from the signal energy and discrete Fourier transform of the ambient audio signal is presented in [13]. In [14] the audio channel is used to exchange data between wireless devices. Also, in [15] isolated sounds and ultrasonic sounds are used as covert channels for mobile device communication. An authentication mechanism based on ambient sound is proposed in [16]. The one-third octave band filtering and the cross-correlation of the recorded sound are used to detect the sound similarity. An audio-based authentication system between a server and two devices is proposed in [17]. The audio signal contains symbols which correspond to a certain frequency between 16 kHz and 19 kHz. A system for device authentication based on the Fast Fourier Transform (FFT) of the audio signal is proposed in [18]. In [19], a key extraction mechanism based on inaudible sounds is discussed. A Wi-Fi connection using audio is depicted in [20]. Also, in [21,22], a smart device pairing system based on ambient noise is proposed.

Not distant from these, there are several lines of work that employed environmental data for secure device pairing. A procedure for device pairing based on ambient sound and on a generated sine wave is presented in [23]. An approach for device pairing based on environmental data extracted from the smartphone accelerometers in distinct transportation modes is proposed in [24]. Solutions for pairing devices based on accelerometer data are proposed in [25,26]. A system for wireless pairing using accelerometer data and heuristic trees is discussed in [27]. Shaking devices for pairing them is proposed in [28,29].

As a summary of findings from related papers, the use of audio channels for data exchange has increasingly gained attention in recent years. The inaudible range above 16 kHz is of specific interest since it does not impede human hearing and the short range of audio signals may offer some security advantages since such channels are harder to eavesdrop (like other short-range interfaces as NFC). Moreover, the use of environment noise, not only from the audio domain but also vibrations recorded by accelerometers, is also attractive as it facilitates the pairing of devices that share a common context.

2. Setup and Methodology

In this section we present the experimental devices from our setup and discuss the methodology. We also present the software implementation architecture.

2.1. Experimental Devices and Methodology for Audio Key Extraction

In the experiments we used several smartphones with different versions of Android, starting from older devices like the Nexus 7 to newer mid-range devices such as the Motorola E6 Plus. This allowed us to determine whether differences between Android APIs and device characteristics can create problems during the acoustic domain pairing. Also, as representative for our main use case, i.e., the in-vehicle scenario, we used an off-the-shelf PX5 Android infotainment unit. A summary of the used devices and some of their specifications are available in Table 2. On the left side of Figure 2 we depict the smartphones used in our experiments and in the right side of the figure we depict the devices placed on the vehicle dashboard.

Table 2. Devices from our experiments.

No.	Device Name	Release Year	Android Version	No. of Microphones
1.	Google Nexus 7	2012	Android 5	2
2.	Samsung Galaxy A3	2014	Android 5	2
3.	Samsung Galaxy S7 Edge	2016	Android 6	1
4.	Motorola E6 plus	2019	Android 9	1
5.	PX5 head unit	2017	Android 9	1

Figure 2. The four phones used in our experiments in-doors (**left**) and on the vehicle dashboard with the head unit (**right**).

To depict the audio acquisition capabilities of the devices, on the left side of Figure 3 we show the power spectrum from the recordings by each device during a linear sweep signal between 20 Hz and 22 kHz that is played by a high-end HiFi system. In this figure, it can be seen that the frequency response of the microphone of the Nexus smartphone is more distinct than for the rest of the devices. On the right side of Figure 3, we depict the power spectrum of a linear sweep signal between 20 Hz and 22 kHz when played by the devices from our experiments and recorded by a calibrated UMIK-1 microphone. In this figure the frequency response of the speaker of the Nexus and Samsung A3 smartphone is

more distinct than the Samsung S7 and Motorola E6 speaker frequency response. For these measurements the distance between the speaker and the microphone was 100 cm.

The distinct frequency responses from the speakers and microphones of our devices explain some of the disparities from the experiments that we present later. For example, for the experiments performed indoors, at 50 cm between devices, when the Nexus was used as emitter and the Motorola E6 as receiver, 100% of the keys were correctly received. However, when the emitter and receiver roles are shifted, i.e., Motorola E6 is the emitter and Nexus the receiver, only 70% of the keys are correct.

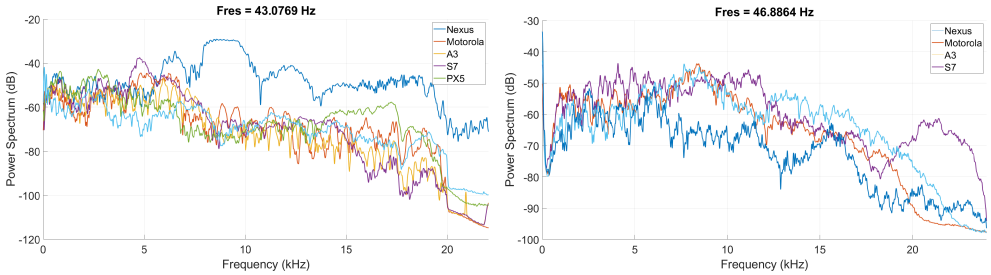


Figure 3. The power spectrum of audio signal recorded (**left**) and played (**right**) by the devices used in our experiments.

2.2. Software Implementation

Since our experiments are exclusively concerned with Android devices, the programming language used for the development was Java and for the cryptographic functions we relied on the SpongyCastle library (https://github.com/rtyley/spongycastle, accessed on 10 August 2021). This is basically the BouncyCastle library (https://www.bouncycastle.org, accessed on 10 August 2021) with specific changes required to work on Android. For processing the audio signal in the frequency domain, we used the fast Fourier transform implemented using radix 2 Cooley-Tukey [30]. For the QR code generation and scanning operation we used a free library, the ZXing Android Embedded library (https://github.com/journeyapps/zxing-android-embedded, accessed on 10 August 2021).

As expected, the development of the APK was done using Android Studio and the result consists of two applications, an emitter and a receiver application, developed using a modular architecture. A detailed view of the modular architecture is presented in Figure 4 which shows each major layer that handles the information. Each class has its own responsibilities, e.g., recording, emitting, symbol conversion, elliptic curve computations, etc., and custom parameters, e.g., duration of a single tone, expected number of symbols, frequency range, etc. At the base of the information flow we place the Android classes that handle the recording and emitting of information, i.e., AudioTrack and AudioRecord. Over this layer we add the FFT class and then the SpongyCastle handler which routes the information to the MainActivity class which is basically the user interface.

