WYDZIAŁ PODSTAWOWYCH PROBLEMÓW TECHNIKI
POLITECHNIKI WROCŁAWSKIEJ

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. Łukasza Krzywieckiego

WROCŁAW 2010

Contents

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 just using sound.

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. Moreover, the sound waves with appropriate, not to high sound intensity, does not penetrate through the walls. It causing that waves cannot be receive e.g. on the street when some protected data are sending in the house. Furthermore receiver as like sender do not need to known each other (unlike Bluetooth) before starting the communication. Sender just start to transmit data and receiver in the same time listen broadcast.

To authentication and key agreement between mobile devices was chosen Anonymous Mutual Authentication (AMA) protocol, created by L. Hanzlik, K. Klucznik, Ł. Krzywiecki and M. Kutyłowski. AMA is symmetric, which means participants execute the same code. One of the advantage is simple to implemented, even on the devices with low power computing. It was originally designed for authenticating electronic documents, it proved to be portable Every key and encryption are sending by audio channel.

Application created on Android system will serve as proof of concept. App is called AKEBySound. Managed to determine that exist this kind of applications but none of theme are use to establishing session keys and they are not open source. To use our app is enough to have mobile phone/smartphone with software Android (version 4.1.2 or higher), working speaker and microphone, of course.

Dissertation is divided on X parts. The first section focus on analysis of the problem. Explain principal issues used to create the application. Due to the fact protocol is executed by audio channel the first subsection is about sound. The next one is one

of the method to exchange keys on which AMA protocol base - Diffie-Hellman Key exchange. Tutaj cos o bezpieczenstwie bedzie itp. Dowod poprawnosci protokolu i takie tam.

Next chapter is about System design. In subsections are described assumptions of project and used cases. Then is shown diagram classes

1   Analysis of the problem

1.1   Sound

Sound is acoustic wave propagating in different substances such as water, air (so called vibrating wire). These waves are causing auditory sensation and these which in appropriate amplitude and frequency are not detected by human organ of hearing. Sound to spread has 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.
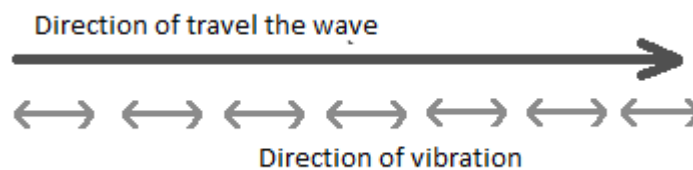
1.1.1   Frequency

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.
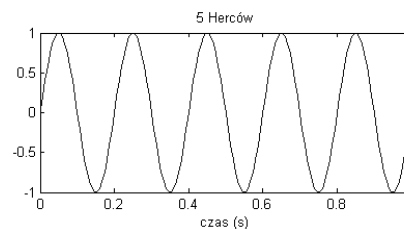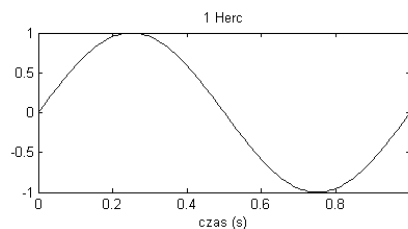
Exist three division of sound as to frequency:

- Infrasound - frequency is lower than 16 Hz.

- Hearable sound - frequency is greater than 16 Hz and lower than 20 kHz.

- Ultrasound - frequency is greater than 20 kHz.

- Hipersound - frequency is greater than 10 GHz.

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

1.1.2   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, don't cause that human hear it two times louder. Human hears its logarithmized value. Logarithmized measure of sound intensity is called sound intensity level. Is measured in decibels and is given by [6]:

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

where:

- L - Sound intensity level.

- I - Sound intensity.

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

1.1.3   Speed of sound

Speed of sound depends on center through which wave passes. Influence on speed has also temperature of the center. The slowest moves in the flue. In the air with 20° 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 spread in solid. For example in steel reach about 5960 m/s (21460 km/h) [7].

