An Adaptive Mechanism for Real-time Secure Speech Transmission over the Internet

Alessandro Aldini, Roberto Gorrieri, Marco Roccetti
Dipartimento di Scienze dell'Informazione
University of Bologna, Italy
E-mail: { aldini, gorrieri, roccetti }@cs.unibo.it

Abstract—
The Internet offers a best-effort service over public networks which do not guarantee privacy. Because of this the provision of secure real time audio applications has received increasing interest and was an active research area in last years. We propose an adaptive packet audio control mechanism, originally designed for controlling and adapting the audio applications to the network conditions, and now enriched with cryptographic features in order to support secure, unicast, voice-based communications over the Internet. We take advantage of the characteristics of the adaptive mechanism, which meets the real time constraints needed by audio transmission applications, in order to realize a lightweight security infrastructure which offers privacy, authenticity and integrity assurances in a simple way and at a negligible cost. Finally, we show the performance of the proposed mechanism and we contrast it with those of other well-known tools designed for the secure audio transmission over the Internet.

Keywords—**Internet, Multimedia Applications, Security, Real Time.**

I. Introduction

SUPPORTING real time audio applications over wide area networks has been the subject of continuous research during the past recent years. Sophisticated applications of Internet multimedia conferencing will become increasingly important only if the speech quality and privacy provided by the communications will be perceived as sufficiently good by their users.

From a performance standpoint, the feasibility and the expected QoS of audio applications over public networks have to be carefully considered, if we wish those applications be successful in spite of the possibly restrictive resources (e.g. bandwidth) they are constrained to work with. In particular a significant issue in interactive sound transmission dealing with restrictive network resources is the problem of minimizing the latency due to each step of the communication. The main problem is that over the Internet only a flat, classless, best-effort service may be offered. For instance, as far as the transmission delay is concerned, the IP model does not consider the provision of QoS guarantees with the proper intensity. As a consequence, real time audio traffic experiences unwanted delay variation (known as jitter) with peaks on the order of $500/1000$ ms for congested Internet links [9]. On the contrary, it is well accepted that telephony users find round trip delays larger than 300 ms more like a half-duplex connection than a real time conversation. In addition, too large audio packet loss rates (over 10%) may have a tremendous impact on speech recognition ([15], [6]). These observations put in evidence the importance of the trade-off between the stochastic end-to-end delay of the played out audio packets and the packet losses, especially when dealing with the problem of unpredictable jitter typical of environments providing a best-effort service. Hence the most important metric affecting the user perception of audio is represented by the average packet audio playout delay vs. the packet loss rate, where with the term playout delay we refer to the total amount of time that is experienced by the audio packets from the time instant they are generated at the sender site to the time instant they are played out at the receiver site ([1], [16], [18]). 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. [16], [18], [22]), which adaptively adjust to the fluctuating network delays of the Internet in order to guarantee, when possible, an acceptable QoS.

In this work we consider the adaptive packet audio control mechanism proposed by one of the authors ([22], [21]) and we cope with the problem of adding security to the audio data flow pipeline generated by such a mechanism. This is done because the problem of considering security constraints when using public networks raises, as real time audio communications are a much less secure service than most people realize. Indeed, it is relatively easy to eavesdrop phone conversations, and the situation is even worse in the case of audio applications over the Internet, because 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 applications. Hence, it is mandatory for audio applications to guarantee authentication, confidentiality, and integrity of data (see e.g. [17] about the authentication problem in the case of video streaming).

The adaptive nature of the mechanism we adopt is particularly suitable to include with a negligible effort an adequate and lightweight security infrastructure. Such an enriched mechanism allows the two trusted parties to have a private conversation by employing a stream cipher, whose cryptanalysis is made much more difficult by the particular behavior of the adaptive algorithm. In particular, it allows the parties taking part into the audio communication to agree on a sequence of session keys, 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 some stream ciphers based on linear feedback shift register require in the best cases 2^{20} to 2^{33} ciphertext bits (with complexity 2^{59} to 2^{21}, respectively) [12], [8].

