FACULTY OF FUNDAMENTAL PROBLEMS OF TECHNOLOGY
WROCŁAW UNIVERSITY OF TECHNOLOGY

CHANNELS ALTERNATIVE TO WIFI IN AUTHENTICATED KEY ESTABLISHMENT PROTOCOLS FOR MOBILE DEVICES

PAWEŁ KĘDZIA

In partial fulfillment of the requirements
for the degree of Master of Engineering

Thesis Supervisor
dr inż. Łukasz Krzywiecki

WROCLAW 2014

Contents

1 Introduction

The goal of the dissertation is to analyze the usability of channels alternative to WiFi in Authenticated Key Establishment (AKE) protocols for mobile devices. Nowadays, more often mobile phones are used to sending secret data. Every this kind of device has embedded speaker and microphone therefore we want to check whether is possible to establish session key using just sound waves.

Mobile devices to transfer data mainly use Internet. Disadvantage of this method is that network not always is available. In that case one can still transmit data using some another embedded components like IrDA, Bluetooth or NFC. Unfortunately, not every device has these kind of accessories implemented. However, basis of handhelds is to have embedded speaker and microphone. That is why, in this dissertation, to communication between devices were chosen audio channel. Availability audio appliance and a lot of uses of this solution are main advantages. Furthermore receiver as like sender do not need to known each other (unlike Bluetooth) before starting the communication. Sender just starts to transmit data and receiver in the same time listens broadcast.

To authentication and key agreement between mobile devices was chosen Anonymous Mutual Authentication (AMA) protocol [14]. AMA is symmetric, which means participants execute the same operation (with some little difference in order of operations). One of the advantage of this protocol is eases implementation on resource limited devices. It was originally designed for authenticating electronic documents, it proved to be portable.

Application created on Android system will serve as proof of concept. There are applications which use sound channel to exchange data, but none of them are used to establishing session keys. To use our application there is need just mobile phone/smartphone with software Android (version 4.1.2 or higher), working speaker and microphone, of course.

The first section focuses on analysis of the problem. It explains principal issues used to create the application. It consists some information about security problem and necessary definitions to create sound channel. In the next chapter is described design of prototype. What was used, how application work etc. Then are shown results of test.

The problem is secure exchange keys between two parties, which have no knowledge of each other, in insecure, unreliable environment e.g. susceptible to eavesdropping.

The first known answer on this problem was published by Witfield Diffie and Martin Hellman. They have showed method to key exchange which is called Diffie-Hellman. Its strength and immunity to eavesdropping is based on Discrete Logarithm Problem. Existing

computers are not able to compute session key from eavesdropped protocol transcript in reasonable amount of time.

Mentioned method is secure against eavesdropping, however it does not authenticate both parties to each other and thus does not guarantee security when adversary can take control over the channel in which the key establishment is being performed. Adversary, taking control over the channel, can perform Man-In-The-Middle-Attack (MITMA). MITMA is that, the adversary try to make some kind of connection with participants of protocol, being not detected. He can manipulate the communication pretending to be one of the party, changing or sending own messages, making them to believe that they are still talking to each other. The goal of these processes is obtain secrete key by attacker and thus decrypt and even modify all messages exchanged between parties.

To exchange data are used mainly electromagnetic radiation channels like Bluetooth, IRDa or Internet. We have wanted to check another options to exchanging data between devices, which does not use electromagnetic radiation. We have decided to study sound channel.

1.1 Related work

There are several ways to establish secure channel generating session key. There are solutions where are used shared secret password which prevent Man-In-The-Middle attacks as in [25, 24]. However, pre-shared passwords are not always practical in many scenarios. There are protocols such as *KEA* [20], *CMQV* [29],*HMQV* [18], *Naxos*, *Naxos+* [12] and *SIGMA* [19] which are in the Authenticated Key Agreement family. The security requirements and security model definition in AKE protocols were proposed by Cenetti and Krawczyk in [30, 23]. To exchange data we used sound channel which is one of the considered out-of-band channel for key exchange protocols. This kind of channel are discussed in [13, 17, 21, 27, 8].

2 Analysis of the problem

In this section we will focus on problem of establishing secure connection in insecure channel between two parties. At first the parties need to authenticate each other and then create common session key, which will be needed to symmetric encryption of the communication. We describe the adversarial model where is assumed that each of the parties secretly holds a long-term private key that enables unique authentication of this party. There is also a trusted mechanism for binding identities with public keys. The private and public key with proper length are bounded in such way that obtain private key from public key is very hard to do. The communication channel can be completely controlled by an adversary that can

intercept, delete, delay, modify or inject messages at will. There is a relation between the requirements of the security model and a well-defined attack scenario. In the second part of this section are described necessary issues to create sound channel.

2.1 Authenticated Key Agreement

Authenticated Key Agreement (or Authenticated Key Exchange) Protocols are such protocols where two parties can establish a common session key which will be use to symmetric encryption in further communication between parties. They base on public key cryptography and trusted authorities which are the most common solution being used. In public key cryptography (asymmetric cryptography) are required two separate keys, where one is private and second one is public. The keys are mathematical related to each other.

AKE protocols have to ensure that attacker can not obtain any information about session key from the protocol transcript. They have to be resistant to exposed session transcript. Adversary should not be able to taking control over the communication channel having this knowledge.

2.2 Anonymous Mutual Authentication

To establish session keys we used in our prototype Anonymous Mutual Authentication protocol which base on Diffie-Hellman key exchange. AMA originally was designed to anonymous authenticate between smart cards. It is characterized by properties:

- *Forward Security* - if long term keys of one of the authenticating parties compromised then established session key does not reveal,
- *Simultability* - it is possible to create protocol transcript between parties without interacting with them,
- *Undetectability* - having protocol transcript one cannot confirm that process really took place,
- *Privacy* - eavesdropping adversary cannot tell about identity of parties performing the protocol
- *Deniability* - no one of the participants can convince third party that he participate in protocol.

Alice		Bob
x_A - private key		x_B - private key
$y_A = g^{x_A}$ - public key		$y_B = g^{x_B}$ - public key
$cert_A$ - certificate for y_A		$cert_B$ - certificate for y_B
	MAIN PROCEDURE	
choose a at random		choose b at random
$h_A := H(a)$		$h_B := H(b)$
$c_A := g^{h_A}$	$\xrightarrow{c_A}$	$c_B := g^{h_B}$
	$\xleftarrow{c_B}$	
$K := c_B^{h_A}$		$K := c_A^{h_B}$
$K_A := H(K,1), K_B := H(K,2)$		$K_A := H(K,1), K_B := H(K,2)$
$K'_A := H(K,3), K'_B := H(K,4)$		$K'_A := H(K,3), K'_B := H(K,4)$
	$\xrightarrow{Enc_{K_A}(cert_A, r_A)}$	check $cert_A$, proceed with random values if $r_A \neq H(y_A^{h_B}, K'_A)$
	$\xleftarrow{Enc_{K_B}(cert_B, r_B)}$	$r_B := H(c_A^{x_B}, K'_B)$
check $cert_B$, proceed with random values if $r_B \neq H(y_B^{h_A}, K'_B)$		
$K_{session} := H(K,5)$		$K_{session} := H(K,5)$

Figure 1: Protocol description - Anonymous Mutual Authentication. Figure from source [14].

Scheme of protocol is shown at the fig. 1. In the scheme are used the following notations:

- *Enc* is a symmetric encryption function. $Enc_K(M)$ means encryption of M using key K.
- H is a cryptographic hash function.
- For confirming public keys are using digital certificates and public key infrastructure (PKI).

At the beginning, parties (Alice and Bob) generate their private key - x_A (respectively, x_B) and public key $y_A = g^{x_A}$ (respectively, y_B). Then parties generate their ephemeral keys. Private is $h_A := H(a)$ (where a is a random number) (respectively h_B, b) and public ephemeral key $c_A := g^{h_A}$ (respectively, c_B). Then participants exchange their ephemeral keys, based on Diffie-Hellman key exchange. Note that in this phase parties do not exchange any identity information.

When parties obtain ephemeral public key, they start compute master key K. Four different one-time keys are established by hashing K and numeric parameter - different for each one-time key.

To authentication, parties encrypt their certificates $cert_A$ and $cert_B$ and the challenge values r_A and r_B which are the hash of two keys. Authenticated party is raising ephemeral public key c_B (respectively, c_A) to power of private key x_A (respectively, x_B). Verifier compute the same value without private key but with the discrete logarithm of c_A (respectively, c_B) which is y^{h_B} (respectively, y^{h_A}). If the compared values are equal then $K_{session}$ is the hash of master key K and new number parameter.

