Efficient Multicast Packet Authentication

Alain Pannetrat, Refik Molva
Institut Eurécom
{Alain.Pannetrat@eurecom.fr, Refik.Molva@eurecom.fr}

Abstract

Providing authentication mechanisms for IP-Multicast streams is paramount for the development of large scale commercial multicast content delivery applications. This need is particularly strong for the delivery of real time content, such as live video/audio news events or financial stock quote distribution. However, this turns out to be a quite challenging problem for many reasons. First, the authentication of the multicast data must be verifiable by a potentially very large number of untrusted recipients. Second, since multicast communication protocols are almost always best effort, the authentication mechanisms needs to authenticate received content despite the potential loss of some packets. Finally, the authentication mechanism needs to be efficient enough to cope with real time data and should have a small communication overhead.

We propose a new multicast authentication scheme designed to authenticate real time multicast packet streams with a potentially unlimited number of recipients. This scheme provides both integrity and nonrepudiation of origin, and in a majority of situations, it performs with less overhead in bytes per packet than previously proposed practical real time stream authentication schemes.

1 Introduction

IP-Multicast [8] allows the scalable delivery of packets to a potentially unlimited number of recipients. As such, it is a very interesting mechanisms for commercial applications that deliver streamed content to a large group of recipients, such as video/audio broadcasting. However, some security issues need to be solved[12] before these application are deployed on a large scale. The most basic needed security mechanisms for large scale commercial multicast applications are confidentiality and authentication. In fact, the key distribution algorithms employed in many multicast confidentiality proposals[16, 30, 23, ...] require a form of authentication to assure that the keys originate from a legitimate key distribution entity. Consequently, we argue that authentication is probably the most needed multicast security mechanism.

To allow packets to be authenticated in a stream, the source must add authentication information to the distributed content. This authentication information is used by recipients to ascertain the origin of the transmitted content. In the context of multicast authentication, we distinguish two types of distributed contents: *pre-recorded* and *real time*. Pre-recorded content describes content that is known in advance to the source, such as a film or music. For such content, the authentication information can be computed and inserted in the stream in advance. On the other hand, real time content describes content that is produced in real time such as live sports event broadcasting, news events or financial stock quotes. Real time content requires some of the authentication information to be computed in real time, which adds further constrains on the efficiency of the authentication algorithm. Thus, an efficient real time authentication algorithm can be used for pre-recorded data while the converse is not necessarily true. Moreover, it seems that real time application naturally have a stronger need for authentication. Consider as an example, the disastrous consequence that source impersonation could have for an application such as stock quote distribution, where a malicious entity could generate bogus financial data. The main goal of this work is to provide a multicast authentication with a emphasis on low communication overhead, for real time data applications where a low delay is acceptable and will not be perceived at the message level. For an approach directed more specifically to pre-recorded data, we refer the reader to [9], [15] and [29].

1.1 Two Levels of Authentication

We distinguish two levels of authentication:

Source Authentication: allows a recipient to verify the origin of the content.

Nonrepudiation (of origin): allows the recipient to prove the origin of the data to a third party.

In traditional two party communications, source authentication is provided with efficient symmetric techniques

using a MAC (Message Authentication Code) which relies on a secret key shared between the two communicating parties. On the other hand, nonrepudiation is provided with a digital signature, using asymmetric cryptographic techniques which have a cost that is several orders of magnitude higher than a MAC.

Canetti et al. have proposed a multiparty extension[7] to MACs in the context of multicast, but their scheme has some drawbacks. Most notably, the communication overhead is important and the security of the scheme is only defined up to a coalition of w malicious recipients forging data for a chosen recipient. Recent work from Boneh et al.[4] suggest more generally that extending symmetric MAC techniques in the multicast setting will not be possible without new advances in cryptography. As we will see, except TESLA[22], current practical multicast authentication techniques are not fully built on symmetric techniques but rely instead partially on asymmetric techniques. As a consequence many of these schemes, including ours, also provide nonrepudiation of origin.

1.2 Real Time Multicast Authentication Challenges

There are two main factors which make multicast stream authentication a challenge:

A Multiparty Factor: we have an unlimited number of untrusted recipients.

A Streaming Factor: we want to authenticate data from a potentially infinite stream of packets transmitted over a lossy channel.

The multiparty factor has a strong impact on the security requirements of a multicast authentication scheme. Indeed, a fundamental difference between multicast and two-party authentication is that in multicast we consider the recipients as potential adversaries. This rules out the use of a symmetric MAC key shared between the source and the recipients, because recipients should not be able to impersonate as the source of the stream.

The streaming factor has several design implications. Firstly, we do not view the stream as a unique object that is authenticated all at once, but rather as a sequence of consecutive chunks of data that need to be authenticated individually as they are received. Secondly, recipients should be able to authenticate packets starting from an arbitrary point in the stream or at least on the boundary of a small block of packets. Multicast is often implemented over UDP and assumes only a best effort delivery mechanism and many multimedia multicast applications tolerate losses with a graceful degradation in playback quality. Consequently, one of the most important design requirement of a multicast authentication scheme is the ability to authenticate packets amid losses in the network (for non-lossy streams, see for example [9]).

From all the observation we made, we can establish several parameters to measure the quality of a real time stream authentication scheme:

- **Robustness:** the ability of the scheme to authenticate received data despite losses in the network.
- **Joinability:** the ability of recipients to start authenticating packets from an arbitrary point in the stream.
- **(Server Side) Buffering:** the maximum number of packets that need to be stored on the server to compute robust authentication information.
- **(Authentication) Latency:** the maximum number of additional packets that need to be received before a packet can be authenticated.
- **Computational Cost:** the computational cost of the scheme.
- **Communication Overhead:** the number of bytes per packets which describe the embed authentication information.

Buffering and Latency appear in some situations where authentication information pertaining to a packet is stored in one or several other packets. Ideally we would like a scheme that has perfect robustness, that is joinable on every packet, has no buffering or latency and has an overhead as well as a cost similar to what is found in a MAC scheme. In practice however, such a perfect scheme does not exist and a compromise needs to be found between these parameters.

1.3 Related Work

A straightforward stream authentication method would be to use a public key signature on each packet of the stream. In theory, this is well suited for real time streams and the authentication is joinable on any packet. However, adding a typical 1024 bit signature[28] (or 128 bytes) to every packet represents a consequent overhead, moreover, the computational cost of a public key signature makes such a solution impractical in many scenarios. Consequently stream authentication proposals have taken two approaches, sometimes in combination: design more efficient signature schemes and amortize the cost of signatures over several packets.

