AdhocPairing: Spontaneous audio based secure device pairing for Android mobile devices

Stephan Sigg, Yusheng Ji
National Institute of Informatics (NII)
Tokyo, Japan
{sigg,kei}@nii.ac.jp

Ngu Nguyen
University of Science
Ho Chi Minh City, Vietnam
nlnngu@gmail.com

An Huynh
Tokyo University
Tokyo, Japan
zanton.zzz@gmail.com

ABSTRACT
We present an implementation of *AdhocPairing*, an audio-based secure pairing application for Android mobile devices. The application implements recent advances in audio-based pairing utilising Fuzzy cryptography. In particular, it generates audio fingerprints from ambient audio of weakly synchronised devices and extracts identical, arbitrary length secure binary keys. Synchronisation in audio samples is achieved by approximative pattern matching without inter-device communication. Additionally, we present results from a case study with android mobile phones and data on the entropy of recorded fingerprints.

Keywords
Secure spontaneous device pairing, Android, Pervasive computing

1. INTRODUCTION
Secure mobile communication among unacquainted devices naturally faces the problem that an authentication of the partner device is hard to accomplish. How can we be sure that the communication is actually conducted with the intended recipient and not with a third party device. Each attempt to generate a secure key among previously unacquainted devices must fail as long as the identity of the remote device can not be verified with high confidence. While classical cryptographic concepts can not provide a solution to this basic problem in mobile communication, Pervasive computing concepts may provide the necessary environmental device-specific information to authenticate also previously unacquainted devices. Common solutions require explicit user input to provide a shared piece of information. For instance, in Bluetooth, mobile phones generate a random text, called PIN code, and depend on their users for verification. However, these approaches merely shift the problem from the device to the user. This is not feasible in many applications due to unavailability of a user. It is also unclear how then the communication channel between users shall be secured and their identity authenticated. We present an application for android-devices which utilises audio-based, spontaneous secure key generation. In the application, the seed for the key is conditioned on ambient audio which sufficiently differs depending on spatial location [25]. Remote, third party devices in another context are then not capable to generate the same key. For the generation of the key, the devices need to agree on a time to take a synchronised audio recording but otherwise do not exchange any information which might be utilised to improve the probability to guess the generated key. Since the seed to the key is implicit with the context, no information that could be used to reconstruct the key is transmitted during key generation. A set of devices willing to establish a common key conditioned on ambient audio take synchronised audio samples from their local microphones. Each device then computes a binary characteristic sequence for the recorded audio: An audio-fingerprint. This binary sequence is designed to fall onto a code-space of an error correcting code. In general, a fingerprint will not match any of the codewords exactly. Fingerprints generated from similar ambient audio resemble but due to noise and inaccuracy in the audio-sampling process, it is unlikely that two fingerprints are identical. Devices therefore exploit the error correction capabilities of the error correcting code utilised to map fingerprints to codewords. For fingerprints with a Hamming-distance within the error correction threshold of the error correcting code the resulting codewords are identical and then utilised as secure keys. This process is explained in detail in [24, 25, 14]. The Hamming distance in fingerprints rises with increasing distance of devices so that distinct devices are unlikely to guess the correct key.
In this work we present a program for android devices that implements the fuzzy cryptography scheme introduced in [24, 25, 14]. Our implementation also exceeds the related work by the utilisation of a pattern based sequence alignment approach and by new case studies conducted in indoor environments as well as statistical tests conducted for this data to estimate the entropy of fingerprints.
We will in section 2 discuss related work on secure spontaneous pairing of mobile devices, audio fingerprints and sequence alignment. Section 3 presents our implementation and discusses discusses the realised pairing of devices (section 3.1), the synchronised recording of audio signals (section 3.2), the correction of synchronisation errors in the recorded sequences (section 3.3) and the creation of fingerprints (section 3.4). In section 3.5 we present results from a case study and data on the entropy of recorded audio samples. Section 4 draws our conclusion.

2. RELATED WORK
The authentication or establishing of a secure key among wireless devices is currently intensively discussed in the research community. A frequent approach is the utilisation of an out-of-band channel to transfer confidential information. Examples are ultrasound or visible laser light [19, 16]. But also audio is utilised as an out-of-band channel since it is typically more restricted in range than RF. For in-

stance, Kim et al. utilise an authenticated audio channel to share cryptographic information for their Short Random String (SRS) approach [12]. A similar concept is followed by Soriente et al. and Claycomb et al. who both rely on the location-limitation of audio as a source for secure information exchange [27, 7]. These approaches, however, mainly rely on the security of the additional channel. When this can be compromised, security of the protocol vanishes.