This protocol is proved to be secure under Random Oracle Model (ROM) for the used hash function H. In ROM, hash function H is modeled as an oracle O_H. For a query x, oracle responds y if in internal table T exist entry (x, y). If there is no such an entry for x, then O_H selects y at random, creates entry (x, y) is in T and answers y. In this model the only available operation concerning hash function H is asking for values for concrete arguments. Hence, having value y the only way to get $x = H^{-1}(y)$ is to remember it from a previous query or ask a query with new x and get y by accident.

Besides ROM, its security base on aforementioned CDH assumption which is hard to solve.

Assumption 1. ***(CDH Assumption)*** *Given a cyclic group G with order q, and (g,g^a,g^b) where g us a random generator of G, and (a,b) where chosen randomly from the set 0,1,...,q − 1 it is computationally intractable to compute the value g^{ab}.*

In the protocol is used symmetric encryption. Security proof of this is shown by series of games, for more details see [14].

The practical part of this paper was to implement AMA on mobile devices as to our best knowledge there is no practical implementation of AMA for smartphones.

2.3 Security Model

There is no one ultimately defined security model for key exchange (KE). However, we will recourse to the requirements as most fundamental for secure key exchange protocol which H. Krawczyk presents in [19].

- Authentication - each party participating in KE needs to uniquely verify identities of other authenticating party.
- Consistency - two parties participating in KE, after establish of session key have to believe that were established common session key with party which participate in

protocol. In other words, Alice has to believe that she established session key with Bob and Bob has to believe that he established session key with Alice.

- Secrecy - if two honest parties established session key then third party should not be able to obtain information about this session key. Third party, which watching and interfering with the protocol run, should not be able to distinguish session key from a random key.

2.3.1 Adversarial Model

There is needed to consider scenario where adversary learns some secrete information from participants of protocol. The worst case is when adversary obtain the long term secrete key from one of the party. Having this key adversary can completely control this party and the party is consider as corrupted. This lead to the fundamental property of key exchange protocols:

- Real-or-Random - this is some kind of game on which basic session key security is based. When uncorrupted parties with successful end the protocol then is determined bit b at random. If $b = 0$ the adversary get the real session key, if $b = 1$ he receives a random value with the same length as the key. Now adversary has to output his own bit b'. If $b' = b$ then adversary wins the game. When the adversary wins the game with probability $P = 1/2 + \varepsilon$, where ε is negligible function then the basis session key security is achieved.

- Perfect Forward Secrecy (PFS) - is the property which ensure that adversary cannot learn session keys from future when one of the long-term private key is compromised.

Above criteria ensure for key exchange protocol that they are resistant to attacks like known-key attacks and replay attacks [10]. For some KE protocols are needed additional requirements e.g. Identity protection - identities of authenticating parties cannot be learn by an adversary.

2.4 Analysis of out-of-band channels

Beyond the security of AKE protocol, we wanted to study not typical communication channel to establish session key, which could hinder the manipulation in protocol transcript.

During the secure exchange data are needed two channels for communication between two parties. First channel, to establish session keys, should be difficult to eavesdrop and taking over control for adversary. The second channel can be insecure because established session key should ensure that content of protected data intended for one of the parties can be encrypt only by the party participated in protocol.

There are a lot of techniques to transfer secret data between devices. The most popular are e.g. Bluetooth, IrDA or WiFi. These techniques use electromagnetic waves. To manipulating, changing or eavesdropping messages are needed devices which operates on electromagnetic radiation (e.g. bug in the phone). So it is worth to consider out-of-band channel which is not so commonly used as electromagnetic devices, and use another medium.

Safety channel could be, for example, two devices connected by wire. For the third party would be very difficult to eavesdrop the session. Attacker should be connected also to this wire but it would be visible for the other parties and easily to detect. Unfortunately, carrying additional cable every time when the parties want to establish session key is inconvenient.

The other possibility could be optical channel [22]. Sharing data using e.g. QR codes is also interesting way to authentication of devices. Attacker would have difficult task to manipulate shared data if e.g. two devices are covered by some walls. However, not every device has camera with appropriate quality that can read QR codes with appropriate big public keys.

Another way is to use vibrations [8]. New smartphones have installed sensor of vibration so it is possible to create such a communication channel. Attacker to manipulate this kind of channel has to generate own vibrations and it would be easily detected by authenticating parties.

Sound channel is also one of the options. Every mobile device has speaker and microphone, so we do not need to worry about no available Internet or that the other party has not installed Bluetooth or IrDA or even camera. Using ultrasounds, the signal will be not detected by humans ear. Additionally mobile phones cannot generate these waves with big intense, so transferring data should be done on small distance. Of course, sound channel could be easily to manipulate. Attacker with proper speaker can generate frequencies which could disturb the communication. However, he needs to be close to the authenticating parties if generated ultrasound has to be detected by party's mobiles microphone or generate sound waves with big intense which could be audible for human. Additionally is worth to consider to use this kind of channel mixing with another channel. For example, ephemeral keys will be sending by sound channel then encrypted values by Bluetooth or one party everything sends by sound waves when the second party uses just electromagnetic channel.

2.5 Proposed usage of out-of-band channel

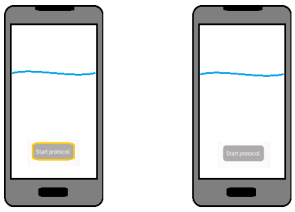
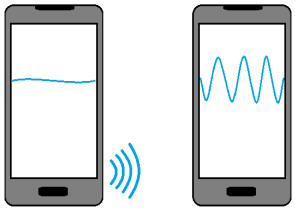
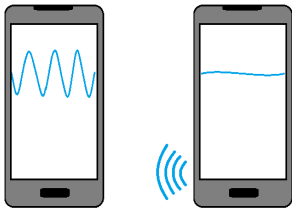
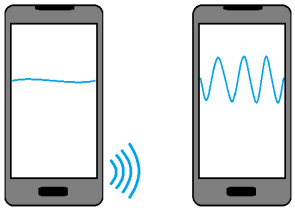
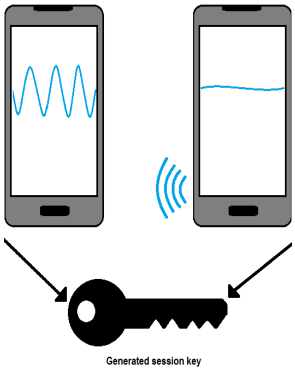
Lets assumed that Alice and Bob want to establish a secure connection between their mobile phones using sound channel. To be sure that every high frequency will be detected, they try to put their phone in a such way that microphone is as close as possible to speaker of the second phone and vice versa. Usually the speaker and microphone are set at the bottom of

device, so mobile phones should be directed bottom to each other. To obtain session key is processed AKE protocol. After this process further exchange of data can be done by WiFi, because of the speed of transferring data which is higher.

- Alice and Bob put their devices into the lead box. Both mobile phone are listening for the first sign. The first one which start generated sound wave will be initiator. For the best quality of transferring data mobiles should be directed bottom to each other no more than couple of millimeters.
- Alice decided to be initiator of the protocol. She says "Start" to the little hole in the box and her phone, after recognized her voice, start sending first data.
- Protocol is divided on couple of phase. First phase is sending ephemeral key by initiator (in our case by Alice) to the receiver. Each phase takes couple of seconds. Every data are encoded in sound waves with appropriate frequencies.
- Bob's phone uses its microphone to obtain the data sent by Alice's device.
- When phase is over, follow switch of the roles. Now Alice's phone will listening and Bob's phone sending data.
- Alice's device will receiving and decode it.

Anonymous Mutual Chip Authentication (described in detail in section above) should ensure that even if some transferring data eavesdrop then adversary will do nothing with this data. Duration of the protocol is about 35 second - more information about duration and performance are described in section 5.

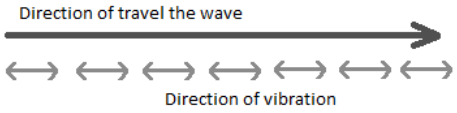
Table 1: Scheme of protocol used in devices.

Alice		Bob
Starting screen Alice starts the protocol as initializator		**Starting screen** Bob starts protocol as receiver.
Send ephemeral key Alice generates ephemeral key, encodes and start sending it by speaker.		**Receive ephemeral key** Bob receives ephemeral key using microphone.
Receive ephemeral key Alice receives ephemeral key using microphone.		**Send ephemeral key** Bob generates ephemeral key, encodes and start sending it by speaker.
Send encrypted authentication data Alice uses her and Bob's ephemeral keys to compute authentication data, encrypts it and sends by speaker.		**Receive and decrypt authentication data** Bob receives and decrypts Alices authentication data.
Receive and decrypt authentication data Alice receives and decrypts Bobs authentication data. If it's not valid, she aborts the protocol.		**Send encrypted authentication data** Bob verifies authentication data received from Alice. If it's valid, he generates his authentication data and send it by microphone. If data is not valid, Bob aborts the protocol.

