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

A. Aldini  R. Gorrieri  M. Roccetti


November 2000

Department of Computer Science
University of Bologna
Mura Anteo Zamboni 7
40127 Bologna (Italy)
Recent Titles from the UBLCS Technical Report Series

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

A. Aldini 1  R. Gorrieri 1  M. Roccetti 1


November 2000

Abstract

We propose an adaptive packet audio control mechanism, originally designed for controlling and adapting the audio applications to the network conditions, and now equipped 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 in order to realize a lightweight security infrastructure which offers a good trade-off between Quality of Service (QoS) and security assurances. From the performance viewpoint, we claim the algorithm meets the real time constraints needed by audio transmission applications. From the security viewpoint, the algorithm aims at guaranteeing the authenticity of the trusted parties, and the confidentiality and integrity of the data during the conversation lifetime, being careful not to degrade the QoS. In order to support this claim, we compare the performance of the proposed mechanism with other well-known tools designed for the secure audio transmission over the Internet.

1. Università di Bologna, Dipartimento di Scienze dell’Informazione, Mura Anteo Zamboni 7, 40127 Bologna, Italy. E-mail: {aldini, gorrieri, roccetti}@cs.unibo.it
1 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. On the one hand, the feasibility and the expected QoS of audio applications over public networks have to be carefully considered, if we wish those applications be successful. On the other hand, the problem of considering security constraints 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.

In general, when we consider real time audio applications, we have to cope with several problems in order to deal with security and quality conditions.

- First, these applications are often constrained to work under very restrictive resources (e.g. bandwidth). 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. In particular, real time audio traffic experiences variable transmission delay along the network, and the most used approach in order to ameliorate the effect of such inevitable problem is to adapt the applications to the variable network delays (see e.g. [15, 17, 21]).

- Second, real time audio applications often have strict security requirements, and the challenge is to guarantee authentication, confidentiality, and integrity of data, being careful that the adopted strategies do not jeopardize the real time behavior of the audio playout process (in the same line of what discussed, see e.g. [16] about the trade-off between authenticity condition and QoS). Moreover, it is well accepted that cryptographic systems used for real time audio applications need to support fast packet rates with a low computational overhead (see e.g. the choices adopted for GSM [4], where a stream cipher is used instead of a more robust, but less performant, block cipher scheme).

From a performance standpoint, 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, and the main challenge in supporting such applications is the need to provide synchronous playout of audio packets in spite of stochastic end-to-end network delays. 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 [14]. These observations put in evidence the importance of the trade-off between the end-to-end delay of the played out audio packets and the packet loss rate, especially when dealing with the problem of unpredictable jitter typical of environments providing a best-effort service (like the Internet). According to these considerations, as put in evidence in a lot of papers (see e.g. [1, 15, 17]), probably 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. The problem of obtaining the optimal trade-off between these two aspects and facing the above mentioned constraints (strict delays and losses tolerated) in an unfavourable platform is addressed by adaptive packet audio control algorithms (see e.g. [15, 17, 21]) which adaptively adjust to the fluctuating network delays of the Internet in order to guarantee, when possible, an acceptable QoS. Unfortunately, as stressed in recent works [15, 1], such mechanisms (i) are not further optimizable 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, (ii)
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.

These arguments put in evidence the importance of minimizing the time spent for any additional feature, in particular security services, which we aim at adding to this class of mechanisms. Based on these considerations, in this work we cope with the problem of adding security (authentication of the parties, secrecy and integrity of data) to the audio data flow pipeline, without noticeably affecting the speech quality and the overall latency. More precisely, we consider the adaptive packet audio control algorithm proposed in [21], and we point out that its adaptive nature 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 stream ciphers require $2^{28}$ to $2^{35}$ ciphertext bits (with complexity $2^{25}$ to $2^{31}$, respectively) [12, 8]. Summarizing, our scheme offers a minimal computational overhead, a low per-packet communication overhead, and, thanks to the original adaptive mechanism, also arbitrary packet loss and delay tolerated. Finally, it provides the receiver with a high assurance of secrecy, integrity, and authenticity, as long as the underlying cryptographic assumptions are enforced.

