EURASIP Journal on Applied Signal Processing 2003:10, 1027–1042
© 2003 Hindawi Publishing Corporation

# On Securing Real-Time Speech Transmission over the Internet: An Experimental Study

**Alessandro Aldini**
*Instituto di Scienze e Tecnologie dell'Informazione (STI), Università degli Studi di Urbino, 61029 Urbino, Italy*
*Email: aldini@sti.uniurb.it*

**Marco Roccetti**
*Dipartimento di Scienze dell'Informazione, Università di Bologna, 40127 Bologna, Italy*
*Email: roccetti@cs.unibo.it*

**Roberto Gorrieri**
*Dipartimento di Scienze dell'Informazione, Università di Bologna, 40127 Bologna, Italy*
*Email: gorrieri@cs.unibo.it*

*Received 27 May 2002 and in revised form 3 January 2003*

We analyze and compare several soft real-time applications designed for the secure transmission of packetized audio over the Internet. The main metrics we consider for the purposes of our analysis are (i) the computational load due to the coding/decoding phases, and (ii) the computational overhead of the encryption/decryption activities, carried out by the audio tools of interest. The main result we present is that an appropriate degree of security may be guaranteed to real-time audio communications at a negligible computational cost if the adopted security strategies are integrated together with the playout control mechanism incorporated in the audio tools.

**Keywords and phrases:** Internet, multimedia applications, real time, security.

## 1. INTRODUCTION

The Internet offers a best-effort service over public networks without security guarantees. Therefore, the provision of secure real-time audio applications over wide area networks (WAN) like the Internet has to be carefully addressed. In particular, the success of such applications depends strictly on the speech quality and the privacy guaranteed by the provided services, which have to be perceived as sufficiently good by their users. Based on these considerations, we concentrate our attention on the steps of the audio data flow pipeline (depicted in Figure 1) which affect the performance and the security features of those applications designed for delivering secure real-time communications over the Internet. More precisely, in this paper, we analyse several audio applications to investigate the overhead, in terms of additional latency, which is caused by the embedded data compression algorithms and securing mechanisms.

In general, the provision of both adequate performance and security for the above-mentioned applications has to be carefully examined and modeled because of some important constricting conditions, illustrated as follows.

(i) These applications are often constrained to work under very restrictive resources (e.g., bandwidth) and congested traffic conditions. In particular, network-based audio applications experience variable transmission delay; hence the most used approach in order to ameliorate the effect of such an inevitable problem is to adapt the application behavior to the variable network delays (see, e.g., [1]).

(ii) Real-time audio applications which employ public, untrusted, and uncontrolled networks have strict security requirements, namely, they have to guarantee authentication, confidentiality, and integrity of the conversation (see, e.g., [2]).

On the one hand, from a performance standpoint, many are the factors that affect the computational cost of real-time audio applications over the Internet, such as codec, network access, transmission, operating system, and sound-card delays; and a significant issue is the problem of the latency due to each of the above components. For instance, an efficient coding of the signal, carried out by the codec activity, is the first factor to be considered if we want to effectively exploit the available transmission rates over the network, and to obtain at the receiver site the same speech quality as that

generated at the sender site [3, 4]. While such a kind of delay is relatively fixed, others depend on variable conditions. This is the case, for example, of network delays, for which the situation is quite crucial. Indeed, since the current Internet service model offers a flat, classless, and best-effort service, real-time audio traffic experiences unwanted delay variation (known as jitter) on the order of 500/1000 milliseconds for congested Internet links [5]. On the contrary, it is well accepted that telephony users find round trip delays longer than 300 milliseconds, more like a half-duplex connection than a real-time conversation (experience suggests that a delay of even 250 milliseconds is annoying despite the fact that message coherence is not affected). In addition, too large audio packet loss rates (over 10%) may have an awful impact on speech recognition [6, 7].

These observations put in evidence the importance of the trade-off between the stochastic end-to-end delays of the played out audio packets and the packet loss percentage, especially when dealing with the problem of unpredictable jitter typical of network environments providing a best-effort service (see, e.g., [1, 8, 9, 10]). The problem of obtaining the optimal trade-off between these two aspects and facing the constraints on strict delays and losses tolerated in an unfavourable platform is addressed by adaptive packet audio control algorithms (see, e.g., [1, 9, 10]), which adaptively adjust to the fluctuating network delays of the Internet in order to guarantee, when possible, an acceptable quality of the audio service.

On the other hand, from a security standpoint, real-time audio communications are a much less secure service than most people realize. It is relatively easy for anyone to eavesdrop phone conversations. Still, critics claim that the Echelon system [11], a world wide high-tech espionage system, is being used for crass commercial theft and a brutal invasion of privacy on a staggering scale. As far as audio applications over the Internet are concerned, anyone with a PC and an access to the public network has the possibility to capture the network traffic, potentially compromising the privacy and the reliability of the provided services. Hence, it is mandatory for audio applications to guarantee authentication, confidentiality, and integrity of data.

In the light of the above considerations, in this paper, we analyse some popular tools designed at the application layer for delivering secure real-time audio communications over the Internet, and we compare them by evaluating the computational overhead introduced by the coding algorithm and by the securing mechanism adopted by those tools. In particular, to carry out such a comparison, we have taken into account the following audio tools: Nautilus [12], PGPfone [13], Speak Freely [14], and BoAT [15]. The abovementioned tools, Nautilus, Pretty Good Privacy Phone (PGPfone) and Speak Freely, have been designed for protecting audio communications, at the application level, on the basis of external cryptographic modules. The motivation behind our choice of taking into account such tools relies on the consideration that they are freeware and downloadable with complete source code. On the other hand, the fourth audio tool we take into consideration, that is, BoAT, integrates, at the application level, the security mechanism together with the playout control algorithm. We point out that the freeware version of such a tool [15] does not include the security infrastructure, whose implementation is an ongoing copyrighted project.

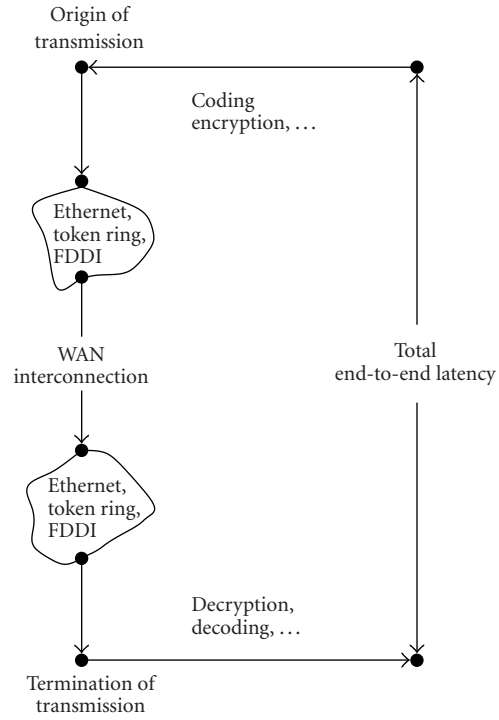



Figure 1: Internetworking audio data flow pipeline issue.

An important consideration concerning Nautilus, PGPfone, and Speak Freely is that these audio software packages do not include adaptive mechanized playout adjustment schemes. In contrast, BoAT has been designed to integrate the mechanism that adaptively adjust the audio playout point to the variable network behavior with the algorithm that makes the conversation secure. On the one hand, the playout control scheme of BoAT offers a minimal per-packet communication overhead, and also tolerable packet loss percentage and playout delays. On the other hand, the securing mechanism, integrated with the playout control algorithm, provides the receiver with a high assurance of secrecy, integrity, and authenticity of the conversation at a negligible computational cost as long as the underlying cryptographic assumptions are enforced. More precisely, the securing algorithm of BoAT allows two trusted parties to have a private conversation by employing a stream cipher (differently from the other considered tools which adopt block ciphers only) whose cryptanalysis is made much more difficult by the integration of this algorithm with the playout control mechanism. In particular, the securing algorithm naturally allows the parties taking part into the audio communication to agree on a sequence of session keys used by the particular stream cipher to encrypt data, where the lifetime of each key

is limited to a temporal interval not greater than one second of conversation (corresponding to less than $2^{12}$ bits of transmitted data), whereas the best-known attacks of stream ciphers require $2^{20}$ to $2^{33}$ ciphertext bits (with complexity from $2^{59}$ to $2^{21}$, respectively) [16, 17].

Other popular tools include strong security mechanisms at the application level like, for example, SecurePhone Professional [18] and SecuriPhone [19], two professional encrypted voice over Internet protocol (IP) tools, WebPhone [20], a shareware TCP/IP-based network phone tool, and MS NetMeeting [21], a freeware real-time Web phone (included in Windows 2000) which employs the MS Crypto APIs to support cryptographic services. In particular, NetMeeting is distributed in a binary form only and a comparison with the other tools would be complicated by the fact that some of its functionalities are embedded into the Windows operating system. On the other hand, there are other popular web audio tools that do not consider security services with the same intensity. For instance, FreePhone [22] does not take into consideration security features at all, while NeVot [23] and rat [24] provide a simplistic privacy service (without authentication mechanisms and key exchange protocols) which consists in encrypting the conversation by using the well-known DES block cipher [25] that exploits a symmetric key somehow decided by the involved parties.

