Proceedings of the 4th WSEAS Int. Conf. on Information Security, Communications and Computers, Tenerife, Spain, December 16-18, 2005 (pp218-223)

A Viability Analysis of a Secure VoIP and Instant Messaging System on a Pocket PC Platform

JOSÉ-VICENTE AGUIRRE[1], RAFAEL ÁLVAREZ[2], JOSÉ NOGUERA[3],
LEANDRO TORTOSA[4] and ANTONIO ZAMORA[5]
Departamento de Ciencia de la Computación e Inteligencia Artificial
Universidad de Alicante
Campus de Sant Vicent del Raspeig, Ap. Correos 99, E-03080, Alicante
SPAIN

This work was partially supported by Generalitat Valenciana grant number GV04B-462

Abstract: - We propose a secure full-duplex VoIP and instant messaging system on a Pocket PC platform, allowing for session key transport using a public-key protocol and encrypted text or voice communication using a private-key algorithm. The full-duplex VoIP scheme presents good performance for long duration communication over LAN networks.

Keywords: - cryptography, security, public-key, VoIP, Pocket PC, Needham-Schroeder protocol, AES, RSA.

1 Introduction

Voice over Internet Protocol [2] (also called VoIP, IP Telephony, Internet telephony, and Digital Phone) is the routing of voice conversations over the Internet or any other IP-based network. The voice data flows over a general-purpose packet-switched network, instead of traditional dedicated, circuit-switched voice transmission lines. VoIP is a technology that has the potential to completely rework the world's phone systems. Secure [3] VoIP includes a Session Initiation Protocol [1] with session key transport.

While secure VoIP providers like Skype [9] have already been around for some time and are growing steadily, there are few implementations available for Pocket PC [8]. The Pocket PC platform has very limited storage and computational resources so a VoIP scheme, which is always very demanding, can be difficult to implement with such limitations. If we consider encryption of all communications, the computational cost can even be higher than what a Pocket PC can perform. This is the reason for this paper: explain the experiences, difficulties and results regarding the implementation of this kind of applications on the Pocket PC platform.

All cryptographic algorithms [5] used are recognized standards in their respective areas [10]. We have chosen RSA as the asymmetric cryptosystem and AES as the symmetric cryptosystem. RSA is already implemented in the Pocket PC cryptographic API, but AES is not available yet, requiring additional libraries [7] that were adapted to the high performance requirements of a VoIP system.

We have chosen C# as the language for the implementation since we consider that the .NET platform has a great future and it is a good opportunity to test the performance even in the non favourable case of a virtual machine based language such as C#. On the other hand, all critical components of the protocol (e.g. AES, RSA) have been implemented as highly efficient native code libraries.

The application consists of three parts: session key transport based on public-key encryption, simple text message exchange and full-duplex VoIP system. The session key transport must be easy to use, fast and secure in its domain. The simple text message exchange must also be fast and easy to use. The VoIP system must assure that the computational cost required for encryption and decryption is not too high for the intended platform. Voice treatment is not part of this paper [4], but the VoIP protocol must be tolerant to connection losses in the network and resynchronizations of the two voice threads, since this kind of errors can occur often in the mobile domain of the pocket PC platform.

2 Preliminaries

We use the following notation in this paper:

$Y_1 \mid Y_2$ denotes the concatenation of Y_1 and Y_2.
$k_1, k_2, \ldots k_n$ are private keys.
KU_x is the public key of X.
KP_x is the private key of X.