Some authors proposed to incorporate general contextual or sensor information of mobile devices as a solution for authentication. For instance, McCune et al. [20] introduced *Seeing-Is-Believing*. This system utilises the camera of a mobile device to capture a 2D barcode which is displayed on the screen of another device. *Loud and Clear* of Goodrich et al. [8] implements a similar scheme but exploits spoken audio. A user reads aloud a text message displayed on one device and a second device recognises the speech for authentication. As a further example of how sensor input can be utilised for device authentication, Mayrhofer et al. [17] presented a mechanism based on accelerometer readings when devices are shaken simultaneously by a single person. The authentication is possible with this approach and it is unlikely for a third person mimicking the movement pattern remotely to also obtain similar acceleration readings. Also, Mayrhofer derived in [15] that the sharing of secret keys is possible with a similar protocol. The proposed protocol repeatedly exchanges hashes of key-sub-sequences until a common secret is found. In contrast, Bichler et al. describe an approach in which noisy acceleration readings can be utilised directly to establish a secure communication channel among devices [2, 3]. They utilise a hash function that maps similar acceleration patterns to identical key sequences. These approaches, similar to others that address the problem of spontaneous device pairing by utilising contextual data [18, 28], are not unobtrusive since they require explicit user interaction.

By utilising a context source that provides a sufficient amount of unique, context-related information, such as audio or radio frequency (RF), it is possible to get the user out of the loop [11]. Early work in this direction exploited the unique channel characteristics of a wireless channel among devices. Wilhelm et al. for instance show that selective channel fading can be utilised to generate a secure key among remote wireless devices [30, 31]. Their approach, however, is not secure against a remote attacker since the generated key is created among any two remote devices and not necessarily with the intended communication partner.

By considering the context of a device also, this drawback, however, can be overcome. Mathur et al. introduced ProxiMate that enables wireless devices in proximity to pair automatically and securely with each other using a common key generated from their shared ambient RF-signals [14]. In their experiments, the keys are extracted from radio frequency signals broadcast by Software Defined Radio (SDR) devices. They generate fingerprints from RF-channel fluctuations and map these onto a codespace of an error-correcting code. Fingerprints are then seen as codewords with added bit errors, such as it would be expected after receiving a codeword transmitted over some error-prone communication channel. By correcting these potential errors in the fingerprints, they are mapped onto the closest regular codeword in the codespace. When the similarity between fingerprints is high, codewords on both devices are identical. This general scheme can be applied to arbitrary contextual data that incorporate sufficient amount of unique information. Marthur et. al utilised FM-radio or TV signals at low frequencies. For mobile devices, the approach can be adapted to higher frequency ranges. Then, however, due to the reduced wavelength and coherence distance, antennas of mobile devices must be close. For instance, in the ISM band, the distance can not be farther than about 6 cm. We utilise audio instead of RF in a similar implementation [25]. While this solution allows an increased range of the system, the instrumentation requires idealised conditions regarding the synchronisation of devices. Also, the implementation requires that a high number of fingerprints is created (201 in the experiments) in order to find one matching fingerprint. For extensive computational load, this is feasible only in an offline approach. The high number of fingerprints created, however, was necessary since the utilised NTP synchronisation is not sufficiently accurate. This problem was later solved with an approximative pattern matching in [22].

The fingerprints utilised in these approaches are well known from audio-processing, where audio fingerprints have typically been used in identification and matching of audio contents [9, 29, 21, 1]. An audio fingerprint is a compact representation of a piece of audio data [5]. It is required to be robust against noise and easy to compute. Wang [29] presented Shazam, a popular music search engine. With a small recorded piece of audio, this service can provide the name of the corresponding song with high accuracy. In this algorithm, the audio segment is converted to a spectrogram and local amplitude peaks are chosen to form a sparse set of time-frequency points. The peaks are combined with a number of their neighbours to create the fingerprint hashes. Haitsma and Kalker [9] claimed that the frequency domain contains the most important features of audio data. Therefore, in their proposed fingerprint extraction technique, they segmented the audio sequence into overlapping frames and applied a Fourier transform on every frame. Then they selected non-overlapping frequency bands of logarithmic scale to form a sub-fingerprint for each frame from bits that represent the energy fluctuation of the audio signal. Ogle and Ellis [23] adapted this approach in identifying recurrent sound events in personal recorded data. Waveprint, a novel audio fingerprint extraction method based on computer vision techniques, was introduced in [1]. The authors generated spectral images of the audio input. For each of these images, top wavelets are extracted. Then, they reduced the wavelets to a binary representation. Finally, the Min-Hash procedure generated the final sub-fingerprints. The locality-sensitive hashing technique was used in the matching process. Chandrasekhar et al. [6] provides a detailed comparison of popular audio-fingerprinting extraction schemes applied in an audio search engine on mobile devices. Our *AdhocPairing* application implements the fingerprint creation method from Haitsma et. al [9] to generate characteristic sequences from ambient audio for devices in proximity.