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 [23] and a key prefixed by the two parties (see e.g. NeVot [25], and rat [13]). In addition, some other hardware and software packages working at the application layer are proposed to offer a secure audio communication over the Internet. In this work we present also a comparison between the performance of our scheme and three well-known application-level tools: Nautilus [11], PGPfone [28], and Speak Freely [27].

The paper is organized as follows. In the next section 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. 3 we present the mechanism and we describe the security services which are met by two different schemes of the mechanism. In Sect. 4 we analyse the performance of these mechanisms and we compare the obtained results with the performance of well-known software tools designed for the secure audio transmission over the Internet. Finally, in Sect. 5 some conclusions are drawn.

2 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 [21]. 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 [22, 20]. 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 the same line of this discussion, a natural extension of the considered audio mechanism, disserted in this paper, is the integration with adequate modules for securing the audio communication over IP. 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].

2.1 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 switched LANs, but not shared 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. It knows every detail of the cryptographical algorithm, and it is in possession of encryption/decryption equipment. Nonetheless the adversary cannot guess secret keys or invert pseudorandom functions with non-negligible probability.

UBLCS-2000-15
3 The Mechanism

The mechanism proposed in [21] 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 sec), 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.

As a proof-of-concept, before detailing the protocol, we present the handshaking phase as follows:

\[ S \rightarrow R: \text{probe message containing the sender time } t_s \]
\[ R \rightarrow S: \text{response message containing the sender time } t_s \]
\[ S \rightarrow R: \text{install message containing the computed } RTT \]
\[ R \rightarrow S: \text{ack message containing the same } RTT \]

For the sake of clarity, we explain 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 [18]), 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 3DES and Blowfish) [23].

3.1 An Adaptive Control Mechanism

As previously explained, one of the goal of such a 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 we can see in the above presented 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 such a 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. Based on the value of the time difference imposed by the above mentioned protocol between the two system clocks (we call such a difference \( \Delta \)), the following strategy may be performed. 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.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Effect on the packet</th>
<th>Motivation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_s &lt; t_r$</td>
<td>discarded</td>
<td>the packet is arrived too late to be played out</td>
</tr>
<tr>
<td>$t_s &gt; t_r + \Delta$</td>
<td>discarded</td>
<td>the packet is arrived too far in advance of its playout time</td>
</tr>
<tr>
<td>$t_r \leq t_s \leq t_r + \Delta$</td>
<td>buffered</td>
<td>the packet is arrived in time for being played out and it is placed in the playout buffer at the receiver site</td>
</tr>
</tbody>
</table>

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 $\Delta$ 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 $\Delta$ 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 represented 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 of the adaptive control mechanism should refer to [21, 1].

3.2 Securing the Mechanism

3.2.1 The Handshaking Protocol

As far as the security is concerned, we exploit the 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 much 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.

From the security point of view, we have specified that during the conversation the audio packets following the generic handshaking phase $i$ are somehow encrypted by the session key $K_i$. The choice of the key is done as follows. After the handshaking protocol the two authenticated parties agree on a precise value, the $RTT$, so we can use such a value as the session key for the next 1 sec chunk of conversation. Simply put, whenever the handshaking phase has a positive outcome, $RTT$ represents the new key $K_i$ and it is used to secure the subsequent audio conversation. The $RTT$ is a secure value which can be precisely computed only by the sender, as it strictly depends on the sender’s clock and can vary on a wide range of values (in support
of this assertion, see e.g. [9]). A typical length for the key is 32 or 64 bits, but longer keys can be used, by simply adding to the transmitted RTT an adequately long sequence of random bits. Since the handshaking protocol is periodically started during the audio conversation, a sequence of keys \( \{ K_i \}_{i \in \mathbb{N}} \) is generated, based on the RTT values of each synchronization phase.

In order to guarantee the correct behavior of the above mechanism, both the sender and the receiver must come to an agreement on the same RTT value. To this aim, the sender site has to know whenever the receiver site has installed the new RTT value at the end of a synchronization handshaking. Due to this fact, 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 RTT value. 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.

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 it does not know the symmetric key used to encrypt them (e.g. it cannot create or alter a response packet with a given timestamp). It 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 delay the packets, but with no information on the overall RTT experienced along the network and computed by the sender, because of the assumptions made on the system model.
- 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, it 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 every key is limited to the time space between two consecutive synchronizations (at most 1 sec 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 [23].
3.2.2 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.