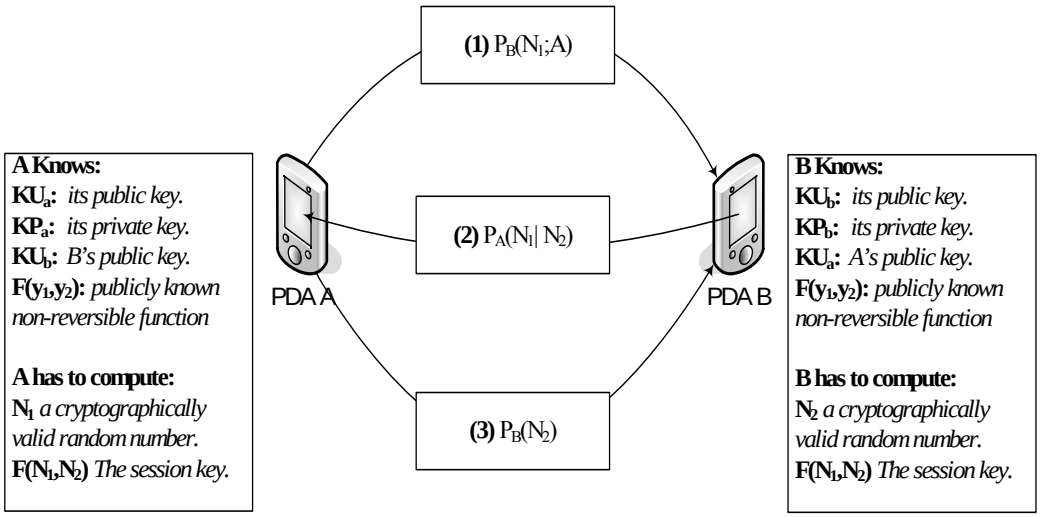

Fig. 1 Needham-Schroeder public-key protocol

$N_1,N_2 \ldots N_n$ are cryptographically valid random numbers.
$E_k(Y)$ denotes encryption (e.g. AES) of data Y using k key.
$D_k(Y)$ denotes decryption (e.g. AES) of data Y using k key.
$P_X(Y)$ denotes public-key encryption (e.g. RSA) of data Y using party X's public key. It is the same as $E_{KUx}(Y)$.
$PD_X(Y)$ denotes public-key decryption (e.g. RSA) of data Y using party X's public key. It is the same as $D_{KPx}(Y)$.
$S_X(Y)$ denotes the result of applying X's signature to the hash or data Y. It is the same as $E_{KPx}(Y)$.
$US_X(Y)$ denotes the result of retrieving signature Y from the original hash or data. It is the same as $E_{KUx}(Y)$.

3 Design

In this section we include a basic explanation of the protocols used in the application.

3.1 Session Key Transport

A well known protocol for session key transport is the Needham-Schroeder public-key protocol [6]. With this protocol, entity authentication, key authentication, and key transport are guaranteed with 3 messages.

Let us assume that A and B possess each other's authentic public-key (If this is not the case, but each party has a certificate carrying its own public key, then one additional message is required for certificate transport). Then the Needham-Schroeder public-key protocol (Prot.1) (see Fig.1) is as follows:

(Step 1.1) $A \rightarrow B: P_B(N_1;A)$
(Step 1.2) $A \leftarrow B: P_A(N_1|N_2)$
(Step 1.3) $A \rightarrow B: P_B(N_2)$

The protocol actions:

a) A sends B message (Step 1.1).
b) B recovers N1 upon receiving the message (Step 1), and returns the message (Step 1.2) to A.
c) Upon decrypting the message (Step 1.2), A checks that the key recovered (N1) is the same as the one sent in message (Step 1.1). (Provided N1 has not been used previously, this gives A both entity authentication of B and assurance that B knows this key.) A sends B message (Step 1.3).
d) Upon decrypting message (Step 1.3), B checks the key N2 recovered agrees with that sent in message (Step 1.2). The session key may be computed as f(N1; N2) using an appropriate publicly known non-reversible function f.

3.2 Voice Transmission

Let us consider two machines, A is the voice origin and B is the voice destination. When A sends voice data to B, the transmission is not instantaneous and a delay occurs. B stores the data received in a local buffer, so when a number of data k is stored, B starts to reproduce the stored information. This buffer is necessary, because the time needed to send the data is not known exactly. We define *recover the buffer* as the process of refilling the buffer with voice data.