3. THE ADHOCPAIRING APPLICATION

We have developed *AdhocPairing*, an audio-based spontaneous device pairing application for android mobile devices. the application was tested on Samsung Nexus S[1] and HTC

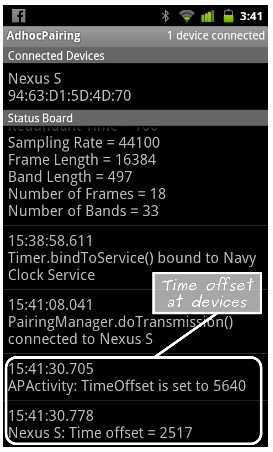
[1]Nexus S Technical Specifications: `http://www.google.com/nexus/tech-specs.html`

Nexus One[2] mobile phones. The Android OS of the Nexus One is CyanogenMode-7.1.0-N1 "cooked" version 2.3.7[3]. The Nexus S device has the official Android OS version 2.3.6. The application implements a scan for devices in proximity via Bluetooth, synchronised audio recording of devices, communication-free alignment of audio sequences and the creation of audio-fingerprints on the devices as seed for cryptographic keys.
Devices in proximity will generate similar fingerprints while for remote devices the Hamming distance in fingerprints increases so that the chance to generate a common key by error correcting codes quickly decreases with distance. A study on the entropy of ambient audio and on the impact of audio properties on the key generation was conducted in [24, 25]. Summarising these results, the entropy of ambient audio can be considered as high. also, a dominant audio source is beneficial for this approach. In particular, in silent environments or environments with a high number of equally loud noise sources (for instance a trafficked road), device pairing is hardly possible. With increasing noise level, the similarity in fingerprints decreases. Due to the utilisation of error correcting codes, however, the sensitivity of the approach to nose can be adapted by modifying the length of the sequence considered so that binary fingerprints that deviate by an arbitrary small amount from random sequences can be clearly distinguished from random sequences.
Since the audio fingerprints utilised are conditioned on energy fluctuation over time [9], time synchronisation among devices is an important issue here. For a first approximate time synchronisation, we utilise the NavyClock application[4]. This ensures that the clocks on remote devices are approximately synchronised. However, this synchronisation alone is not sufficient since also the hardware for recording ambient audio might have different time offsets as experienced in our related studies [22]. As described in [24, 25], audio-recordings of remote devices should not differ by more than few ten milliseconds in order to achieve sufficiently matching fingerprints. In our case, however, we frequently experienced a delay of about 0.5 seconds. Therefore, we utilise a pattern-based communication-free alignment of sequences (cf. section 3.3 and [22]).

3.1 Pairing of devices

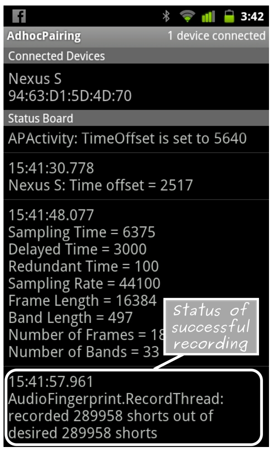
After the start of the application, a device will bound itself to the NavyClock clock synchronisation and wait for connections via Bluetooth in a client mode. Figure 1a shows a screenshot of the program with debug messages. In the top right, the application displays its status, which is to listen for Bluetooth pairing requests. The screen is further divided into two sections, showing the currently connected devices and a status board with debug messages. These messages show the default parameter of the *AdhocPairing* program and a status message that the device was successfully bound to the NavyClock synchronisation. The default parameters specify that an audio sequence of 6375 milliseconds will be recorded with 44100 samples per second. The 6375 millisec-