In the following we propose two schemes of our mechanism which offer a different trade-off between performance and assurances on the above mentioned security properties.

3.2.3 Scheme I: Encrypting the Whole Packet

As a first approach, we can simply decide to use the session key and 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 Algorithm 1 we describe such an approach which guarantees a secure audio conversation. In particular, we denote with \( \{ P_j \}_K \), the audio packet \( P_j \) encrypted by using the stream cipher starting from the session key \( K_i \).

**Algorithm 1**

**Sender**
1. \( P_j = \{ t_s, M_j \} \)
2. Send \( P_j^* = \{ P_j \}_K \)

**Receiver**
1. Receive \( P_j^* \)
2. Compute \( t_s \) and \( M_j \) by means of \( K_i \)

This 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 Algorithm 1 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.

**Secrecy** As far as the secrecy is concerned, as a minimal requirement for our mechanism, we are interested in preventing an attacker from an effective real-time cryptanalysis; more precisely, the trusted parties should have a high assurance of the privacy of the transmitted data during the conversation lifetime. As an example of the meaning of real-time cryptanalysis, 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. The authors of [7] claim that a first novel attack requires two minutes of data and one second of processing time, whereas a second novel attack requires two seconds of data and several minutes of processing time. Based on these considerations, we show that our mechanism satisfies the real-time secrecy property, in the sense that it prevents anyone from real-time cryptanalysis. Indeed, 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 we use at least 120 different session keys, so that the quantity of data that can be analyzed for a single key (about \( 2^{12} \) bits) is not sufficient to reveal such a key and, consequently, the audio conversation. This method prevents the first attack of [7], whereas the second kind of attack (which has at its disposal at most one second of conversation) probably requires more than several minutes to compute the key.
Anyway, it is worth noting that revealing a single key allows an adversary to decipher only one second of conversation with no information about the remaining encrypted data. Other attacks of generic stream ciphers can be prevented by the assumptions of our algorithm. As a matter of fact, in the recent literature, the best known attacks of most stream ciphers, proposed in [8], have complexity $2^{51}$, require $2^{90}$ bits of ciphertext and $2^{38}$ bytes of memory, and are based on some restrictive assumptions on the characteristics of the stream cipher. In [12], a novel attacks has a complexity gain ($2^{21}$ with no memory requirement), but it requires $2^{33}$ bits of ciphertext and in some cases the cipher can resist this attack.

In general, our mechanism can be even more robust, depending on the effective robustness of the particular stream cipher used in its implementation and by choosing a suitable lifetime of every session key. On the one hand, if we adopt a strong stream cipher (in the sense that it prevents both real time and off-line cryptanalysis), our mechanism is guaranteed to provide strong security, but in this case we have to carefully consider the effect of such a cipher on the performance. On the other hand, the relatively short lifetime of every session key can be exploited to improve the secrecy guarantees for any cryptographic algorithm. As an example, unlike GSM, our mechanism guarantees the real time secrecy property by using the same stream cipher of GSM. 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.

**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 secure 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. 3.2, 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 and encrypted by means of the session key $K_i$, 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 $i$th handshaking synchronization phase. Due to these considerations, 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 it does not know the session key) or spoofing (as it 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.

**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 and (ii) compute $M_j$ in order to playout the audio packet. 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 Algorithm 1, 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 $P_j^*$. Even in this case the correctness is guaranteed: in fact an adversary cannot alter (or create) a packet by fooling the receiver if he does not know the session key used by the stream cipher.

### 3.2.4 Scheme II: Achieving Fast Encryption

The above algorithm is very simple and it is clear that it offers privacy, authenticity, and integrity. However, in order to minimize the overhead on the QoS due to the privacy infrastructure, we could exploit the characteristics of both the mechanism and the IP model we are relying on. In the following we investigate a second approach.

It is clear that the correct temporal ordering of audio samples is fundamental in order to ensure the human perception of a transmitted speech. Nevertheless, the timestamps piggybacked by our playout delay control mechanism on each emitted audio packet represent the only information that the receiving site of an audio communication may exploit in order to reconstruct the temporal order of the transmitted audio samples. Several studies in the literature proved that transmission (via UDP) results in significant numbers of out of order packet arrivals with rates from 2% to 30% (see e.g. [6]). Moreover, the practical experience has shown that an average rate of about 10% is more than enough for making speech unintelligible. Given that a not authenticated party should cope with an amount of lost and disordered packets and that jitter and disorder make speech unintelligible, we can deduce that the real time reproduction of the temporal order of a transmitted audio stream without the piggybacked timestamps is made hard. Based on these considerations, in order to obtain the real time privacy, we decide to encrypt only the few bytes that constitute the timestamp attached to each audio packet.