Faster digital signatures designed with stream authentication in mind where proposed by Rohatgi [27], as well as Wong and Lam [31]. These proposals come however with a communication overhead that makes them impractical

in many situations. The BiBa scheme proposed by Perrig [21] offers a significantly improved broadcast signature scheme which has a lower computationnal overhead but still a communication overhead that is only slightly lower than a traditionnal public key signature. On the other hand, these schemes including the one in [9], still have the advantage of offering a fully real time authentication (ie. with no delay at all).

A complementary approach is to amortize the signature over several packets in a *block*. The stream is itself divided into many small blocks that have each a unique digital signature that is combined with hash/MAC techniques to authenticate the packets in the block. We refer to these techniques as well as the one we propose in this work as *hybrid* approaches. Wong and Lam proposed one of the first hybrid approaches in their hash tree construction[31], which is robust to any number of losses in a stream but has a consequent overhead per packet, even larger than the size of a digital signature. Instead of being robust to any type of packet loss, recent stream authentication proposals have been designed to adapt to loss patterns that are more specific to the Internet. This allows a significant gain in terms of overhead. First, based on the observation that losses usually occur in bursts in TCP/IP[20], Golle and Mogadugo[10] proposed a scheme that could tolerate (1 or several) bursty loss(es) of at most n packets in a block. Packets are linked together in a "hash chain", the last packet of which is digitally signed. However, the scheme has some drawbacks, and in particular, the transmission of the signature is not clearly addressed. Independently Perrig et al. proposed a more complex "hash chain" construction called EMSS[22] which is adapted to multiple losses and which better addresses signature transmission. Recently, in a scheme called SAIDA[19], which shares similarities with our work, Park et al. used the IDA (information dispersal algorithm) to transmit authentication information pertaining to each block of packets in a stream. We discuss these related proposals more in detail in section 5.

As a complementary approach to their EMSS scheme Perrig *et al.*[22] proposed a very efficient time based stream authentication scheme called TESLA. It provides source authentication but does not offer nonrepudiation, which is not a problem for many applications. Its most interesting feature is that it tolerates arbitrary packet loss with a low overhead. Its main drawback is that it requires all the recipients to establish a loose clock synchronization with the source through a initial unicast exchange which may not be always practical in a large multicast group.

1.4 Overview of our Scheme

Our scheme uses a combination of hash and signature techniques with FEC, or more precisely, erasure codes. The two most employed techniques to achieve reliable delivery of packets in computer communication protocols are ARQ (Automatic Repeat reQuest) techniques and FEC (Forward Error Correction). ARQ techniques are used every day in Internet protocols such as TCP, while FEC techniques have long been confined to the telecommunications world. However, there has been recently a surge in interest for FEC techniques in the Internet world, often in combination with more traditional ARQ approaches[18, 6]. While in the telecommunications world FEC techniques are used most often to detect and correct errors occurring in the transmission of a stream of bits, they are used in the Internet world to recover from the loss of packet sized objects. Indeed, in the Internet world a packet is either received or lost. A packet can be considered lost if it does not arrive after a certain delay or perhaps if it has bad checksum. Our idea was first to use FEC to transmit the signature alone, but we soon realized that FEC could also be used as an alternative to hash trees[31] or chains[22, 10] to transmit authentication information, with lower overhead per packet in most cases than any other scheme suitable for *real time* broadcasts.

The central contribution of this work is the proposal of a *joinable real time robust* stream authentication scheme with nonrepudiation of origin. It uses Erasure Codes to provide a lower overhead per packet than previous real time authentication stream proposals, while being adapted to realistic multicast Internet loss patterns.

A brief overview of erasure codes will be presented in the next section. Our scheme is formalized in section 3 as well as its relationship with Internet loss patterns which are modeled with a Markov chain. Section 4 discusses the cost and overhead of our scheme and presents its use in a few concrete scenarios. Finally, we review other real time lossy stream authentications schemes in section 5 and compare them with our approach.

2 Background

2.1 Erasure Codes

An erasure code generation algorithm $C_{k,r}$ takes a set $X = \{x_1, ..., x_k\}$ of k source packets in a block and produces $(k + r)$ code packets:

$$\{y_1, ..., y_{(k+r)}\} \leftarrow C_{k,r}(X)$$

The main property of the set $Y = \{y_1, ..., y_{(k+r)}\}$ is that any subset of k elements of Y suffices to recover the source data X with the help of a decoding algorithm D_k. To be exact, the decoding algorithm D_k needs to know the position, or index, of the k received elements in Y to recover X. This information can often be derived by other means (such as the packet sequence number) and we will assume in the remaining discussion that this information

is available implicitly to D_k. If the first k code packets are equal to the source packets, that is $\{y_1, ..., y_k\} = X$ where $\{y_1, ..., y_{(k+r)}\} \leftarrow C_{k,r}(X)$, we call the code *systematic* and the extra redundancy packets $\{y_{(k+1)}, ..., y_{(k+r)}\}$ are called parity packets. Systematic codes are very useful since they do not require any additional processing from the recipient in the case where no loss occurs.

It is important to note that Erasure Codes are not used in the same context in the Internet as in telephony. Here the codes are not designed to recover damaged packets but rather the loss of full packets in a block of several packets. Intuitively, an individual packet can therefore be viewed more like a single code symbol rather than a set of symbols. For a good introduction to practical erasure codes we refer the reader to the work of L. Rizzo[26] where Reed-Solomon erasure codes are described. These codes operate in $GF(2^n)$ and may not be efficient for large data blocks of packets (several hundred kilobytes). However, they are suitable in our scenario since we work on data units that are much smaller than a packet (typically 16 or 20 bytes), as shown below. For faster codes, we refer the reader to the work of M. Luby et al. on Tornado Codes [14, 6], where codes with near linear coding and decoding times are described.

In the remaining of this work, $C_{k,r}(.)$ will describe a practical systematic erasure code generation algorithm which takes k source packets and produces $(k + r)$ code packets. If $X = \{x_1, ..., x_k\}$ is the source data and Y are the r extra generated parity packets, we will write $\{X; Y\} \leftarrow C_{k,r}(X)$. The corresponding decoding algorithm will be denoted $D_k(.)$ and if Z describes the set of received elements and X the source data, we will write $X \leftarrow D_k(Z)$ to describe the recovery process.