This paper is a full version of [26], which in turn is based on some preliminary ideas [27] proposed to embed security services into the audio tool BoAT [10, 15, 28]. Here, we report on a complete performance/security comparative analysis conducted on the audio tools, Nautilus, PGPfone, Speak Freely, and BoAT, by measuring the computational overhead due to the coding/decoding phases, and the computational overhead due to the encryption/decryption phases. The main results we obtained emphasize that the computational costs payed by the security mechanism are quite low with respect to those due to the coding activity, by each of the considered tools. In particular, thanks to its integrated approach, the security platform of BoAT pays a computational cost which turns out to be about two orders of magnitude lower than that of the other tools. We wish to conclude these considerations by observing that our experimental results put clearly in evidence the kind of influence that the coding activity exerts on the cost of the security mechanism in terms of computational overhead. Simply put, the higher the compression level imposed by the codec, the lower the computational overhead due to the securing algorithm is (since less data are to be encrypted).

The remainder of the paper is organized as follows. In Section 2, we discuss the general problem of guaranteeing secure real-time audio communications over IP platforms, provide a succinct survey of those tools that guarantee security at the application level through external cryptographic modules, and provide the reader with the system and adversary model which all the tools have to cope with. In Section 3, we present a detailed survey of the BoAT architecture. In Section 4, we introduce the experimental scenario we have developed for carrying out our analysis. In Sections 5 and 6, we provide, respectively, the results of our experimental analysis with respect to the coding activity and the security mechanism. Finally, in Section 7, some conclusions terminate the paper.

## 2. SECURE AUDIO TRANSMISSION OVER IP

The need to consider security constraints when developing applications over IP is well accepted. The IP underlies large academic and industrial networks as well as the Internet. IP's strength lies in its easy and flexible way to route packets; however, its strength is also its weakness. In particular, the way IP routes packets makes large IP networks vulnerable to a range of security risks, for example, spoofing (meaning that a machine on the network masquerades as another) and sniffing (meaning that a third party listens in a transmission between two other parties). In order to protect audio communications in such a scenario, different approaches can be exploited depending on the particular layer chosen to be equipped with a complete set of security services. In the following two subsections, we briefly introduce two different audio security approaches; the former amounts to the use of the network level secure protocol termed IP-Sec, while the latter consists in equipping the networked audio applications with appropriate security mechanisms.

### 2.1. *Securing speech at the network level*

According to this approach, networked audio applications rely on the underlying securing internetworking structure to satisfy their security requirements. Usually, this transparent management of security is obtained by making the network layer secure. This is the case of IP-Sec [29], a collection of protocols and mechanisms adopted to extend the classical IP layer with authentication and security features.

The IP security (IP-Sec) protocol suite [29], developed by the Internet Engineering Task Force (IETF), defines a set of IP extensions for the provision of a secure, virtual, and private network which is as safe as or safer than an isolated local area network (LAN), but built on an unsecured, public network. The set of security services that IP-Sec can provide includes access control, connectionless integrity, data origin authentication, rejection of replayed packets (a form of partial sequence integrity and defence against unauthorized resending of data), confidentiality (encryption), and limited traffic flow confidentiality. IP-Sec technology seeks to secure the network itself instead of the applications that use it, as shown in Figure 2. Just as IP is transparent to the average user, so are the IP-Sec-based security services. Unlike classical specific application-level methods for protecting communications, this protocol suite guarantees security for any application using the network. To make the network level secure, the IP-Sec exploits three main traffic security technologies: (i) the Authentication Header (AH) through which authentication of packets is allowed, (ii) the Encapsulation Payload (ESP) through which encryption of data is offered, and (iii) the Internet Key Exchange (IKE) protocol that allows users to agree on keys and every related information. As already

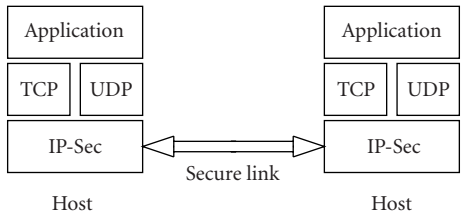



Figure 2: IP-Sec within the network layers.

Table 1: Packet size overhead due to the ESP header when the audio packet is generated by the codecs implemented in three different tools.

| GSM (Speak Freely) | GSM (PGPfone) | LPC-10 (Nautilus) |
|---|---|---|
| +9% | +57% | +39% |

mentioned, these mechanisms are designed to be algorithm-independent.

Besides the optimal level of interoperability guaranteed by this standard, the computational costs imposed by its implementation must be carefully considered, especially when real-time applications have to be supported. In particular, these costs are associated with (i) the memory needed for the IP-Sec code and data structures, and (ii) the computational overhead due to the activities of header management and of data encryption and decryption, to be carried out on a per-packet basis. This per-packet computational cost amounts to increased latency and reduced throughput. In addition, the use of the IP-Sec also imposes bandwidth utilization overhead on the transmission/switching/routing elements of the Internet infrastructure even if those components do not implement the IP-Sec. This is due to the increase in the packet size resulting from the addition of dedicated IP-Sec headers and from the increased traffic associated with key management protocols. For instance, the ESP header consists of a 10-byte long segment, an additional padding of variable length (0–255 bytes), and finally, a message authenticating code (MAC) whose length depends on the particular algorithm used to compute it. Such an additional header increases the packet size and can jeopardize the application throughput. This situation is particularly exacerbated when short audio samples are transmitted with each packet. To make this problem explicit, in Table 1, we show the packet size overhead due to the ESP header, in the case the audio packet is generated by the GSM codecs of Speak Freely and PGPfone (see columns 1 and 2), and by the LPC-10 codec of Nautilus (see column 3), when the particular algorithm used for the MAC is the MD5 [30]. It is also worth considering the analysis developed in [31], where the authors evaluate the performance of digital video transmission with the IP-Sec over IPv6 networks using an ordinary PC platform. By adding the IP-Sec infrastructure, the throughput degrades to 1/9 with respect to the performance without authentication or encryption.

In conclusion, the provision of reliable real-time audio quality data transmission over the Internet can be a hard task when using IP-Sec. For this reason and due to the fact that IP-Sec is not yet widely used over the Internet, other specific application-level security methods are to be considered in order to provide an adequate trade-off between security and performance.

### 2.2. Securing speech at the application level

According to this approach, we can exploit suitable hardware and software packages that, working at the application layer, are able to offer a secure real-time audio communication over the Internet. Usually, these applications are responsible for taking the audio samples, and then continuously digitizing, compressing, and encrypting them. After the encryption phase, the obtained audio packets are sent out through the network to the receiver site, where the reverse process is executed.

In this section, we make an overview of the software tools termed Nautilus, Speak Freely, and PGPfone, which provide secure audio communications over the Internet by exploiting, at the application level, appropriate external security modules. As already mentioned, instead, BoAT guarantees secure conversations by merging the security strategy together with the playout control mechanism. Due to the novelty of the approach adopted in BoAT, the general architecture of this tool together with the related design issue are presented separately in Section 3. For the sake of completeness, in Section 2.2.4, we formally provide the system model and the threat model with which the above audio tools have to cope.

#### 2.2.1. Nautilus

Nautilus [12] is a popular audio tool that digitizes, encrypts, transmits, and playouts audio packets either on ordinary phone lines using modems or over TCP/IP networks including the Internet. This tool provides usable speech quality at bandwidths as low as 4800 bps. The current version of Nautilus supports linear predicting coding [3] and exploits three different encryption functions.

The securing algorithm of Nautilus first generates an encryption key in one of two ways. In the former case, the key is generated from a secret passphrase that the users share. In the latter case, Nautilus generates the key by employing the Diffie-Hellman key exchange algorithm. Once the key is agreed on, by using one of the two ways, it is used for the encryption of the rest of the conversation by means of one of three block ciphers (Triple DES, Blowfish, and IDEA [25]), to be selected by the user. It is worth noting that Nautilus has been the first audio tool of this type freely distributed with source code (written in C) and it has been through four public beta test releases.

This tool is supported by two hardware platforms: IBM PC-compatibles and desktop Sun Sparcstations. In the former case, it supports (i) Windows platforms including Windows 95, 98, and NT, (ii) Linux, and (iii) Solaris X86. In the latter case, SunOS or Solaris are needed.

#### 2.2.2. Speak Freely

Speak Freely [14] is an audio tool for Windows machines and a variety of Unix workstations (Windows and Unix

machines can intercommunicate) which are usable across a local network or the Internet. Speak Freely is full duplex and provides a variety of compression modes, but if no compression mode is selected, it requires the network to reliably transmit 8000 Bps. Speak Freely incorporates a software implementation of the compression algorithm used in GSM digital cellular telephones that permits operations over Internet links of modest bandwidth. For instance, by using GSM compression, in conjunction with sample interpolation, the data rate can be reduced to about 9600 bps. Moreover, Speak Freely supports ADPCM compression to halve the data rate, and LPC-10 which compresses audio down to the limit of 346 Bps, thus yielding a compression factor of more than 26 to 1. Within Speak Freely, audio packets can be encrypted with either IDEA, DES, Blowfish, or a method based on a binary key supplied in a file. Speak Freely cooperates with PGP [32] to automatically exchange session keys with users on the same public key ring.

Speak Freely supports multicasting and can interoperate with other Internet voice programs supporting the Internet Real-Time Transport Protocol (RTP) or the Lawrence Berkeley Laboratory Visual Audio Tool (VAT) protocol, a widely used Unix conferencing program.