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 [16] 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 Algorithm 2 we present this version of the mechanism for the sender and the receiver. For the sake of simplicity we denote with $MAC(K_i, P_j)$ the message authenticating code for the packet $P_j$ and obtained by resorting to the session key $K_i$, without regarding on the particular mechanism we choose to adopt.

#### Algorithm 2

**Sender**

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

**Receiver**

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

Also in the case of Algorithm 2, the authentication and integrity conditions continue to be valid, and a weaker form of privacy is preserved.

*Secrecy* As far as the secrecy is concerned, the above scheme prevents an attacker from an effective real time cryptanalysis, as the temporal reordering of the received packets without the piggybacked timestamps is made hard by both jitter and disorder. In spite of this, an attacker is not prevented from an off-line dedicated analysis aiming at the reconstruction of most of the conversation. Such an attack consists of finding the exact permutation of the audio samples which allows an adversary to obtain a perceivable human conversation. The complexity of such an analysis does not depend on the robustness of the cryptographic algorithm, and is less expensive than
the cryptanalysis of the stream cipher as described in the previous section. As a consequence, this scheme is less secure than the first one; in particular, an upper bound for this second scheme is that it can prevent real time cryptanalysis only.

**Authenticity** As far as the authenticity is concerned, we can resort to the same assumptions presented for the first scheme. In particular, in this case the receiver is guaranteed that the packets authenticated by means of the MAC, and played out according to the piggybacked timestamp encrypted by means of the key $K_i$ have been generated at (and sent by) the sender site.

An adversary cannot create a new packet, as it cannot authenticate it by means of the MAC and the session key $K_i$, and whenever it resends or delays a packet, it cannot change the encrypted timestamp, so that the receiver can discard such packet.

**Integrity** As far as the integrity is concerned, the same remarks presented for Algorithm 1 are in order. As far as the correctness of the algorithm is concerned, when a packet $P_j$ (created by the sender by following Algorithm 2) arrives, the receiver can (i) compute the timestamp in order to schedule the playout instant of the packet and (ii) check the MAC, in order to verify the integrity of $M_j$. As far as the adversary behavior is concerned, we distinguish the following scenarios:

- the adversary alters 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 $MAC(K_i, P'_j) \neq MAC(K_i, P_j)$;

- the adversary sends a new packet $P_j$ to the receiver, but he knows neither the session key nor the playout instant of the audio sample $M_j$ he intends to forge.

4 Experimental Assessment

A working prototype of the secure audio control scheme illustrated in the previous section, called BoAT, was implemented the last year 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 different schemes presented in the previous section. After this, we compare the obtained results with the performance of some software tools equipped with cryptographic features in order to offer a secure audio communication over the Internet. In particular, we consider the audio tools Nautilus [11], PGPfone [28], and Speak Freely [27]. 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 [23] 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 for each 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 [23], whereas the message authenticating code of each packet is represented by the encryption of the output of the MD5 message-digest algorithm [19]. The packets of the handshaking protocol are encrypted by using the block cipher Blowfish [23], and the temporal interval between two consecutive synchronizations is exactly one second.

4.1 Performance Results

In Table 1 we report the computing time (expressed in ms) experienced during a second of audio conversation by the two schemes presented in Sect. 3, by putting in evidence 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 (only for the second scheme).

The results of Table 1 put in evidence the following facts. The overall computing overhead is negligible for both the schemes (equal to few tens of μsec in both cases). The extremely low use of the block cipher (for the packets of the handshaking phase only) is such that our mechanism does not suffer the well-known heavy computational cost of this kind of ciphers (with respect to stream ciphers). Moreover, because of the fact that the MD5 algorithm is faster than the RC4 algorithm (about 17 MBps vs. 13.7 MBps), the second scheme of the mechanism is slightly faster than the first one. Compatible results on the performance of RC4 and MD5 are also presented in [3, 5, 24, 26]. It is worth noting 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.