1.1.4 Ultrasounds

The present app largely, to sending data, uses ultrasound. Ultrasound 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 [10]. 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.

1.2 Fourier transform

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

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

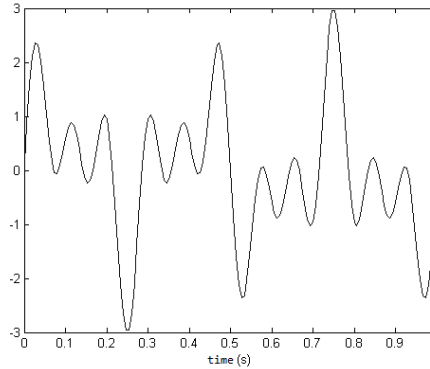Sum of three functions with frequencies 3m 7m and 11 Hz on a graph:

Figure 1: $f(x) = \sin(2\pi \cdot 3t) + \sin(2\pi \cdot 7t) + \sin(2\pi \cdot 11t)$, where t is current time.

Passing function $f(x)$ from figure 1 through Fourier transform we will get the following graphs:
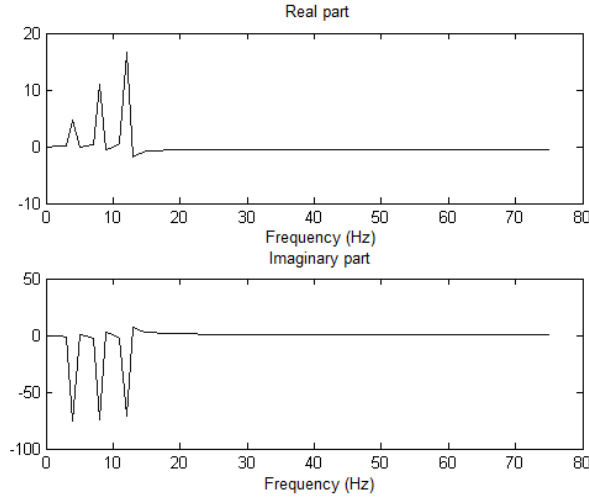
Figure 2: On the top real part results of Fourier transform, on the bottom imaginary part.

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

1.2.1 Discrete Fourier Transform

Application analyzes periodically finite number of samples of sound. To converse to frequency domain is used Discrete Fourier transform (DFT). Input 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]:

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 [9].
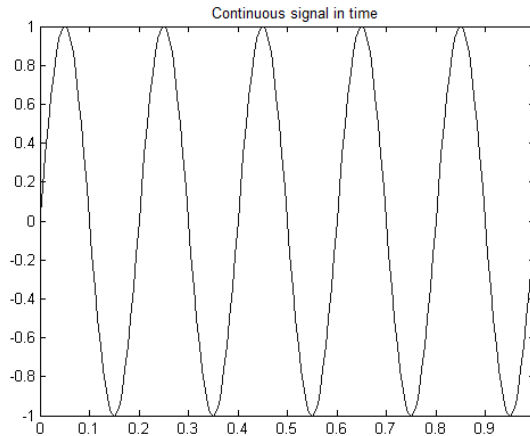
Figure 3: Continuous signal in duration 1 second.

1.2.2 Fast Fourier Transform

Computational complexity of discrete Fourier transform is not enough to fast process more data. Therefore developed a lot of algorithms, which improve its calculation. Exist 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 mentioning that FFT is not approximate value of DFT. FFT returns the same values but do this much faster.

1.3 Diffie-Hellman Key exchange

1.4 Random Oracle Model

2 System design

2.1 Assumptions

System should:

- Works on every middle class mobile devices with system Android.