The Sample Time is Tm, Tr is the transmission delay (lapse between recording at origin and reproduction at destination) and Tp is the processing time defined in equation (1).

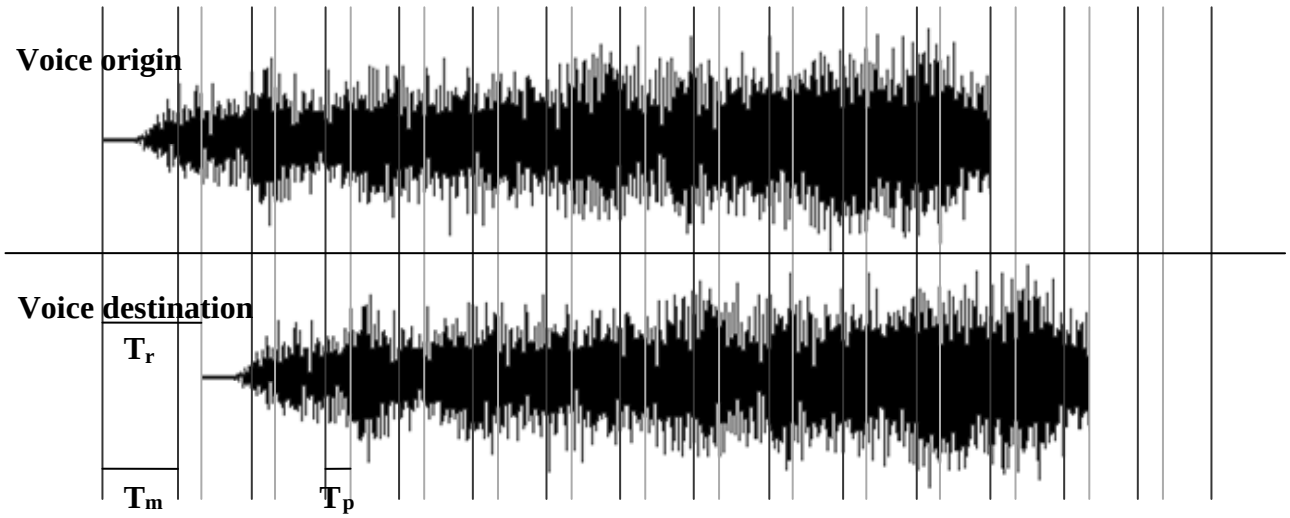

Fig. 2 Voice transmission diagram

$$T_p = T_{Crypt} + T_{Send} + T_{Decrypt} \quad (1)$$

The time required to encrypt the recorded data (measured in Tm units of time) is denoted by T_{Crypt}. This time is established by A, although it can theoretically be considered constant. $T_{Decrypt}$ is the time used to decrypt the data received at the destination and can also be considered constant.

T_{Send} is the time employed to send the data from the origin to the destination. The network establishes this time and, therefore, can't be calculated at the origin, but it can, nevertheless, be estimated.

The relation between times is defined in equations (2) and (3).

$$Tp < Tm \quad (2)$$
$$Tr = k \cdot Tm + Tp \quad (3)$$

The parameter k is the size of the buffer. Equation (3) is always true, while equation (2), in some cases, can be false. When this occurs, the data stored in the buffer is used, guaranteeing the continuity of reproduction. If this occurs k times it is called *buffer overflow* and, if the buffer isn't recovered, a silence of duration T_p - T_m time units will be listened at the destination or, if the buffer is recovered, the silence will be of T_r time units. If the buffer isn't recovered, the next time (2) is false another buffer overflow will occur. If the buffer is recovered then it is necessary that (2) is false another k times for a new buffer overflow to occur.

Two kinds of algorithms for voice transmission are developed for this article. The first one (Twisted client-server approach) considers each machine as a voice server that records and sends to the clients that are connected. For this approach it is necessary that each machine works as a voice server and voice client simultaneously. This is necessary in order to obtain a real full-duplex communication, but it is computationally expensive. It also requires a mechanism to synchronise both threads of communication.