[2]Nexus One Technical Specifications: `http://www.google.com/phone/detail/nexus-one`
[3]CyanogenMode: `http:/www.cyanogenmod.com`
[4]We use the implementation of Navy-Clock to synchronise local time of devices: www.androidpit.de/de/android/market/apps/app/com.cognition.navyclock/Navy-Clock-II



```
3:41
AdhocPairing                1 device connected
Connected Devices
Nexus S
94:63:D1:5D:4D:70
Status Board
Sampling Rate = 44100
Frame Length = 16384
Band Length = 497
Number of Frames = 18
Number of Bands = 33

15:38:58.611
Timer.bindToService() bound to Navy
Clock Service

15:41:08.041
PairingManager.doTransmission()
connected to Nexus S

15:41:30.705
APActivity: TimeOffset is set to 5640

15:41:30.778
Nexus S: Time offset = 2517
```

(a) Client device listening for connections



```
3:42
AdhocPairing                1 device connected
Connected Devices
Nexus S
94:63:D1:5D:4D:70
Status Board
APActivity: TimeOffset is set to 5640

15:41:30.778
Nexus S: Time offset = 2517

15:41:48.077
Sampling Time = 6375
Delayed Time = 3000
Redundant Time = 100
Sampling Rate = 44100
Frame Length = 16384
Band Length = 497
Number of Frames = 18
Number of Bands = 33

15:41:57.961
AudioFingerprint.RecordThread:
recorded 289958 shorts out of
desired 289958 shorts
```

(b) Menu of the application

Figure 2: Comparison of time offsets and recording of audio

onds for one sequence are chosen such that 18 frames with 15618 samples fit into one sequence. Furthermore, to tolerate hardware-originated recording offsets, $2 \cdot 100$ additional milliseconds (in total a sequence of 6575 milliseconds) of audio are recorded 3 seconds after a pairing request is initiated. Also, the time-frames and frequency-bands for the fingerprint creation are detailed (more information on this process, is provided in [24, 25, 14]). The menu indicates further actions in this state (cf. figure 1b). In particular, by choosing 'connect a device', the device will change into server mode and send connection requests to other devices. Devices can be selected manually as displayed in figure 1c. A successful connection is indicated in the status bar and in the connected devices frame. Afterwards, the device reports the debug information that the pairing was successful (cf. figure 1d).

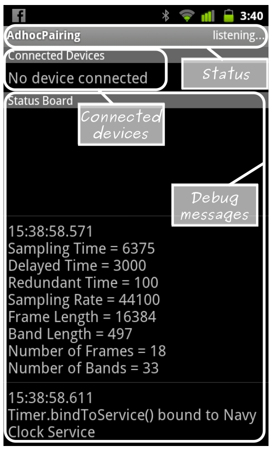
3.2 Recording of audio signals

To create distributed, synchronised audio sequences for device pairing, devices use their Bluetooth connection to initialise a recording. For debugging purposes, the software enables the sharing of synchronisation information. For instance, figure 2a shows the debug information of the local time offset of both devices to the NavyClock server (here: 2517 milliseconds and 5640 milliseconds).

The menu option '*Get audio sample*' on the server device (cf. figure 1b) will cause the server to publish a recording time to all connected devices, which in turn wait until that time instant and then start recording synchronised. By default, the recording time is set to 3 seconds after the '*Get audio sample*' menu option was selected (cf. status information in figure 1a). After recording, the status Board of the application displays a debug information reporting the successful caption of audio (cf. figure 2b).

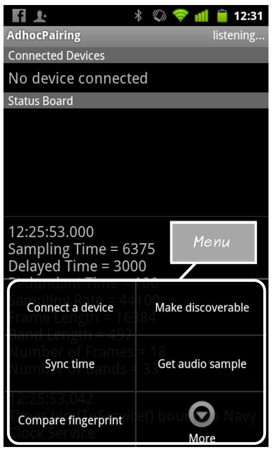
3.3 Alignment of sequences

As mentioned above, the synchronisation by NavyClock alone is not sufficient to achieve sufficiently synchronised audio recordings. The reason for this is that the recording hard-