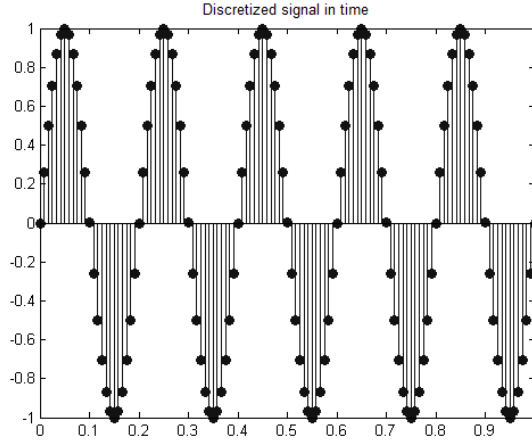- Generate keys and encryption during authorization process.

Figure 4: Discretized signal in constant time period. In this case in 1 second are 121 samples.

- Use sound wave to sending encoded keys and encryptions to another party to establish authorization.

- Receiving sound wave and decode received data to keys and encryption from another party to establish authorization.

2.2   Activity diagram

2.3   Anonymous Mutual Authentication protocol - Description

Scheme of protocol is shown at the 5. In scheme are use the following notation:

- *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).

Protocol is very simple to implement. In the first state, parties (Alice and Bob) generate their private key - x_A (respectively, x_B) and public key $y_A = g^{x_A}$ (respectively, y_A). Then starts main procedure. Parties generate ephemeral keys. Private is $h_A := H(a)$ (where a is a random number) (respectively h_B, b) and public ephemeral

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
$h_A := H(a)$		$h_B := H(b)$
$c_A := g^{h_A}$	$\xrightarrow{\quad c_A \quad}$	$c_B := g^{h_B}$
	$\xleftarrow{\quad c_B \quad}$	
$K := c_B^{h_A}$		$K := c_A^{h_B}$
$K_A := H(K,1),\ K_B := H(K,2)$		$K_A := H(K,1),\ K_B := H(K,2)$
$K'_A := H(K,3),\ K'_B := H(K,4)$		$K'_A := H(K,3),\ K'_B := H(K,4)$
$r_A := H(c_B^{x_A}, K'_A)$		
	$\xrightarrow{Enc_{K_A}(cert_A, r_A)}$	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 5: Protocol description - Anonymous Mutual Authentication. Figure from source [?].

key $c_A := g^{h_A}$ (respectively c_B). After this parties exchange between themselves public ephemeral key. This is standard Diffie-Hellman key agreement.

When parties obtain ephemeral public key of another party, start compute master key K. Four different one-time keys are establish by hashing K and number parameter - different for each one-time key.

Authenticated party is raising ephemeral public key c_B (respectively, c_A) to power of private key x_B (respectively, x_A). On this base authentication. Verifier compute the same value without private key but with the discrete logarithm of c_A (respectively, c_B) $((y_A)^{(h_B)})$

2.3.1 Security model assumption

3 Prototype Design

3.1 Mobile Devices

AMA attempts to be simple in implement and 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 the 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.

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 app. 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, the rest of interface are created by standard Android's libraries.

Figure 3.

3.5 Cryptography library

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. Ensures symmetric and asymmetric cryptography along with signatures and secure hash function. Is used by such companies as Microsoft or Symantec.

3.6 Activity diagram

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 application model

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 method, was created MACWrapper.h where definition of methods looks like below.

```
/*
 * Class:     com_example_androidake_MutualAuthenticateChip
 * Method:    prepareMACCPP
 * Signature: (Z)V
 */
JNIEXPORT void JNICALL
    Java_com_example_androidake_MutualAuthenticateChip_prepareMACCPP
  (JNIEnv *, jobject, jboolean);
```

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

```
JNIEXPORT void JNICALL
    Java_com_example_androidake_MutualAuthenticateChip_prepareMACCPP
(JNIEnv *env, jobject thisObj, jboolean jinit) {
  bool init = jinit;
  if(init == true) {
    mac = new MutualAuthenticationChip(init);
```

```
6    } else {
7      mac_B = new MutualAuthenticationChip(init);
8    }
9  };
```