The second one (client-server approach) establishes a communication between both machines. Each time a voice packet is sent, the other machine replies with the voice packet corresponding to the same time. Therefore, the synchronization mechanism is embedded into the algorithm. Also, it is less expensive computationally.

4 Implementation

We improve the memory management of the *player* and *recorder* classes included in the public library OpenNETCF [7]. The main functions analysed in this application follow next.

4.1 Session Key Transport

Needham-Schroeder public-key protocol implementation (Prot.2):

(Step 2.1) A → B: $P_B(N_1|ID_A|Alg|sizeKey)$
(Step 2.2) A ← B: $P_A(N_1|\ N_2)$
(Step 2.3) A → B: $P_B(N_2)$

The algorithm defined in this design is used for the implementation of the session key transport. Fixed sizes for the Keys, secrets and other variables involved in the algorithm are needed for the implementation. The sizes used are as follows:

- Secrets N_1 and N_2 are of 16 bytes of length (128 bits) each one.

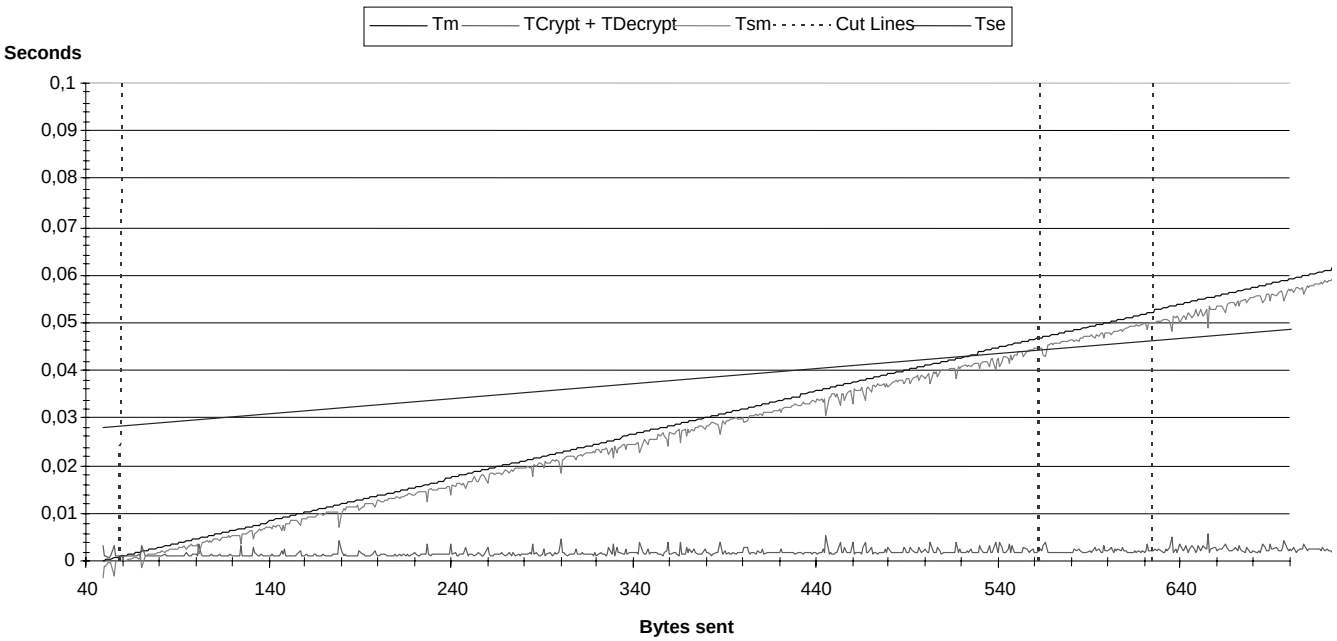

Fig. 3 Time variations in size increments

- ID (13 bytes) identifies each machine.
- Alg (1 byte) identifies the algorithm used for encryption.
- sizeKey (2 bytes) is the size of the key used in Alg.