#### 2.2.3. PGPfone

PGPfone [13] is another popular tool which exploits external software modules in order to provide secure audio communications over the Internet. In particular, the audio transmission starts by transparently and dynamically negotiating the keys between the two parties by using the Diffie-Hellman key exchange protocol. Then, the voice stream is encrypted by means of either triple DES, CAST, or Blowfish, depending on the user.

The tool architecture allows any speech compression algorithm to be negotiated between the two parties as long as both parties support the same algorithm in their respective versions of PGPfone. Currently, it supports the GSM speech compression algorithm and the ADPCM compression for higher bandwidth connections such as ISDN. PGPfone is copyrighted freeware for noncommercial use and available for Windows machines and Apple Macintosh.

#### 2.2.4. The system model and the threat model

In this section, we define the environment in which the considered audio tools are expected to work, and the threat model such mechanisms should deal with, which basically reflects the assumptions of the Dolev-Yao model [33].

An ideal network can be expected to provide some precise properties; for instance, it should guarantee message delivery, deliver messages in the same order they are sent, deliver one copy of each message, and support synchronization between the sender and the receiver. All these properties are favourable in order to support real-time applications such as packetized audio transmission or multimedia conferencing over wide area networks.

However, the underlying network upon which we operate has certain limitations in the level of the service it can provide. Some of the more typical limitations on the network we are going to consider are that it may

(i) drop messages,
(ii) reorder messages,
(iii) deliver duplicate copies of a given message,
(iv) limit messages to some finite size,
(v) deliver messages after an arbitrarily long delay.

A network with the above limitations is said to provide a *best-effort* level of service, as exemplified by the Internet. This model adequately represents the Internet as well as shared LANs, but not switched LANs. All the dissertations and the results presented in the next sections are obtained under such model of the network.

As far as the adversary model is concerned, we argue that the audio tools we consider are also secure in the presence of a powerful adversary with the following capabilities:

(i) the adversary can eavesdrop, capture, drop, resend, delay, and alter packets;
(ii) the adversary has access to a fast network with negligible delay;
(iii) the adversary computational resources are large, but not unbounded. The adversary knows every detail of the cryptographic algorithm, and is in possession of encryption/decryption equipment. Nonetheless he cannot guess secret keys or invert pseudorandom functions with nonnegligible probability.

## 3. A SURVEY OF BoAT

In this section, we describe in detail the playout control software mechanism of [10], which has been originally designed for controlling and adapting the audio application to the network conditions. In [27], some proposals have been discussed to extend the above algorithm with security features. Here, we give a detailed and formal explanation of the approach proposed in [26], which integrates the playout control activity together with the security mechanism.

The original playout control algorithm of BoAT has been passed through intense functional and performance analysis [8], which revealed its adequacy to guarantee real-time constraints and it has been implemented in a software tool called BoAT [15]. Such a mechanism follows an adaptive approach and operates as follows. At the sending site, audio samples are periodically gathered, packetized, encrypted, and then transmitted to the receiving site, where the provision of a synchronous playout of the received audio packets is achieved by queueing the packets into a smoothing buffer and delaying their playout so as to maximize the percentage of packets that arrive before their playout point.

The playout control mechanism of BoAT assumes neither the existence of an external mechanism for maintaining an accurate clock synchronization between the sender and the receiver, nor a specific distribution of the end-to-end transmission delays. Such a scheme relies on a periodic synchronization between the sender and the receiver in order to obtain an estimation of the upper bound for the packet

Table 2: Steps of the handshaking protocol.

| Direction | Message Type | Contents of packets |
|---|---|---|
| $S \rightarrow R$ | probe | sender time $t_s$ |
| $R \rightarrow S$ | response | sender time $t_s$ |
| $S \rightarrow R$ | install | RTT computed by $S$ |
| $R \rightarrow S$ | ack | RTT computed by $S$ |

transmission delays experienced during the conversation. This upper bound is computed using round trip time (RTT) values obtained from packet exchanges of a handshaking protocol periodically performed (about every second) between the two parties. The handshaking protocol can be exploited for a two-fold goal:

(i) it allows the receiver to generate a synchronous playout of audio packets in spite of stochastic end-to-end network delays;

(ii) it allows the two authenticated parties to agree on a sequence of secret keys used to encrypt the conversation.

Before detailing the handshaking protocol and the related playout mechanism, we briefly explain the notation we adopt: $S$ is the sender, $R$ is the receiver, $M_j$ is a chunk of conversation contained in a packet, and $P_j$ denotes a packet composed of a timestamp and an audio sample $M_j$. We denote by $K_0$ a symmetric key agreed on during a preliminary authentication phase (e.g., by using a regular digital signature scheme such as RSA [34]), and by $K_i$ any subsequent session key agreed on between the two authenticated parties. Moreover, we assume that the packets of the handshaking phase are encrypted with $K_0$ by using any one of the block ciphers for the symmetric cryptography such as AES and Blowfish [25].

### 3.1. The playout control algorithm of BoAT

The first purpose of the synchronization protocol of BoAT is the provision of an adaptive control mechanism at the receiver site in order to properly playout the incoming audio packets. This is typically achieved by buffering the received audio packets and delaying their playouts so that most packets, in spite of stochastic end-to-end network delays, will have been received before their scheduled playout points. The success of such a strategy depends on a correct estimation of an upper bound for the maximum transmission delay. The technique we describe to achieve such an estimation is based on a three-way handshake protocol.

The first handshaking protocol precedes the conversation. As shown in Table 2, the sender begins the packet protocol exchange by sending a *probe* packet timestamped with the time value shown by its own clock ($t_s$). At the reception of this packet, the receiver sets its own clock to $t_s$ and sends immediately back a *response* packet. Upon receiving the response packet, the sender computes the value of the RTT by subtracting the value of the timestamp $t_s$ from the current value of its local clock. At that moment, the difference between the sender clock $C_S$ and the receiver clock $C_R$ is equal to an unknown quantity (say $t_0$) which may range from a theoretical lower bound of 0 (i.e., all the RTT values have been consumed on the way back from the receiver to the sender), and a theoretical upper bound of RTT (i.e., all the RTT values have been consumed when the probe packet is transmitted from the sender to the receiver). Then, the sender transmits to the receiver an *installation* packet with the calculated RTT value attached. Upon receiving this packet, the receiver sets the time of its local clock by subtracting from the current value of its local clock the value of the transmitted RTT. At that moment, the difference between $C_S$ and $C_R$ is equal to a value given by

$$\Delta = C_S - C_R = t_0 + \mathrm{RTT}, \tag{1}$$

where $\Delta$ ranges in the interval $[\mathrm{RTT}, 2 \times \mathrm{RTT}]$, depending on the unknown value of $t_0$, that in turn may range in the interval $[0, \mathrm{RTT}]$. Hence, the receiver is provided with the sender's estimate of an upper bound for the transmission delay that can be used in order to dynamically adjust the playout delay and buffer. In essence, a maximum transmission delay equal to $\Delta$ is left to the audio packets to arrive at the receiver in time for playout, and consequently a playout buffering space proportional to $\Delta$ is required for packets with early arrivals.

During the audio conversation, the sender timestamps each emitted audio packet $P_j$ with the value of its local clock $t_s$ at the moment of the audio packet generation. When an audio packet arrives, its timestamp $t_s$ is compared with the value $t_r$ of the receiver clock, then a decision is taken according to the rules shown in Table 3. Simply put, packets that arrive too late to be played out ($t_s < t_r$) are immediately discarded. In the same way, packets arriving too far in advance ($t_s > t_r + \Delta$) are discarded since their playout instant is beyond the temporal window represented by the buffer size. Instead, if $t_r \leq t_s \leq t_r + \Delta$, the packet arrives in time for being played out and is placed in the first empty location in the playout buffer. Then, the playout buffering space allows the packets that arrive in time for being played out to be scheduled according to the following rules. The playout instant of each packet that arrive in time is scheduled after a time interval equal to the positive difference between the values of $t_s$ and $t_r$. Using the same rate adopted for the sampling of the original audio signal at the sender site, the playout process at the receiver site fetches audio packets from the buffer and sends them to the audio device for playout. More precisely, when the receiver clock shows a value $t_r$, the playout process searches in the buffer for the audio packet with timestamp $t_r$. If such a packet is found, it is fetched from the buffer and sent to the audio device for immediate playout.

In order for the proposed policy to adaptively adjust to the highly fluctuant end-to-end delays experienced over wide area, packet-switched networks (like the Internet), the above mentioned synchronization technique is first carried out prior to the beginning of the conversation, and then periodically repeated throughout the whole audio communication. The adopted period is about 1 second in order to prevent the two clocks (possibly equipped with different clock rates) from drifting apart. Thus, each time a new RTT is

computed by the sender, it may be used by the receiver for adaptively setting the value of its local clock and the playout buffer size. This strategy guarantees that both the introduced additional playout time and the buffer size are always proportioned to the traffic conditions. However, it may be not possible to replace on the fly the current value of the receiver's clock and the dimension of its playout buffer. In fact, such an instantaneous adaptive adjustment of the parameters might introduce either *gaps* or even *time collisions* inside a talkspurt period during which the audio activity is carried out.