2.6 Sound

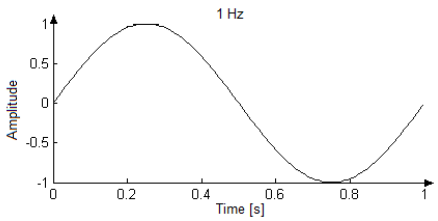
Sound and its properties have one of the important role in our dissertation. All of the data are sent using sound waves, so in this section are explained necessary issues associated with it.

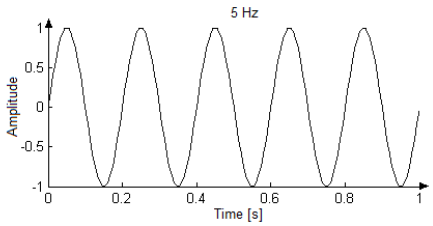
Sound is an acoustic wave propagating in various substances such as water, air (so called vibrating wire). There are waves which cause auditory sensation and at the same time, in appropriate amplitude and frequency, are not detected by human organ of hearing. Sound to spread needs to have some medium, that is why not propagate in vacuum. Furthermore, sound is longitudinal waves, which means that particles of the medium is in the same direction as the direction of travel of the wave.



2.6.1 Frequency

Audio frequency is measured in hertz (Hz), where 1 Hz means one cycle per second. Below figures show graphs of audio frequency 1 Hz and 5 Hz.





There exist three division of sound as to frequency:

- Infrasound - frequency is lower than 16 Hz. Too low frequency to be audible for human. Infrasounds have very large length of wave, more than 17 meters. They have influence on humans comfort. On high level we can feel pressure in ears, discomfort, excessive fatigue or somnolence.
- Hearable sound - frequency is greater than 16 Hz and lower than 20 kHz.
- Ultrasound - frequency is greater than 20 kHz. Too high to be audible for human. Ultrasounds are used inter alia in medicine (USG) or in sonars.

- Hipersound - frequency is greater than 10 GHz. They can spread only in crystals, because of such a high frequency. The wave is smaller than distance between molecules in the environment. They disappear in the gases.

Frequency specify also pitch of sound. A high frequency sound wave corresponds to a high pitch of sound and low frequency sound wave corresponds to a low pitch of sound.

2.6.2 Sound intensity

Loudness of the sound is dependent to his intensity. It is expressed by W/m^2 unit. This is ratio the power of the acoustic wave to surface area to which wave energy falls perpendicular during 1 second [4]. Human's ear works in such way that doubling intensity of sound, do not cause that human hear it two times louder. Human hears its logarithmized value. Logarithmized measure of sound intensity is called sound intensity level. It is measured in decibels and is given by [6]:

$$L = 10\log_{10}\left(\frac{I}{I_o}\right)[dB], \tag{1}$$

where:

- L - Sound intensity level,
- I - Sound intensity,
- I_o - Contractual value of the intensity which determines hearing range, which is $10^{-12}\frac{W}{m^2}$.

2.6.3 Speed of sound

Speed of sound depends on center through which wave passes. Influence on speed has also temperature of the center. The slowest it moves in the flue. In the air with 20^o temperature reach in approximation 343 m/s (1230 km/h). In water with the same temperature speed is 1482 m/s (5335 km/h). The fastest sound spreads in solid. For example in steel it reach about 5960 m/s (21460 km/h) [7].

2.6.4 Ultrasounds

The present application largely, to sending data, uses ultrasound. Ultrasounds are sound waves, which are inaudible for human, on the grounds of their too high frequency. Have adopted that ultrasounds starts from 20 kHz, but really everything depends on how good hearing human has. Children hear the highest frequencies. Increasing age the upper limit of hearing descend [26]. Ultrasounds end at frequencies of several GHz, where hipersounds start. Contractual it is 10 GHz. Ultrasounds have many applications. Using them one can

detect objects and specify distance to them. Are used in medicine where whereby ultrasonography (USG) we can find not available for human eye diseases, watching e.g. internal organs.

2.6.5 Fourier transform

In our prototype microphone receives signal as a sinus wave. Because of that we need to know the frequency of this signal, we have to use some transform which turn sinus wave into the frequency spectrum. The solution is to use *Fourier transform*.

Using *Fourier transform* one can change function $f(x)$ in time domain to $F(s)$ in frequency domain. In other words, it distributes the function $f(x)$ to series of periodic functions, in such way that it shows how frequencies consist of the function $f(x)$. The formula for the *Fourier transform* looks as fallows:

$$F(s) = \int_{-\infty}^{\infty} f(x) e^{-i2\pi xs} dx. \tag{2}$$

Sum of three functions with frequencies 3 Hz, 7 Hz and 11 Hz on a graph:

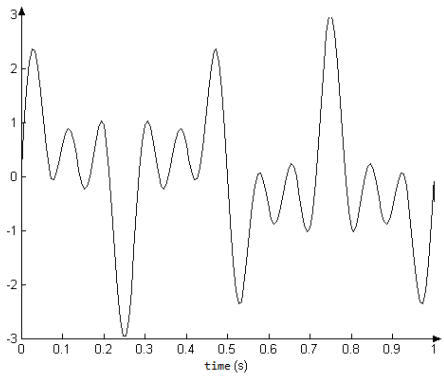


Figure 2: $f(x) = \sin(2\pi \cdot 3t) + \sin(2\pi \cdot 7t) + \sin(2\pi \cdot 11t)$, where t is current time.

Passing function $f(x)$ from fig. 2 through *Fourier transform* we will get the following graphs:

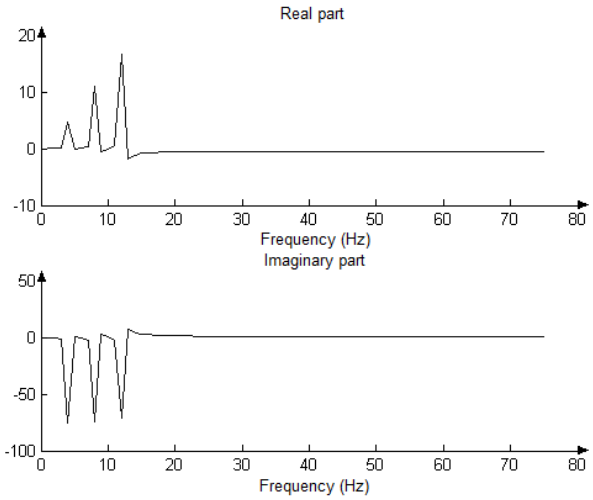


Figure 3: On the top real part results of *Fourier transform*, on the bottom imaginary part.

As you can see on above figure, time domain was changed on frequency domain. On graphs are bigger values for frequencies which had functions summed.

2.6.6 Discrete Fourier Transform

Application analyzes periodically finite number of samples of sound. To converse to frequency domain is used *discrete Fourier transform* (*DFT*). Inserted samples are in the form of complex numbers, but in practice used only real numbers. Output samples are complex numbers. The following graphs show continuous signal and discretized signal [1]:

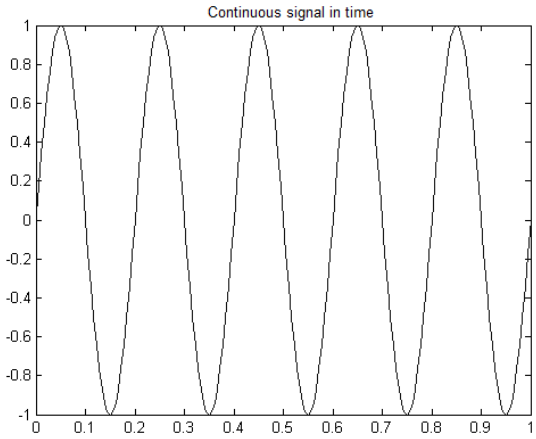


Figure 4: Continuous signal in duration 1 second.

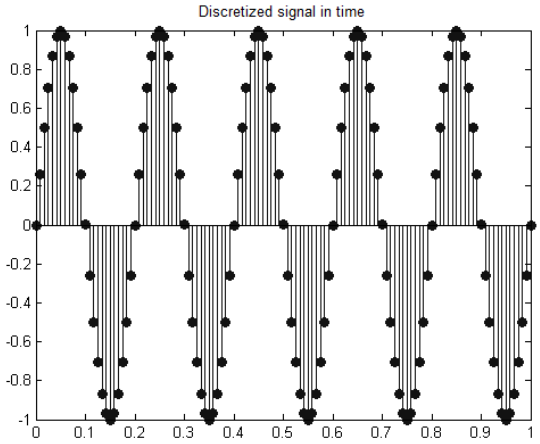


Figure 5: Discretized signal in constant time period. In this case in 1 second are 121 samples.