Summarizing, our scheme offers a minimal per-packet communication overhead, and, thanks to the original adaptive mechanism, also arbitrary packet loss and delay tolerated. Moreover, it provides the receiver with a high assurance of secrecy, integrity, and authenticity, as long as the underlying cryptographic assumptions are enforced. The negligible cost of such an integration is supported by some experimental results, presented also with a comparison with the performance of other tools.

To the best of our knowledge, in the literature the tools based on adaptive playout adjustment schemes either do not consider security services (see e.g. FreePhone [2]) or simply enable encryption of data by using the well-known DES block cipher [24] and a key prefixed by the two parties (see e.g. NeVot [26], and rat [13]). In addition, some other software packages working at the application layer are proposed to offer a secure audio communication over the Internet (see e.g. Nautilus [11], PGPfone [29], and Speak Freely [28]), but they do not include mechanized adaptive playout adjustment schemes.

The paper is organized as follows. In Sect. II we first discuss how to approach the problem of guaranteeing real time secure audio communications over IP, and then we specify the characteristics of the system (and the adversary) which the mechanism we propose is able to cope with. In Sect. III we present the mechanism and we describe the security properties which are met by our algorithm. In Sect. IV we analyze the performance of the mechanism together with the performance of well-known software tools designed for the secure audio transmission over the Internet. Finally, in Sect. V some conclusions are drawn.

II. Real-time Secure Audio Transmission

In this section we briefly describe how to get over the problems introduced when taking into account real time and security requirements, and then we fix the features of the models of (i) the system we rely on and (ii) the adversary we should cope with.

In this work we consider an adaptive audio control software mechanism, originally designed for controlling and adapting the audio application to the network conditions [22]. The mechanism has been passed through intense functional and performance analysis [1], which revealed its adequacy to guarantee real time constraints, and it has been recently implemented in a software tool called BoAT [23]. The motivation under the development of this kind of mechanism is that in the absence of network support to provide quality guarantees of Internet voice software, an interesting alternative to deal with jitter and high packet loss is to use adaptive control mechanisms. In fact, jitter-free, in-order, on-time packet delivery rarely, if ever, occurs in today's packet-switched networks. The provision of a synchronous playout of audio packets at the receiver site, in spite of stochastic end-to-end network delays, is typically achieved by buffering received audio packets and delaying their playouts, so that most of packets will have been received before their scheduled playout times. At the sending site, packet audio mechanisms operate by periodically gathering audio samples, packetizing them, and transmitting the packets to the receiving site. On the other site, received packets are queued into a smoothing buffer and the playout of packets is adaptively delayed. A strict connection exists between this additional delay introduced by the receiver buffer and the number of lost packets due to late arrivals, and the goal of these mechanisms consists in achieving the optimal trade-off.

Actually, these adaptive packet audio control mechanisms do not consider all the security problems, in that they do not guarantee confidentiality of the audio conversation, integrity of the transmitted data, and authentication of the involved parties. The need to consider such problems when modeling applications over IP is well accepted. The Internet Protocol 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. Indeed, the way IP routes packets makes large IP networks vulnerable to a range of security risks, e.g. 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 sensitive communications in such a scenario several approaches have been suggested. For instance, the Internet Engineering Task Force (IETF) has developed the IP Security (IP-Sec) protocol suite [14], a set of IP extensions that provide security services (such as access control, integrity, data origin authentication, and confidentiality) at the network level. IP-Sec technology seeks to secure the network itself, instead of the applications that use it; just as IP is transparent to the average user, so are IP-Sec based security services. Some key issues that need to be addressed concern the level of interoperability reachable by this standard and the computational performance costs imposed by the use of IP-Sec. In particular, these costs are associated with the memory needed for IP-Sec code and data structures, the computation of integrity check values, encryption and decryption, and added per-packet handling. The per-packet computational costs are manifested by increased latency and, possibly, reduced throughput. This is due to the increase in the packet size resulting from the addition of IP-Sec dedicated headers, and the increased packet traffic associated with key management protocols. In general, IP-Sec cannot be optimized for special-purpose applications. These considerations are emphasized especially in the case of transmission of real time audio packets.