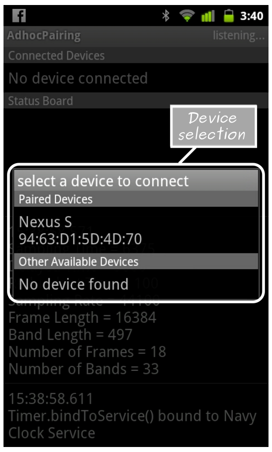
```
AdhocPairing                listening...
Connected Devices
No device connected
Status Board
15:38:58.571
Sampling Time = 6375
Delayed Time = 3000
Redundant Time = 100
Sampling Rate = 44100
Frame Length = 16384
Band Length = 497
Number of Frames = 18
Number of Bands = 33
15:38:58.611
Timer.bindToService() bound to Navy
Clock Service
```

(a) Client device listening for connections



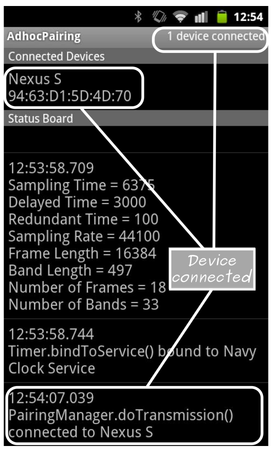
```
AdhocPairing                listening...
Connected Devices
No device connected
Status Board
12:25:53.000
Sampling Time = 6375
Delayed Time = 3000
```

(b) Menu of the application



```
select a device to connect
Paired Devices
Nexus S
94:63:D1:5D:4D:70
Other Available Devices
No device found
Frame Length = 16384
Band Length = 497
Number of Frames = 18
Number of Bands = 33
15:38:58.611
Timer.bindToService() bound to Navy
Clock Service
```

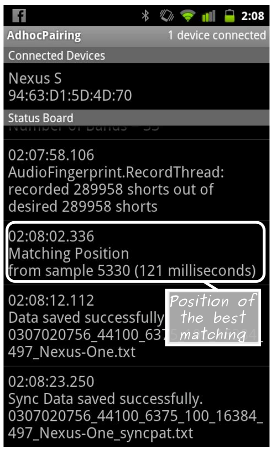
(c) Selection of a device for connection



```
AdhocPairing         1 device connected
Connected Devices
Nexus S
94:63:D1:5D:4D:70
Status Board
12:53:58.709
Sampling Time = 6375
Delayed Time = 3000
Redundant Time = 100
Sampling Rate = 44100
Frame Length = 16384
Band Length = 497
Number of Frames = 18
Number of Bands = 33
12:53:58.744
Timer.bindToService() bound to Navy
Clock Service
12:54:07.039
PairingManager.doTransmission()
connected to Nexus S
```

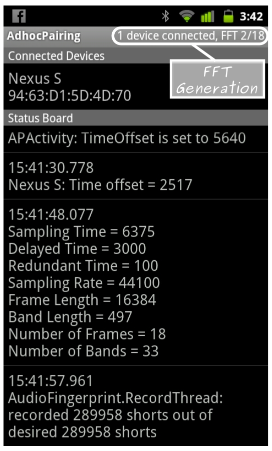
(d) Server device with one connected client

Figure 1: Screenshots of the *AdhocPairing* application with debug information after launching the application



```
AdhocPairing         1 device connected
Connected Devices
Nexus S
94:63:D1:5D:4D:70
Status Board
02:07:58.106
AudioFingerprint.RecordThread:
recorded 289958 shorts out of
desired 289958 shorts
02:08:02.336
Matching Position
from sample 5330 (121 milliseconds)
02:08:12.112
Data saved successfully
0307020756_44100_637
497_Nexus-One.txt
02:08:23.250
Sync Data saved successfully.
0307020756_44100_6375_100_16384_
497_Nexus-One_syncpat.txt
```

(a) Finding a best matching between pattern and sequence



```
AdhocPairing   1 device connected, FFT 2/18
Connected Devices
Nexus S
94:63:D1:5D:4D:70
Status Board
APActivity: TimeOffset is set to 5640
15:41:30.778
Nexus S: Time offset = 2517
15:41:48.077
Sampling Time = 6375
Delayed Time = 3000
Redundant Time = 100
Sampling Rate = 44100
Frame Length = 16384
Band Length = 497
Number of Frames = 18
Number of Bands = 33
15:41:57.961
AudioFingerprint.RecordThread:
recorded 289958 shorts out of
desired 289958 shorts
```

(b) Application of the Fast Fourier Transformation

Figure 3: Comparison of time offsets and recording of audio

ware at devices might induce further, unpredictable delays. We implement the approximate pattern matching detailed in [22] to account for this property. In particular, the application utilises a short (100 samples; approximately 0.003 seconds) predefined audio pattern which is identical for all devices to find in the 200 millisecond time offset of the recording a position that best matches this predefined pattern.