2.2 Notations

In this work we will consider a stream to be divided in consecutive blocks of b packets. Since a stream does not necessarily exactly contain a number of packets which is an exact multiple of b we allow the use of dummy padding packets at the very end of the stream to match a b packet boundary. Our authentication scheme is parameterized by b the block size in packets and $p \in [0..1[$ the maximum expected loss rate per block.

We will denote H as a cryptographic hash function such as SHA[17] or MD5[25] which produces hashes of h bytes. The couple $(\mathcal{S}, \mathcal{V})$ will denote the digital signature and verification algorithms respectively associated with the source of the packet stream, such as RSA[28, 1] for example. The size of the signatures will be expressed as s bytes. For RSA, a typical value for s is 128 bytes (or 1024 bits).

3 Stream Authentication

3.1 Authentication Tags

Consider a block as a sequence of b packets $[P_1, ..., P_b]$. Let $\{h_1, ..., h_b | h_i \leftarrow H(P_i)\}$ be the set of hash values of these packets with a cryptographic hash function $H(.)$. From this hash set we build a set of b authentication tags $\{\tau_1, ..., \tau_b\}$ with the following algorithm $\mathcal{T}_{[b,p]}$ which uses some of the notations introduced in the previous section:

Tag generation: $\mathcal{T}_{[b,p]}$

INPUT: $\{h_1, ..., h_b\}$

OUTPUT: $\{\tau_1, ..., \tau_b\}$

$\{X; \overline{X}\} \leftarrow C_{b,\lceil pb \rceil}(X)$ (1)

$\sigma \leftarrow \mathcal{S}(H(h_1||...||h_p))$ (2)

$\{Y; \overline{Y}\} \leftarrow C_{\lfloor b(1-p) \rfloor, \lceil pb \rceil}(\overline{X}||\sigma)$

where $Y = \overline{X}||\sigma$ (3)

Split $\{Y; \overline{Y}\}$ into

b equal length tags $\{\tau_1, ..., \tau_b\}$. (4)

We propose a more visual representation of the tag algorithm on figure 1.

We observe that $\mathcal{T}_{[b,p]}$ uses two different erasure codes, in steps (1) and (3). The values $\{Y; \overline{Y}\}$ on line (3) is of total length that is a multiple of b bytes, because we have $b = \lfloor b(1-p) \rfloor + \lceil pb \rceil$. This allows us to divide $\{Y; \overline{Y}\}$ into equal length tags on line (4). To exploit the tag generation algorithm we will first define our authentication criterion:

Authentication criterion: In this work we say that a packet P_i is *fully authenticable* in a block if, given the set of hashes $\{h_1, ..., h_b\}$ of packets in the block and their signature $\sigma = \mathcal{S}(H(h_1||...||h_p))$, we can verify that both $\mathcal{V}(\sigma, H(h_1||...||h_p)) = true$ and $H(P_i) = h_i$.

The proposed schemes in this work are based on the following property of the tag generation algorithm.

Proposition 1. Let $\Pi = [P_1, ..., P_b]$ be a block of b packets and $\{h_1, ..., h_b | h_i \leftarrow H(P_i)\}$ its associated hash set. If we compute $A = \{\tau_1, ..., \tau_b\} \leftarrow \mathcal{T}_{[b,p]}(\{h_1, ..., h_b\})$ then any subset of at least $\lfloor b(1-p) \rfloor$ packets in Π can be authenticated using any subset of at least $\lfloor b(1-p) \rfloor$ tags in A.

Proof. Define $r = \lfloor b(1-p) \rfloor$. Let $\Pi' = [P_{e_1}, ..., P_{e_r}]$ be a subset of r packets in Π and let $A' = [\tau_{a_1}, ..., \tau_{a_r}]$ be a subset of r packets in A. We can compute $\{\overline{X}||\sigma\} \leftarrow D_{\lfloor b(1-p) \rfloor}(A')$ since A' contains $r = \lfloor b(1-p) \rfloor$ elements. Let $E = \{h_{e_1}, ..., h_{e_r} | h_{e_i} \leftarrow H(P_{e_i})\}$ be the hashes of the received packets. We can recover

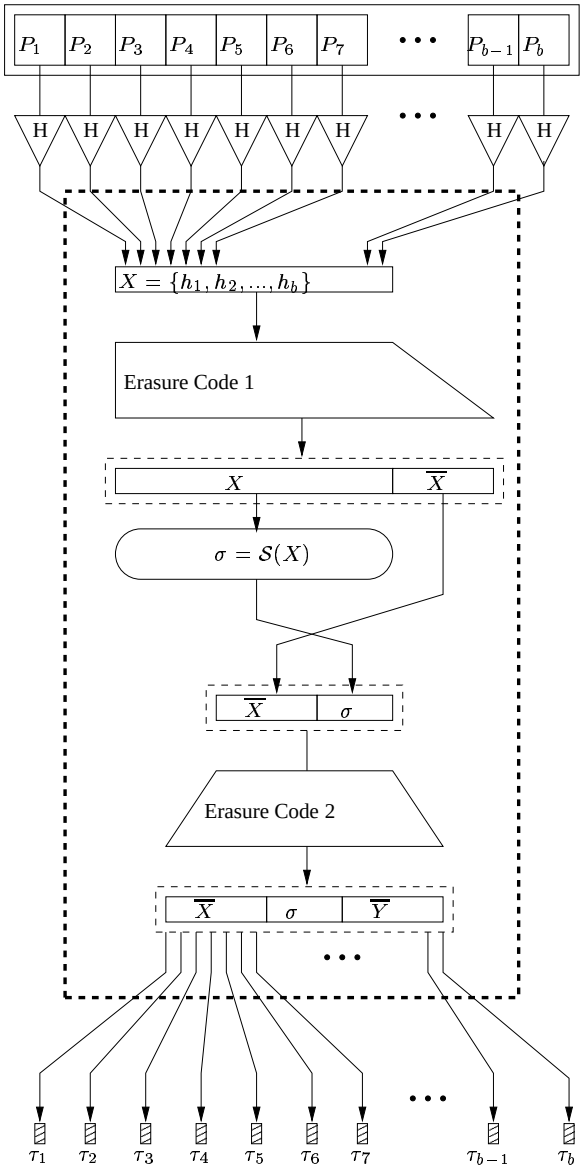


Figure 1. the tag generation algorithm

$\{h_1, ..., h_b\}$ form B and $\overline{X}$ by computing $\{h_1, ..., h_b\} \leftarrow D_b(E||\overline{X})$. Finally we can compute $\mathcal{V}(\sigma, \{h_1, ..., h_b\})$ to authenticate the received packets Π' to verify our authentication criterion. $\Diamond$

A direct corollary of the proposition above is that both a block of packets and their authentication tags can withstand a loss rate of at most $\lceil pb \rceil$ elements while allowing us to authenticate the remaining packets.