In the same line of the above discussion, in this paper we consider an application-level extension of the audio mechanism of [22] in order to provide the audio communication over IP with all the main security services. Our approach in equipping this mechanism with security modules is said to be lightweight, because the used cryptography infrastructure exploits the particular adaptive audio control scheme in order to make secure the application with a minimal computational cost.

Before presenting the characteristics of the extended algorithm, we need to define (i) the environment in which the mechanism is expected to work and (ii) the threat model such a mechanism should deal with, which basically reflects the assumptions of the Dolev-Yao model [10].

A. *The System Model and the Threat Model*

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

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

- drop messages;
- reorder messages;
- deliver duplicate copies of a given message;
- limit messages to some finite size;
- deliver message 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 a model of the network.

As far as the adversary model is concerned, we argue that our mechanism is secure also in the presence of a powerful adversary with the following capabilities:

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

III. The Mechanism

The mechanism proposed in [22] has been originally designed to dynamically adapt the playout delay of the received audio packets to the network conditions assuming 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, in periodic intervals (at most 1 second), an estimation of the upper bound for the packet transmission delays experienced during an audio conversation. Such an upper bound is periodically computed using round trip time (RTT) values obtained from packet exchanges of a handshaking protocol performed among the two parties. In this work we exploit the handshaking protocol for a twofold goal:

- it allows the receiver to generate a synchronous playout of audio packets, in spite of stochastic end-to-end network delays;
- it allows the two authenticated parties to agree on a sequence of exchanged keys such that a third party cannot know it.

In the paper we adopt the following notation: S is the sender, R is the receiver, M_j is a chunk of audio conversation contained in a packet, P_j denotes a packet composed of a timestamp and an audio sample M_j. We denote with K_0 a symmetric key agreed during a preliminary authentication phase (e.g. by using a regular digital signature scheme such as RSA [19]), and with K_i any subsequent session key agreed among 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 RC6 and Blowfish) [24].

A. An Adaptive Playout Control Algorithm

As previously explained, one of the goal of the handshaking synchronization protocol consists of providing 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 of packets, in spite of stochastic end-to-end network delays, will have been received before their scheduled playout times. This is achieved as follows. The first handshaking protocol precedes the audio conversation and then is carried out every second along the conversation lifetime. As a proof-of-concept, before detailing the protocol, we present the handshaking phase as follows:

Direction	Message Type	Contents of packet
$S \rightarrow R$	*probe*	sender time t_s
$R \rightarrow S$	*response*	sender time t_s
$S \rightarrow R$	*install*	RTT computed by the sender
$R \rightarrow S$	*ack*	RTT computed by the sender

As shown in the above scheme, 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. Then it sends to the receiver an *installation* packet, with attached the calculated RTT value. 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 the end of this protocol, 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. Based on the value of the time difference (called Δ) between the two system clocks imposed by the protocol the following strategy may be followed. 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 following rules:

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

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 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 essence, a maximum transmission delay equal to Δ is left to the audio packets to arrive at the receiver in time for playout. In particular, the playout instant of each packet arrived in time is scheduled after a time interval equal to the positive difference between the values of t_s and t_r. The playout buffering space is proportional to Δ and allows the packets with early arrivals to be scheduled according to the above rules. Packets arrived too far in advance are discarded because their playout instant is beyond the borderline of the temporal window determined by the buffering space. The proposed scheme adaptively adjusts to the fluctuating network delays of the Internet thanks to the periodic clock synchronization carried out throughout the entire conversation lifetime. Whenever a new synchronization activity is conducted, a new RTT value is computed and depending on its value the clock values, the buffering delay and the buffer dimension are updated. This method guarantees that both the introduced additional playout time and the buffer dimension are always proportioned to the traffic conditions. The reader interested in more technical details and proofs related to the adaptive control mechanism should refer to [22], [1].

B. Securing the Mechanism

B.1 The Handshaking Protocol

As far as the security is concerned, we exploit the existing handshaking protocol in order to exchange among the two authenticated parties fresh session keys, and more precisely a key for each synchronization phase. Such a key will be used to secure the audio conversation and will have a lifetime equal to at most one second, namely the time between two consecutive synchronizations. More precisely, we use such a key as the session key of a stream cipher used to encrypt data. A stream cipher is a symmetric encryption algorithm which usually is 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 (a sequence of bits used as a key) 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 (used by about 130 million GSM customers in Europe to protect the over-the-air privacy of their cellular voice and data communication), RC4 (by the RSA's group), and SEAL.