For N signal samples $(x_0, x_1, x_2, ..., x_{N-1})$, *discrete Fourier transform* gives series X_k for $k = 0, 1, 2, 3, ..., N-1$ using notation [1]:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi \frac{k}{N} n}. \tag{3}$$

Computational complexity this notation is $O(N^2)$, because for N elements X_k series compute sum for N samples [16].

2.6.7 Fast Fourier Transform

Computational complexity of *discrete Fourier transform* is not enough to fast process more data. Therefore were developed a lot of algorithms, which improve its calculation. There are algorithms, which do *discrete Fourier transform* having computational complexity $O(NlogN)$, where for large number of data is significant improvement. They use design paradigm divide and conquer. Precursors of these solutions are James William Cooley and John Tukey, who elaborated first faster algorithm to compute *DFT*, known as *fast Fourier transform* (*FFT*). It is worth to mention that *FFT* is not approximate value of *DFT*. *FFT* returns the same values but do this much faster.

2.6.8 Hanning window

Window functions are functions which are used with *DFT*. Often during process of *DFT* occur blur of spectrum, which mean that on the spectrum are seeing values for frequencies different than the real frequency of signal. To minimize this effect are used window functions. In time domain it is multiply every sample of signal by corresponding value of sample of window. The simplest window is rectangular window. Spectrum of this windowing is shown in 6

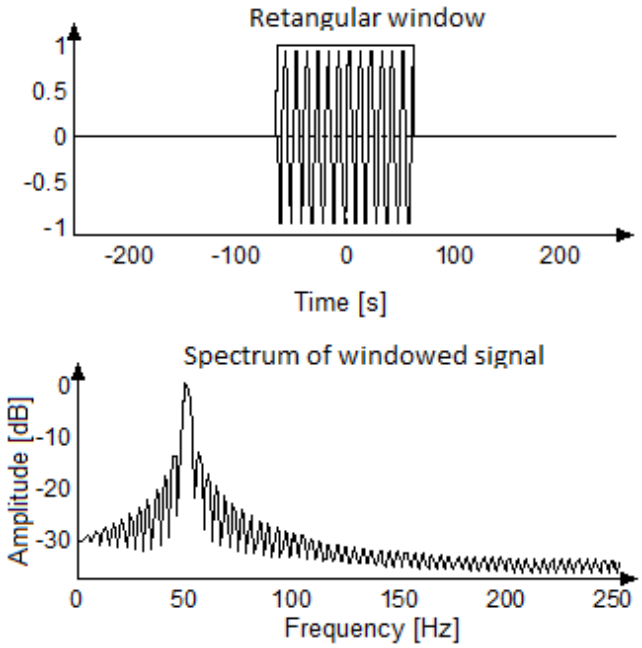


Figure 6: Rectangular window and its effect in spectrum of signal.

In our protocol is used Hanning window which is defined by:

$$w(n) = 0.5(1 - \cos\left(\frac{2\pi n}{N-1}\right)), \tag{4}$$

where N - amount of analyzed samples, n - nth sample.

As one can see at fig. 7 the pitch is wider, hence is easier to determine the correct frequency.

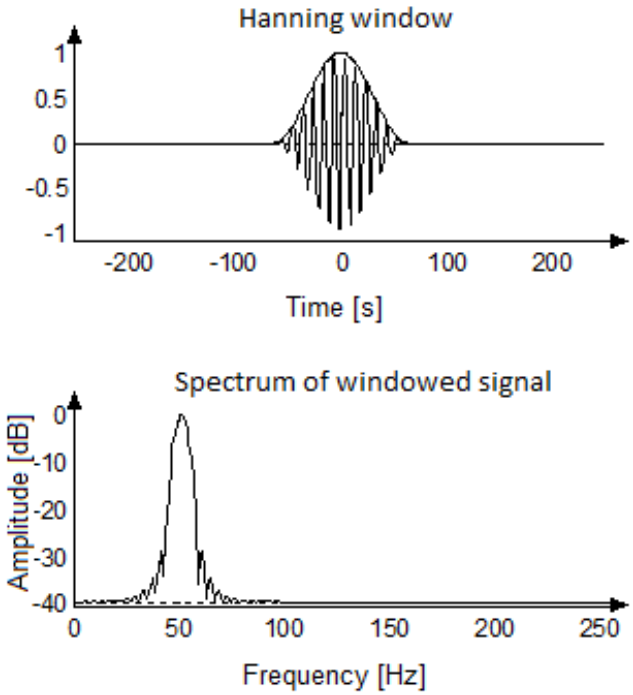


Figure 7: Hanning window and its effect in spectrum of signal.

2.6.9 Pulse-Code Modulation

Pulse-Code Modulation (*PCM*) is a method used to digitally represent sampled analog signals. *PCM* is based on representation of the instantaneous value of the signal (Sampling) at uniform intervals (sampling rate). The sampled analog data are represent by binary data. During the demodulation, sample rate should be at least two times greater that frequency of receiving analog signal (greater than the Nyquist frequency $f_s/2$). This ensure that sampled values represent analog signal without errors. In our prototype sample rate is 44,1 kHz. The highest frequency which we use in application is about 20,5 kHz so 44,1 kHz is sufficient.

2.7 Sound channel parameters

Using Diffie-Hellman key exchange and asymmetric cryptography should be use the longest key length as possible. In NIST specification SP 800-131 is mentioned that recommended key length is 3072. However, in our prototype we used 1024 key length. The reason is low speed of transferring data, which is 200b/s. Decreasing time of sending one character had influence on analyzing received data. It have been taking too long time (frequency of getting data increased, so algorithm did not follow in analyze so much data in short time).

2.8 Microphones and speakers in mobile devices

Specification of this appliances are very different and depends on devices. During the tests could be seen that microphone from more expensive SAMSUNG Galaxy Tab is better than microphone from LG Swift L9. SAMSUNG could detect higher frequency. However, microphone from LG Swift L9 was completely enough to registered sufficiently large frequencies. Speakers also are a little different, but they have no problem to generate opportunely frequencies.

2.9 Security analysis

The key exchange protocol is considered to be secured on the assumptions that underlying problems are hard e.g. factorization, discrete logarithm, computational Diffie-Hellman etc. One interesting question is what happens if those assumptions are false. It could happened if implementation of cryptography module has some error or an adversary has super efficient quantum computer that is capable of solving the above problems in reasonable time.

The adversary can have different powers over the communication channel. In the following paragraphs we will discuss different powers of the adversary and their influence over the security of the protocol.

The security analysis below does not include security proofs of AMA but the different aspects of an adversary capturing the transmission if he was able to break the aforementioned assumptions.

2.9.1 No access to protocol transcript

At first the adversary might not be able to eavesdrop the transcript of the protocol at all e.g. in environment where capturing the communication requires sufficient distance to obtain sound signal from the authenticating devices. In that case there are not a lot of requirements for the key exchange protocol being used.

This case is trivial as we can imagine a very simple game in which two parties are hidden from an adversary and he has to guess whether the parties performed a key exchange protocol. Without access to the messages exchanged between the parties, an adversary can only guess with probability $P = \frac{1}{2}$ if the parties performed the protocol. As a consequence he cannot win the "Real-or-Random" game concerning the session key in underlying CK or eCK models for the chosen AKE protocol [23, 31, 30]. Similarly, using the same reasoning, the adversary cannot guess the identities of parties mutually authenticating in the protocol.

However in practice it is possible for an adversary to partially eavesdrop the communication between devices.

2.9.2 Partial access to protocol transcript

Partial access to protocol transcript in sound channel depends on many factors. Participant's microphone, speakers and adversary's microphone have to have appropriate conditions which allow adversary to eavesdrop the transcript. For more details see section 2.9.7 about eavesdropping.

Factors described in 2.9.7 contribute to some probability $\bar{p}$ that adversary and participants will have this type of devices satisfying the conditions. Otherwise, adversary is not being able to receive all frequencies. He will gain transcript with data having no sense. So it would be similar situation when adversary has no access to the transcript.

2.9.3 Full access to protocol transcript

Imagine that an adversary uses some kind of microphone with high sensitivity so he can capture all communication between authenticating devices. As mentioned in the previous section in order to secure against eavesdropping the key exchange protocol must use difficulty to inverse problems like the discrete logarithm problem used in the Diffie-Hellman key exchange scheme.