Finally, from the construction of the algorithm above we can determine the size of an authentication tag:

Proposition 2. Let h define the length of our cryptographic hashes and s the size of the signatures. The size of an individual authentication tag is expressed as a function $\delta(b, p)$ of both the number of packets in a block and p the maximum expected loss rate per block, as follows:

$$\delta(b, p) = \frac{\mathcal{R}_{\lfloor (1-p)b \rfloor}(s + \lceil p.b \rceil \ h)}{\lfloor (1-p)b \rfloor}$$

where $\mathcal{R}_n(z)$ is an integer function which returns the lowest multiple of n greater or equal to z.

Proof. Let y denote the size of the value $\{Y; \overline{Y}\}$ and x the size of $\overline{X}||\sigma$ padded to the proper length, both on line (3) of the algorithm. We have $\delta(b, p) = y/b$. From the erasure code parameters on line (3) we have $y = x \frac{\lfloor (1-p)b \rfloor + \lceil p.b \rceil}{\lfloor (1-p)b \rfloor} = x \frac{b}{\lfloor (1-p)b \rfloor}$ and thus $\delta(b, p) = \frac{x}{\lfloor (1-p)b \rfloor}$. The value of x is the the sum of the size of $\overline{X}$ and the signature σ, *padded to the appropriate length* for the erasure code of line (3). From line (1) we compute the size of $\overline{X}$ as $\lceil p.b \rceil \ h$ and write s as the size of σ which yields $x = \mathcal{R}_{\lfloor (1-p)b \rfloor}(s + \lceil p.b \rceil \ h)$. $\Diamond$

3.2 Proposed Schemes

In our stream authentication scheme we propose to piggyback authentication tags in the packets of a block and use Proposition 1 to authenticate received packets when the loss rate in a block is less than p. We propose 3 different variants of our scheme which only differ by the positioning of the authentications tags.

In this section we will denote a stream as a set of m blocks $B_1, ..., B_m$. The individual b packets in each block B_i are identified as $P[i, 1], ..., P[i, b]$. The corresponding authentication tags are identified as $\tau[i, 1], ..., \tau[i, b]$. The packets $P[i, j]$ are a combination of just two things: stream data packet $D[i, j]$ and an authentication tag.

ECU: The unbuffered sender scheme. In this scheme we use packets in a block $B_{(i+1)}$ to piggyback authentication tags pertaining to block B_i. The j^{th} packet in a block B_i is thus defined as $P[i, j] = \{D[i, j]||\tau[i-1, j]\}$. This requires the sender to create an extra padding dummy

block $B_{(m+1)}$ to allow the last block B_m to be authenticated. This scheme has the particularity that it does not require any stream data packet buffering from the sender, only the hashes of the packets in the current block need to be stored by the sender who can then compute the necessary authentication tags to be piggybacked in the next block. In this sense, this scheme is truly an *real time* authentication scheme. The tradeoff of this construction is that the receiver will experience a latency of two blocks in the worst case before he can authenticate the first packet in a blocks he received.

This construction creates a dependency between two consecutive blocks, thus in the event of a loss that exceeds the threshold p and in particular if a whole block B_i is lost than we will not be able to authenticate $B_{(i-1)}$.

An interesting aspect of the ECU scheme is that it also gives an extra amount of time for the sender to compute the signature of a block and the second authentication code. Recalling line (3) of the tag generation algorithm we have $\{Y;\overline{Y}\} \leftarrow C_{\lfloor b(1-p)\rfloor,\lceil pb\rceil}(\overline{X}||\sigma)$ where $\{Y;\overline{Y}\}$ is split in b authentication tags. Accordingly we can rewrite $\{Y;\overline{Y}\}$ as $\{\overline{X}||\sigma;\overline{Y}\}$, thus the first $l_{|\overline{X}|}$ authentication tags will contain elements representing $\overline{X}$, then the next group of $l_{|\sigma|}$ tags will represent the signature σ and finally the last group of tags will represent the $b - l_{|\overline{X}|} - l_{|\sigma|}$ associated parities. Consequently, the first authentication coding operation on line (1) of our algorithm needs to be produced before sending block $B_{(i+1)}$, however, the signature on line (2) only needs to be computed after the first $l_{|\overline{X}|}$ packets of $B_{(i+1)}$ and the second code on line (3) only needs to be ready after the $l_{|\overline{X}|} + l_{|\sigma|}$ first packets of $B_{(i+1)}$.

EC2: The double buffer scheme. Instead of piggybacking tags in the next block, we examine the possibility of piggybacking tags in the previous block. In other words, the tags of block B_i are put in packets of block $B_{(i-1)}$ and packets in a block B_i are defined as $P[i,j] = \{D[i,j]||\tau[i+1,j]\}$. This requires the sender to create an extra padding dummy block at the beginning of the data stream. The main advantage of this construction is that the receiver can authenticate each received packet immediately upon reception. The main drawback of this scheme is that it requires the sender to buffer two blocks at a time. In this sense it is not a truly *real time* scheme but in some applications, our double buffering is still acceptable.

This construction also creates a dependency between blocks similar to ECU, with similar consequences.

EC1: The single buffered scheme. The most obvious construction and perhaps the one that offers the best compromise between the sender buffering and the receiver authentication latency is to piggyback the tags of a block B_i in the block B_i itself. Packets in a block are simply defined as $P[i,j] = \{D[i,j]||\tau[i,j]\}$. This scheme requires the sender to buffer one block and adds a maximum verification latency of one block for the receiver.

A advantage of this scheme is that it does not create a dependency between blocks, thus if a block losses packets beyond the expected maximum loss rate p, the authentication of neighboring blocks in the stream remains unaffected.

3.3 Parameter Choice

Until now we proposed a method which can authenticate a block when a threshold of less than pb packets are lost in a block of b packets. However we need to relate these parameters to concrete average network loss patterns and we will now discuss the choice of the two main parameters of our scheme: b the block size and p the maximum loss rate per block.

The goal of an hybrid scheme is to amortize the cost of a signature over several packets. Thus the greater the block size, the less often we will need to compute a signature. On the other hand the block size influences the authentication latency and/or the sender buffer size, depending on which scheme is chosen. The EC2 has the lowest possible authentication latency (1 packet) but the biggest buffering, whereas ECU has no sender-side packet buffering but a maximum 2 block authentication latency. As we said above, EC1 seems to be a good compromise in most situations with both a buffering and a maximum authentication latency of one block. Once a scheme is chosen, we recommend to choose the largest possible block size b within the constraints of the application authentication latency requirements.