In our protocol, 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). We point out that the packets of the handshaking phases, instead of being encrypted by employing the particular stream cipher, are encrypted by using a block cipher (such as RC6 and Blowfish). Whenever the handshaking protocol has a positive outcome, K_i is the new key used to secure the subsequent chunk of audio conversation. Since the handshaking protocol is periodically started during the audio 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 the sender and the receiver must come to an agreement. In particular, the sender site has to know whenever the receiver site has received the new RTT and key. Because of this, 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 it is exactly the new key K_i. For instance, 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 do not arrive at their destination, both the sender and the receiver carry on the communication by using the old key. Indeed the sender begins to encrypt with the new key only if it receives the ack packet, and the receiver begins to decrypt with the new key as soon as it receives an audio packet whose flag has been changed with respect to the previous audio packets. The presented policy does not require additional overhead on the original scheme, because it relies on the original handshaking protocol.

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

- an adversary cannot forge any packet, as 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). He can cheat neither the sender nor the receiver by resending any packet, because of the presence of the timestamp t_s (in the case of the probe and response packets) and also the RTT (in the case of the install and ack packets).
- 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 one second to the whole duration of the conversation; in this way, many more data and time are at disposal of a cryptanalysis attempt. A possible solution consists of masquerading the handshaking packets as normal audio packets, by filling the audio sample with rubbish. An additional bit is used to distinguish at the receiving site among the different packets, whereas a third party cannot guess anything because all the data are encrypted. With this assumption 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 our 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 general, 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 spread spectrum frequency open, in the sense that during a conversation the transmission frequency is frequently changed in order to avoid interception and alteration. In the case of our mechanism, the duration of each key is limited to the time space between two consecutive synchronizations (at most 1 second for normal executions), thereby this policy allows for making difficult for a not authenticated party to decode the encrypted data, and practically guarantees to be robust to trivial breaks [24].

B.2 Security Properties

From the security point of view, we aim at proving that our scheme guarantees the properties of secrecy, authentication, and integrity. As far as the secrecy is concerned we show that the robustness of our mechanism depends on both the particular stream cipher we adopt and the adaptiveness of the algorithm. As far as the 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) is exactly the same packet P_j sent with timestamp t by the sending trusted party.

B.3 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 entire 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 message authenticating code (MAC). In particular we can adopt a mechanism similar to the HMAC-MD5 used also in [17] 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.

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

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 created with the above algorithm and received in time for its playout, the receiver can (i) decide its playout instant and (ii) verify its integrity and the authenticity of the sender.*

Algorithm

Sender

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

Receiver

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

B.3.a Secrecy. As far as the secrecy is concerned, our mechanism guarantees that the trusted parties have a high assurance of the privacy of the data transmitted during the conversation lifetime. In fact, we have shown that (i) our protocol does not reveal any information about the secret keys exchanged between the trusted parties and (ii) an adversary as specified in Sect. II cannot guess secret keys. The secrecy is a crucial condition that the recent literature shows to be not met in glaring cases. For instance, let us consider the attack on the A5/1 algorithm (used in GSM systems) proposed in [7], 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 [7] claim that a novel attack requires two minutes of data and one second of processing time to decrypt the conversation. Now let us assume that the particular cipher we choose to adopt is as weak as the A5/1 algorithm. In the mechanism we propose, 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 audio conversation. Moreover, in support of the robustness of our approach we point out that in the recent literature the best known attacks of some stream ciphers, proposed in [8], 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 [12], a novel attack has a complexity gain (2^{21}), but it requires 2^{33} bits of ciphertext and it can be used under specified conditions. We can add that guessing somehow a 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 [1] revealed that too short lifetimes (e.g. less than 0.5 sec) cause a worsening of the speech quality, therefore a massive resort to such an expedient should be carefully analyzed.