2.9.4 Possible vectors of attack on a sound channel

There are two ways to attack the sound channel. The first one is trying to generate sound wave which can drown or change sending data by one of the authenticate parties. Additionally one can in some ways force microphone to gaining frequencies in another bandwidth. Soundproofing box or case could make it more difficult. Good solution would be mixing of the channels. For example, first phase (exchanging ephemeral keys) would be process by out-of-band channel but second phase (encrypted identity information) by commonly used radiation channel (WiFi, Bluetooth). Then adversary would have to has devices which detect two types of waves - electromagnetic and acoustic.

2.9.5 Manipulating speaker

One of the vector attack is manipulating of the speaker. Assume that adversary want to run malicious program which take control over the speaker. He can generates waves what he wants, hence he can send malicious data and encryption due to which he can learn something about the protocol. System has to ensure that during process of key exchange any kind of background application cannot interrupt this. Android ensure security in that way but do not forget about different type of viruses. Unfortunately we do not have influence on this. Manipulation from external is based on using additionally speaker which will generate sound in used bandwidth. This method could mislead of the receiving microphone. It would be easy to disturb the signal but then recorder will gain data with no sense. Attacker would has to start generate sound in perfect time with appropriate intense. Large distance could causing that would be necessary some calculation about speed of sound. High intense (which would be needed if microphone would has to receive high frequencies from distance) could cause that signal would be audible for human, hence attacker could be exposed. Inserting authenticated parties to some soundproofing box could minimize effectiveness of these methods of attacks.

2.9.6 Manipulating microphone

Manipulation of microphone would be possible only if attacker would have access to the application. He could e.g. reduce or increase bandwidth of frequencies. Adversary could totally change the bandwidth, outside of the know range for authenticated parties and at the same time generate his sound waves in chosen bandwidth.

2.9.7 Eavesdropping

Eavesdropping in sound channel depends on many factors. At first microphone has to have appropriate sensitivity level. Sensitivity level is given by:

$$L_m = 20log_{10}\left(\frac{M}{M_0}\right)[dB], \tag{5}$$

where M is sensitivity [mV/Pa] and M_0 is the 1000 mV/Pa (1 V/Pa) reference output ratio. $M = \frac{U}{p_A}$ where U is the effective voltage microphone and p_A is the sound pressure. Microphone sensitivity M depends also on frequency and the angle of incidence. The important role of receiving signal have speakers generating this signal. Sensitivity level of speaker is given by:

$$L_s = 10log_{10}\left(\frac{p_A}{p_0}\right)[dB], \tag{6}$$

where p_A is sound pressure generated by speaker powered $1V \cdot A$, p_0 is a reference pressure which has value $2 * 10^{-5}$ Pa. Frequency also has influence on sensitivity level of

speaker. The next thing which should be consider is distance microphone from speaker. Too big distance and too low sensitivity of microphone could causing that microphone will not able to gain sounds wave or at least some frequencies. Sound pressure decreases proportionally with increase of distance. So when sound level L_1 is measured at a distance r_1, the sound level L_2 at the distance r_2 is given by:

$$L_2 = L_1 + 20log_{10}\left(\frac{r_1}{r_2}\right)[dB]. \tag{7}$$

There are exist a lot of devices to eavesdropping. One of the example is microphone with high sensitivity which allow to eavesdrop the sounds even from a large distance e.g. parabolic or shotgun microphone. To gain ultrasounds for eavesdropper would be useful instrument detecting ultrasonics from bats. The next one is using laser [28]. The special laser is a transmitter and the infrared measure of surface vibration. This method also allows to gain data from a large distance.

2.9.8 Malicious implementation

An adversary could try to swap a genuine application for the key exchange on the authenticating device with a malicious one and thus gain the session key. To ensure both parties that they use genuine software, it should be signed with a certificate of a trusted authority and it should be downloaded from a secure server. This case is mentioned in security assumptions in section 3.3.

3 Prototype Design

3.1 Mobile Devices

AMA does not require high-power computing. So it is ideal to implement it on chip cards. Due to the fact the chip cards are slowly replacing by mobile phones we have decided to implement AMA on this devices. Operating system which we have chosen is Android, because it is one of the most frequently used software on mobile phones. The lowest version, for which program should work without problems is Android 4.1.2. Core of the AMA was implemented in C++. Sound channel and layout was made using Android's libraries.

3.2 Operating System

Operating system which was chosen is Android version 4.1.2 and higher. This is the newest version which was available for device which has been using in tests. Second device to tests has Android 4.2.2. This software has every required libraries whereby is possible to create sound channel between two devices (*AudioTrack* and *AudioRecord*). Additionally, Android allows to use of modules written in C++.

3.3 Security of Android

The operating system should ensure that running application cannot interact with another. Each application is run in separate process having own user ID. This sets up a kernel-level Application Sandbox, where by default application cannot interact with each other and has limited access to OS.

Furthermore, every application has to be signed by developer. Applications without signature will rejected by Google Play or the package installer on Android device. Signed certificate is verified by Package Manager after installation of application [11].

3.4 Application architecture

Application consist of three parts: cryptographic module, sound channel module and graphic interface. Cryptographic module is written in *C++* and to use it in Android, was needed to write a wrapper using *JNI* (*Java Native Interface*). *NDK* (*Native Development Kit*) is a toolset that allows to add native-code languages such as *C* and *C++* to Android's application. User interface consists initial screen where user choose which frequencies have to be use during sending data. In next screen is dynamic graph with recording sound wave and button which start the protocol. To show sound as sinusoid wave we used external library *AChartEngine* [9], the rest of interface are created by standard Android's libraries.

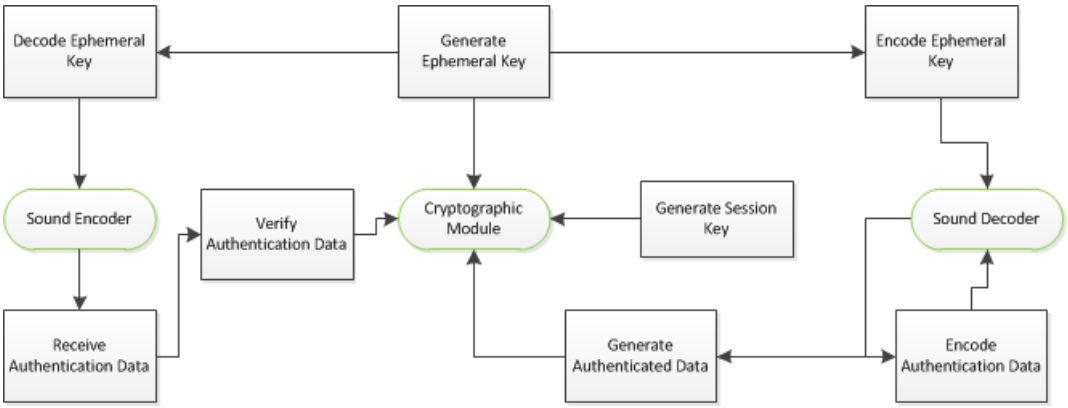


Figure 8: Application architecture.

3.5 Cryptography library

There are exist few of cryptography libraries which are written in Java (e.g. *javax.crypto*, *BouncyCastle*), but we wanted to create cryptography module which is independent of the platform. The choice fell on *CryptoPP*. It is free, open source cryptography library written

in *C++*. The library is objectives and very easy to use. Library ensures symmetric and asymmetric cryptography along with signatures and secure hash function. It is used by such companies as Microsoft or Symantec.

3.6 Design pattern

Application architecture base on Model-View-Controller pattern. As model is included whole cryptographic module, controller receive and send data to appropriate places and view is responsible for sending and receiving data associated with AMA protocol.

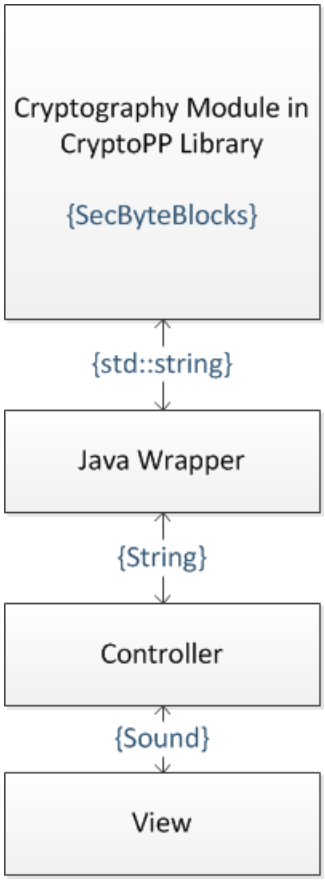


Figure 9: Design pattern.