The parameter p depends on the loss pattern of our network. There has been quite a few studies about Internet loss patterns for applications such as Audio Unicast/Multicast [2], Internet Telephony[3], Multicast [32, 33], TCP[20] TCP/UDP[5]. These studies differ on their analysis and their scope, however there is a general consensus among most studies that:

1. Packet losses are not independent. When a packet is lost the probability that the next packet will be lost increases, which means that losses in the Internet are often *bursty*.

2. However the majority of bursts are small (from 1 to 6-7 packets).

3. There are some very rare long bursts, lasting up to a few seconds (In [5] the authors suggest that these bursts could attributed to network disruption or maintenance).

In this work, we propose to refer to a model often suggested to describe bursty losses in Internet traffic which is a simple 2 state Markov chain [3, 34] also called the Gilbert model, where state 0 represents a packet received and state 1 a packet lost by the recipient. If r denotes the probability of going from state 0 to state 1 and q the probability of going from state 1 to state 0 we have the following transition matrix[11]:

$$M = \begin{bmatrix} (1-r) & r \\ q & (1-q) \end{bmatrix}$$

This model simulates well the fact that the loss probability of packet increases when the previous packet is lost ($r < 1 - q$), rather than being uncorrelated ($r + q = 1$). The probability that k consecutive packets are lost is equal to $(1-q)^{k-1}q$ which describes a geometric distribution of mean $\mu = 1/q$. According to [3], the head of the distribution seems to model Internet loss patterns well with some inaccuracies in the tail. But in any case, if a very long burst rarely occurs, with extremes such as those stated in point 3 above, it does not make sense to invest much effort to make our scheme robust for those bursts since most the data that needs to be authenticated is likely to be lost itself. The long term average loss rate π_1 is given by solving the equation $(\pi_0, \pi_1).M = (\pi_0, \pi_1)$, which yields $\pi_1 = \frac{r}{r+q}$. We further note that Perrig *et al.*[22] as well as Park *et al.[19]* have used this model in their own stream authentication schemes.

The strategy we followed in this work was first to choose b, then to simulate a Markov chain over a very large number of blocks and adjust the parameter p such that most blocks would be verifiable (we chose an arbitrary value of 99% verifiable blocks). The Markov chain parameters were derived from μ: the average loss rate and π_1: the average burst length. Note that here the number of losses in a block of b packets can be successfully modeled as a the number of successes in trials of a Bernoulli process with parameter π_1, which is approached by the normal distribution. This approximation could also give us some analytical results but we found the simulations to be more informative.

4 Discussion

4.1 Computational Cost

Our scheme involves 3 types of operations:

- cryptographic hash computations.
- a digital signature.
- 2 coding and decoding operations.

For each block, the source needs to compute b hash operations, a digital signature (which includes a hash), and generate the 2 codes. Here, the hashing and signing costs are equivalent to other hybrid schemes such as EMSS[22] or Hash Chains[10]. The amount of computation done by the recipient depends on the losses in the network. In an ideal situation we just computes b hashes and verifies a signature. If packets are lost some additional decoding operations will be needed. The codes are used to recover hashes of packets, rather then the packets themselves, thus we will be manipulating small amounts of data. In traditional uses of Erasure Codes, the packets size L is typically over a thousand bytes, while here, we are looking at figures ranging from $L = 1$ to $L = 150$ bytes in the most extreme cases.

If we take a simple Reed-Solomon Erasure Code[26], the computational decoding cost is $\mathcal{O}(m.e.L)$ where m is the number of original message packets, and e the additional parities needed (corresponding to the loss) and L the size of a packet. The coding cost is similarly in $\mathcal{O}(m.k.L)$ where k is the number of parities.

For demanding situations, we can turn to more efficient codes such as Tornado Codes[14]. These codes are probabilistic and come with what is called a slight "decoding inefficiency": $(1 + \varepsilon)m$ packets are needed to recover m original packets with high probability. These codes use the binary XOR operation as a basic operation as opposed to Galois Field operations in the Reed Solomon case, thus we achieve very efficient coding and decoding times of $\mathcal{O}((m + k)\ln(1/\varepsilon)L)$. Note that the use of tornado codes would thus conduct us to modify our definitions in section 3 to take the decoding efficiency into account. However, in [6] significant values of $\varepsilon \approx 0.05$ are considered, thus the results we propose in this work should not be significantly different with such a small overhead increase if we use Tornado Codes.

Compared to other hybrid real time authentication streams, the main tradeoff of our scheme is in the is the additional computational cost generated by the erasure code. However, since we are operating on small code packet size, the cost over a block should remain very reasonable. We will show in the next section that the substantial gain we can achieve in terms of overhead per packet is clearly worth the extra computational effort.

4.2 Overhead

4.2.1 Evaluation

The overhead in bytes per packet of our 3 schemes is uniquely defined by the size of an authentication tag. Thus, recalling Proposition 2 in section 3 we can express the overhead as a function $\delta(b, p)$ of the maximum expected loss rate per block p and the number b of packets in a block:

$$\delta(b,p) = \frac{\mathcal{R}_{\lfloor (1-p)b \rfloor}(s + \lceil p.b \rceil\ h)}{\lfloor (1-p)b \rfloor}$$

where $\mathcal{R}_n(z)$ is an integer function which returns the lowest multiple of n greater or equal to z.

We would like to emphasize again that this overhead *includes the signature overhead*. Table 1 presents a sampling of $\delta(p,b)$ for different values of p and b, with $s = 128$ bytes (1024 bit RSA) and $h = 16$ (MD5[25]). Note that $\delta(b,p)$ remains surprisingly small if either b large or p is reasonably low.

$p \backslash b$	16	32	64	128	256	512	1024
0.05	10	6	4	2	2	2	1
0.10	12	7	5	3	3	3	2
0.25	16	11	8	7	6	6	6
0.50	32	24	20	18	17	17	17
0.75	80	64	56	56	50	49	49

Table 1. Overhead bytes per packets for different values of p and b

4.2.2 Case studies

To be more concrete we applied our scheme to the two case studies Perrig *et al.* propose in their work for the EMSS[22] real time stream authentication scheme. We recall their first case study:

A municipality wishes to collect traffic information from sensors distributed over the streets. The system requirements are as follows:

- The data rate of the stream is about 10 Kbps, about 20 packets of 64 bytes each are sent every second.
- The packet drop rate is at most 5% for some recipients, where the average length of burst drops is 5 packets.
- The verification latency should be less than 10 seconds.