B.3.b Authenticity. As far as the 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 step, only the legitimate parties (i) know the value of the symmetric key agreed during this phase and (ii) can carry out the first packet exchange of the handshaking protocol by means of the symmetric key. In particular, as we have shown in Sect. III-B.1, an adversary cannot start, carry out, and complete the packet exchange of this 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 (i) only once and (ii) only if it arrives in time for being played out according to the adaptive adjustment carried out during the ith 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 creating new packets (as he does not know the session key and he cannot authenticate the packet) or spoofing (as he can resend or delay packets, but the timestamp allows the receiver to discard such packets). Finally, we point out that (i) the key K_{i+1} is agreed by resorting to a packet exchange encrypted by means of a secret key, and (ii) 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.

B.3.c Integrity. As far as the 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 such an 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 (i) compute the timestamp in order to schedule the playout instant of the packet, (ii) compute M_j in order to playout the audio packet and (iii) 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. Note that it is computationally infeasible, given a packet P_j and the message authenticating code $MAC(K_i, P_j)$ to find another packet P_j' such that $MAC(K_i, P_j) = MAC(K_i, P_j')$. On the other hand, 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.

IV. Experimental Assessment

A working prototype of the secure audio control scheme illustrated in the previous section, called BoAT, was implemented in 1999 using the C programming language and the development environment provided by both the Linux and the BSD Unix operating systems. In this section we present the results we have derived by analysing such a mechanism under the assumptions of the previous section. After this, we present also the performance of some software tools which offer secure audio communications over the Internet. In particular, we consider the audio tools Nautilus [11], PGPfone [29], and Speak Freely [28]. The above methods adopt block cipher algorithms 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, IDEA, Blowfish, and CAST (see [24] for the technical details of these algorithms). The experiments have been conducted with a 133 MHz Pentium processor, 48 MB RAM, ISA Opti 16 bit audio card, and a 200 MHz MMX Pentium processor, 64 MB RAM, PCI Yamaha 724 audio card. The workstations have been connected by means of two 10/100 Mbit Ethernet network cards. Both Linux (RedHat 6.0) and Windows 98 operating systems, where available, have been used.

In order to provide the reader with an understanding of the reported values, we first specify the scenario in which such results have to be considered.

From a performance standpoint, an efficient coding of the signal is the first factor to consider, in order (i) to work with the available transmission rates over networks, and (ii) to obtain the same speech quality as generated at the sender site. As an example, telephone quality of speech needs 64 Kbits, but in most cases such a bandwidth is not reachable over the Internet. Codecs are used in order to cope with this lack, but as the compression level increases (and the needed bandwidth decreases), the generated speech degrades itself, by turning misunderstandable. In general, a trade-off exists between the quantity of data to be encrypted (specified by the particular codec) and the quality of the transmitted speech. In the case of BoAT, the used codec guarantees high quality at a sending rate of about 850 Bps, corresponding to 25 34-bytes long packets per second of conversation.

As far as the cryptographic algorithms are concerned, the following remarks are in order. The particular stream cipher we have considered in our experiments is the RC4 algorithm [24], whereas the message authenticating code of each packet is computed as the encryption of the output of the MD5 message-digest algorithm [20]. The packets of the handshaking protocol are encrypted by using the block cipher Blowfish [24], and the temporal interval between two consecutive synchronizations is exactly one second.

A. Performance Results

In Table I we report the computing time (expressed in ms) experienced during a second of audio conversation by a sending site that follows the scheme presented in Sect. III, by singling out the different steps of the mechanism:

- encryption of the handshaking packets by means of the block cipher,

- encryption of the audio packets by means of the stream cipher,
- computation of the MAC.

The results of Table I put in evidence the following facts. The overall computing overhead is negligible (equal to few tens of μsec). The extremely low use of the block cipher, which is used for the packets of the handshaking phase only, justifies the almost null computational cost deriving from such an operation. Substantially, we note that the computational overhead is equally divided between the encryption step, performed resorting to the RC4 algorithm (whose performance is about 13.7 MBps), and the authentication step, performed resorting to the MD5 algorithm (whose performance is about 17 MBps). It is worth noting that compatible results on the performance of RC4 and MD5 are also presented in [3], [5], [25], [27]. 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.