3.7 NIST recommendation for security parameters

NIST recommends using 3072 bit length keys to ensure low probability of calculating discrete logarithm. This length is set as default in *CryptoPP*, however in our prototype we are

using 1024 bit. It is caused by duration of sending data by sound. Now, sending 1024 bit length keys take some time and for needs of presentation we decided to stay with this length. For more information please check [5].

3.8 Shared cryptographic module

Cryptographic module is designed in such way that it will be easy to use it on different platforms. Application treat it as a black box where are exchanging strings with data. Implementation is made in C++ based on *Diffie-Hellman* and *Authenticated-Diffie-Hellman* classes from CryptoPP library. It performs following cryptographic operation:

- Modular exponentiation
- Long term and ephemeral key generation (Diffie-Helman)
- Certificate generation and verification
- Secure hashing (*SHA1*)
- Symmetric encryption (*AES*)

Core of this module is class *MututalAuthenticationChip*. It is responsible for inputting, preparing and outputting data to Application. All sensitive data is kept internally using secure byte blocks. Messages generated in the protocol are converted to strings. For group arithmetic operations and modular exponentiation we use build in Integer class provided by the library.

The overall implementation tightly follows the protocol specification and the only overhead is connected with conversions to provide compatibility with the rest of the application.

```
1  void KeyGenerator::GenerateEphemeralKeyPair(CryptoPP::RandomNumberGenerator &rng,
       byte *privateKey, byte *publicKey) const
2  {
3      Integer a = Integer(rng, keySize);
4      byte *aEncoded = new byte[keySize];
5      a.Encode(aEncoded, keySize);
6      privateKey = HashClass::getSHA1(aEncoded, keySize);
7      CryptoPP::Integer exponent(privateKey, keySize);
8      CryptoPP::Integer cA = a_exp_b_mod_c(g, exponent, p);   //ca = g^H(a)
9      cA.Encode(publicKey, keySize);
10 }
```

Listing 1: Example of CryptoPP usage for ephemeral key generation.

```
1  byte * HashClass::getSHA1(const byte * input, int length)
2  {
3    CryptoPP::SHA1 sha;
4    byte *digest = new byte [CryptoPP::SHA1::DIGESTSIZE];
5    sha.CalculateDigest(digest, input, length);
```

```
6   return digest;
7 }
```

Listing 2: Example of CryptoPP usage for hash function.

3.9 Java Native Interface - C++ Wrapper

To be able to use cryptographic part written in *C++* in Android, we had to create *C++* wrappers using *JNI* (*Java Native Interface*) [2]. Main class which communicate with *C++* code has native methods, for which, after appropriate compilation, was created *MACWrapper.h* where definition of methods looks like below.

```
1 /*
2  * Class:     com_example_androidake_MutualAuthenticateChip
3  * Method:    prepareMACCPP
4  * Signature: (Z)V
5  */
6 JNIEXPORT void JNICALL
      Java_com_example_androidake_MutualAuthenticateChip_prepareMACCPP
7   (JNIEnv *, jobject, jboolean);
```

Listing 3: Example of JNI usage for method definition.

Implementations of methods are in *MACWrapper.cpp*. There are made conversion from *C++* data type to Java data type and vice versa.

```
1 JNIEXPORT void JNICALL
      Java_com_example_androidake_MutualAuthenticateChip_prepareMACCPP
2 (JNIEnv *env, jobject thisObj, jboolean jinit) {
3   bool init = jinit;
4   if(init == true) {
5     mac = new MutualAuthenticationChip(init);
6   } else {
7     mac_B = new MutualAuthenticationChip(init);
8   }
9 };
```

Listing 4: Example of JNI usage for method implementation.

3.10 Base64

One of the biggest problem was decided in what representation data should be sending. First choice was standard binary representation with 0 and 1. During creation of sound channel we have found that sending one ephemeral key, which has 1024 bits will take too long. So we decided to reduce sending data using *Base64* conversion. Six bits are transform to one of the 64 defined signs. Hence, instead of sending 1024 sings we send just about 172. Data which come from cryptographic module are in Hexadecimal representation. To converse we have used library created by Robert Harder [15].

Character	Frequency [kHz]
A	10
B	10,15
C	10,3
⋮	⋮
/	19,25
,	19,4
.	19,55

Table 2: Mapping characters to frequencies.

```
1  public static String fromHexToBase64(byte[] hex_byte) {
2    String hex_str = ConverterJava.ByteToString(hex_byte);
3    byte[] decodedHex = null;
4    try {
5      decodedHex = Hex.decodeHex(hex_str.substring(0,
6          hex_str.length() - 1).toCharArray());
7    } catch (DecoderException e) {
8      e.printStackTrace();
9    }
10   String encodedHexB64 = Base64.encodeBytes(decodedHex);
11   return encodedHexB64;
12 }
```

Listing 5: Example of code converting from Hex String to Base64 String

3.11 Sound channel

Sound channel was one of the most difficult part during creation of application. One of the reason was inaccessibility of library whereby we could transform received sound data to cryptographic data. Whole process of analysis received data had to be created from the scratch. At beginning we had doubt about sensitivity and precision of speaker and microphone in mobile devices, especially when we wanted to use ultrasonic waves which could be hard to interpreter. Fortunately, doubts were dispelled after first testing of this channel. Results of working will be describe in 4 section. Here we focus on principle of operation.

3.11.1 Generating sound wave

Class responsible for generating sound wave get data presented in Base64 (Check section with Base64). Every character A-Z, a-z, 0-9, + and / is mapped to one of the 64 frequencies.

Additional two characters "," and "." represent begin of data and end of data. Sample rate is 44,1 kHz. Every character is sending by 30 ms (this is the lowest value which we

achieve during optimization of prototype), which means that for each pitch is needed 1323 samples. To create sinus wave each sample for $i \in (0, 1322)$ is calculating by formula:

$$F(i) = \sin\left(\frac{f \cdot \pi \cdot 2 \cdot i}{44100}\right)$$

where f is the target frequency. These samples are processed by *AudioTrack* library to sound using *PCM* encoding. In the listing is shown peace of code preparing sinus waves in appropriate frequencies.

```
/* --------- Constants and variable used in code -------------------------*/
// Constants.DEFAULT_NUM_SAMPLES = 1323; - Number of samples needed to
//                        encode one character.
// Constants.FREQUENCIES - all 66 used frequencies;
// Constants.DEFAULT_BUFFER_SIZE = 8192;
// Constants.RAMP = 30; - on the edge of buffor with samples is needed
//              to decrease the amplitude to avoid scratching
// Input:
// List<Integer> inofsign; - signs which will be converted on frequencies
/*-----------------------------------------------------------------------*/

public ArrayList<Buffer> encodesDataToBuffers(List<Integer> inofsign) {

  ArrayList<Buffer> queue_with_data_AL = new ArrayList<Buffer>();
  for (int index : inofsign) {

    int totalCount = Constants.DEFAULT_NUM_SAMPLES;
    double per = (double) (( Constants.FREQUENCIES[index] * 2 * Math.PI )  /
        Constants.SAMPLING);
    double d = 0;

    int mFilledSize = 0;
    Buffer buffer = new Buffer();
    int ramp = totalCount / Constants.RAMP;
    short[] buffer_data = new short[Constants.DEFAULT_BUFFER_SIZE];

    for(int i = 0; i < totalCount ; ++i){

      if(mFilledSize >= Constants.DEFAULT_BUFFER_SIZE - 1){
        buffer.setBufferShort(buffer_data);
        buffer.setSize(mFilledSize);
        mFilledSize = 0;
        queue_with_data_AL.add(buffer);
        buffer = new Buffer();
        buffer_data = new short[Constants.DEFAULT_BUFFER_SIZE];
      }
      double out = (double) Math.sin(d);
      final short val;
      if(i < ramp){
        val = (short) ((out * Short.MAX_VALUE * i / ramp));
      }else if(i < totalCount - ramp){
        val = (short) ((out * Short.MAX_VALUE));
      }else{
        val = (short) ((out * Short.MAX_VALUE * (totalCount-i)/ramp));
      }

      buffer_data[mFilledSize++] = (short) ( val );
```

```
47          d+=per;
48
49        }
50
51
52        buffer.setBufferShort(buffer_data);
53        buffer.setSize(mFilledSize);
54        queue_with_data_AL.add(buffer);
55
56        mFilledSize = 0;
57
58      }
59      return queue_with_data_AL;
60    }
```

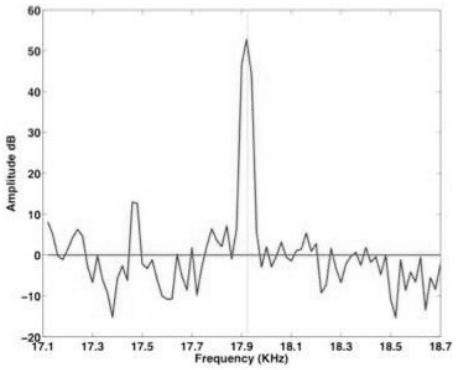
Listing 6: Method converting characters to sinus waves.