When recorded audio is similar on devices, even though the predefined audio pattern might be very different to the recorded audio, chances are good that the 'best matching position' is similar for both recorded audio sequences. After this pattern matching, the devices then utilise the audio sequences directly following the best matching position to create a fingerprint. Notice that the devices do not transmit any information about the recorded audio since a predefined pattern is utilised. Furthermore, as the sequence after the best matching position is utilised to create a fingerprint, the information obtained from the predefined audio pattern on the recorded sequence is small. In particular, the 'best' matching might be very dissimilar to the predefined pattern, even though it is the best possible matching.

Figure 3a shows the debug information on the screen of the server-device after the matching was computed. In the figure, a best matching was found starting at sample 5330, which translates to 121 milliseconds offset from the start of the recording. The implemented approximative matching of the predefined pattern and the recorded sequence is based on the Smith-Waterman algorithm [26], an approximative pattern matching technique.

The specific pattern used for matching was extracted randomly from consecutive samples of an arbitrary audio sequence. In our experiments, its length is 100 samples. Longer patterns increased the running time of the algorithm without improving the accuracy of the matching. Due to the fact that our sampling rate is 44100Hz, a 100-sample pattern is equivalent to a 0.003-second audio chunk. After finding the matching position, we can eliminate all samples preceding this position and generate the audio fingerprint from the remaining ones. Due to the noise caused by recording hardware and software diversity, the best matching position on one device may not correspond to the best one on the other. However, since the alignment matrix contains all matching scores, no recalculation is required, we can easily find the top k matching locations of the pattern on each audio file.

3.4 Fingerprint creation

Fingerprints are created following the approach described in [10] as further detailed in [25]. In particular, for each of the frames (here 18), the Fast Fourier Transform (FFT) is calculated and the result is equally split into 33 bands[5]. Afterwards, binary fingerprints are created by comparing the energy fluctuation in successive frames (for further details on this process, please refer to [25]. Figure 3b depicts a screenshot of the program during FFT generation. The status in the upper right reports the status of the FFT computation. The created fingerprints are then stored on the

[5]these figures can be modified in the configuration

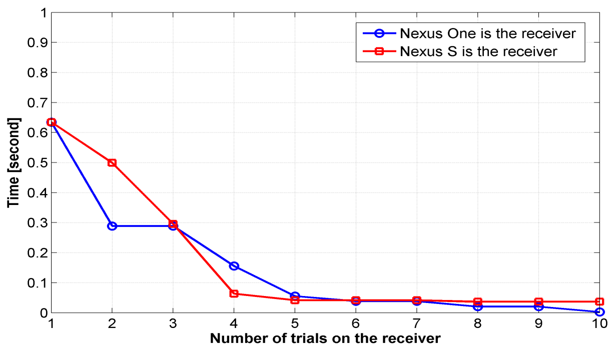


Figure 4: Time difference of audio sequences after alignment when a transmitter uses only the best matching position to generate the key while the receiver decrypts up to ten times with the corresponding matching positions.

device as a seed to create the cryptographic key. We have demonstrated in [25] that fingerprints of proximate devices are more similar than those of remote devices and that the remaining errors in fingerprints can be corrected without communication among devices by utilising error correcting codes to create identical cryptographic keys.

3.5 Secret keys with sequence alignment

In the case study, a Samsung Google Nexus S and a HTC Google Nexus One, are placed in the same distance d from an audio source. Ten audio sequences are recorded on each phone in a distance of 20cm.

In a pairing session of two phones, each of them records a $6.375 + 0.2$ second audio file. To improve the alignment results, the top 10 matching positions of a common pattern in each audio sequence are found. Then, the audio fingerprints to create the cryptographic keys are extracted. After that, each phone has 10 keys, which are obtained without exchanging any information between the devices. In this case study, each device has one of two roles, transmitter or receiver, alternately. The transmitter uses the key created from the best matching position to encrypt a pre-defined piece of information and send the encrypted data to the receiver. At the opposite site, the receiver tries to decrypt the encrypted data chunk, using its key from the best matching result. If it fails, it tries again with the key generated from the *second best* matching location, and continues for 10 trials until it can recover the data. For sufficiently similar fingerprints, a time offset in the order of 10 milliseconds is required for devices in the same audio context (cf. [24, 25]). With this scheme, we evaluate the efficiency of our pattern-based alignment method. Figure 4 depicts the median time difference after alignment when increasing the number of trials in the receiver from 1 to 10. In our experiments, we can achieve a synchronisation in the accuracy of 2 milliseconds with 10 trials at the receiver.