Summarizing, the results put in evidence the negligible computational overhead of the implemented security infrastructure, in particular with respect to the overall latency due to the adaptive audio control mechanism, equal to tens of ms, as shown in [1], [16], [18].

TABLE I

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

B. Comparison

In this section we report the experimental results for some well-known tools designed for the secure audio transmission over the Internet, namely Nautilus [11], PGPfone [29], and Speak Freely [28]. In particular, the results are obtained by employing the same architecture presented in Sect. IV. As far as the software tools are concerned, the following remarks are in order. Nautilus provides the Unix version only, released in 1996; PGPfone 2.1 and SpeakFreely 7.1 were released in the last two years: the former tool is available for Windows 9x operating systems only, whereas the latter one is available for both Unix and Windows 9x. Each tool employs codecs in order to reduce the quantity of data to be transmitted (as in the case of BoAT), and block ciphers for the encryption/decryption of data.

The codecs implemented in such tools offer different tradeoffs among efficiency of compression, loss of fidelity in the compression process, and the amount of computation required to compress and decompress, and last but not least, they determine the length of data that have to be encrypted and transmitted along the network. We recall that the codec activity is very important in order to establish a suitable QoS. For instance, due to the fact that the Nautilus release is 5 years old, it has been developed considering data rates up to 14400 Bps, so it has been equipped with codecs which have a high compression level in order to cope with low bandwidth, but this choice is paid at the cost of poor quality of the transmitted speech. In this paper we are interested neither in presenting and measuring the performance of the different codecs nor in evaluating the trade-off between QoS and efficiency of such compression mechanisms.

In order to provide the reader with a better understanding of the reported results, we just point out 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; the different versions of the GSM codecs offer high speech quality in spite of a higher compression level (we point out that the performance and quality features of the GSM and BoAT codecs are very close); finally, LPC-10 offers poor speech quality with the maximum level of compression. A summarization of the quantity of data compressed during a second of audio transmission by the different codecs embedded in each tool is reported in Table II; for a comparison, we recall that in the case of BoAT the codec works at about 850 Bps, corresponding to 25 34-bytes long packets per second of conversation.

TABLE II

AUDIO PACKET DIMENSION AND NUMBER OF TRANSMITTED AUDIO PACKETS PER SECOND OF CONVERSATION FOR EACH CODEC.

	Speak Freely 7.1	
	GSM	ADPCM
Bytes per packet	336	496
Packets per sec	5	8.4

	PGPfone 2.1	
	GSM 4.4	ADPCM
Bytes per packet	70	327
Packets per sec	14	13

	Nautilus
	LPC-10
Bytes per packet	56
Packets per sec	5.5

The experimental results are shown in Tables III to V; for each block cipher implemented in the different tools the tables report the computing time experienced during a second of conversation by the encryption step (the reader interested in a survey of these ciphers should refer to [24]). It is worth mentioning that all the results are obtained as mean values of repeated experiments, whose individual duration is 30 sec, and the relative variancy is reported too.

The first interesting point illustrated by our tables is that in all cases the computational overhead of the privacy mechanisms is restricted to few milliseconds (the upper bound is represented by the case of Speak Freely with the block cipher DES and the codec ADPCM in Table III with 20.8 ms). If we observe these results and the ones reported in Sect. IV-A, 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 μsec w.r.t. few milliseconds). This because our mechanism adopts a lightweight ciphering mechanism that is very adequate when integrated with the original handshaking protocol.

TABLE III
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

A comparison among the performance of the different tools strictly depends on the particular codec that is used to compress data. For instance, it may be significant to contrast the performance of PGPfone implementing GSM and BoAT, as the related codecs offer the same QoS 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 ms for Boat and about $2 - 7$ ms for PGPfone) confirm once again our claim that BoAT outperforms the other tools. As far as the results obtained with other codecs are concerned, it is worth mentioning that the good performance offered by Nautilus with LPC-10 (see Table V) depends on the fact that such a codec uses a high compression factor (note that the output of the LPC-10 compression algorithm per second of audio 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 mechanisms of the other tools. An interesting remark is in order in the case of the ADPCM codec implemented in PGPfone and Speak Freely (see Table IV and III). Indeed for such a codec we can observe an overhead of the encryption phase of several ms, especially in the case of 3DES, because it offers toll quality of the transmitted speech in spite of a low compression level (thousands of bytes per second of audio conversation).