RSA is the public key system used for encryption and decryption. Since all packets sent in the session key transport, are not bigger than 32 bytes, no fragmentation is needed. This restriction defines the size of the rest of the variables.

Secrets N_1 and N_2 are the random numbers used to create the session key and 16 bytes for each one is the biggest size possible to avoid RSA packet fragmentation.

Alg and sizeKey are set to the minimum size required and the rest of bytes are for ID. With this distribution we have 2^{104} possible machines, 2^8 possible encryption algorithms and a maximum possible key size of 2^{16} bits.

The session key is computed as $f(N_1; N_2)$. Where $f(N_1; N_2)$ is the function that concatenates N_1 and N_2 and gets bytes from N_1.length – (Key.length/2) to N_1.length + (Key.length/2)

4.1.1 Client side algorithm

a) A encrypts, with B's public key, message (Step 2.1), and sends it to B.
b) A receives message (Step 2.2) and decrypts it with its private key.
c) Upon decrypting message (Step 2.2), A checks that the key N_1 recovered agrees with that sent in message (Step 2.1). (Provided N_1 has never been used previously, this gives A both entity authentication of B and assurance that B knows this key.)
d) A sends B message (Step 2.3).
e) The session key is computed as $f(N_1; N_2)$.

4.1.2 Server side algorithm

a) B receives message (Step 2.1) and decrypts it with its private key.
b) B recovers N_1 upon receiving message (Step 1), and returns message (Step 2.2) to A.
c) B receives message (Step 2.3).
d) Upon decrypting message (Step 2.3), B checks that the key N_2 recovered agrees with that sent in message (Step 2.2).
e) The session key may be computed as $f(N_1;N_2)$

4.2 Text Transmission

In this application, text transmission consists of encrypting text with the session key and sending it to the destination machine.

Each machine is always a text message server, waiting to receive a text message from a client. At the same time, each machine is a client that can send text to another machine if they have interchanged a session key.