For the sake of completeness, we report in Table 2 the performance derived by simulating the use of the GSM A5/1 stream cipher algorithm instead of the RC4 algorithm. The second scheme is slightly faster than the first one, even if the difference is less evident with respect to the results of Table 1, the reason being that A5/1 spends more time in the setup phase during the generation of the keystream. However, we can observe also in this case that the computational cost of the algorithm is reduced to tens of μsec.

Summarizing, in both tables 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 put in evidence in [1, 15, 17].

<table>
<thead>
<tr>
<th>Computing Time (ms)</th>
<th>Scheme I</th>
<th>Scheme II</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block Cipher</td>
<td>0.008</td>
<td>0.008</td>
</tr>
<tr>
<td>Stream Cipher</td>
<td>0.0591</td>
<td>0.0013</td>
</tr>
<tr>
<td>MAC</td>
<td>0.0474</td>
<td>0.0671</td>
</tr>
</tbody>
</table>

Table 1. Computational overhead of the securing mechanism of BoAT during a second of conversation.

<table>
<thead>
<tr>
<th>Computing Time (ms)</th>
<th>Scheme I</th>
<th>Scheme II</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block Cipher</td>
<td>0.008</td>
<td>0.008</td>
</tr>
<tr>
<td>Stream Cipher</td>
<td>0.052</td>
<td>0.0026</td>
</tr>
<tr>
<td>MAC</td>
<td>0.0474</td>
<td>0.058</td>
</tr>
</tbody>
</table>

Table 2. Computational overhead of the securing mechanism of BoAT during a second of conversation, by using the GSM A5/1 stream cipher.
4.2 Comparison

In this section we report the experimental results for some well-known tools designed for the secure audio transmission over the Internet, as obtained by employing the same architecture presented in Sect. 4. 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 year: 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. Each of the implemented codecs in such tools offers different trade-offs 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, it determines 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 4 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 each codec is reported in Table 3; 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 for each second of conversation.

The experimental results are reported in Tables 4 to 7. It is worth mentioning that all results are obtained as mean values of repeated experiments, whose individual duration is 30 sec, and the relative variancy is reported too. As far as the encryption overhead is concerned, we report the performance of the particular block ciphers implemented in the different tools. The reader interested in a survey of these ciphers should refer to [23].

If we compare these results with the ones reported in Sect. 4.1, we can observe that BoAT performs better than the other tools (tens of μsec vs. few ms) thanks to its lightweight ciphering mechanism, the main reason being that stream ciphers are designed to be exceptionally fast with respect to any block cipher. As put in evidence in the previous section, we point out that in our mechanism the use of stream ciphers coupled with the adaptive audio control scheme allows us to obtain an adequate security level, which is comparable to the security guarantees offered by any application-level tool implementing block ciphers. At the same time, from a security viewpoint, BoAT is better than other mechanisms which implement stream ciphers, as seen in the case of the cryptanalysis of the GSM privacy algorithm [7], where the performance gain obtained by using the A5/1 stream cipher is paid at the cost of a weaker secrecy level.

The comparison among BoAT and the other tools is even more significant in the case of the mechanisms implementing the GSM codec. Indeed this codec and the BoAT codec offer the same QoS and the same quantity of data to be encrypted and transmitted per second of conversation.
<table>
<thead>
<tr>
<th>Computing Time (ms)</th>
<th>CODEC</th>
<th>CODEC</th>
<th>CODEC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GSM</td>
<td>ADPCM</td>
<td>LPC-10</td>
</tr>
<tr>
<td>Blowfish</td>
<td>0.921</td>
<td>2.10</td>
<td>0.29</td>
</tr>
<tr>
<td></td>
<td>0.001</td>
<td>0.003</td>
<td>0.0005</td>
</tr>
<tr>
<td>IDEA</td>
<td>5.58</td>
<td>11.7</td>
<td>0.003</td>
</tr>
<tr>
<td></td>
<td>0.0056</td>
<td>0.001</td>
<td></td>
</tr>
</tbody>
</table>

Table 4. Speak Freely 7.1 (Linux RedHat 6.0)