We also investigate the entropy of these audio fingerprints with the DieHarder statistical test suite [4]. For a 512-bit fingerprint, each test in the suite produces a p-value [13] which denotes the probability of the fact that this binary sequence is generated from a truly random bit generator. We run all tests in the DieHarder suite for 20 times. Because the suite contains 114 single tests, the total number of statistical tests applied on the set of fingerprints is 2280. The percentage of "PASSED" results is 92%. Figure 5 shows the histogram plot of p-values. The results suggest that there is no bias in the fingerprints generated from recorded ambient audio. Therefore, an attacker can not gain significant information from collecting encrypted messages.

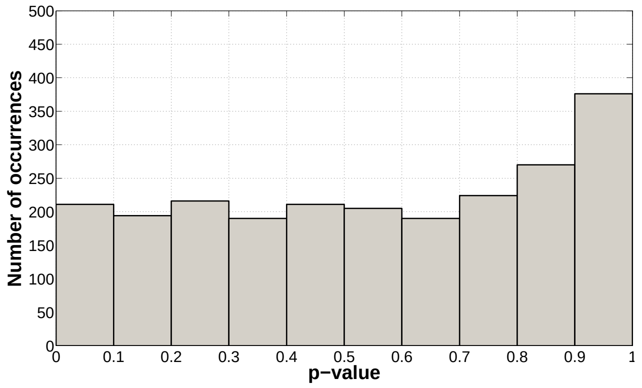


Figure 5: Histogram of p-values for all DieHarder tests.

4. CONCLUSION

We introduced *AdhocPairing*, an android application to generate spontaneous secure keys from ambient audio of devices in proximity. For synchronisation of recorded audio sequences the application utilises an approximative pattern matching that does not require inter-device communication to synchronise audio recordings from remote devices. The application was instrumented in case studies with different android mobile devices and the entropy of recorded audio was studied.

5. ACKNOWLEDGMENTS

This work was supported by a fellowship within the Postdoc-Programme of the German Academic Exchange Service (DAAD)

6. REFERENCES

[1] S. Baluja and M. Covell. Content fingerprinting using wavelets. In *Proceedings of the Conference of Visual Media Production*, London, UK, 2006.

[2] D. Bichler, G. Stromberg, and M. Huemer. Innovative key generation approach to encrypt wireless communication in personal area networks. In *Proceedings of the 50th International Global Communications Conference*, 2007.

[3] D. Bichler, G. Stromberg, M. Huemer, and M. Loew. Key generation based on acceleration data of shaking processes. In J. Krumm, editor, *Proceedings of the 9th International Conference on Ubiquitous Computing*, 2007.

[4] R. G. Brown. Dieharder: A random number test suite. http://www.phy.duke.edu/~rgb/General/dieharder.php, 2011.

[5] P. Cano, E. Batlle, T. Kalker, and J. Haitsma. A review of audio fingerprinting. *Journal of VLSI Signal Processing Systems*, 41 Issue 3, 2005.

[6] V. Chandrasekhar, M. Sharifi, and D. Ross. Survey and evaluation of audio fingerprinting schemes for mobile audio search. In *International Symposium on*

Music and Information Retrieval (ISMIR), Miami, Florida, October 2011.

[7] W. Claycomb and D. Shin. Secure device pairing using audio. In *43rd Annual International Carnahan Conference on Security Technology*, pages 77–84, 2009.

[8] M. T. Goodrich, M. Sirivianos, J. Solis, G. Tsudik, and E. Uzun. Loud and clear: Human-verifiable authentication based on audio. In *Proceedings of the 26th IEEE International Conference on Distributed Computing Systems*, 2006.

[9] J. Haitsma and T. Kalker. A highly robust audio fingerprinting system. In *3rd International Conference on Music Information Retrieval (ISMIR 2002)*, pages 107–115, Paris, France, 2002.

[10] J. Haitsma and T. Kalker. A highly robust audio fingerprinting system with an efficient search strategy. *Journal of New Music Research*, 32(2), October 2003.

[11] L. E. Holmquist, F. Mattern, B. Schiele, P. Schiele, P. Alahuhta, M. Beigl, and H. W. Gellersen. Smart-its friends: A technique for users to easily establish connections between smart artefacts. In *Proceedings of the 3rd International Conference on Ubiquitous Computing*, 2001.