On the one hand, a gap occurs when a given sequence of audio packets is artificially contracted (or truncated) by the playout control mechanism, thus causing at the receiver an arbitrary skipping of a number of consecutive audio samples. This unwanted situation arises when an improvement of the traffic conditions of the underlying network causes a reduction of the estimated RTT. In such a case, as soon as the current synchronization is completed and the receiver installs new parameters, the receiver's clock suddenly advances from its current value to a larger value. In [10], it is shown that, in order for the receiver to playout all the audio packets generated by the sending site without skipping any audio sample, it suffices that the sender transmits the installation packet as soon as the first silence period not smaller than an amount of time proportional to the improvement of the traffic conditions is elapsed. Since no audio packet is generated during the silence period, at the moment the receiver sets a new value for its own clock, no audio packet is waiting for its playout instant in the receiver's buffer.

On the other hand, a time collision occurs when audio packets that would be too late for playout according to the current synchronization may instead be considered in time for playout if they are processed by the receiver's buffer as soon as a new synchronization has been completed. This situation arises in case of a deterioration of the traffic conditions over the underlying network. In such a case, the installation of a new synchronization causes the receiver's clock to be moved back from its current value; thus, in order to avoid collisions, it is necessary that the receiver does not play out, when the new synchronization is active, any audio packet that was generated when the old synchronization was active. Again, in [10], it is shown that in order to circumvent the problem raised by such a scenario, it suffices that the receiver installs the new value for its own clock only at the beginning of a silence period signalled by the sender.

In general, the installation at the receiver of the values of the receiver's playout clock and of the buffer dimension is carried out only during the periods of audio inactivity, when no audio packets are generated by the sender (i.e., during silence periods between different talkspurts). The reader interested in the proofs concerning the policies described above should refer to [10].

### 3.2. The securing algorithm of BoAT

In this section, we show how to integrate the playout control algorithm of BoAT with security services allowing for confidentiality, integrity, and authenticity to be preserved. This is obtained in two steps. On the one hand, we guarantee the handshaking packets against spoofing and sniffing attempts. On the other hand, we employ the handshaking protocol to make secure the whole audio conversation.

Table 3: Playout rules at the receiver site.

| Condition | Effect on the packet | Motivation |
|---|---|---|
| $t_s < t_r$ | discarded | it arrived too late to be played out |
| $t_s > t_r + \Delta$ | discarded | it arrived too far in advance of its playout |
| $t_r \leq t_s \leq t_r + \Delta$ | buffered | it arrived in time for being played out |

As far as secrecy is concerned, we show that the robustness of the privacy mechanism of BoAT depends on (i) the particular stream cipher we adopt and (ii) the lifetime of the secret keys used during the conversation. As far as authenticity is concerned, we show that after a preliminary authentication phase, the two trusted parties are provided with data origin authentication during the conversation lifetime. As far as the integrity is concerned, we show that the receiving trusted party can unambiguously decide that a received packet $P_j$ (timestamped with a value $t_s$) is exactly the same packet $P_j$ sent at the instant $t_s$ by the sending trusted party.

#### 3.2.1. The handshaking protocol

The original handshaking protocol of BoAT is exploited in order to exchange fresh session keys between the two authenticated parties, more precisely providing a key for each synchronization phase. Such a key will be used to secure the conversation and will have a lifetime equal to at most 1 second, namely, the time between two consecutive synchronizations. More precisely, we adopt the exchanged key as the session key of a stream cipher used to encrypt audio data. A stream cipher is a symmetric encryption algorithm which is usually faster than any block cipher. While block ciphers operate on large blocks of data, stream ciphers typically operate on smaller units of plaintext, usually bits. A stream cipher generates what is called a keystream starting from a session key $K$ which is used as a seed for the pseudorandom generation of the keystream. Encryption is accomplished by combining the keystream with the plaintext, usually with the bitwise XOR operation. Examples of well-known stream ciphers are A5/1 [35] (used by about 130 million GSM customers in Europe to protect the over-the-air privacy of their cellular voice and data communication), RC4 [25] (by the RSA's group), and SEAL [36].

The packets of the handshaking phases, instead of being encrypted with the particular stream cipher, are encrypted by employing the initial key $K_0$ and a block cipher that can use long keys in order to strengthen the security assumptions (e.g., up to 448-bit keys in the case of Blowfish, or up to 2040-bit keys in the case of RC6). During the generic handshaking phase $i$, the two authenticated parties agree on a 128-bit session key $K_i$ (e.g., exchanged in the install packet). Whenever the handshaking protocol has a positive outcome, $K_i$ is the

new key used to secure the subsequent chunk of conversation. Since the handshaking protocol is periodically started during the conversation, a sequence of keys $\{K_i\}_{i\in\mathbb{N}}$ is generated.

In order to guarantee the correct behavior of the above mechanism, both sender and receiver must come to an agreement. In particular, the sender site has to know if the receiver site has received the new key $K_i$ in order to decide to employ such a key to encrypt the following audio samples. Hence, upon receiving the installation packet, the receiver sends back an *ack* packet. At the reception of this packet, the sender starts to use the new key. An additional information for each audio packet is used as a flag in order to inform the receiver that the key is changed and is exactly $K_i$. For instance, by following a policy inspired by the alternating bit protocol, if each packet encrypted with the key $K_i$ is transmitted with a flag bit set to 0, then whenever a new synchronization phase is completed, each subsequent packet is transmitted with the bit set to 1. It is worth noting that if either the installation packet or the ack packet does not arrive at their destination, both sender and receiver carry on the communication by using the old key. Indeed, on the one hand, the sender begins to encrypt the outgoing audio packets with the new key only if it receives the ack packet. On the other hand, the receiver begins to decrypt the ingoing audio packets with the new key as soon as it receives a packet whose flag has been changed with respect to the previously received packets. The presented policy does not require additional overhead on the original scheme because it relies on the handshaking protocol only.

As far as the secrecy, authenticity, and integrity conditions of the handshaking protocol are concerned, the following remarks are in order.

(i) An adversary can try to corrupt the result of the handshaking protocol so that the two parties, after such a negotiation, disagree on the new key used for securing the conversation. In particular, he may try to forge or alter some packets of the handshaking phase, but he does not know the symmetric key used to encrypt them (e.g., he cannot create or alter a response packet with a given timestamp). In addition, he can cheat neither the sender nor the receiver by reusing any packet because of the presence of the timestamp $t_s$ in case of the probe and response packets, and also the presence of of the RTT in case of the install and ack packets (e.g., during the generic handshaking phase $i$, he cannot masquerade as the receiver by transmitting to the sender the response and ack packets intercepted during a previously completed handshaking phase $i - j$).

(ii) An adversary can try to drop systematically the messages of the handshaking protocol so that the lifetime of the old session key is extended from 1 second to the whole duration of the conversation; in this way, many more data and time are at disposal of a cryptanalysis attempt. Such a problem may be avoided by adopting the following policy. For each handshaking message, we create a packet containing the synchronization information encrypted with the block cipher and the audio sample filled with rubbish. Such a packet is first enriched with an additional field to inform the receiver that this is a handshaking packet and then encrypted with the stream cipher, thus masquerading it as a normal audio packet. Finally, in order to make it harder to reveal the handshaking packets, the time instant a new phase is started by the sender can be randomly chosen, instead of being scheduled once per second as in the original proposal of the algorithm. With these assumptions in view, an adversary can only try to drop some packets in a random way and, as a consequence, he can break off several consecutive handshaking phases with a negligible probability. In spite of this, an intensive traffic analysis during a full-duplex conversation could significantly restrict the temporal interval in which the two parties are expected to send packets of the handshaking phase. If we want the security mechanism to be more robust against this unlikely attack, we can shut down the conversation whenever more than $n$ consecutive handshaking phases are not completed, for some suitable $n$ depending on the strength of the cryptographic algorithm.

In essence, the handshaking protocol does not reveal any information flow allowing an adversary to spoof or sniff the conversation. Moreover, the same mechanism is robust to lost and misordered packets and makes no assumption on the service offered by the network. The described policy is similar to some well-known protocols for radio communications which are based on using spread spectrum frequency, in the sense that during a conversation, the transmission frequency is frequently changed in order to avoid interception and alteration. In the case of the securing mechanism of BoAT, the duration of every key is limited to the time space between two consecutive synchronizations (at most one second for normal executions), thereby this policy allows for making it difficult for a not authenticated party to decode the encrypted data, and practically guarantees to be robust to trivial breaks [25].

#### 3.2.2. Securing the conversation

The session key exchanged during the handshaking phase is used by the particular stream cipher for the encryption of both the timestamp and the whole audio packet. More precisely, each audio packet belonging to the chunk of conversation $i$ between the two consecutive synchronizations $i$ and $i+1$ is encrypted by resorting to the particular stream cipher and the session key $K_i$.

In order to guarantee authenticity and integrity of data, we employ this mechanism in conjunction with a MAC. In particular, we can adopt a mechanism similar to the HMAC-MD5 used also in [2] to ensure authenticity and integrity of the audio packets. Alternatively, we can encrypt (by the particular stream cipher) the output of a 1-way hash function applied to the audio packet to ensure authenticity and integrity of the same packet. Examples of well-known hash functions are MD5 and SHA [25].

In Algorithm 1, we show such an approach which guarantees a secure conversation. We denote by $\{P_j\}_{K_i}$ the audio packet $P_j$ encrypted by using the stream cipher starting from the session key $K_i$, and by $\mathrm{MAC}(K_i, P_j)$ the message authenticating code for the packet $P_j$ obtained by resorting to the session key $K_i$.