3.11.2 Receiving sound

Retrieving sound is more complicated than sending. In separate process, using *AudioRecord* library, in every 15 milliseconds (if sending of one character takes 30 ms), are taken samples and then, as one buffer, are appended to array of buffers. Separate process, in the same time, takes first buffer from array and analyses it.

At this moment, the samples are as short integer values which represent sound in time domain (sinusoid). To check in which frequency was received sound is needed to convert the sampled function from time domain to the frequency domain. We used to this *discrete Fourier transform* . Library which we have chosen to run *DFT* is [3].

Before change of domain, we need to window samples, using Hanning windowing. Thereby, chart in frequency domain is more readable and is easier to find the place with the highest pitch. Windowed samples are put into *DFT* method and on the output we have result in frequency domain. The effect on the graph looks like at 3.11.2.



Now, algorithm tries to find the highest place in the output. The reason of getting data in two times shorter time than sending one character is that, in one buffer can be two frequencies. In the worst case, algorithm can find one frequency in three buffers in a row, even if in first and third buffer was also another frequency, but because of lower sound intensity or less samples representing this frequency, would not be found as a highest pitch. Dividing time of receiving by two ensure that always, at least once, expected frequency will be registered. If he finds pitch with amplitude higher than 50, for frequency lower than 14 kHz (this value was chosen after testing of prototype, these frequencies always exceed this limit), or 10 for higher than 14 kHz (this different follows from the smaller volume for the higher frequency by hardware limitation) is checked whether caught frequency is in our dictionary (with margin of error + - 50 Hz).

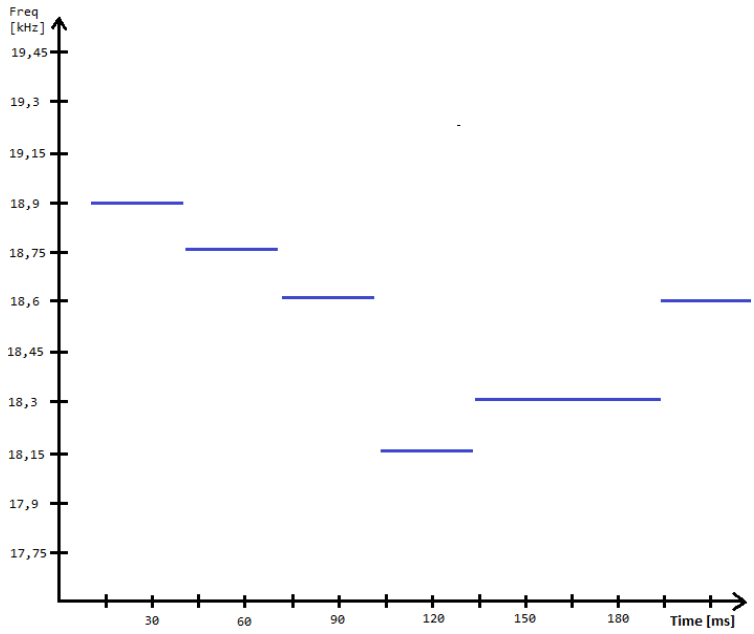


Figure 10: Figure shows received frequencies in time.

Caught value is put to the memory and wait for the next result. If current frequency is different than before we change the previous frequency to one of the 64 signs. Otherwise we increment the counter. If counter gain value three (maximal number of occurrences on received character in three further buffers), counter is reset and return adequate character.

```
1 /* --------- Constants and variable used in code -------------------------*/
2 // Constants.AMPLITUDE_LT_14K = 50; - Limit of amplitude for frequencies
3                         lower than 14 kHz
4 // Constants.AMPLITUDE_GT_14K = 10; -  Limit of amplitude for frequencies
5                       greater than 14 kHz
6 // Constants.SAMPLING = 44,1 kHz - sampling rate of sound
```

```
// Constants.K14 = 2660 - place in fft array when starts frequencies greater than
    14 kHz.
// Constants.START_ST = 1800 - place from which algorithm must start to searching
    peak.
// Input:
// FFT fft; - data prepared by Discrete Fourier Transform.
/*-------------------------------------------------------------------------*/
private int findPitch(FFT fft) {
  float max_band = 0;
  int max_peak = 0;
  int k14 = Constants.K14;
  for (int i = Constants.START_ST; i < fft.specSize(); i++) {
    if (i < k14) {
      if (fft.getBand(i) > Constants.AMPLITUDE_LT_14K)
        if (fft.getBand(i) > max_band) {
          max_peak = i;
          max_band = fft.getBand(i);
        }
    } else {
      if (fft.getBand(i) > Constants.AMPLITUDE_GT_14K)
        if (fft.getBand(i) > max_band) {
          max_peak = i;
          max_band = fft.getBand(i);
        }
    }
  }
  int frequency = max_peak * (Constants.SAMPLING / 2) / fft.specSize();
  return frequency;

}
```

Listing 7: Method finding frequency with highest aplitude.

```
/* --------- Constants and variable used in code -------------------------*/
// Constants.FREQUENCIES - all 66 used frequencies;
// Constants.BORDER = 50 - error range
Input:
int freq; - Found frequncy by method findPitch(FFT)

/*-------------------------------------------------------------------------*/

private void checkPitch(int freq) {
  String sign = "";
  if (freq > Constants.FREQUENCIES[0] - 100) {
    if (current_freq != null) {
      if (current_freq.getFrequency() - Constants.BORDER < freq
      && current_freq.getFrequency() + Constants.BORDER > freq) {
        current_freq.increaseCount();
        sign = current_freq.foundAndReturnChar();
        if (!sign.equals(Constants.NOEND_STR)) {
          vrs.onRecognition(String.valueOf(sign));
        }
      } else {
        sign = current_freq.returnChar();
        if (!sign.equals(Constants.NOEND_STR)) {
          vrs.onRecognition(String.valueOf(sign));
        }
        current_freq = new FrequencyTime();
```

```
26            current_freq.setFrequency(freq);
27          }
28        } else {
29          current_freq = new FrequencyTime();
30          current_freq.setFrequency(freq);
31        }
32      } else {
33        if (current_freq != null) {
34          sign = current_freq.returnChar();
35        if (!sign.equals(Constants.NOEND_STR)) {
36          vrs.onRecognition(String.valueOf(sign));
37        }
38        current_freq = null;
39        }
40      }
41    }
```

Listing 8: Method changing founded freqency on appropriate character.

4 Tests

To all tests was used two devices: LG Swift L9 with system Android 4.1.2 and tablet Samsung GALAXY Tab 2 with Android 4.2.2.

During tests we have focused on below factors:

- Distance between devices,
- Position of devices terms of placement of speaker and microphone,
- Distance between chosen frequencies,
- Working during sounds in the background,
- Volume of sound.

Firstly, we have tested distance between chosen frequencies. Every character from *Base64* coding is mapped to appropriate frequency. The smaller distance then is possible to use higher frequencies. We have choose three spaces:

Distance [Hz]	The lowest frequency [kHz]	The highest frequency [kHz]
150	10,5	20,25
130	11,8	20,25
100	13,8	20,3
100	14	20,5
100	14,3	20,8

Table 3: Mapping characters to frequencies.

We did not want to go below 10 kHz and higher frequency than 20,5 is hard to detect for mobile phone (especially for LG). Distance 100 Hz is the smallest to test because of implemented algorithm (margin of error is + - 50 Hz). During the tests, proved to be that for almost every test cases, protocol was successful in 80%. Except the last one which always end by failure. Character which represent end of sending data has always the highest frequency. In the last case it was 20,8 kHz. Only LG Swift cannot registry that frequency - Samsung dealt with it flawlessly. Duration of protocol is about 34,5 seconds.

There are several reason why the result was not 100%. Sometimes, if we send, for example, frequency 20 kHz then second device receive it as 19,9. It can be caused by some distortion on the microphone or even on the speaker. What happened was that the speaker make sound different than usual and then was needed restart of device. Sometimes it is sufficient that one of the device froze on some milliseconds and then receiving device lose some characters.