The communication between classes is depicted in the class diagram from Figure 5. The MainActivity class schedules all the work based on the user input. It also constructs all the needed objects (the elliptic curve handler class, the emitter, recorder and the task for extracting noise) for the application startup, passing them the basic information, e.g., sound duration, volume, number of bits in the transmitted symbols, etc. Once the object initialization is over, user input is expected for selecting the curve type and whether the seed from the environment is to be used or not. Also, debugging information can be logged in external files. Once these options are selected, they will be passed to the responsible objects. The ECHandler object will need to obtain only the curve type, while the Emitter and Recorder will need to be informed if the seed is predefined or extracted. The MainActivity class will coordinate the transmission of sounds, using the Emitter class, and the reception, using the Recorder class, then extracts the information, using the ECHandler class.

Application	
MainWindow Emitter	MainWindow Recorder
+startEmit() +startRecord() +sendDataFinished() +workWithRecordedData(String) *+OnEnvironmentExtractionRecordingHasFinished (ArrayList<Integer>)* *+SynchronizationTaskStartEnvironmentExtraction (Emitter, Recorder)*	
SpongyCastle library	
+getParameterSpec(String) *+getEncoded(bool)* *+decodePoint(byte[])* *+multiply()* *+normalize()*	
FFT algorithm class	
+calculateFFT(byte[])	
Android AudioTrack	Android AudioRecord
+play() +write(byte[], int, int)	+createAudioRecord() +startRecording()

Figure 4. Implementation stack.

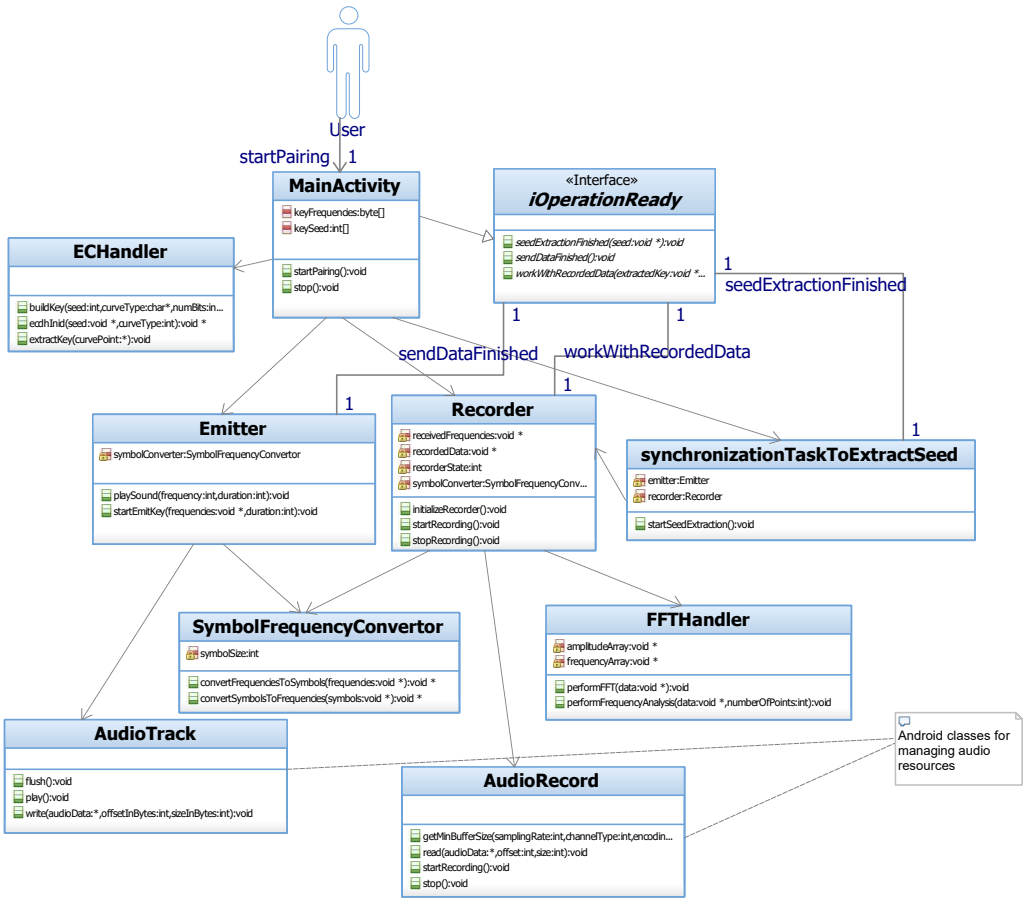
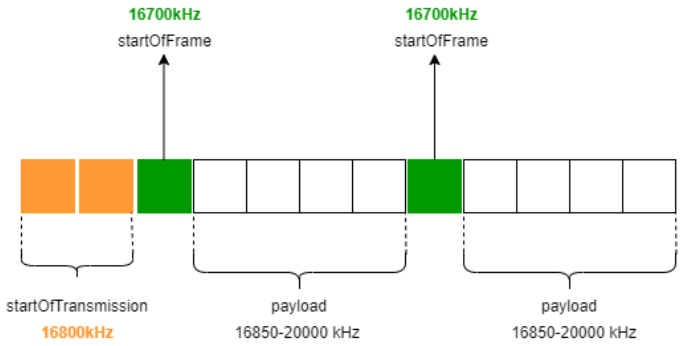


Figure 5. Class diagram.

Regarding the use-flow, firstly we initiate the seed generation (this is used inside the car for extracting the seed based on the environment noise; otherwise the seed can have a predefined value). After the seed is generated, the key share on the elliptic curve is computed and the operation of emitting, or recording, starts. The seed generation (which is handled by the synchronizationTaskToExtractSeed class), emit and record actions are asynchronous operations, done on different threads. Thus, the main class required an interface through which it could be signaled when the action is over. This is the role of the iOperationReady interface. Before the key is emitted or after the key is received, both the Emitter and Recorder classes use the SymbolFrequencyConvertor class to translate the values to keys and vice versa. Moreover, the Recorder class uses the FFTHandler to compute the recorded sound (even if it is during the seed or the key extraction, the operation is the same). For brevity, some class functionalities, methods or members have been left apart from the figure, e.g., file handling, conversion from 8 to 6 bits and vice versa or the mapping between frequencies and values, but some of these are nonetheless presented in what follows.

The left side of Figure 6 shows how the signals are encoded in each frame. There are 4 frequencies, and for synchronization purposes, the fifth is a predefined StartOfFrame frequency. Also, to mark the start of transmission, we will use a special frequency. The shared key is reconstructed from 64 symbols. Each symbol is carried by a tone with a frequency in the 16.85–20 kHz range at 50 Hz resolution.

