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 analyse the usability of channels alternative to WiFi in Authenticated Key Establishment (AKE) protocols for mobile devices. Authentication and communication protocol in audio channel for mobile devices.

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 devices 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 listening broadcast.

To authentication and key agreement between mobile devices was choosen Anonymous Mutual Authentication (AMA) protocol, created by L. Hanzlik, K. Klucznik, Ł. Krzywiecki and M. Kutyłowski. AMA is simmetric, which means participants execute the same code. One of the advantage is simple to implemented, even on the devices with low power computing. Every key and encryption are sending by audio channel.

Application created on Android system will serve as proof of concept. Application is called AKEBySound. Managed to determine that exist this kind of applications but none of theme are use to authentication and key exchange protocol. To use this app is enough to have mobile phone/smartform with software Android (version 4.1.2 or higher), working speaker and microphone, of course.

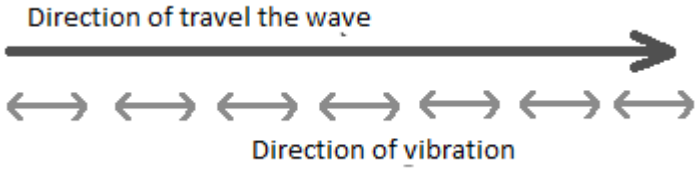
Dissertation is divided on X parts. The first section focus on analysis of the problem. Explain principal issues used to create the application. Due to the fact protocol is executed by audio channel the first subsection is about sound. The next one is one of the method to exchange keys on which AMA protocol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

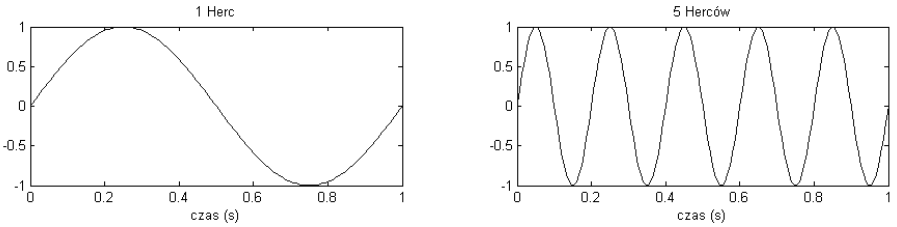
1.1 Sound

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.

1.2 Diffie-Hellman Key exchange

1.3 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.
- 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 Fig. 1. In scheme are use the fallowing 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 $y_A = g^{x_A}$ - public key $cert_A$ - certificate for y_A		x_B - private key $y_B = g^{x_B}$ - public key $cert_B$ - certificate for y_B
	MAIN PROCEDURE	
choose a at random $h_A := H(a)$ $c_A := g^{h_A}$	$\xrightarrow{c_A}$ $\xleftarrow{c_B}$	choose b at random $h_B := H(b)$ $c_B := g^{h_B}$
$K := {c_B}^{h_A}$ $K_A := H(K,1)$, $K_B := H(K,2)$ $K'_A := H(K,3)$, $K'_B := H(K,4)$ $r_A := H(c_B^{x_A}, K'_A)$		$K := {c_A}^{h_B}$ $K_A := H(K,1)$, $K_B := H(K,2)$ $K'_A := H(K,3)$, $K'_B := H(K,4)$
	$\xrightarrow{Enc_{K_A}(cert_A, r_A)}$	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 [**?**].

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 [3].

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) [1]. 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.

```
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);
```

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   };
```

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 [4].

```
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 ,ändrepresent 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.

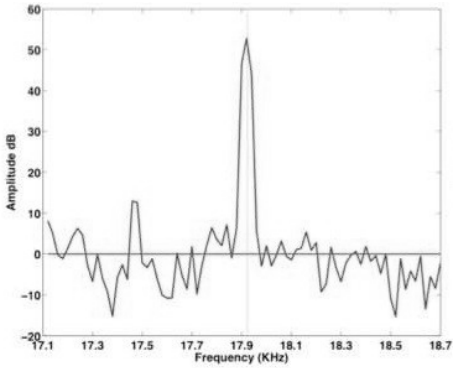
3.11.2 Receiving sound

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 [2].

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).

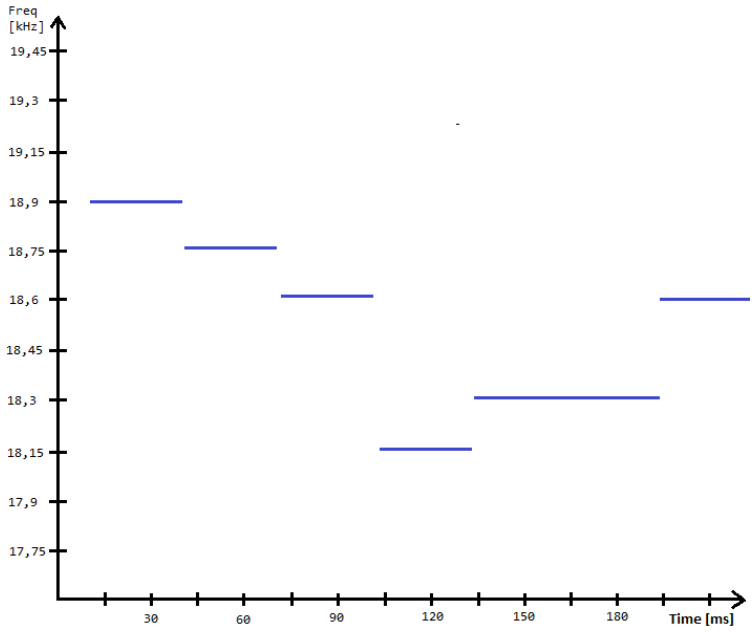


Figure 2: Figure show received frequencies in time.

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

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

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

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