We propose to use the ECU scheme since the sensors may have limited memory, thus the verification latency of 10 seconds allows us to use a block of 100 packets (200/2 since a block is authenticated by the next one). Given the drop rate and the average length of bursts, we constructed a corresponding two state Markov chain with $r = 0.010526, q = 0.2$ and simulated it over 10000 blocks of 100 packets. For Markov chain simulation techniques we referred to Häggström[13]. We found that 99% of those blocks experienced a loss less than 27 packets, thus we decided to choose $p = 0.27$. The overhead[1] per packet is then only $\delta(100, 0.27) = 8$ bytes !

The second case study proposed by Perrig *et al.* is related to real-time video broadcasting, with the following requirements:

- The data rate of the stream is about 2Mbps, or 512 packets of 512 bytes each every second.
- The packet drop rate is at most 60% for some recipients, with an average length of burst drops of 10 packets.
- The verification latency should be less than 1 second.

We propose again the EC1 scheme and because of the verification latency, we have to limit b to 512 packets. We simulated the corresponding Markov model over 10000 blocks and found that 99% of those blocks experienced a loss of less than 375 packets. We decided to choose $p = 0.73 = 375/512$, which gives us an overhead per packet of $\delta(512, 0.73) = 45$ bytes.

	loss rate	av. burst length	b	p	$\delta(p,b)$
Example 1	5%	5	100	0.27	8
Example 2	60%	10	512	0.73	45
Example 3	10%	3	32	0.47	22
Example 4	10%	50	512	0.50	18
Example 5	80%	10	200	0.905	160
Example 6	5%	5	1024	0.1	2

Table 2. A few case studies.

As a complement to the two proposed scenarios above, Table 2 shows a few of our other simulation results, following the same approach as above for different average burst loss lengths and loss rates. Example 1 and 2 simply repeat the two case scenarios above. Example 3 shows that with a small block size, parameter p is significantly higher than the network loss rate. Similarly, an extreme average burst length increases the value of p as shown in example 4. Finally we have two extreme examples of the parameters of our scheme: first in a very lossy network which requires 160 bytes of overhead per packet which more than the size of a public key signature, and to finish we have an ideal case, with a small loss and a long block size which gives us a surprisingly low overhead per packet of 2 bytes !

4.3 Denial of Service

In their work on stream authentication for pre-recorded streams[15], Miner and Staddon briefly discuss the use

[1] If we had chosen the EC1 scheme instead, we would have $b = 200$, $p = 0.2$ and $\delta(b,p) = 5$.

of Erasure Code techniques as an additional robustness mechanism. Their objective is different from ours here, since they use Erasure Codes as a mean to "reinforce" their "hash and MAC chain" rather than as a substitute as we do. However, they make an interesting remark that Erasure Code techniques may be vulnerable to "Denial of Service" since an adversary who modifies the transmitted parities may render the authentication of the received packets impossible. We observe that this remark is also valid for our scheme: if a some packets are lost and if an adversary modifies the tags piggybacked on the data packets the verification process may not function properly if the decoding algorithm requires those parities. In our scheme, the authentication information used for one block is not used to compute the authentication information of another block, and thus there is no authentication chain across several blocks. This implies that a DoS that affects a block in our scheme will not impact another block and will not disrupt the rest of the communication stream.

Other stream authentication schemes based on a chaining mechanism are less vulnerable to this type of DoS attack. In protocols such as Hash Chains[10] and in EMSS[22] several signature packets are sent to authenticate a block. An adversary thus needs to modify several more packets than in our scheme to prevent a recipient from authenticating a block.

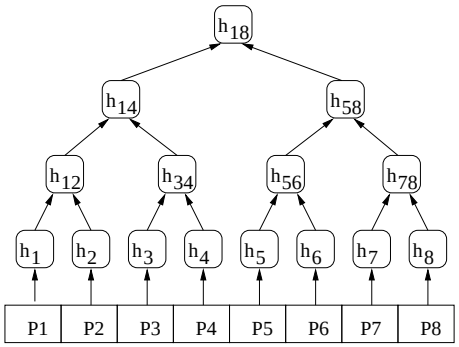
The stream authentication schemes that are the most resistant to DoS are the ones that include a signature with each packet such as [31], [9], [27] and [21]. In those schemes a modified packet can be discarded immediately if the signature verification fails.

5 Comparison

5.1 Hash Trees

Wong and Lam[31] proposed the construction of hash trees, in a scheme that can authenticate received packet in a block no matter how many packets are lost. In their most interesting scheme, using a cryptographic hash function H, they construct a complete balanced binary tree, where the leafs are the hashes h_i of the packets P_i in the block and the other vertices are hashes of their two children as shown on the example on figure 2.

The source computes a signature of the value representing the root of the binary tree and sends it to the recipients. Each data packet P_i is augmented with the minimum set A_i of complementary values it needs to recompute the value associated to the root of the tree. This set A_i is the set of vertices that are siblings to all the vertices on the path from h_i to the root of the tree. For example on figure 2, packet P_2 is sent with $A_2 = \{h_1, h_{34}, h_{58}\}$ and the recipient can verify the signature of the root $h_{18} = H(H(H(h1|H(P_2))|h_{34})|h_{58})$. To allow the received



(a)

Figure 2. Authentication tree for an 8 packet block

packets to be authenticated independently (and make the scheme joinable on any packet), the authors of [31] suggest to append the signature of the root of the tree to every packet which leads to an overhead per packet of $s + (\ln_2(b) - 1).h$ bytes. This schemes has thus an even larger overhead per packet than the "sign each" approach, though the signature only needs to be computed once for each block. Just like EC1 (and EC2), the scheme requires the sender to buffer the whole block before the first packet of that block can be sent.

5.2 Hash Chains

Based on the observations of Paxson[20] who conducted a large scale survey of TCP/IP Internet communications and who showed that losses often occur in bursts, Golle and Modadugu[10] proposed a stream authentication mechanisms designed to tolerate the loss of packets in bursts of at most β packets in a block. They construct a directed acyclic graph between the packets of the block, by putting the cryptographic hash of a packet in one or several other packets. If a packet P is signed then any packets P' for which there exists a path in the graph joining P' to P can be authenticated. In their work, Golle and Modadugu propose methods to design such acyclic graphs in an optimal way regarding bursty packet losses. Their simplest scheme is constructed as shown on the example of figure 3: the hash of a packet P_i is stored both as part of the following packet P_{i+1} and as part of $P_{i+1+\beta}$. Finally the hashes of the last $(\beta + 1)$ packets are sent, along with a signature of these $(\beta + 1)$ hashes for verification.