One of our primary objectives was to use the highest frequency that we could so that the discomfort for the users would be minimal. For this we mapped one frequency to a 6 bit value and we set the distance between frequencies at 50 Hz. Consequently, in order to be able to send all the data, we needed a total bandwidth of 3200 Hz as it can be seen on the right of Figure 6. This means that we could emit a message in a range starting from 16.7 kHz (inaudible for most people) up to 20 kHz. This upper frequency of 20 kHz was used because for sampling audio signals Android recommends a sampling of 44.1 kHz or 48 kHz (https://developer.android.com/ndk/guides/audio/sampling-audio, accessed on 10 August 2021) and according to the Nyquist-Shannon sampling theorem we could not use frequencies higher than 22.05 kHz to avoid sampling issues. Also, during the experiments, the results for frequencies above 20 kHz were not very good due to the limited power of the signal, so we chose 20 kHz as the maximum frequency in our application.



Frequency	Symbol
16,700	Start of frame
16,800	Start of transmission
16,850	0
16,900	1
16,950	2
17,000	3
...	...
19,950	62
20,000	63

Figure 6. Frame structure (**left**) and symbol encoding as a tone (**right**).

In Figure 7 we show the 160 bit key when emitted by a device with carrier frequencies between 16.7 kHz and 20 kHz, the representation is in the time domain (left) and in the frequency domain with use of the Fast Fourier Transform (FFT) on the audio signal (right).

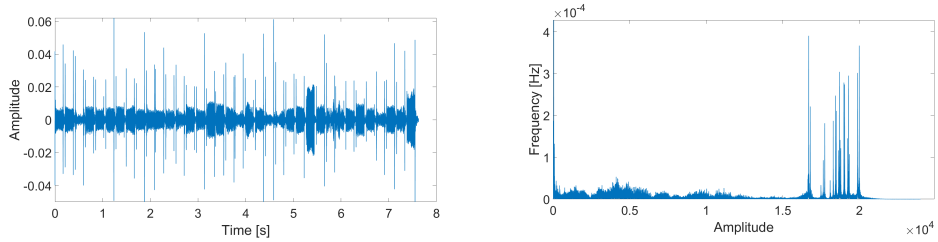


Figure 7. The audio signal in the time domain (**left**) and frequency domain (**right**) for the recorded key.

To extract the correct frequencies in real-time by the recording device and to avoid device synchronization problems we split each window corresponding to a tone in another 5 windows. The same procedure is useful in removing the influences of the silence periods appearing at the transitions between tones. We compute the Fast Fourier Transform for each new window and then we apply a majority voting to extract the correct frequency.

3. Protocol and Results

In this section we discuss details regarding the employed security protocol and then provide the results from our experiments.

3.1. Protocol

In Figure 8 we present the flowchart of the proposed key exchange protocol. The protocol includes two applications that are running in parallel on the two devices involved in the key exchange, i.e., the emitter and the recorder application. Both applications are waiting for the user interaction and then they compute the share to emit between each other, i.e., aP in the case of the emitter and bP the in case of the recorder application. The emitter starts to send the share, aP, on the audio channel. The receiver is recording and when the predefined start frequency is detected, the collected frequencies are extracted from the frame (for frame structure see Figure 6) and are stored into an array. The recording is stopped when the expected number of frequencies are received and the values of the acquired frequencies are out of range. Next, the recorder computes the session key based on the received aP and the calculated bP. The Wi-Fi SSID and password are encrypted using the computed session key. The encrypted information and the value of bP are embedded in a QR code which is displayed on the screen. The emitter scans the QR code and extracts the share bP and the encrypted data. Based on the value of aP and the received value of bP, the session key abP is computed (this session key will be used to decrypt the Wi-Fi SSID and password which are used as an example in our application).

As stated before, in the experiments involving the head unit we also used the environment noise to obtain a seed for key generation as can be seen in the flowchart in Figure 8 marked with a dotted line to suggest this as an optional feature. To synchronize the two devices to start recording the noise, the emitter sends a predefined audio signal and both devices start recording. After finishing the recording and performing the FFT, an algorithm to extract the seed is applied. The algorithm consists in selecting only the desired number of frequencies (in our case 4) with the highest amplitudes and these frequencies will form the seed. After this, the flow continues as explained above.

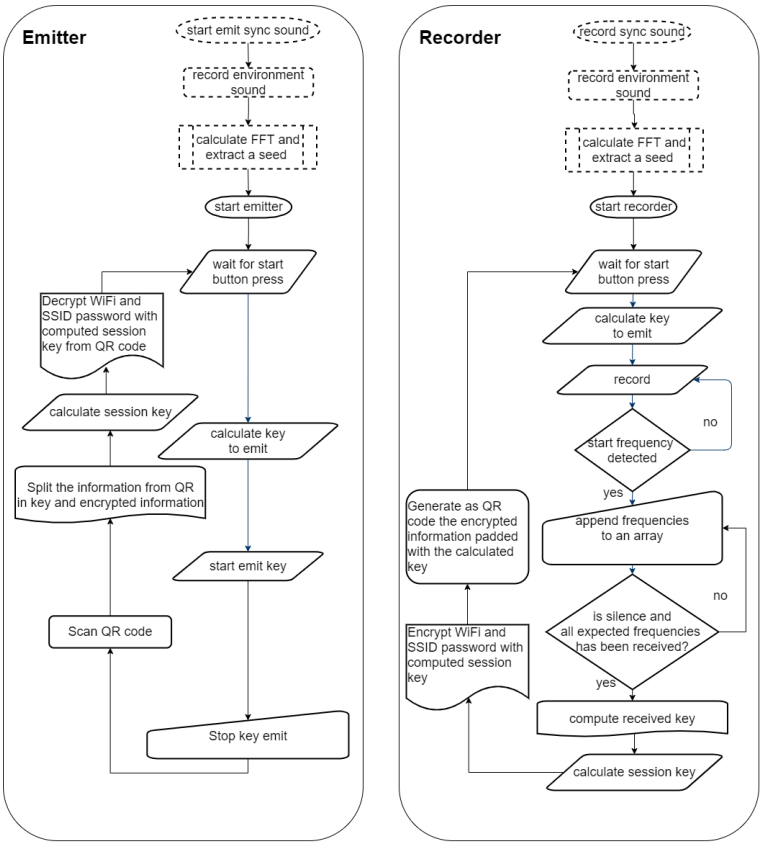


Figure 8. The flowchart of the proposed protocol.

3.2. Computational Requirements and Implementation of SPEKE

The SPEKE protocol [5] is not part of the regular Spongy Castle cryptography toolset. For this reason we had to implement it ourselves with the help of the elliptic curve support from Spongy Castle. The SPEKE protocol is a modified version of the Diffie-Hellman protocol which rather than using a generator P in the key exchange uses a generator that is extracted based on a common secret $P = h(S)$, where S represents the secret (for example a password) and h is a hash function. The steps of the key exchange protocol are the usual Diffie-Hellman protocol steps: $A \rightarrow B : aP$, $B \rightarrow A : bP$, with the common extracted key abP, the only difference being in the generation of P from the secret S. For practical implementations, there is no hash function to directly map a secret S to a group generator, but such a group generator can be found by successive attempts. When the resulting P is not a group generator, the security level of the key exchange will be lower, but a key can still be correctly extracted. For our scenario, the security is based mostly on the inability of the outsiders to hear the noise inside the car while any device inside the car can extract the same noise, a reason for which extracting a regular point on the curve, and not necessarily a generator is sufficient.