*Sender*

1. $P_j = \{t_s, M_j\}$
2. Send $P_j^* = \{\{P_j\}_{K_i}, \text{MAC}(K_i, P_j)\}$

*Receiver*

1. Receive $P_j^*$
2. Compute $t_s$ and $M_j$ by means of $K_i$
3. Verify the MAC

Algorithm 1: Securing algorithm.

The algorithm guarantees secrecy and satisfies the properties of authentication and integrity. More precisely, it guarantees the following condition. For each audio packet $P_j^*$, which is generated with the above algorithm and received in time for its playout, the receiver can decide its playout instant and verify its integrity and the authenticity of the sender.

### *Secrecy*

As far as secrecy is concerned, the security mechanism of BoAT offers to the trusted parties a high assurance of the privacy of the data transmitted during the conversation lifetime. In fact, we have shown that the handshaking protocol does not reveal any information about the secret keys exchanged between the trusted parties, and that an adversary as specified in Section 2 cannot guess secret keys. Secrecy is a crucial condition that the recent literature shows to be not met in glaring cases. For instance, we consider the attack on the A5/1 algorithm (used in GSM systems [35]) proposed in [37], in which a single PC is proved to be able to extract the conversation key in real time from a small amount of generated output. In particular, the authors of [37] claim that a novel attack requires two minutes of data and one second of processing time to decrypt the conversation. Now we assume that the particular cipher we choose to adopt is as weak as the A5/1 algorithm. In the approach of BoAT, in the absence of a powerful adversary able to identify and drop the handshaking messages, during two minutes of conversation, at least 120 different session keys are used so that the quantity of data that can be analyzed for a single key is not sufficient to perform the attack and to reveal the key and, consequently, the conversation. Moreover, in support of the robustness of the approach of BoAT, we point out that, in the recent literature, the best known attacks of some stream ciphers, proposed in [16], have complexity $2^{59}$ and require $2^{20}$ bits of ciphertext and are based on some restrictive assumptions on the characteristics of the stream cipher. In [17], a novel attack has a complexity gain $2^{21}$, but it requires $2^{33}$ bits of ciphertext, and, in certain cases, the cipher can resist this attack. Because of this, we have that an adversary can guess somehow a session key with a negligible probability; anyway, we recall that each session key may allow an adversary to decipher just one second of conversation with no information about the remaining encrypted data. In general, it is worth noting that the relatively short lifetime of every session key improves the secrecy guarantees for any cryptographic algorithm. Anyway, a study conducted in [8] revealed that too short lifetimes (e.g., less than 0.5 seconds) cause a worsening of the speech quality, therefore a massive resort to such an approach should be carefully analyzed.

### *Authenticity*

As far as authenticity is concerned, we first assume a preliminary authentication phase carried out by the two parties before the conversation (e.g., by resorting to a regular digital signature scheme). After this initial secure step, only the legitimate parties know the value of the symmetric key agreed on during this phase, and can carry out the first packet exchange of the handshaking protocol by means of the symmetric key. In particular, as we have shown in Section 3.2.1, an adversary cannot start, carry out, and complete the packet exchange of such a synchronization protocol with any of the trusted parties. Later on, during the conversation, each packet is timestamped with the sender clock value at the moment of the audio packet generation, encrypted by means of the session key $K_i$, and authenticated by means of the MAC, so that each received packet can be played out only once, and only if it arrives in time for being played out, according to the adaptive adjustment carried out during the $i$th handshaking synchronization phase. The receiver is guaranteed that the audio packets encrypted by means of the key $K_i$ and played out according to the piggybacked timestamp have been generated at (and sent by) the sender site. In fact, an adversary cannot behave as a "man in the middle" by generating new packets (as he does not know the session key and he cannot authenticate the packets) or spoofing (as he can resend or delay packets, but the timestamp allows the receiver to discard such packets). Finally, we point out that the key $K_{i+1}$ is agreed on by resorting to a packet exchange encrypted by means of a secret key, and such a negotiation does not reveal any information about the new session key. For these reasons, we deduce that the authentication condition is preserved along the conversation lifetime.

### *Integrity*

As far as integrity is concerned, the following remarks are in order. As a first result, we argue about the correctness of the algorithm, and then, we show that an adversary cannot alter the content of the conversation obtained by applying the above-presented algorithm. In a first simplified scenario, we assume the system model without malicious parties. We consider a packet $P_j^*$ generated by the sender and arriving at the receiver site in time for its playout. As the trusted parties share the same session key, the receiver can compute the timestamp in order to schedule the playout instant of the packet, compute $M_j$ in order to playout the audio packet, and check the MAC in order to verify the integrity of $M_j$. The effect of this behavior cannot be altered by an adversary, and we prove this fact by considering the potential moves of a malicious party. We assume the audio packets generated by the sender and managed by the receiver as seen in the above algorithm, and we show that all the played out packets can be neither generated nor altered by an adversary

with the capabilities specified in the threat model. In the case the adversary eavesdrops, captures, drops, or delays a packet $P_j^*$, then the proof is trivial. In fact, in these cases the adversary can only prevent the receiver from receiving or playing out $P_j^*$. The most interesting case arises whenever the adversary tries to alter $P_j^*$. In particular, he can alter the encrypted timestamp, the plaintext $M_j$, or the MAC, but in this case, the receiver notices the alteration by verifying the MAC, and therefore he discards the packet. It is worth noting that it is computationally infeasible, given a packet $P_j$ and the message authenticating code $\mathrm{MAC}(K_i, P_j)$, to find another packet $P_j'$ such that $\mathrm{MAC}(K_i, P_j) = \mathrm{MAC}(K_i, P_j')$. In addition, the adversary cannot send a new packet $P_j$ to the receiver because he knows neither the session key nor the playout instant of the audio sample $M_j$ he intends to forge.

## 4. EXPERIMENTAL SCENARIO

In this section, we describe the experimental scenario we have constructed to conduct the analysis of the audio tools of interest, namely, Nautilus, PGPfone, Speak Freely, and BoAT.

The experiments have been conducted with the two following machines, namely, a 133-MHz Pentium processor, 48-MB RAM, and ISA Opti 16-bit audio card, and a 200-MHz MMX Pentium processor, 64-MB RAM, and PCI Yamaha 724 audio card. These workstations have used two 10/100-Mbit Ethernet network cards to transmit packets over the underlying network. Both Linux (RedHat 6.0) and Windows 98 operating systems have been used depending on the analyzed audio tool.

In order to perform measurements of the computational overhead introduced by both securing and coding activities, while avoiding the issue of taking into account network delays, all the experiments were conducted as described in the following. For all the analyzed audio tools, each audio sample was first compressed by the codec employed within that tool, then encrypted by the corresponding securing algorithm, and finally, transmitted over the network by the adopted Ethernet card. Hence, for each conducted experiment, we took, at the sending site, measurements of the time intervals between the packet generation instant and its transmission instant over the network. This policy has allowed us to take experimental measurements of the packetization/compression/encryption delays not affected by the problem of managing variable network delays. The reverse process was executed at the receiving site in order to evaluate the decompression/decryption delays. Each of those experiments was repeated 30 times with an individual duration (for each experiment) of 30 seconds. All the results (reported in the two following sections) have been obtained by averaging the experimental measurements taken in each repeated experiment.

In order to allow the reader to understand the meaning of the reported experimental results, some important considerations concerning codecs are discussed in the following section.

### 4.1. Codecs

An efficient coding of the signal is the first factor to consider in order to allow speech to be reduced to a bandwidth fitting the network availability, and to obtain the same speech quality as generated at the sender site. For instance, telephone quality of speech needs 64 Kbits, but in most cases, such bandwidth is not reachable over the Internet. Codecs are used to cope with this lack, but as the compression level increases (and the needed bandwidth decreases), the generated speech degrades itself by turning misunderstandable. This issue has been passed down to the codec development efforts of the International Telecommunication Union (ITU). Hence, several codecs that work well in the presence of the scarce network bandwidth constraint have been designed. As an example, the ITU codecs G.729 and G.723.1 [38] have been designed for transmitting audio data at bit rates ranging from 8 Kbps to 5.3 Kbps.

In general, a trade-off exists between loss of fidelity in the compression process and amount of computation required to compress and decompress data. In turn, the more the data are compressed, the faster is the encryption/decryption process since less data is to be encrypted.

In the remainder of this section, we briefly survey the most characterizing features of the codecs embodied in all the audio tools of interest, namely, GSM, ADPCM, LPC-10, and the wavelet-based codec of BoAT.

GSM compression employs the global system mobile algorithm used by European digital cellular phones [39]. Speak Freely supports the standard GSM version that can produce audio at a data rate of 1650 bytes per second, thus reducing the PCM basic data rate by a factor of almost five with little degradation of voice-grade audio. In turn, PGPfone supports two versions of GSM: standard GSM and a version called "GSM lite," that provides the same speech quality as full GSM, but with less effort and less bandwidth needed. PGPfone provides GSM and GSM lite with a range of sampling frequencies. More precisely, voice can be sampled at various sampling rates, ranging through 4410, 6000, 7350, 8000, and 11025 samples per second. The faster GSM is sampled, the better the voice quality, but at the cost of a considerable computational load. Like most voice codecs, GSM is asymmetrical in its computational load for compression and decompression; indeed, decoding requires only about half the computation as encoding.