The same authors further refined their hash chain construction, to create "Augmented Chains", which require to buffer a few packets, but allows a smaller set of hashes to be signed at the end. The principle remains the same and we refer the reader to their work [10] for details. It

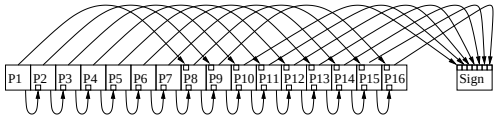


Figure 3. Augmented chain resisting to bursts of 6 packets in a 16 packet block.

is worth noting that their first scheme can tolerate several bursts in a block while the augmented chain construction may have difficulties in some situations if there are several bursts in the same block, consequently we will focus on their first scheme in this comparison.

Hash Chain Overhead: The authors of [10] do not detail how to choose β nor do they provide a clear method to deal with signature loss except to suggest the transmission of several copies of the signature. If these signatures are transmitted far enough apart, we can consider that their loss probabilities are uncorrelated. If we assume that γ signatures are transmitted, we can approximate the cost of the hash chain construction as $\delta_{HC}(\gamma, b) = \gamma\frac{(\beta+1).h+s}{b} + 2h$ bytes per packets, with the notations already used throughout this work. The size of b is essentially constrained by the authentication latency, which here is at most the distance between the first packet of the block and the γ^{th} redundant signature that is transmitted for that block. Since the simple hash chain construction is not sender side buffered similarly to ECU, the γ signatures pertaining to a block are transmitted after the last packet of that block.

Recalling the Markov chain model of section 3 we know that the probability that a burst of k lost packets occurs is $q.(q-1)^{(k-1)}$ with an average length of $1/q$ packets in a burst. Consequently we will choose β in the hash chain such that the probability that a burst exceeds β is low, for example such that $1 - \sum_{k=(\beta+1)}^{\infty} q(1-q)^{k-1} \leq 99\%$. If we refer to the two case studies we borrowed from EMSS in section 3, we would have:

- **Case 1:** We propose $b = 160$, $\gamma = 2$, $\beta = 21$ since $q = 0.2$. We would transmit the first signature at the end of the block and the second signature 20 packets later (1 second). The probability that one of the signature arrives is approximately $1 - 0.05^2 = 0.9975$ and the overhead per packet is $\delta_{HC}(\gamma, b) \approx 38$ bytes.

- **Case 2:** This case is more problematic because the network is extremely lossy and the signature has a high probability of being lost. Indeed if we take $\gamma = 8$ the probability that one redundant signature at least arrives is $1 - 0.6^8 \approx 0.99$ (if we take $\gamma = 4$ the signature arrival probability is lowered to 0.87). But this means that each block is transmitted along with 4 to 8 signatures and it becomes difficult to define a reasonable size for $b < 512$. If we choose b small then we need to compute several signatures per second and we need to send several copies of each them during the same time (without a guaranty that losses will be independent). If we choose b larger then the probability of authenticating a packet within the authentication latency becomes lower. As a indication, if $b = 256$, $\gamma = 8$, $\beta = 43$ since $q = 0.1$, we have $\delta_{HC}(\gamma, b) \approx 58$ bytes.

No matter how good the network conditions are and no matter how long the block size is, the hash chains have at least an overhead of $2.h$ per packets (with perhaps 1 or 2 extra bytes for the signature). Comparatively, our scheme has clearly a lower overhead when the network is not too lossy, with such extremes shown as in Example 6 in table 2. For more lossy streams, our scheme maintains a high authentication probability despite the losses, without encountering the problems we described here in Case 2.

5.3 EMSS

Perrig et al. used a similar hash chain idea in their EMSS[22] scheme. Their work is targeted at more general loss patterns and proposes a method to deal with signature loss. As opposed to the work of Golle and Modadugu which uses a deterministic edge relationship pattern among the packets in the chain, the EMSS scheme uses randomly distributed edges. Moreover, packets are chained across blocks, thus event if all the redundant signatures pertaining to a block are lost the signature in the next block can be used to authenticate the data (extending the authentication latency). They performed several simulations in order to tune the right number of hashes to include in each packet depending on the loss characteristics of the stream. The signature of a block is transmitted several times to allow it to reach the recipient with high probability, depending on the characteristic of the network.

Since we borrowed our 2 test cases directly from EMSS, we can recall their results here as a comparison. The simulations conducted in the EMSS scheme, give an overhead of 22 bytes in the traffic information scenario (with an average verification probability per packet of 98,7%) and an overhead of 55 bytes in the video stream scenario (with a minimum verification probability of 90%). In the latter scenario, the signature of a block alone which is transmitted twice only has an estimated probability of arrival of $0.64 \approx 1 - 0.6^2$, but since there is linking between blocks a packet may be verified by the signature of future blocks, however in this case we understand that the verification latency limit of a packet will be exceeded. We would also like to highlight that their scheme used 80 bit key-less hashes while we use 128 bit hashes (MD5). A

similar value in our scheme would have given an even lower overhead per packet and also a lower overhead in the Hash Chain construction.

Despite longer hashes, in both cases, our scheme has lower overhead and a higher probability of block verification within the required maximum authentication latency.

5.4 SAIDA

Recently, J. Park et al. proposed a stream authenticastion scheme called SAIDA[19] which shares a lot of similarities with our work. The skeleton of their scheme is similar to the ECU scheme : they devide the stream into blocks and compute the hashes of each packet in a block, next they use the IDA (information dispersion algorithm[24]) to compute authentication information with added redundancy. This authentication information is then piggybacked in the next block. Provided that a certain threshold of (1-*p*).*b* out of *b* packets are recieved in a block, the reciepients can recover the hashes of the packets and the signature of a block.

There is however an important difference with our scheme : in SAIDA, *only* the piggybacked authentication information is used to recover the hashes and the signature of a block. In our scheme some hash values are computed from the recieved blocks themselves. By definition, this allows our scheme to have a lower communication overhead per packet than SAIDA. Using the previously introduced notations in this work, the overhead in bytes per packet of SAIDA can be computed as $\frac{\mathcal{R}_{\lfloor(1-p)b\rfloor}(s+bh)}{\lfloor(1-p)b\rfloor}$ versus $\frac{\mathcal{R}_{\lfloor(1-p)b\rfloor}(s+\lceil p.b\rceil h)}{\lfloor(1-p)b\rfloor}$ in our scheme. This advantage can be attributed to the use of a two level erasure code in our scheme, where SAIDA uses only a single redundancy code.