The code is presented in Figure 9 and explained next. By using the `seed` resulted from the environment noise, we initialize a pseudo-random number generator `prngA` that will be used on A's side (where A is one of the participants in the key exchange). Then, `prngA` is used to generate the masking bits, i.e., `maskA`, which are going to alter the default point (which is set to `xA` in the implementation) by XOR-ing it with the mask, i.e., `ekeshA[i] = (byte) (maskA[i] ^ xA[i])`. Note that on the side of the other participants, since the environment noise is identical and an identical seed is thus received, the same mask and the same random point on the curve will be obtained. This will ensure that both protocol participants will generate the same random point on the curve. However, the newly generated X coordinate by XOR-ing may not be part of the elliptic curve group and thus the attempt to decode the point with `ecpointA = specA.getCurve().decodePoint(ekeshA)`

may return an exception. To circumvent this, we add a try/catch block and regenerate until a point on the curve is reached (on average this required two attempts).

```
SecureRandom prngA = new SecureRandom(seed);
byte[] maskA = new byte[xA.length];
byte[] ekeshA = new byte[xA.length];
ECPoint ecpointA = null;
boolean asucc = false;
    while (asucc == false) {
        try {
            prngA.nextBytes(maskA);
            ekeshA[0] = (byte) xA[0];
            for (int i = 1; i < xA.length; i++) {
                ekeshA[i] = (byte) (maskA[i] ^ xA[i]);
            }
            ecpointA = specA.getCurve().decodePoint(ekeshA);
            asucc = true;
            Log.i(TAG, "ekeshA: " + Arrays.toString(ekeshA));
            } catch (Exception e) {
                asucc = false;
            }
        }
ECPoint aP = ecpointA.multiply(aa);
aP = aP.normalize();
byte[] aPtoB = aP.getEncoded(true);
return aPtoB;
```

Figure 9. Code snippet for the SPEKE protocol implementation in Java.

In Table 3 we show the computational time for the pairing operation on our smartphones using SPEKE. All smartphones from our experiments can handle the SPEKE protocol in no time, i.e., a dozen milliseconds or so. This proves that handling the cryptography part of the protocol is not an issue for modern smartphones.

Table 3. Computational time of SPEKE in the pairing operation.

	SPEKE-160 bit (secp160r1)	
Phone	**Share**	**Recover**
Motorola E6 Plus	17 ms	52 ms
Samsung Galaxy S7 Edge	37 ms	8 ms
Samsung Galaxy A3	18 ms	13 ms
Google Nexus 7	17 ms	7 ms
PX5 head unit	27 ms	5 ms

3.3. Test Scenarios and Results

We center our experiments around two environments: indoors and inside a vehicle. For each scenario, we exchange several pairs of keys between each pair of smartphones (or with an Android head unit in case of the in-vehicle experiments). The results of these experiments are detailed next.

3.3.1. In-Door Experiments with Smartphones

It was natural for us to expect that the success of the key exchange will decrease and the BER will increase with the distance. To get a more accurate image on the exact rate at which these values decrease and increase, we perform 30 indoors experiments with the Motorola E6 and Samsung S7 smartphones at different distances, i.e., 20 cm, 50 cm, 100 cm and 200 cm and depicted the results in Figure 10. The results confirm our expectations, the BER increases with the distance and the percentage of correct keys decreases. These changes are not linear and not identical between the devices due to the different hardware

with which they are equipped. For example, the increase and decrease are much more pronounced in the Motorola E6 when compared to the Samsung S7. We marked with green the results of the receiver app running on Motorola E6 and with black the results of the receiver app running on Samsung S7.

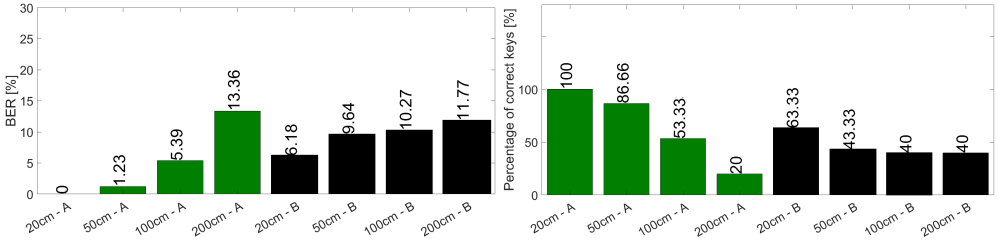


Figure 10. The BER (**left**) and the percentage of correct keys (**right**) for the 30 measurements with Motorola E6 (marked with green when receiver app running on Motorola E6) and Samsung S7 (marked with black when receiver app running on Samsung S7) at 20 cm, 50 cm, 100 cm and 200 cm.

Afterwards, with all the other smartphones we established three scenarios, with devices placed at different distances, i.e., 20, 50 and 100 cm (as shown on the left side of Figure 11) and frequency range between 16.7 kHz and 20 kHz. We have calculated the bit error rate (BER) for each pair of smartphones as follows: $BER = \frac{no_{err}}{no_{all}}$, where no_{err} is the number of wrongly received bits and no_{all} is the total number of received bits. In Table 4 we show the percentage of the correct key received and in Table 5 we show the BER for each distance. In some cases the BER is influenced by which smartphone is the sender and which is the receiver, e.g., in case of experiments at 20 cm between Samsung S7 and Motorola E6, when Samsung S7 is the sender and Motorola E6 is the receiver, the BER is 5.92% and when Motorola E6 is the sender and S7 is the receiver the BER is 0%. This behavior is also observed between Samsung A3 and Samsung S7 at 20 cm. For all in-door experiments scenarios the range of the BER is between 0% and 9.66%. In case of 50 cm and 100 cm, in our table we marked with *NW* the pair of devices, which are not able to receive all frequencies. In Figure 12 we depict a graphical representation of the results for our indoors experiments for each smartphone pair, except the Samsung S7–Samsung A3 pair where in some cases the exchange did not work. For each pair of smartphones, we have three bars marked with blue when the distance between devices was 20 cm, red for 50 cm and yellow for 100 cm. We depict on the abscissa the pair of smartphones using the following acronyms: N (Nexus), M (Motorola E6), S7 (Samsung S7) and S3 (Samsung A3).

Table 4. Pairing success rate for indoors experiments (16.7–20 KHz range).