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 [8].

```
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 1: 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 beginning. 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 Test 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.

A = 10 kHz,

B = 10,15 kHz,

C = 10,3 kHz,

...

/ = 19,55 kHz

Additional two characters ¸ änd ¨ represent begin of data and end of data. Sample rate is 44,1 kHz. Every character is sending during 30 ms, 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:

$$\sin\left(\frac{Frequency * PI * 2 * i}{44100} \right)$$

These samples are processed by AudioTrack library to sound using PCM encoding.

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 Transform Fourier. 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
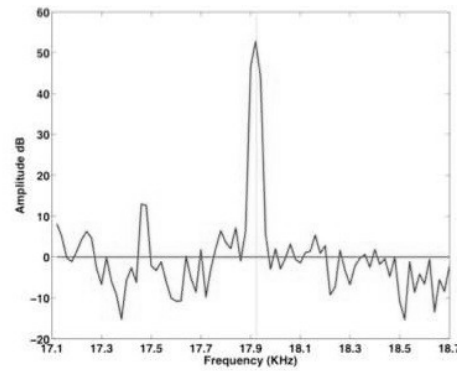
because of lower sound intensity or less samples representing this frequency, would not be found as a highest pitch. Dividing time of receiving by two ensure that always, at least once, expected frequency will be registered. If he finds pitch with amplitude higher than 50, for frequency lower than 14 kHz, 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).
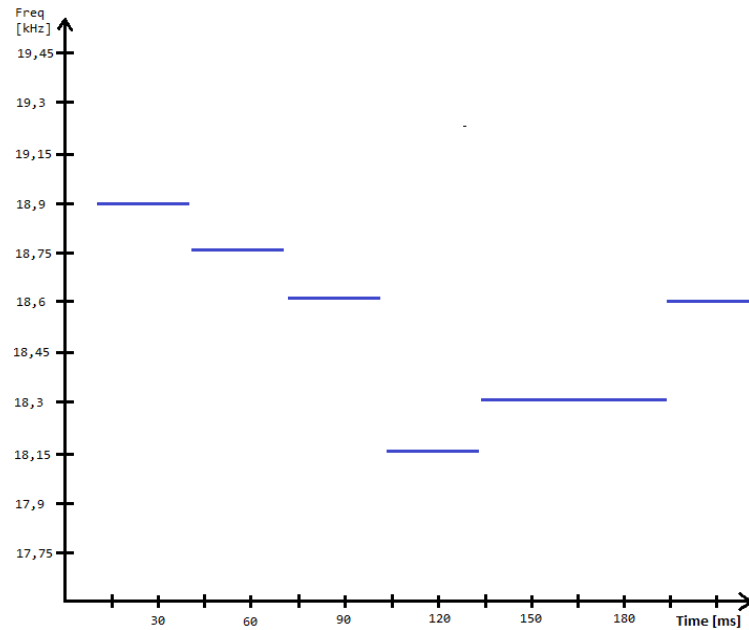

Figure 6: Figure show received frequencies in time.

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.

4 Tests

5 Installing and implementation

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] NatÄ™ĹĹ'enie fali dĹşwiÄ™kowej. `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 natÄ™ĹĹ'enia dĹşwiÄ™ku. `http://www.fizykon.org/akustyka/akustyka_poziom_natezenia.htm`.

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

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

[9] Steve Haynal and Heidi Haynal. Generating and searching families of fft algorithms. `http://jsat.ewi.tudelft.nl/content/volume7/JSAT7_13_Haynal.pdf`.

[10] Shintaro Takeda, Ikuharu Morioka, Kazuhisa Miyashita, Akeharu Okumura, Yoshiaki Yoshida, and Kenji Matsumoto. Age variation in the upper limit of hearing. `http://link.springer.com/content/pdf/10.1007%2FBF00243505`.