Conclusion

In this work we propose a new approach to real time lossy stream authentication, which is joinable on block boundaries. Though the general idea of using erasure codes in an authentication scheme is not completely new in itself, we believe that our scheme uses them in a quite novel way. Where previous proposals used hash linking, we use erasure codes to achieve a lower communication overhead per packet. Moreover, we propose a concrete mechanism describing how to transmit the authentication information as well as the digital signature associated to a block with equivalent recovery probabilities. We proposed buffered and unbuffered variations of our scheme which offer an interesting alternative to other real time stream authentication mechanisms in many situations.

References

[1] M. Bellare and P. Rogaway. Optimal asymmetric encryption. In A. D. Santis, editor, *Advances in Cryptology - EuroCrypt '94*, pages 92–111, Berlin, 1995. Springer-Verlag. Lecture Notes in Computer Science Volume 950.

[2] J. Bolot and H. Crépin. Analysis and control of audio packet loss over packet-switched networks. Technical report, INRIA, 1993.

[3] J.-C. Bolot, S. Fosse-Parisis, and D. F. Towsley. Adaptive FEC-based error control for internet telephony. In *INFOCOM (3)*, pages 1453–1460, 1999.

[4] D. Boneh, G. Durfee, and M. Franklin. Lower bounds for multicast message authentication. In *Theory and Application of Cryptographic Techniques*, pages 437–452, 2001.

[5] M. Borella, D. Swider, S. Uludag, and G. Brewster. Internet packet loss: Measurement and implications for end-to-end QoS. In *International Conference on Parallel Processing*, Aug. 1998.

[6] J. Byers, M. Luby, M. Mitzenmacher, and A. Rege. A digital fountain approach to reliable distribution of bulk data. In *proceedings of ACM SIGCOMM '98*, September 1998.

[7] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas. Multicast security: A taxonomy and some efficient constructions. In *Proceedings of IEEE Infocom'99*, 1999.

[8] S. E. Deering. RFC 1112: Host extensions for IP multicasting, Aug 1989.

[9] R. Gennaro and P. Rohatgi. How to sign digital streams. In B. S. K. Jr., editor, *Advances in Cryptology - Proceedings of CRYPTO 97*, volume 1294 of *Lecture Notes in Computer Science*, pages 180–197, Santa Barbara, California, USA, August 1997. Springer Verlag.

[10] P. Golle and N. Modadugo. Streamed authentication in the presence of random packet loss. In *to appear in NDSS 2001.*, 2001.

[11] C. M. Grinstead and J. L. Snell. *Introduction to Probability*. McGraw Hill, 2000.

[12] T. Hardjono and G. Tsudik. IP multicast security: Issues and directions. *Annales des Telecommunications*, to appear in 2000.

[13] O. Häggström. *Finite Markov Chains and Algorithmic Applications*. Cambridge, June 2002.

[14] M. Luby, M. Mitzenmacher, M. A. Shokrollahi, D. A. Spielman, and V. Stemann. Practical loss-resilient codes. In *ACM Symposium on Theory of Computing*, pages 150–159, 1997.

[15] S. Miner and J. Staddon. Graph-based authentication of digital streams. In *2001 IEEE Symposium on Security and Privacy*, May 2001.

[16] S. Mittra. Iolus: A framework for scalable secure multicasting. In *Proceedings of the ACM SIGCOMM'97 (September 14-18, 1997, Cannes, France)*, 1997.

[17] National Institute of Standards and Technology. Secure hash standard, 1995.

[18] J. Nonnenmacher, E. W. Biersack, and D. Towsley. Parity-based loss recovery for reliable multicast transmission. *IEEE/ACM Transactions on Networking*, 6(4):349–361, 1998.

[19] J. M. Park, E. K. P. Chong, and H. J. Siegel. Efficient multicast packet authentication using signature amortization. In *2002 IEEE Symposium on Security and Privacy*, Berkeley, California, May 2002.

[20] V. Paxson. End-to-end internet packet dynamics. *IEEE/ACM Transactions on Networking*, 7(3):277–292, 1999.

[21] A. Perrig. The BiBa one-time signature and broadcast authentication protocol. In *ACM Conference on Computer and Communications Security*, pages 28–37, 2001.

[22] A. Perrig, R. Canetti, J. Tygar, and D. Song. Efficient authentication and signing of multicast streams over lossy channels. In *IEEE Symposium on Security and Privacy*, May 2000.

[23] A. Perrig, D. Song, and D. Tygar. ELK, a new protocol for efficient large-group key distribution. In *IEEE Symposium on Security and Privacy*, May 2001.

[24] M. O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the ACM*, 36(2):335–348, 1989.

[25] R. Rivest. The MD5 message-digest algorithm, April 1992.

[26] L. Rizzo. Effective erasure codes for reliable computer communication protocols. *ACMCCR: Computer Communication Review*, 27, 1997.

[27] P. Rohatgi. A compact and fast hybrid signature scheme for multicast packet authentication. In *Proceedings of the 6th ACM conference on Computer and Communications Security*, pages 93–100, Singapore, November 1999. Association for Computing Machinery.

[28] RSA Security Inc. PKCS-1 v2.1: RSA cryptography standard, 1999.

[29] D. Song, J. D. Tygar, and D. Zuckerman. Expander graphs for digital stream authentication and robust overlay networks. In *2002 IEEE Symposium on Security and Privacy*, Berkeley, California, May 2002.

[30] C. K. Wong, M. Gouda, and S. S. Lam. Secure group communications using key graphs. In *ACM SIGCOMM 1998*, pages 68–79, 1998.

[31] C. K. Wong and S. S. Lam. Digital signatures for flows and multicasts. *IEEE/ACM Transactions on Networking*, 7(4):502–513, 1999.

[32] M. Yajnick, J. Kurose, and D. Tosley. Packet loss correlation in the Mbone multicast network. Technical Report 96-32., UMCASS CMPSCI, 1996.

[33] M. Yajnick, J. Kurose, and D. Tosley. Packet loss correlation in the mbone multicast network. In *IEEE Global Internet Conference*, London, Nov. 1996.

[34] M. Yajnik, S. Moon, J. Kurose, and D. Towsley. Measurement and modeling of the temporal dependence in packet loss. In *IEEE INFOCOM*, New York, Mar. 1999.