	20 cm				**50 cm**				**100 cm**			
Snd. \ **Rcv.**	**E6**	**S7**	**A3**	**Nexus**	**E6**	**S7**	**A3**	**Nexus**	**E6**	**S7**	**A3**	**Nexus**
E6	X	100%	80%	100%	X	70%	60%	70%	X	50%	60%	80%
S7	70%	X	70%	100%	40%	X	NW	90%	30%	X	NW	70%
A3	90%	100%	X	50%	60%	NW	X	40%	60%	NW	X	30%
Nexus	80%	100%	90%	X	100%	90%	100%	X	60%	60%	100%	X

Table 5. BER for indoors experiments (16.7–20 KHz range).

	20 cm				50 cm				100 cm			
Snd. \ **Rcv.**	**E6**	**S7**	**A3**	**Nexus**	**E6**	**S7**	**A3**	**Nexus**	**E6**	**S7**	**A3**	**Nexus**
E6	X	0%	2.59%	0%	X	3.33%	8.14%	1.11%	X	4.85%	9.36%	1.22%
S7	5.92%	X	4.81%	0%	5.57%	X	NW	1.11%	9.36%	X	NW	1.85%
A3	0.37%	0%	X	9.36%	3.38%	NW	X	9.66%	6.29%	NW	X	9.36%
Nexus	5.16%	0%	0.37%	X	0%	0.37%	0%	X	3.71%	7.43%	0%	X

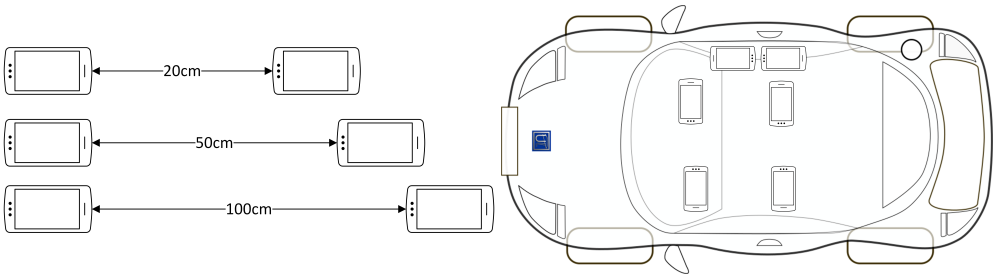


Figure 11. Distance between phones at indoor measurements (**left**) and in-vehicle measurements (**right**).

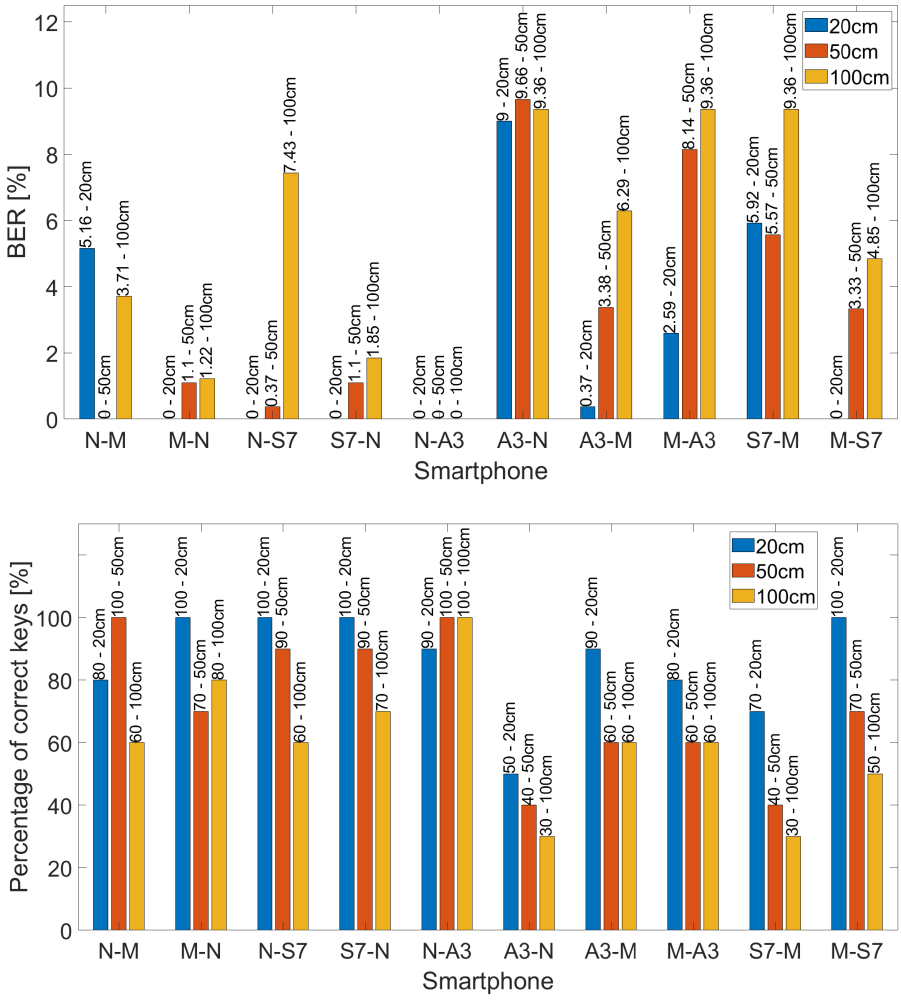


Figure 12. The BER (**up**) and the percentage of correct keys (**down**) for each smartphone pair.

3.3.2. In-Vehicle Experiments with Smartphones

We have tested several scenarios inside the vehicle, all using the frequency range between 16.7 kHz and 20 kHz but requiring different placements of the smartphones: (i) both smartphones placed on the dashboard, (ii) one smartphone placed on the driver seat and the other one on the passenger seat and (iii) both smartphones inside the door pocket as is shown in the right side of Figure 11. In Table 6 we show the percentage of the correctly received keys and in Table 7 we show the BER for each scenario. When the smartphones were placed in the door pocket we obtained a lower BER, between 0% and 4.83%. We consider that this is due to the good acoustics and short distance between the smartphones, approximately 5 cm. On dashboard the BER is between 0% and 5.92% and on the front seats the BER is between 1.1% and 6.96%.

In Figure 13 we depict the BER (up) and the percentage of correct keys (down) for in-vehicle experiments with smartphones for each smartphone pair except the Samsung A3–Samsung S7 pair (for which in some cases the exchange does not work). For each pair of smartphones, we have three bars marked with blue when the devices were placed in the door pockets, red for dashboard and yellow when the devices were placed on the front seats. We depict on the abscissa the pair of smartphones as we already specified above.

Table 6. Pairing success rate for in-vehicle experiments (8.7–12 KHz range).