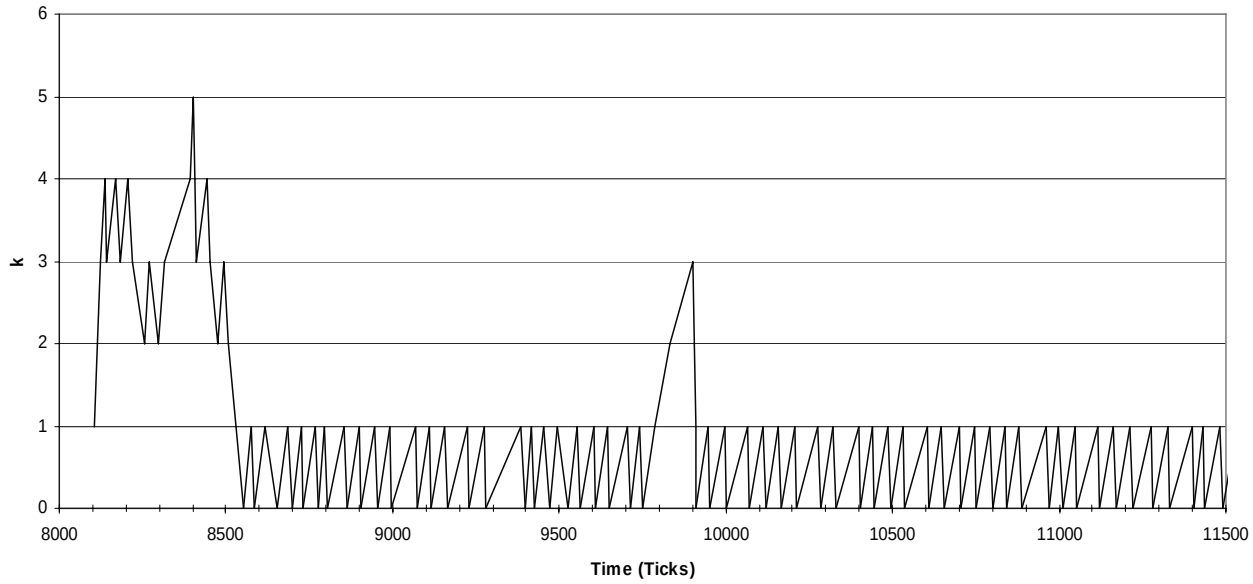

Fig. 4 Buffer state

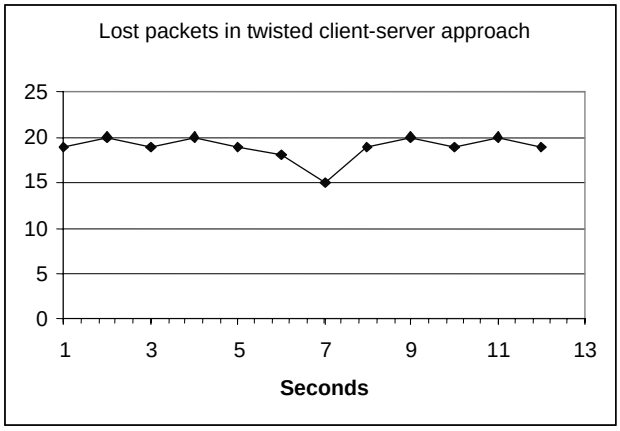

Fig. 5 Lost packets

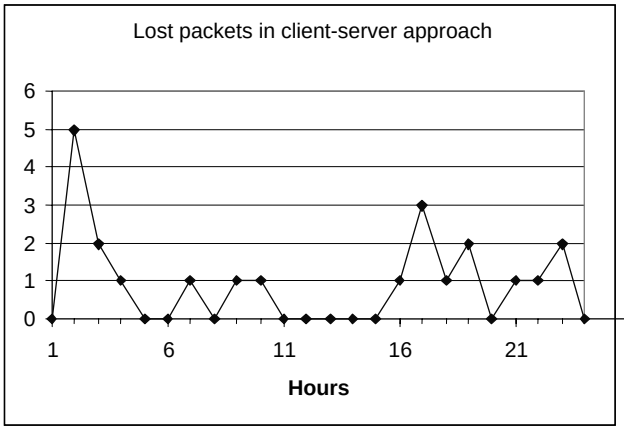

Fig. 6 Lost packets

4.3 Voice Transmission

In this application, voice transmission consists in sending voice data encrypted with the session key and receiving voice data from another machine and decrypting it. Voice is recorded and stored using the standard WAV format which has a header that permits storing information about sound format (bits, channels, frequency). In this version, sound format is constant, but a good enhancement could be implementing dynamic sound format.

When voice connection between two machines is activated, the client machine sends a synchronization byte. After this byte is received, at the same time, the two machines start a record process that records voice blocks of 576 bytes (set by experimentation). These blocks are stored in the send buffer, which is cyclical and has 4 elements (set by experimentation). This limitation prevents that slow networks will not increase the delay between voice recording and playback (t_r). If a network is too slow for sending the voice blocks fast enough, then a jump in reproduction is heard in the target machine. This is caused by the cyclical buffer. Another process sends the voice blocks to the destination machine. To do this, there are two approaches.

4.3.1 Twisted client-server approach

In this approach, each machine is a voice server that receives voice data from the clients that are connected and plays back the sound. For this approach it is necessary that each machine works as a server and client simultaneously. This is similar to text transmission, only that more bandwidth is required. This is the reason why this approach obtains catastrophic results on Pocket PC systems with WiFi communication. A great number of

collisions occur and a big delay is produced in the communications. Half-duplex communication works fine with this approach, but full-duplex communication is not possible.

4.3.2 Client-server approach

The second approach establishes a communication between the two machines. Only one acts as a server and the other is the client. In this communication each time a machine sends a voice packet, the other machine replies with the corresponding packet to the same time. Therefore, the synchronization mechanism is embedded into the algorithm. Finally, there is a process that plays back voice blocks. This process gets data from the received buffer and plays it.