Measure of allowed distance between devices we have started with stick microphone from LG with speaker from Samsung. This gave us the best results. Received amplitude was the biggest, hence easier, for algorithm, was to find interesting for us frequency. Then we increased distance to 2 cm. In this case results were also good. Sessions keys were established almost every time. Problem was started increasing distance to 6 cm. Higher frequencies were not registered by microphone. Placing the devices parallel the protocol always end by failure. The LG was not able to gain all frequencies in such way. Samsung again has been receiving data without problems.

The next test rely on checking influence of background sound. As we know, from the previous test, the best results were when devices were close to each other (maximal 2 cm from each other), so we have tested it in such position. At beginning we have checked it at the University during the break between lectures. A lot of students was talking and laughing but even this, did not interfere with protocol. Almost every attempt was ended with success. Then we have used third device which was generating constantly chosen frequency. We have put this device very close to the participants. First disturbing frequency was 8 kHz. Because of that bandwidth is from 10,5 to 20,25 we have suspected that 8 kHz will not cause the problem and we was right. Algorithm analyzing data has not mistaken. The problem was just after increasing disturbing frequency above 10,5 kHz, but was needed to stick speaker of this device to microphone of protocol participant. Otherwise, the protocol always ends with success.

Volume set on 70% on both devices was sufficient to run the protocol. Set this value below 70% cause that character denoting start of transmission was not detected.

5 Possible development of the project

Lets imagine that Alice and Bob want to establish secure channel being far away from each other. They can authenticate each other using telephone line. It is possible to run application during the calling but the microphone receiving the signal is unavailable in the application. So to this experiment would be needed second pair of mobile phones which will be used as intermediary. Each intermediary phone should be set as handsfree because of loudness of sound. Authenticated phones should be apply as close as possible to intermediary phone. Now when intermediary phones established the connection, the authenticated devices can start the protocol, the sound signal will be transferring by telephone line. Device on the other side will be receiving the signal. There is possible use computers as intermediary. There are a lot of voice communicator which would be useful.

Disadvantage in our protocol is speed of transferring data. In every 15 milliseconds are received samples which are put to the queue. Another process gets these samples and analyze them. For our test devices, receiving data in every 15 second was the most optimal method. Algorithm analyzing this data follow with computing, so in queue is not accumulate to much data. Decreasing period receiving data to 10 millisecond cause that decoder of sound did not keep up and outcome was sometimes couple of seconds after the last received character. To improve that is worth to considering client-server model like in [13]. Mobile phone as a client gathering the data and send to the external server. Server with much more computing power could decode the data definitely faster and returns the result to the client. Server could do much more complex computing, hence the results would be with better precision.

For now the sound channel use bandwidth from 10,5 kHz to 20,25 kHz. For the reason please check section with *Base64* conversion. These frequencies (especially below 19 kHz) could be audible for the most of the people (it sounds like squeaking). If we would want to exchange data using sound which is inaudible, we could use just simple binary representation. Sending data using 0 and 1 would be needed only two frequencies, so taking the highest as possible (for example 19,9 kHz and 20,10) would ensure that exchanging of data would be inaudible for the most people. Of course at the expense of length of sending data. It would be achievable also with hexadecimal representation. Would be needed to use bigger bandwidth than from binary representation but the length of data would be smaller.

6 Summary

In the dissertation is showed that is possible to use sound channel to establish session key. There is no need Internet connection or another electromagnetic channels, so proposed protocol Anonymous Mutual Authenticated could be called even on the place without reach,

using mobile devices with embedded speaker and microphone.

In the prototype we used the bandwidth of frequencies which are audible for human, however it is possible to use only these frequencies which are not detected by humans ears. Our mobile devices, which are medium priced products did not have problem to send and receive frequency between 19-20,5 kHz which are inaudible for the most people.

There is showed that using this type of channel could surprise the attacker which could not be preparing to eavesdrop/manipulate exchanging data in acoustic waves. Mixing the sound channel with another kind of channels, for sure, could make it even more difficult. Duration of the protocol execution is not so fast as in radiation channels, however it is good enough to be applied in practical solution. Establishing the authentication could be much faster if would be use client-server model. Computer as a server would analyze data faster and results would be better.

References

[1] The descrete transform fourier (dft): Definition and numerical examples. http://www.nbtwiki.net/doku.php?id=tutorial:the_discrete_fourier_transformation_dft#.UND9vuT6Jv8.

[2] Java programming tutorial - java native interface (jni). https://www3.ntu.edu.sg/home/ehchua/programming/java/JavaNativeInterface.html.

[3] Minim fft class. http://code.compartmental.net/minim/javadoc/ddf/minim/analysis/FFT.html.

[4] Natężenie fali dźwiękowej. http://www.fizykon.org/akustyka/akustyka_natezenie_fali.htm.

[5] Nist sp 800-131 recommendation for key length. http://csrc.nist.gov/publications/nistpubs/800-131A/sp800-131A.pdf.

[6] Poziom natężenia dźwieku. http://www.fizykon.org/akustyka/akustyka_poziom_natezenia.htm.

[7] The speed of sound. http://www.sengpielaudio.com/calculator-speedsound.htm.

[8] Vibrations usage in out-of-band channel.

[9] AChartEngine. http://www.achartengine.org/.

[10] Scott A. Vanstone Alfred J. Menezes, Paul C. van Oorschot. *Handbook of Applied Cryptography*. CRC Press.

[11] Android. `https://source.android.com/devices/tech/security/`.

[12] Anton Mityagin Brian LaMacchia, Kristin Lauter. *Stronger Security of Authenticated Key Exchange*. Microsoft Research, 1 Microsoft Way, Redmond, WA.

[13] Chirp. `http://chirp.io/`.

[14] Lucjan Hanzlik, Kamil Kluczniak, Łukasz Krzywiecki, and Mirosław Kutyłowski. *Mutual Chip Authentication*. Faculty of Fundamental Problems of Technology, Wrocław University of Technology.

[15] Robert Harder. Base64. `http://iharder.sourceforge.net/current/java/base64/`.

[16] Steve Haynal and Heidi Haynal. *Generating and Searching Families of FFT Algorithms*. `http://jsat.ewi.tudelft.nl/content/volume7/JSAT7_13_Haynal.pdf`.

[17] M.K. Reiter J.M McCune, A. Perrig. *Seeing-is-believing: using camera phones for human-verifiable authentication*. Carnegie Mellon Univ., Pittsburgh, PA, USA.

[18] Hugo Krawczyk. *HMQV: A High-Performance Secure Diffie-Hellman Protocol*. IBM T.J.Watson Research Center, Yorktown Heights, NY.

[19] Hugo Krawczyk. *SIGMA the 'SIGn-and-MAc' Approach to Authenticated Diffie-Hellman and its Use in the IKE Protocols*. EE Department, Technion, Haifa, Israel, and IBM T.J. Watson Research Center.

[20] Anton Mityagin Kristin Lauter. *Security Analysis of KEA Authenticated Key Exchange Protocol*.

[21] Joshua D. Guttman Wenjing Lou Kui Ren Ming Li, Shucheng Yu. *Secure Ad Hoc Trust Initialization and Key Management in Wireless Body Area Networks*.

[22] Pawel Nuzka. *Authentication and communication protocol for mobile devices in optical channel*. Faculty of Fundamental Problems of Technology, Wroclaw University of Technology.

[23] Hugo Krawczyk Ran Canetti. *Analysis of key-exchange protocols and their use for building secure channels. Advances in Cryptology*. EUROCRYPT, 2001.

[24] M. Merrit S. Bellovin. *Encrypted key exchange: Password-based protocols secure against dictionary attacks.* In Proceedings of the IEEE Symposium on Security and Privacy, 1992.

[25] M. Merrit S. Bellovin. *Augmented encrypted key exchange: a password-based protocol secure against dictionary atttacks and password file compromise*. In Proceedings of the ACM Conference on Computer and Communications Security (CCS), 1993.

[26] Kazuhisa Miyashita Akeharu Okumura Yoshiaki Yoshida Shintaro Takeda, Ikuharu Morioka and Kenji Matsumoto. *Age variation in the upper limit of hearing*.

[27] Nisha Panwar Michael Segal Shlomi Dolev, Łukasz Krzywiecki. *Certificating Vehicle Public Key with Vehicle Attributes*.

[28] Mike Szczys. Laser mic makes eavesdropping remarkably simple. `http://hackaday.com/2010/09/25/laser-mic-makes-eavesdropping-remarkably-simple/`.

[29] Berkant Ustaoglu. *Obtaining a secure and efficient key agreement protocol from (H)MQV and NAXOS*.

[30] SaanJae Moon Xinghua Li, Jianfeng Ma. *On the security of Cenetti-Krawczyk model*.

[31] Zheng Yang. *Efficient eCK-secure Authenticated Key Exchange Protocols in the Standard Model*.