	Door Pockets				**Dashboard**				**Front Left-Right**			
Snd. \ **Rcv.**	**E6**	**S7**	**A3**	**Nexus**	**E6**	**S7**	**A3**	**Nexus**	**E6**	**S7**	**A3**	**Nexus**
E6	X	100%	90%	100%	X	80%	70%	100%	X	60%	60%	60%
S7	80%	X	70%	100%	60%	X	60%	90%	50%	X	NW	70%
A3	90%	100%	X	60%	70%	80%	X	60%	50%	NW	X	50%
Nexus	100%	100%	100%	X	70%	100%	80%	X	90%	80%	80%	X

Table 7. BER for in-vehicle experiments (16.7–20 KHz range).

	Door Pockets				**Dashboard**				**Front Left-Right**			
Snd. \ **Rcv.**	**E6**	**S7**	**A3**	**Nexus**	**E6**	**S7**	**A3**	**Nexus**	**E6**	**S7**	**A3**	**Nexus**
E6	X	0%	0.93%	0%	X	4.29%	5.57%	0%	X	6.47%	5.57%	5.12%
S7	1.85%	X	4.83%	0%	5.84%	X	5.23%	0.53%	6.96%	X	NW	3.85%
A3	0.37%	0%	X	3.83%	5.92%	1.87%	X	4.46%	7.43%	NW	X	6.72%
Nexus	0%	0%	0%	X	3.33%	0%	2.59%	X	1.1%	1.74%	2.31%	X

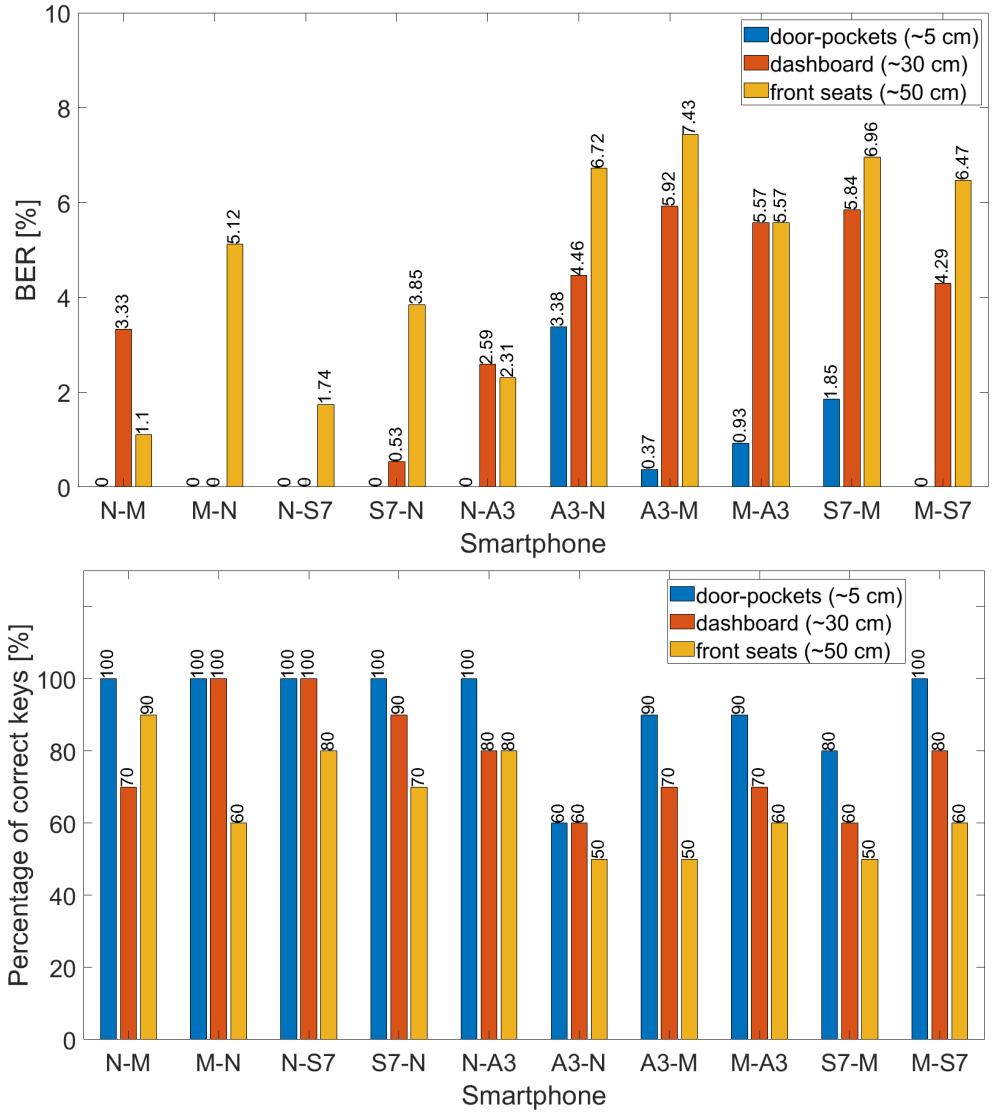


Figure 13. The BER (**up**) and the percentage of correct keys (**down**) for each smartphone pair for in-vehicle experiments with smartphones.

3.3.3. In-Vehicle Experiments with Smartphones and Head Unit

In our experiments we relied on an Android PX5 head unit which was connected to the car radio using an AUX cable. Thus we were able to use the car speakers and the microphone from the head unit. We carried out the experiments in two hatchback cars, a BMW 120d and an older Volkswagen Golf 3, which do not benefit from exquisite isolation from the outside noise. This type of cars should cover a sufficiently large market segment, while on more expensive cars equipped with superior isolation from the outside noise, the results should be even better. As stated in the introduction, the short range of audio signals is beneficial from a security point of view since such channels are harder to eavesdrop. While indeed the driver and the front passenger are favored by being closer to the head unit, we expect that getting a phone closer to the infotainment unit is practical in most scenarios.

During the experiments the engine was running and thus the engine noise also influenced our experiments. This kind of typical in-vehicle noise is useful to generate a random seed (the same on both devices) for the key exchange protocol. Due to the poor quality of the microphone from the head unit, we were not able to use high frequencies and we had to lower these to the range of 8.7 kHz to 12 kHz. Also, we reduce the distance between smartphone and head unit to 5 cm during the audio transmission for the head

unit to be able to record the frequencies. In Table 8 we present the BER and the key receive rate for each smartphone paired with the head unit. The BER values are between 0% and 9.25%. Also in Table 9 we present the same values but we used the environment noise to generate the seed for the keys. Also in Figures 14 and 15 we created bar charts using the results from the tables for better visualization.