[12] Y. S. Kim, S. H. Kim, and S. H. Jin. Srs-based automatic secure device pairing on audio channels. In *International Conference forInternet Technology and Secured Transactions (ICITST)*, pages 1–6, 2010.

[13] N. Kuiper. Tests concerning random points on a circle. In *Proceedings of the Koinklijke Nederlandse Akademie van Wetenschappen*, volume Series a 63, pages 38–47, 1962.

[14] S. Mathur, R. D. Miller, A. Varshavsky, W. Trappe, and N. B. Mandayam. Proximate: proximity-based secure pairing using ambient wireless signals. In A. K. Agrawala, M. D. Corner, and D. Wetherall, editors, *MobiSys*, pages 211–224. ACM, 2011.

[15] R. Mayrhofer. The Candidate Key Protocol for Generating Secret Shared Keys from Similar Sensor Data Streams. *Security and Privacy in Ad-hoc and Sensor Networks*, pages 1–15, 2007.

[16] R. Mayrhofer and H. Gellersen. On the security of ultrasound as out-of-band channel. In *Proceedings of the 21st IEEE International Parallel and Distributed Processing Symposium (IPDPS 2007)*, 2007.

[17] R. Mayrhofer and H. Gellersen. Shake well before use: Authentication based on accelerometer data. *Pervasive Computing*, pages 144–161, 2007.

[18] R. Mayrhofer and H. Gellersen. Spontaneous mobile device authentication based on sensor data. *information security technical report*, 13(3):136–150, 2008.

[19] R. Mayrhofer and M. Welch. A human-verifiable authentication protocol using visible laser light. In *Proceedings of the 2nd International Conference on Availability, Reliability and Security (ARES 2007)*, 2007.

[20] J. M. McCune, A. Perrig, and M. K. Reiter. Seeing-is-believing: Using camera phones for human-verifiable authentication. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy*, 2005.

[21] M. Mueller, F. Kurth, and M. Clausen. Audio matching via chroma-based statistical features. In *Proceedings of the 6th International Conference on Music Information Retrieval (ISMIR)*, pages 288–295, 2005.

[22] N. Nguyen, S. Sigg, A. Huynh, and Y. Ji. Pattern-based alignment of audio data for ad-hoc secure device pairing. In *Proceedings of the 16th annual International Symposium on Wearable Computers (ISWC)*, 2012.

[23] J. P. Ogle and D. P. W. Ellis. Fingerprinting to identify repeated sound events in long-duration personal audio recordings. In *IEEE International Conference on Acoustics, Speech and Signal Processing 2007 (ICASSP 2007)*, 2007.

[24] S. Sigg and D. Schuermann. Secure communication based on ambient audio. *IEEE Transactions on Mobile Computing (TMC)*, 2012. Accepted for publication.

[25] S. Sigg, D. Schuermann, and Y. Ji. Pintext: A framework for secure communication based on context. In *Proceedings of the Eighth Annual International ICST Conference on Mobile and Ubiquitous Systems:Computing, Networking and Services (MobiQuitous 2011)*, 2011.

[26] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197, Mar. 1981.

[27] C. Soriente, G. Tsudik, and E. Uzun. Hapadep: Human-assisted pure audio device pairing. In T.-C. Wu, C.-L. Lei, V. Rijmen, and D.-T. Lee, editors, *Information Security*, volume 5222 of *Lecture Notes in Computer Science*, pages 385–400. Springer Berlin / Heidelberg, 2008.

[28] A. Varshavsky, A. Scannell, A. LaMarca, and E. de Lara. Amigo: Proximity-based authentication of mobile devices. *International Journal of Security and Networks*, 2009.

[29] A. Wang. An industrial-strength audio search algorithm. In *Proc. 2003 ISMIR International Symposium on Music Information Retrieval*, 2003.

[30] M. Wilhelm, I. Martinovic, and J. B. Schmitt. Secret keys from entangled sensor motes: implementation and analysis. In *Proceedings of the third ACM conference on Wireless network security*, WiSec '10, pages 139–144, 2010.

[31] M. Wilhelm, I. Martinovic, E. Uzun, and J. Schmitt. Sudoku: Secure and usable deployment of keys on wireless sensors. In *6th IEEE Workshop on Secure Network Protocols (NPSec)*, pages 1–6, 2010.