ADPCM [3] compression uses adaptive differential pulse code modulation and delivers high sound quality (the loss in fidelity is barely perceptible) with low computing loads, but at a cost of a higher bit rate with respect to GSM. Both Speak Freely and PGPfone support ADPCM.

LPC-10 [3] uses a version of the linear predictive coding algorithm (as specified by United States Department of Defense Federal Standard 1015/NATO-STANAG-4198) and achieves the greatest degree of compression, but like GSM, it is extremely computationally intensive. LPC-10 requires many calculations to be done in floating point and may not run in real time on a machine without an FPU. Audio fidelity in LPC-10 is less than what may be achieved with GSM.

The high degree of compression achieved by LPC-10 permits to use low-speed Internet links. In addition, the computational overhead per each audio packet due to encryption/decryption activities decreases since those packets have a typical small size. Both Nautilus and Speak Freely support LPC-10.

As far as BoAT is concerned, its control mechanism exploits a wavelet-based software codec designed to encode audio samples with a variable bit rate [28]. This codec exploits a flexible compression scheme based on the discrete wavelet transform of audio data; the wavelet coefficients are quantized using a successive approximation algorithm and are encoded according to a run-length entropy coding strategy that uses variable length codes. The quantization scheme has the property that the bits in the audio stream are generated in order of importance, yielding a fully embedded code. In such a way, the encoder can terminate the encoding at any point, thus allowing any precise (and variable) target bit rate. Based on this codec, a control mechanism that encodes and transmits audio samples with a data rate that is always proportioned to the network traffic conditions has been devised. In essence, the control mechanism devised within BoAT establishes a feedback channel between the sender and the receiver in order to assess periodically, in 5-second measurement intervals, the network congestion estimated under the form of average packet loss rate (and transmission delay variation). On the basis of this periodical feedback information, the following control process is carried out at the sender's site in order to match the sending rate to the current network capacity. When the experienced average loss percentage surpasses a given upper threshold, the control process gradually decreases the sending rate according to a predefined decreasing scheme which takes into direct account the value of the measured jitter. Such one gradual decrease is obtained by exploiting the possibility of terminating at any point the encoding activity provided by the wavelet-based variable-bit-rate codec embedded in BoAT. Conversely, if the average loss rate falls below a certain lower threshold, the sending rate is gradually increased in order to match the improved connection capacity. In summary, the use of one such wavelet-based variable-bit-rate codec guarantees that BoAT is always able to encode audio samples at a speed proportioned to the current network performances.

As a final remark, it is also worth mentioning that BoAT may guarantee a gradual adjustment of the sending rate to range from 8000 Bps (corresponding to the toll quality provided by PCM) to 700 Bps (corresponding to the synthetic quality provided by the LPC encoding strategy). For the purposes of the experiments concerning the coding computational load, we have used only the two limiting rates: 700 Bps and 8000 Bps. Instead, as far as the experiments related to the encryption/decryption computational overhead are concerned, we have used only the data rate 850 Bps, corresponding to a speech quality comparable with that offered by the GSM lite codec.

Summarizing, in Table 4, we report the costs of all the above-mentioned codecs in terms of both relative CPU cost and needed bandwidth. The CPU costs reported in that table are calculated taking the value 1 as the basis for the time needed by ADPCM to encode a second of speech. For instance, based on the fact that ADPCM takes one time slot to encode a second of speech, the wavelet-based codec of BoAT requires about 9 time slots to encode the same quantity of data, even if it guarantees a better level of compression since it requires 700 Bps instead of 4000 Bps.

Table 4: Relative CPU cost and bandwidth (bytes per second) requirements of various codecs.

| Codec | CPU | Bandwidth |
|---|---|---|
| ADPCM | 1 | 4000 |
| GSM | 39 | 1650 |
| LPC-10 | 53 | 346 |
| BoAT | 9 | 700 |

To conclude this section, in Table 5, we report the quantity of data compressed during a second of audio transmission by the different codecs embedded in each analyzed tool. Such values are particularly meaningful, especially when evaluating the performance of the securing algorithms, because they specify the quantity of data to be encrypted and decrypted.

## 5. COMPRESSING DATA: EXPERIMENTAL RESULTS

In this section, we report the experimental results obtained by calculating the computational overhead due to the codec activity for all the analyzed tools. The motivation behind our study relies on the fact that the coding activity is an important step of the audio data flow pipeline, and also affects the performance of the encryption/decryption activities. Hence, a significant comparison among the different tools must also take the performance of the coding/decoding activities into consideration.

All the results of interest are reported in Tables 6, 7, 8, and 9, and were obtained by employing the same architecture presented in Section 4. In particular, our tables show the computing time (expressed in milliseconds) needed for coding (meaning that digitized speech samples are converted into a compressed form) and decoding a second of conversation.

As already mentioned previously, the codecs implemented in each tool offer different trade-offs between the loss of fidelity and the amount of computation required to compress and decompress the data. In turn, the more the compression level, the faster the encryption/decryption process. These considerations motivate the very different results shown in our tables. In particular, we have that a lower compression level implies a better speech quality, a lower coding computational load, and, consequently, a high quantity of data to be encrypted and transmitted.

As a first result, since ADPCM offers the lowest compression level and the best quality of speech, we can observe that its computational load is limited to a few milliseconds (3 to 6

Table 5: Audio packet size and number of transmitted audio packets per second for each codec.

| | Speak Freely 7.1 | | PGPfone 2.1 | | Nautilus | BoAT |
|---|---|---|---|---|---|---|
| | GSM | ADPCM | GSM 4.4 | ADPCM | LPC-10 | |
| Bytes per packet | 336 | 496 | 70 | 327 | 56 | 34 |
| Packets per second | 5 | 8.4 | 14 | 13 | 5.5 | 25 |

Table 6: Speak Freely 7.1 (Windows 98).

| Computing time (ms) | CODEC | | | |
|---|---|---|---|---|
| | GSM | | ADPCM | |
| | Mean | Variancy | Mean | Variancy |
| Coding | 114.1 | 10.5 | 3.58 | 0.01 |
| Decoding | 32.2 | 3.23 | 3.67 | 4.51 |

Table 7: PGPfone 2.1 (Windows 98).

| Computing time (ms) | CODEC | | | |
|---|---|---|---|---|
| | GSM lite | | ADPCM | |
| | Mean | Variancy | Mean | Variancy |
| Coding | 115 | 296 | 6.08 | 2.84 |
| Decoding | 79.1 | 274.3 | 5.88 | 2.75 |

Table 8: Nautilus 1.5a (Linux RedHat 6.0).

| Computing time (ms) | CODEC | |
|---|---|---|
| | LPC-10 | |
| | Mean | Variancy |
| Coding | 172 | 1.95 |
| Decoding | 80.8 | 18.6 |

Table 9: BoAT (Linux RedHat 6.0).

| Computing time (ms) | CODEC | | | |
|---|---|---|---|---|
| | 8000 Bps | | 700 Bps | |
| | Mean | Variancy | Mean | Variancy |
| Coding | 31.03 | 12.11 | 105.5 | 46.5 |
| Decoding | 36.1 | 22.6 | 143.7 | 337.6 |

depending on the tool, see Tables 6 and 7) with respect to tens of milliseconds experienced by the other codecs. On the contrary, LPC-10, which offers the maximum level of compression, experiences the worst computational load (Table 8). Instead, both the BoAT codec (at 700 Bps) and the GSM codec have a workload corresponding to about a hundred of milliseconds (Tables 6, 7, and 9).

As another significant result, it is worth noting that each codec, except for the ADPCM codec of Speak Freely (Table 6) and for the wavelet-based codec of BoAT (Table 9), is appreciably faster during the decoding than the encoding phase. Only in the cases of ADPCM and of the BoAT codec (at 8000 Bps), the coding and decoding activities present a comparable computational load.

Again, as far as the wavelet-based codec of BoAT is concerned, the amount of computation required to compress data is very low when using the 8000 Bps data rate with respect to the 700 Bps data rate (Table 9). We point out that our experiments reveal that the codec of BoAT and the codec of GSM represent a good trade-off between speech quality and coding computational load since they both outperform LPC-10 from a performance standpoint, and guarantee a voice quality only a little lower than that provided by the codec of ADPCM in spite of an appreciably lower needed bandwidth.

## 6. SECURING DATA: EXPERIMENTAL RESULTS

In this section, we report the experimental results obtained by calculating the computational overhead due to the security mechanism for all the analyzed tools. In particular, the results are obtained by employing the same architecture presented in Section 4.

As far as BoAT is concerned, we make the following assumptions. The particular stream cipher we have considered in our experiments is the RC4 algorithm [25], while the message-authenticating code of each packet is computed as the encryption of the output of the MD5 message-digest algorithm [30]. The packets of the handshaking protocol are encrypted by using the block cipher Blowfish [25], and the temporal interval between two consecutive synchronizations is exactly one second.

In Table 10, we report the computational overhead (expressed in milliseconds) experienced during a second of conversation by a sending site that follows the algorithm illustrated in Algorithm 1 by singling out the different steps of the mechanism:

(i) encryption of the handshaking packets by means of the block cipher,
(ii) encryption of the audio packets by means of the stream cipher,
(iii) computation of the MAC.