The experiments performed on this set of distinct devices led us to several observations. Smartphones with distinct hardware characteristics might not work identically when running the same application. We noticed some synchronization problems when playing the tones, e.g., the delay between the frequencies was not the same on the Samsung A3 as on the other three devices. This is likely because of clock issues, so we had to increase the duration of the tones only on the Samsung A3 so that the pairing could be made. Other factors that we expected to impact our experiments were the quality of the speakers, microphones and digital-to-analog converters which are all variable on all distinct smartphone models.

Table 8. Pairing success rate and BER for infotainment experiments without environment noise.

	Key rcv. Rate A	Key rcv. Rate B	BER A	BER B
PX5-Motorola E6 Plus	70%	100%	2.96%	0%
PX5-Samsung Galaxy S7 Edge	80%	90%	1.48%	0.74%
PX5-Samsung Galaxy A3	70%	60%	8.51%	9.25%
PX5-Google Nexus 7	60%	60%	6.67%	4.46%

Table 9. Pairing success rate and BER for infotainment experiments with environment noise.

	Key rcv. Rate A	Key rcv. Rate B	BER A	BER B
PX5-Motorola E6 Plus	40%	50%	20.09%	16.4%
PX5-Samsung Galaxy S7 Edge	30%	50%	32.89%	19.81%
PX5-Samsung Galaxy A3	30%	40%	40.2%	22.96%
PX5-Google Nexus 7	30%	40%	29.62%	25.09%

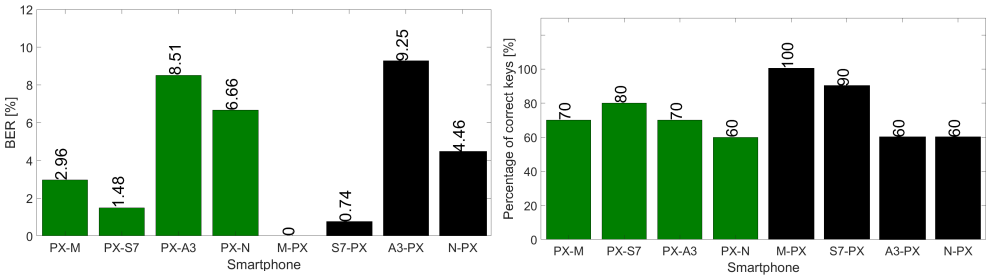


Figure 14. The BER (**left**) and the percentage of correct keys (**right**) for the measurements with the infotainment unit without environmental noise; marked with green when the receiver app is running on PX5 infotainment unit and with black when the receiver app is running on the smartphones.

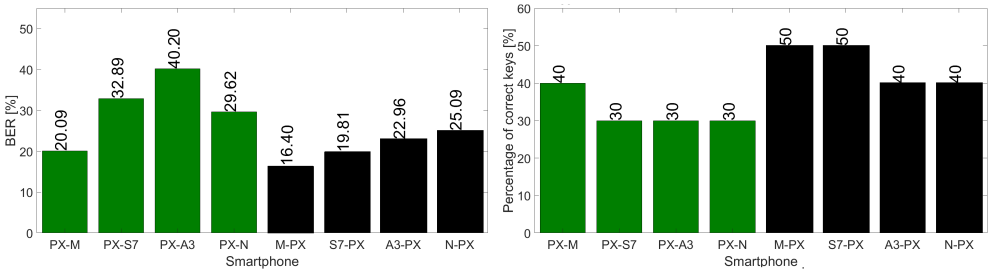


Figure 15. The BER (**left**) and the percentage of correct keys (**right**) for the measurements with the infotainment unit with environmental noise; marked with green when receiver app running on PX5 infotainment unit and with black when the receiver app in running on the smartphones.

4. Conclusions

The designed solution shows that a secure Diffie-Hellman key-exchange can be efficiently performed between smartphones and in-vehicle head units in the audio domain. This is further complemented by a visual channel which uses a QR code. The bit error rate in the audio transmission is generally below 10% and the key recovery rate greater than 50%. As a more general conclusion that can be drawn from the experiments over all devices, the decay in the key recovery rate and the increase in the BER are highly dependent on the device characteristics. That is, the quality of the speakers and microphones proved to be very non-uniform over the devices which we tested. If environment noise is further extracted for a more resilient pairing, the error rate of the channel doubles. In this respect, leaving the audio channel one-sided, i.e., only from the smartphone to the head unit, and using a visual channel from the head unit to the phone will increase the pairing success rate while maintaining the security offered by the extracted in-vehicle noise. While the bandwidth of audio-visual channels is clearly lower, they are still effective for exchanging small elements such as cryptographic keys that can be later used to enforce the security of traditional Wi-Fi channels. This hybrid approach should be beneficial both from a security and performance perspective. So far, our experiments did not address the pairing of multiple devices, but we leave this as potential future work. The cryptographic operations required by the elliptic curve version of the Diffie-Hellman protocol are very easy to handle by modern smartphones as proved by our experimental results. Since the SPEKE protocol is not supported by default in the Android cryptography toolset, we had to implement this on our own, but this required only a moderate effort and good understanding of the elliptic curve library.

Author Contributions: Conceptualization, B.G.; methodology, B.G.; software, A.A. and P.I.; validation, A.A. and P.I.; investigation, A.A.; resources, A.A. and B.G.; writing—original draft preparation, A.A., A.B., P.I. and B.G.; writing—review and editing, A.A., A.B., P.I. and B.G.; supervision, B.G.; project administration, B.G. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Scheuermann, C.; Werner, M.; Kessel, M.; Linnhoff-Popien, C.; Verclas, S.A. Evaluation of barcode decoding performance using zxing library. In Proceedings of the Second Workshop on Smart Mobile Applications, SmartApps, Newcastle, UK, 18–22 June 2012.
2. Liu, Y.; Yang, J.; Liu, M. Recognition of QR Code with mobile phones. In Proceedings of the 2008 Chinese Control and Decision Conference, Yantai, China, 2–4 July 2008; pp. 203–206.
3. Diffie, W.; Hellman, M. New directions in cryptography. *IEEE Trans. Inf. Theory* **1976**, *22*, 644–654. [CrossRef]
4. Zhang, L.; Zhu, X.; Wu, X. No More Free Riders: Sharing WiFi Secrets with Acoustic Signals. In Proceedings of the 2019 28th International Conference on Computer Communication and Networks (ICCCN), Valencia, Spain, 29 July–1 August 2019; pp. 1–8.
5. Jablon, D.P. Extended password key exchange protocols immune to dictionary attack. In Proceedings of the IEEE 6th Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, Cambridge, MA, USA, 18–20 June 1997; pp. 248–255.
6. Hao, F.; Shahandashti, S.F. The SPEKE protocol revisited. In Proceedings of the International Conference on Research in Security Standardisation, London, UK, 16–17 December 2014; Springer: Berlin/Heidelberg, Germany, 2014; pp. 26–38.
7. Goodrich, M.T.; Sirivianos, M.; Solis, J.; Soriente, C.; Tsudik, G.; Uzun, E. Using audio in secure device pairing. *Int. J. Secur. Netw.* **2009**, *4*, 57–68. [CrossRef]
8. Mathur, S.; Miller, R.; Varshavsky, A.; Trappe, W.; Mandayam, N. Proximate: Proximity-based secure pairing using ambient wireless signals. In Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services, Bethesda, MD, USA, 28 June–1 July 2011; pp. 211–224.