The voice blocks size and send buffer length are parameters set by experimentation.

$$T_p = T_{Crypt} + T_{Send} + T_{Decrypt} \quad (4)$$

$$T_m = T(wav, n_{wav}) \quad (5)$$

$$T_p < T(wav, n_{wav}) \quad (6)$$

$$n_{wav} = 11025 \cdot Tm \textit{ in bytes} \quad (7)$$

$$T_r = k \cdot Tm + T_p \quad (8)$$

$$n_{wav} = n_{send} - 48 \textit{ in bytes} \quad (9)$$

$$T_{sm} = T_m - (T_{Crypt} + T_{Decrypt}) \quad (10)$$

$$T_{se} < T_{sm} \quad (11)$$

We have a function T(wav, n) that returns the duration time of the WAV data. This time is always equal to T_m (Sample Time). As we see in the design, T_p has to be less than T(wav,n) in the average case (6). T_p and T(wav,n) are related by the parameter n, and n is related to T_m and the WAV format. If T_m increases then n increases as well, and if we choose a better WAV format then n increases too. We choose the WAV format to be 8 bits, mono and 11 kHz; this is a small format and has phone quality. With this format we have the relation (7) between T_m and n. Finally we want T_r to be as small as possible, and k to be as big as possible. We can take T_m smaller, but we have to consider the restriction of T_p, that must include the encryption, transmission and decryption of the n bytes in less time that T_m.

After this, n_{send} is set to 576 bytes so, T_m is 52 milliseconds. This is set by experimentation (see Fig.3). T_{se} is the estimated send time and T_{sm} is the maximum send time. The relation (11) is true with a n_{send} higher than 562 and n_{send} is set to 576 that provides T_m a bit bigger than a 1/20 of second with good performance results.

Finally, k is chosen to be four, so T_r is less than 260 milliseconds. With this value we have a buffer a bit bigger than 1/5 of second. This value of k produces good performance in our case of study (see Fig.4). So, T_r is less than a ¼ of second, and that is a good enough value.

With this values and a good WiFi connection between machines, minimum loss of packets was detected in a communication with duration longer than 24 hours (see Fig.5 and Fig.6). No loss of audio performance was detected either.

5 Conclusion

Observing the results, we can conclude that Pocket PC platform is good enough to implement the hardest part of a secure VoIP and instant messaging system. This application allows session key transport using a public-key protocol and encrypted text or voice communication using a private-key algorithm. The full-duplex VoIP scheme has good performance for communications with duration longer than 24 hours and no degradation of the system was detected.

The application can be extended to more than two machines [8] using Windows CE certificates for the session key transport. A more complex system for voice treatment could also be advisable. But all enhancements are only add-ons that do not change the computational requirements shown in this paper.

References:

[1] Handley, M., Schulzrinne, H., Schooler, E., Rosenberg, J., Session Initiation Protocol (SIP). *The Internet Society* 1999

[2] H.323v5, *ITU* 2003

[3] Koblitz, N. *A Course in Number Theory and Cryptography.* Springer-Verlag. 1987.

[4] Linden, J., Achieving the Highest Voice Quality for VoIP Solutions, *Global IP Sound.* 2004

[5] Menezes, A., van Oorschot, P., Vanstone, S., Handbook of Applied Cryptography. CRC Press. Florida. 2001.

[6] Needham, R., Schroeder, M., Using encryption for authentication in large networks of computers, *Communications of the ACM*, 21(12):993-999, 1978.

[7] OpenNETCF http://www.opennetcf.org/

[8] Salman A., Baset and Henning Schulzrinne, An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol. *Department of Computer Science Columbia University*. Technical Report CUCS-039-04, 2004

[9] Skype http://www.skype.com/

[10] Stallings, W. *Cryptography and Network Security. Principles and Practice. Third Edition.* Prentice Hall. New Jersey. 2003