The results of Table 10 put in evidence the following facts. The overall computational overhead is negligible (equal to few tens of microseconds). The extremely low use of the block cipher, which is used for the packets of the handshaking phase only, motivates the almost null computational cost derived from such an operation. Substantially, we note that the computational overhead is equally divided between the encryption phase, performed resorting to the RC4 algorithm (whose performance is about 13.7 MBps), and the

Table 10: Computational overhead of the securing mechanism of BoAT per second of conversation.

| | Computing time (ms) |
|---|---|
| Block cipher | 0.008 |
| Stream cipher | 0.0591 |
| MAC | 0.0474 |
| Total latency | 0.1145 |

authentication phase, performed resorting to the MD5 algorithm (whose performance is about 17 MBps). It is worth noting that these results are compatible with those on the performance of RC4 and MD5 presented in [40, 41, 42, 43]. Moreover, we point out that we have not considered SEAL-like stream ciphers, because from the performance viewpoint these algorithms do not seem to be appropriate if the key needs to be changed frequently (see, e.g., [36]).

Summarizing, the results put in evidence the negligible computational overhead of the implemented security mechanism, especially with respect to the latency introduced by the additional delay calculated by the adaptive playout control algorithm, equal to tens of milliseconds, as shown in [1, 8, 9].

Now, we contrast the above results with the performance obtained by analyzing the other application-level methods, namely, Nautilus, PGPfone, and Speak Freely. Before presenting the results of our experiments, we point out the following remarks. Unlike BoAT, we have that the above methods adopt block ciphers only in order to encrypt each audio packet to be transmitted along the network. More precisely, they employ some well-known cryptographic algorithms such as DES, 3DES, IDEA, Blowfish, and CAST (see [25] for the technical details of these algorithms).

In order to provide the reader with a better understanding of the reported results, we just recall the following remarks on the considered codecs. ADPCM offers toll quality of speech at the cost of a high quantity of data to be encrypted and transmitted. GSM codecs offer high speech quality in spite of a higher compression level. Finally, LPC-10 offers poor speech quality with the maximum level of compression. The experimental results are shown in Tables 11, 12, and 13. For each block cipher implemented in the tools of interest, the tables report the computing time experienced during a second of conversation by the encryption phase.

The first interesting point illustrated by our tables is that in all cases the computational overhead of the privacy mechanism is restricted to a few milliseconds. The upper bound is represented by the case of Speak Freely with the block cipher DES and the codec ADPCM with 20.8 milliseconds (Table 11). If we compare these results with those reported in Table 10, we can conclude that the securing mechanism of BoAT outperforms the other tools; in particular, BoAT turns out to be about 2 orders of magnitude better than the other tools (tens of microseconds with respect to a few milliseconds). This is because the integrated mechanism of BoAT adopts a lightweight ciphering mechanism that is very adequate when incorporated within the original handshaking protocol.

Table 11: Speak Freely 7.1 (Windows 98).

| Computing time (ms) | CODEC | | | |
|---|---|---|---|---|
| | GSM | | ADPCM | |
| | Mean | Variancy | Mean | Variancy |
| Blowfish | 2.47 | 0.01 | 5.22 | 0.15 |
| IDEA | 3.94 | 0.01 | 9.08 | 0.05 |
| DES | 9.77 | 0.20 | 20.8 | 0.16 |

Table 12: PGPfone 2.1 (Windows 98).

| Computing time (ms) | CODEC | | | |
|---|---|---|---|---|
| | GSM lite 4.4 | | ADPCM | |
| | Mean | Variancy | Mean | Variancy |
| Blowfish | 2.09 | 0.06 | 4.72 | 0.02 |
| CAST | 2.08 | 0.002 | 4.43 | 0.07 |
| 3DES | 6.35 | 0.14 | 16.8 | 0.56 |

The results of a comparison among the performance of the different tools depend strictly on the particular codec that is used to compress data. Indeed, the more the data are compressed, the faster is the encryption/decryption process. For instance, it may be significant to contrast the performance of PGPfone with the codec GSM (Table 12) and BoAT (Table 10) since the related codecs offer the same quality of speech and the same quantity of data to be encrypted and transmitted per second of conversation (850 bytes in the case of BoAT and 980 bytes in the case of the PGPfone GSM 4.4). The results (about 0.1 milliseconds for BoAT and about 2–7 milliseconds for PGPfone) confirm once again our claim that BoAT outperforms the other tools.

As far as Nautilus is concerned, it is worth mentioning that the very low computational overhead of its securing algorithm (Table 13) depends on the fact that Nautilus uses the LPC-10 codec that exploits a very high compression factor (note that the output of the LPC-10 compression algorithm per second of conversation is few hundreds of bytes). In particular, the speech quality offered by this codec is noticeably poorer than the high quality guaranteed by the codecs of the other considered tools.

An interesting remark is in order in the case of the ADPCM codec implemented in Speak Freely and PGPfone (see Tables 11 and 12). Indeed, when using such a codec, we can observe an overhead of the encryption phase of several milliseconds, especially in the case of 3DES, because ADPCM is based on a low compression level (thousands of bytes per second of conversation) in order to offer toll quality of the transmitted speech.

To conclude this section, we can summarize the obtained results by observing that the integrated mechanism of BoAT, thanks to its handshaking protocol which allows the two parties to share the session keys, has turned out to be very suitable to extend the playout control algorithm with security

Table 13: Nautilus 1.5a (Linux RedHat 6.0).

| Computing time (ms) | LPC-10 | |
|---|---|---|
| | Mean | Variancy |
| Blowfish | 0.32 | 0.0004 |
| IDEA | 0.48 | 0.004 |
| 3DES | 0.84 | 0.0009 |

Table 14: Performance comparison.

| | Computing time (ms) |
|---|---|
| BoAT | 0.229 |
| Speak Freely | 19.54 |
| PGPfone | 12.7 |
| Nautilus | 1.68 |

features in a simple and cheap way. Hence, adding security modules to the audio data flow pipeline may be done without jeopardizing the overall end-to-end delay because the presented approach has been revealed to be neither a noticeable computational penalty nor a performance bottleneck in real-time speech traffic. As far as the other application-level audio tools of interest are concerned, the performance results put in evidence that the computational overhead of the security mechanism is limited to a few milliseconds, and that such a result is about 2 orders of magnitude worse than the performance offered by BoAT.

## 7. CONCLUSION

In this paper, we have considered an adaptive packet audio control mechanism called BoAT and three application-level tools for the secure speech transmission over the Internet, namely, Nautilus, PGPfone, and Speak Freely. The former offers a scheme which adaptively adjusts to the fluctuating network delays typical of the Internet and integrates in such an algorithm security features. The other tools have been designed at the application layer in order to add speech compression and strong cryptographic protocols, as separated external modules, to the audio transmission over untrusted networks.

The comparison among the above audio tools has been conducted by measuring the computational overhead of both codec activity and security mechanism. In the former case, we have put in evidence the role played by codecs in the generation of the audio data flow pipeline and how they also affect the performance of the security mechanism. In the latter case, we have emphasized the low-computational cost of the cryptographic algorithms for each considered tool. In particular, we have shown the adequacy of BoAT in adding security with a negligible overhead. As an example, an interesting summarization of the results of Section 6 is reported in Table 14, where we show the computing time experienced by both encryption (at the sending site) and decryption (at the receiving site) during a second of conversation. In such a table, we consider the tools BoAT, Speak Freely with the codec GSM and the block cipher DES, PGPfone with the codec GSM lite 4.4 and the block cipher 3DES, and Nautilus with the codec LPC-10 and the block cipher 3DES. The results reveal that the provision of security has a computational cost of a few milliseconds and BoAT performs better than the other tools (tens of microseconds with respect to a few milliseconds).

A final consideration may be done concerning the particular approach adopted by the designers of BoAT for guaranteeing security. This approach has permitted to add appropriate security services without affecting the overall playout latency introduced by the playout control mechanism. This is a very relevant result for all those audio tools that incorporate dynamic mechanisms to adapt the playout process to the network conditions. In fact, as stressed in recent works [8, 9], it is not possible to interfere with the playout values decided by the control mechanisms without jeopardizing the strict real-time constraints imposed by audio applications.

## ACKNOWLEDGMENTS

We are grateful to the EURASIP JASP reviewers for their useful comments on the first version of this paper. This research has been funded by a Progetto MIUR and by a grant from Microsoft Research Europe.

## REFERENCES

[1] R. Ramjee, J. Kurose, D. Towsley, and H. Schulzrinne, "Adaptive playout mechanisms for packetized audio applications in wide-area networks," in *Proc. 13th IEEE Infocom Conference on Computer Communications (Infocom '94)*, pp. 680–688, Toronto, Ontario, Canada, June 1994.

[2] A. Perrig, R. Canetti, J. D. Tygar, and D. Song, "Efficient authentication and signing of multicast streams over lossy channels," in *Proc. IEEE Symposium on Security and Privacy*, pp. 56–73, Oakland, Calif, USA, May 2000.

[3] R. Westwater, "Digital audio presentation and compression," in *Handbook of Multimedia Computing*, B. Furht, Ed., pp. 135–147, CRC Press, Boca Raton, Fla, USA, 1999.

[4] R. Steinmetz and K. Nahrstedt, *Multimedia: Computing, Communications and Applications*, Innovative Technology Series. Prentice-Hall, Upper Saddle River, NJ, USA, 1995.

[5] L. Cottrell, W. Matthews, and C. Logg, *Tutorial on Internet Monitoring & PingER at SLAC*, Stanford Linear Accelerator Center, 2000.