To conclude this section, we can summarize the obtained results by observing that the nature of BoAT (in particular the handshaking protocol which allows the two parties to share the session keys) seems to be very suitable to extend the original mechanism with security features in a natural and cheap way. Hence, adding security modules to the audio data flow pipeline may be done without compromising the overall end-to-end delay, because the presented approach turns out to be neither a noticeable computational penalty nor a performance bottleneck in real time speech traffic. As far as other well-known tools are concerned, the performance results put in evidence that the computational overhead of the security platform is limited to few milliseconds, and that such a result is about 2 orders of magnitude worse than the performance of BoAT.

TABLE IV
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

TABLE V
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

V. CONCLUSION

In this paper we have presented a control mechanism for the transmission of real time audio over the Internet which offers:

- a packet audio control algorithm that adaptively adjusts to the fluctuating network conditions in order to maximize the QoS,
- a complete security infrastructure providing authentication of the parties, privacy and integrity of the transmitted data.

On the one hand, the original adaptive playout adjustment scheme offers good performance close to the optimum. In particular, as stressed in recent works ([16], [1]), packet audio control algorithms are optimizable with difficulty because there are intrinsic limits for improving their QoS (e.g. the traffic conditions do not allow us to reduce the playout delay without compromising the human perception of transmitted speech), and at the same time, the QoS they guarantee is borderline, because an higher overall latency cannot be tolerated by the real time constraints typical of such kind of services.

On the other hand, the adaptive playout adjustment scheme has revealed its adequacy to include security services in a simple way and with a negligible overhead. This because we have chosen to exploit the features of the existing adaptive mechanism, instead of e.g. treating the packet audio control mechanisms and the security mechanisms as separate layers in the protocol hierarchy. From the performance standpoint, the adequacy of our algorithm is put in evidence also in the experiments. As an example, an interesting summarization of the results of Sect. IV is reported in Table VI, 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.

TABLE VI
PERFORMANCE COMPARISON

	Computing Time (ms)
BoAT	0.229
Speak Freely	19.54
PGPfone	12.7
Nautilus	1.68

In particular, 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 (i) in general the provision of security has a computational cost of few milliseconds, and (ii) in particular BoAT performs better than the other tools (tens of μsec w.r.t. few milliseconds).

Acknowledgements This research has been funded by Progetto MURST Cofinanziato TOSCA and by a grant from Microsoft Research Europe.

References

[1] A. Aldini, M. Bernardo, R. Gorrieri, M. Roccetti, *"Comparing the QoS of Internet Audio Mechanisms via Formal Methods"*, Tech. Rep. UBLCS-99-04, University of Bologna (Italy), to appear in ACM Trans. on Modeling and Computer Simulation (TOMACS), 2001, `ftp://ftp.cs.unibo.it/pub/techreports/99-04.ps.gz`

[2] J-C. Bolot, A. Vega-Garcia, *"Control mechanisms for packet audio in the Internet"*, IEEE Infocom'96, San Fransisco, CA, 1996

[3] A. Bosselaers, R. Govaerts and J. Vandewalle, *"Fast hashing on the Pentium"*, Advances in Cryptology, Proceedings Crypto'96, LNCS 1109, N. Koblitz, Ed., Springer-Verlag, pp. 298-312, 1996

[4] M. Briceno, I. Goldberg, D. Wagner, *"A pedagogical implementation of A5/1"*, 1999, `http://www.scard.org`

[5] A. Bosselaers, *"Even faster hashing on the Pentium"*, presented at the rump session of Eurocrypt'97, 1997

[6] J. Boyce, R. Gaglianello, *"Packet Loss Effects on MPEG Video Sent Over the Public Internet"*, ACM Multimedia 98, 181-190, Electronic Edition, 1998

[7] A. Biryukov, A. Shamir, D. Wagner, *"Real Time Cryptanalysis of A5/1 on a PC"*, in Fast Software Encryption Workshop, NYC, 2000