9. Sigg, S.; Ji, Y.; Nguyen, N.; Huynh, A. AdhocPairing: Spontaneous audio based secure device pairing for Android mobile devices. In Proceedings of the 4th International Workshop on Security and Privacy in Spontaneous Interaction and Mobile Phone Use, IWSSI/SPMU, Newcastle, UK, 18–22 June 2012; Volume 12.
10. Durmus, Y.; Langendoen, K. Wifi authentication through social networks—A decentralized and context-aware approach. In Proceedings of the 2014 IEEE International Conference on Pervasive Computing and Communication Workshops (PERCOM WORKSHOPS), Budapest, Hungary, 24–28 March 2014; pp. 532–538.
11. Lafuente, C.B.; Titi, X.; Seigneur, J.M. Flexible communication: A secure and trust-based free wi-fi password sharing service. In Proceedings of the 2011 IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications, Changsha, China, 16–18 November 2011; pp. 706–713.
12. Shang, J.; Wu, J. AudioKey: A usable device pairing system using audio signals on smartwatches. *Int. J. Secur. Netw.* **2020**, *15*, 46–58. [CrossRef]
13. Schürmann, D.; Sigg, S. Secure communication based on ambient audio. *IEEE Trans. Mob. Comput.* **2011**, *12*, 358–370. [CrossRef]
14. Soderi, S. Acoustic-Based Security: A Key Enabling Technology for Wireless Sensor Networks. *Int. J. Wirel. Inf. Netw.* **2020**, *27*, 45–59. [CrossRef]
15. Deshotels, L. Inaudible sound as a covert channel in mobile devices. In Proceedings of the 8th USENIX Workshop on Offensive Technologies (WOOT 14), San Diego, CA, USA, 19 August 2014.
16. Karapanos, N.; Marforio, C.; Soriente, C.; Capkun, S. Sound-proof: Usable two-factor authentication based on ambient sound. In Proceedings of the 24th USENIX Security Symposium (USENIX Security 15), Washington, DC, USA, 12–14 August 2015; pp. 483–498.
17. Luo, J.N.; Tsai, M.H.; Lo, N.W.; Kao, C.Y.; Yang, M.H. Ambient audio authentication. *Math. Biosci. Eng. MBE* **2019**, *16*, 6562–6586. [CrossRef] [PubMed]
18. Chen, D.; Mao, X.; Qin, Z.; Wang, W.; Li, X.Y.; Qin, Z. Wireless device authentication using acoustic hardware fingerprints. In Proceedings of the International Conference on Big Data Computing and Communications, Taiyuan, China, 1–3 August 2015; Springer: Berlin/Heidelberg, Germany, 2015; pp. 193–204.
19. Lu, Y.; Wu, F.; Tang, S.; Kong, L.; Chen, G. FREE: A Fast and Robust Key Extraction Mechanism via Inaudible Acoustic Signal. In Proceedings of the Twentieth ACM International Symposium on Mobile Ad Hoc Networking and Computing, Catania, Italy, 2–5 July 2019; pp. 311–320.
20. Liu, L.; Han, Z.; Fang, L.; Ma, Z. Tell the Device Password: Smart Device Wi-Fi Connection Based on Audio Waves. *Sensors* **2019**, *19*, 618. [CrossRef] [PubMed]
21. Nguyen, N.; Sigg, S.; Huynh, A.; Ji, Y. Using ambient audio in secure mobile phone communication. In Proceedings of the 2012 IEEE International Conference on Pervasive Computing and Communications Workshops, Lugano, Switzerland, 19–23 March 2012; pp. 431–434.
22. Miettinen, M.; Asokan, N.; Nguyen, T.D.; Sadeghi, A.R.; Sobhani, M. Context-based zero-interaction pairing and key evolution for advanced personal devices. In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, 3–7 November 2014; pp. 880–891.
23. Mei, S.; Liu, Z.; Zeng, Y.; Yang, L.; Ma, J.F. Listen!: Audio-based Smart IoT Device Pairing Protocol. In Proceedings of the 2019 IEEE 19th International Conference on Communication Technology (ICCT), Xi'an, China, 16–19 October 2019; pp. 391–397.
24. Groza, B.; Berdich, A.; Jichici, C.; Mayrhofer, R. Secure Accelerometer-Based Pairing of Mobile Devices in Multi-Modal Transport. *IEEE Access* **2020**, *8*, 9246–9259. [CrossRef]
25. Kirovski, D.; Sinclair, M.; Wilson, D. *The Martini Synch: Using Accelerometers for Device Pairing*; Technical Report, Technical Report MSR-TR-2007-123; Microsoft Research: Washington, DC, USA, 2007.
26. Studer, A.; Passaro, T.; Bauer, L. Don't bump, shake on it: The exploitation of a popular accelerometer-based smart phone exchange and its secure replacement. In Proceedings of the 27th Annual Computer Security Applications Conference, Orlando, FL, USA, 5–9 December 2011; ACM: New York, NY, USA, 2011; pp. 333–342.
27. Groza, B.; Mayrhofer, R. SAPHE: Simple accelerometer based wireless pairing with heuristic trees. In Proceedings of the 10th International Conference on Advances in Mobile Computing & Multimedia, Bali, Indonesia, 3–5 December 2012; ACM: New York, NY, USA, 2012; pp. 161–168.
28. Bichler, D.; Stromberg, G.; Huemer, M.; Löw, M. Key generation based on acceleration data of shaking processes. In Proceedings of the International Conference on Ubiquitous Computing, Innsbruck, Austria, 16–19 September 2007; Springer: Berlin/Heidelberg, Germany, 2007; pp. 304–317.
29. Jiang, Q.; Huang, X.; Zhang, N.; Zhang, K.; Ma, X.; Ma, J. Shake to Communicate: Secure Handshake Acceleration-based Pairing Mechanism for Wrist Worn Devices. *IEEE Internet Things J.* **2019**, 6, 5618–5630. [CrossRef]
30. Sedgewick, R.; Wayne, K. *Computer Science—An Interdisciplinary Approach*; Pearson/Addison Wesley: Boston, MA, USA, 2017. Available online: https://introcs.cs.princeton.edu/java/home/ (accessed on 18 October 2020).