[6] N. Jayant, "Effects of packet loss on waveform coded speech," in *Proc. 5th Data Communications Symposium*, pp. 275–280, Atlanta, Ga, USA, October 1980.

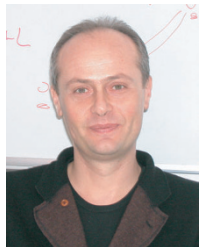
[7] J. Boyce and R. Gaglianello, "Packet loss effects on MPEG video sent over the public Internet," in *Proc. 6th ACM International Multimedia Conference (Multimedia '98)*, pp. 181–190, Bristol, UK, September 1998.

[8] A. Aldini, M. Bernardo, R. Gorrieri, and M. Roccetti, "Comparing the QoS of Internet audio mechanisms via formal methods," *ACM Transactions on Modeling and Computer Simulation*, vol. 11, no. 1, pp. 1–42, 2001.

[9] S. B. Moon, J. Kurose, and D. Towsley, "Packet audio play-

out delay adjustment: performance bounds and algorithms," *ACM Multimedia Systems*, vol. 6, no. 1, pp. 17–28, 1998.
[10] M. Roccetti, V. Ghini, G. Pau, P. Salomoni, and M. E. Bonfigli, "Design and experimental evaluation of an adaptive playout delay control mechanism for packetized audio for use over the Internet," *Multimedia Tools and Applications*, vol. 14, no. 1, pp. 23–53, 2001.
[11] N. Hager, *Secret Power*, Craig Potton Publishing, Nelson, New Zealand, 1996.
[12] B. Dorsey, P. Rubin, A. Fingerhut, B. Soley, and P. Mullarky, "Nautilus documentation," 1996, http://www.nautilus.berlios.de/.
[13] P. R. Zimmermann, "PGPfone: Owner's manual," 1996, http://www.pgp.com.
[14] J. Walker and B. C. Wiles, "Speak freely," 1995, http://www.fourmilab.ch/.
[15] M. Roccetti, V. Ghini, D. Balzi, and M. Quieti, *BoAT: Bologna optimal Audio Tool*, Department of Computer Science, University of Bologna, Bologna, Italy, 1999, http://radiolab.csr.unibo.it/BoAT/src.
[16] A. Canteaut and M. Trabbia, "Improved fast correlation attacks using parity-check equations of weight 4 and 5," in *Advances in Cryptology - EUROCRYPT '00, International Conference on the Theory and Application of Cryptographic Techniques*, vol. 1807 of *Lecture Notes in Computer Science*, pp. 573–588, Springer-Verlag, Bruges, Belgium, May 2000.
[17] E. Filiol, "Decimation attack of stream ciphers," in *Proc. First International Conference on Cryptology in India (INDOCRYPT 2000)*, vol. 1977 of *Lecture Notes in Computer Science*, pp. 31–42, Springer Verlag, 2000.
[18] Information Security Corporation, "SecurePhone Professional," 2002, http://www.infoseccorp.com.
[19] EarthSpeak International LLC, "SecuriPhone V. 1.09," 2002, http://www.jechtech.com/securiphone.htm.
[20] NetSpeak Corporation, "NetSpeak Webphone User's Guide," 1998, http://www.k2nesoft.com/webphone/.
[21] Microsoft, "NetMeeting 3 Resource Kit," 1999, http://www.microsoft.com/windows/netmeeting/.
[22] J.-C. Bolot and A. Vega-Garcia, "Control mechanisms for packet audio in the Internet," in *Proc. 15th IEEE Infocom Conference on Computer Communications (Infocom '96)*, San Francisco, Calif, USA, March 1996.
[23] H. Schulzrinne, "Voice communication across the Internet: a network voice terminal," Tech. Rep., University of Massachusetts, Amherst, Mass, USA, 1992, http://www.cs.columbia.edu/~hgs/rtp/nevot.html.
[24] V. Hardman, M. A. Sasse, and I. Kouvelas, "Successful multiparty audio communication over the Internet," *Communications of the ACM*, vol. 41, no. 5, pp. 74–80, 1998.
[25] B. Schneier, *Applied Cryptography*, John Wiley & Sons, New York, NY, USA, 2nd edition, 1996.
[26] A. Aldini, R. Gorrieri, and M. Roccetti, "An adaptive mechanism for real-time secure speech transmission over the Internet," in *Proc. 2nd IP-Telephony Workshop (IP-Tel '01)*, H. Schulzrinne, Ed., pp. 64–72, Columbia University, New York, NY, USA, April 2001.
[27] M. Roccetti, "Secure real time speech transmission over the Internet: performance analysis and simulation," in *Proc. Summer Computer Simulation Conference (SCSC '00)*, B. Waite and A. Nisanci, Eds., pp. 939–944, Society for Computer Simulation International, Vancouver, British Columbia, Canada, July 2000.
[28] M. Roccetti, "Adaptive control mechanisms for packet audio over the Internet," in *Proc. SCS Euromedia Conference (EUROMEDIA '00)*, F. Broeckx and L. Pauwels, Eds., pp. 151–155, Society for Computer Simulation International, Antwerp, Belgium, May 2000.
[29] Internet Engineering Task Force, "IP security protocol," in *Proc. 43th IETF Meeting*, Orlando, Fla, USA, December 1998, Internet Drafts available at http://www.ietf.org.
[30] R. L. Rivest, *The MD5 Message-Digest Algorithm*, MIT Laboratory for Computer Science and RSA Data Security, 1992.
[31] S. Ariga, K. Nagahashi, M. Minami, H. Esaki, and J. Murai, "Performance evaluation of data transmission using IPSec over IPv6 networks," in *Proc. The Internet Global Summit: Global Distributed Intelligence for Everyone, 10th Annual Internet Society Conference*, Yokohama, Japan, July 2000.
[32] S. Garfinkel, *PGP: Pretty Good Privacy*, O'Reilly & Associates, Sebastopol, Calif, USA, 1994.
[33] D. Dolev and A. C. Yao, "On the security of public key protocols," *IEEE Transactions on Information Theory*, vol. 29, no. 2, pp. 198–208, 1983.
[34] R. L. Rivest, A. Shamir, and L. M. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.
[35] M. Briceno, I. Goldberg, and D. Wagner, "A pedagogical implementation of A5/1," 1999, http://www.scard.org/.
[36] P. Rogaway and D. Coppersmith, "A software-optimized encryption algorithm," *Journal of Cryptology*, vol. 11, no. 4, pp. 273–287, 1998.
[37] A. Biryukov, A. Shamir, and D. Wagner, "Real time cryptanalysis of A5/1 on a PC," in *Proc. 7th Fast Software Encryption Workshop (FSE '00)*, pp. 1–18, New York, NY, USA, April 2000.
[38] ITU-T Recommendation G.729-G.723.1, 1996, http://www.itu.int/publications/maim_publ/itut.html.
[39] S. Redl, M. Weber, and M. Oliphant, *GSM and Personal Communications Handbook*, Artech House Publishers, Norwood, Mass, USA, 1998.
[40] A. Bosselaers, R. Govaerts, and J. Vandewalle, "Fast hashing on the Pentium," in *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference*, N. Koblitz, Ed., vol. 1109 of *Lectures Notes in Computer Science*, pp. 298–312, Springer-Verlag, Santa Barbara, Calif, USA, 1996.
[41] A. Bosselaers, "Even faster hashing on the Pentium," in *Proc. Rump Session of Eurocrypt (Eurocrypt '97)*, Konstanz, Germany, May 1997, http://www.esat.kuleuven.ac.be/~bosselae/publications.html.
[42] B. Schneier and D. Whiting, "Fast software encryption: designing encryption algorithms for optimal software speed on the Intel Pentium Processor," in *Proc. 4th Fast Software Encryption Workshop (FSE '97)*, pp. 242–259, Springer-Verlag, Haifa, Israel, January 1997.
[43] J. Touch, "Performance analysis of MD5," in *Proc. Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '95)*, pp. 77–86, Cambridge, Mass, USA, August–September 1995.

**Alessandro Aldini** is an Assistant Professor of computer science at the STI Centro of the University of Urbino, Italy. He received the Laurea (with honors) and the Ph.D. degrees in computer science from the University of Bologna, in 1998 and 2002, respectively. His current research interests include theory of concurrency, formal description techniques and tools for concurrent and distributed computing systems, and performance evaluation and simulation.



**Marco Roccetti** is a Professor of computer science in the department of Computer Science of the University of Bologna, Italy. From 1992 to 1998, he was a Research Associate in the Department of Computer Science of the University of Bologna, and from 1998 to 2000, he was an Associate Professor of computer science at the University of Bologna. Marco Roccetti authored more than 70 technical refereed papers that appeared in the proceedings of several international conferences and journals. His research interests include protocol design, implementation and evaluation for wired/wireless multimedia systems, performance modeling and simulation of multimedia systems, and digital audio for multimedia communications.

**Roberto Gorrieri** is a Professor of computer science in the Department of Computer Science, University of Bologna, Italy. He received the Laurea and the Ph.D. degrees in computer science, both from the University of Pisa, Italy, in 1986 and 1991, respectively. From 1992 to 2000, he was an Associate Professor of computer science at the University of Bologna. Roberto Gorrieri is a member of the European Association for Theoretical Computer Science and Chairman of IFIP WG 1.7 on Theoretical Foundations of Security. His research interests include theory of concurrent and distributed systems, formal method for security, and real-time and performance evaluation.