[8] A. Canteaut, M. Trabbia, *"Improved Fast Correlation Attacks using Parity-check Equations of weight 4 and 5"*, Eurocrypt'2000, Bruges, 2000

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

[10] D. Dolev, A. C. Yao, *"On the Security of Public Key Protocols"*, IEEE Transactions on Information Theory, 29(2), 1983

[11] B. Dorsey *et al., "Nautilus Documentation"*, 1996, `http://www.lila.com/nautilus/`

[12] E. Filiol, *"Decimation Attack of Stream Ciphers"*, in Cryptology ePrint Archive, Report 2000/040, 2000

[13] V. Hardman, M.A. Sasse, I. Kouvelas, *"Successful Multi-Party Audio Communication over the Internet"*, in Comm. of the ACM 41:74-80, 1998

[14] Internet Engineering Task Force, *"IP Security Protocol"*, in Proc. of the 43 IETF Meeting, 1998, Internet-Drafts available at `http://www.ietf.org`

[15] N. Jayant, *"Effects of Packet Loss on Waveform Coded Speech"*, in Proc. of Fifth Int. Conference on Computer Communications, pp. 275-280, 1980

[16] S.B. Moon, J. Kurose, D. Towsley, *"Packet Audio Playout Delay Adjustment: Performance Bounds and Algorithms"*, in ACM Multimedia Systems 6:17-28, 1998

[17] A. Perrig, R. Canetti, D. Tygar, D. Song, *"Efficient Authentication and Signing of Multicast Streams over Lossy Channels"*, in Proc. of IEEE Security and Privacy Symposium, 2000

[18] R. Ramjee, J. Kurose, D. Towsley, H. Schulzrinne, *"Adaptive Playout mechanisms for Packetized Audio Applications in Wide-Area Networks"*, in Proc. of *INFOCOM '94*, 1994

[19] R. L. Rivest, A. Shamir, L. M. Adleman, *"A method for obtaining digital signatures and public-key cryptosystems"*, Communications of the ACM, 21(2): 120-126, 1978

[20] R. L. Rivest, *"The MD5 Message-Digest Algorithm"*, MIT LCS & RSA Data Security, Inc., 1992

[21] M. Roccetti, *"Secure Real Time Speech Transmission over the Internet: Performance Analysis and Simulation"*, in Proc. of 2000 Summer Computer Simulation Conference (SCSC'2000), (B. Waite, A. Nisanci Eds.), The Society for Computer Simulation International, Vancouver (CA), pp. 939-944, 2000

[22] M. Roccetti, V. Ghini, G. Pau, P. Salomoni, M. E. Bonfigli, *"Design and Experimental Evaluation of an Adaptive Playout Delay Control Mechanism for Packetized Audio for Use over the Internet"*, Tech. Rep. UBLCS-98-04, University of Bologna (Italy), to appear in Multimedia Tools and Applications, an International Journal, Kluwer Academic Publishers, 2001, `ftp://ftp.cs.unibo.it/pub/techreports/98-04.ps.gz`

[23] M. Roccetti, V. Ghini, D. Balzi, M. Quieti, *"BoAT: the Bologna optimal Audio Tool"*, University of Bologna (Italy), 1999, `http://radiolab.csr.unibo.it/BoAT/src`

[24] B. Schneier, *"Applied Cryptography, 2nd Edition"*, John Wiley & Sons, 1996

[25] B. Schneier, D. Whiting, *"Fast Software Encryption: Designing Encryption Algorithms for Optimal Software Speed on the Intel Pentium Processor"*, Fast Software Encryption, Fourth International Workshop Proceedings'97, Springer-Verlag, pp. 242-259, 1997

[26] H. Schulzrinne, *"Voice Communication across the Internet: a Network Voice Terminal"*, Tech. Rep., University of Massachusetts, Amherst (MA), 1992

[27] J. Touch, *"Performance Analysis of MD5"*, in Proc. of Sigcomm '95, Boston MA, 1995

[28] J. Walker, B. C. Wiles, *"Speak Freely"*, 1995, `http://www.fourmilab.ch`

[29] P. Zimmermann, *"PGPfone: Owner's Manual"*, 1996, `http://www.pgpi.com`