<table>
<thead>
<tr>
<th>Computing Time (ms)</th>
<th>CODEC</th>
<th>CODEC</th>
<th>CODEC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GSM</td>
<td>ADPCM</td>
<td>LPC-10</td>
</tr>
<tr>
<td>Blowfish</td>
<td>2.25</td>
<td>5.32</td>
<td>0.52</td>
</tr>
<tr>
<td></td>
<td>0.0002</td>
<td>0.0005</td>
<td>0.0001</td>
</tr>
<tr>
<td>IDEA</td>
<td>5.58</td>
<td>11.7</td>
<td>0.003</td>
</tr>
<tr>
<td></td>
<td>0.0056</td>
<td>0.001</td>
<td></td>
</tr>
</tbody>
</table>

Table 5. Speak Freely 7.1 (Windows 98)

<table>
<thead>
<tr>
<th>Computing Time (ms)</th>
<th>CODEC</th>
<th>CODEC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GSM 8</td>
<td>GSM lite 4.4</td>
</tr>
<tr>
<td>Blowfish</td>
<td>2.27</td>
<td>2.09</td>
</tr>
<tr>
<td>CAST</td>
<td>2.19</td>
<td>2.08</td>
</tr>
<tr>
<td>3DES</td>
<td>7.2</td>
<td>6.35</td>
</tr>
<tr>
<td></td>
<td>0.02</td>
<td>0.06</td>
</tr>
<tr>
<td></td>
<td>0.05</td>
<td>0.002</td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>0.14</td>
</tr>
</tbody>
</table>

Table 6. PGPfone 2.1 (Windows 98)

<table>
<thead>
<tr>
<th>Computing Time (ms)</th>
<th>LPC-10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blowfish</td>
<td>0.32</td>
</tr>
<tr>
<td>IDEA</td>
<td>0.48</td>
</tr>
<tr>
<td>3DES</td>
<td>0.84</td>
</tr>
<tr>
<td></td>
<td>0.0004</td>
</tr>
<tr>
<td></td>
<td>0.004</td>
</tr>
<tr>
<td></td>
<td>0.0009</td>
</tr>
</tbody>
</table>

Table 7. Nautilus 1.5a (Linux RedHat 6.0)
Moreover, it is worth mentioning that in the case of ADPCM and LPC-10, the quantity of data compressed in a second of audio conversation is few thousands of bytes and few hundreds of bytes, respectively. Indeed, in the case of ADPCM we can observe an overhead of the encryption phase of several ms, especially in the case of 3DES; on the other hand, if we resort to LPC-10, the encryption algorithms experience a computational cost closer to the performance of BoAT, but with a poorer quality of the transmitted speech.

To conclude this section, we can summarize the obtained results by observing that we are able to add security modules to the audio data flow pipeline without compromising the overall end-to-end delay. 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 mechanism with security features in a natural and cheap way. On the other hand, the cost of the addition of security modules to the infrastructure of purpose-made application-level tools (such as Nautilus, Speak Freely, or PGPfone) is expressed by the higher overhead reported in the experimental results presented in this section.

5 Conclusion

In this paper we have presented an adaptive packet audio control algorithm equipped with security modules in order to guarantee a secure transmission of audio over the Internet. The original adaptive playout adjustment scheme has revealed its adequacy to be enriched with security services with a negligible cost. The main reason is that its form of adaptiveness (in particular, the handshaking protocol) allowed us to realize a lightweight mechanism for the secure transmission of real time audio over untrusted public networks.

We have proposed two schemes of the mechanism with different computational overhead and security conditions. As far as the second scheme is concerned, it would be interesting to carry out a complexity analysis of an algorithm devoted to the reconstruction of the conversation in lack of the timestamps piggybacked to the audio packets, in order to compare the (real time) secrecy features of the proposed schemes.

With respect to other application-level tools proposed for the secure transmission of audio over the Internet (see e.g. [11, 28, 27]), our mechanism exploits (i) stream ciphers for the encryption and (ii) a synchronization phase in order to frequently change the session key and, as a consequence, to improve the security level of the encryption algorithm. For instance, we could adopt the A5/1 stream cipher algorithm used in GSM, with guarantees on the real time secrecy, authentication, and integrity of the audio packets played out at the receiver site (unlike the GSM protocol, which has been subjected to real time cryptanalysis). Finally, a comparison with the above mentioned application-level tools has put in evidence that our mechanism is adequate from the performance viewpoint, as it requires a negligible computational overhead and guarantees a good QoS.

Acknowledgements

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

References


