A Tree-based Forward Digest Protocol to Verify Data Integrity in Distributed Media Streaming

Ahsan Habib† Dongyan Xu‡ Mikhail Atallah‡ Bharat Bhargava‡ John Chuang†
†School of Information Management and Systems
University of California, Berkeley, CA 94720
{habib,chuang}@sims.berkeley.edu
‡Department of Computer Science
Purdue University, West Lafayette, IN 47907
{dxu,mja,bb}@cs.purdue.edu

Abstract

We design a Tree-based Forward Digest Protocol (TFDP) to verify data integrity in distributed media streaming for content distribution. Several challenges arise, including the timing constraint of streaming sessions, the involvement of multiple senders, and the untrustworthiness of these senders. A comprehensive comparison is presented on the performance of existing protocols and TFDP, with respect to communication and computation overhead. Both simulation and Internet-based experimental results are presented to demonstrate the effectiveness of TFDP.

Keywords: Data integrity, message digest, and media streaming.

I. Introduction

Consider the following media data distribution scenario: a source content server called "Hollywood" first starts streaming a movie to its clients. When sufficient number of clients have been served and have agreed to serve as re-distributors, they will begin streaming the movie to other clients in the system. The distribution is supervised by the server: it authenticates *requesting* clients and gives them credentials to be served by the *supplying* clients, who will perform distributed media streaming only if proper credentials are presented. One key property of such distributed media streaming is that each streaming session involves *multiple* supplying clients, due to the limited bandwidth contributed to the session by each of them. We note that, unlike traditional file sharing systems, a media streaming session allows continuous playback of media data *during* the session.

In such a "many-to-one" media streaming session, an untrustworthy supplying client (or "supplier") may corrupt any block of the media data. The verification of media data *integrity* thus becomes a critical task and poses a number of challenges. First, unlike authentication for multicast streaming [1], [11], [8], the suppliers cannot be assumed as trusted. In distributed media streaming, packets signed by

This work is supported in part by NSF-ITR 0085879, NSF-ANI 0219110, NSF-IIS 0209059, NSF-EIA 9903545, NSF-ISS 0219560, and CERIAS. This work was partially done when Ahsan Habib was at Purdue University.

one client may not be acceptable to other clients. Therefore, a client needs a point of reference to verify the media data it receives. Second, due to the real-time constraint of media streaming, data integrity verification needs to be performed in real-time. Third, the objective of verifying data integrity is not only to verify that the data are not corrupted, but also to validate that the media file is really what the client has requested. We note that other security problems exist in distributed media streaming, such as how to prevent the disclosure of copyright-protected media data to unauthorized parties. These issues are outside the scope of this paper.

By presenting a comprehensive survey of existing protocols for data integrity verification, we show that they are either inapplicable or too expensive for distributed media streaming. We adopt the method of *message digest*, and propose a Tree-based Forward Digest Protocol (TFDP). TFDP uses Merkle tree [7], and distributes data digest delivery overhead over the duration of a streaming session. The protocol works well with unreliable transport protocols. This is achieved by using Forward Error Correction (FEC) codes, especially for data digests. Our experiments show that TFDP is able to verify media data integrity with low communication and computation overhead.

TFDP is also applicable to Bittorrent-like file sharing applications that adopt the "multiple senders and single receiver" model similar to distributed media streaming. In addition, users are required to perform simultaneous uploads and downloads. By using TFDP, a receiver can verify data received from multiple suppliers block by block during a download session, and thus, it can become an uploading supplier at the same time.

The rest of the paper is organized as follows: Section II surveys related work. Section III presents TFDP. A comprehensive comparison among different solutions is presented in Section IV. Section V presents experimental results. Section VI concludes this paper.

II. Related Work

To the best of our knowledge, there has been no prior study on data integrity verification for many-to-one distributed media streaming. In this section, we survey current related solutions and identify

their limitations in supporting distributed media streaming.

Digital signature. A straightforward method to verify data integrity is to let the source server sign every packet (*packet* indicates the minimum unit of media transport, *not* an IP packet) or the hash of each packet with its private key using digital signature. A client can then verify the signed data using the server's public key. The RSA signature [14] verification has high computation overhead and is not suitable for real-time applications. Unlike RSA, one-time signature schemes such as [5], [9], [13] incur low verification overhead and latency. These schemes are usually used to sign multicast or broadcast streams. Rohatgi [15] proposed a k-time signature scheme which is more efficient than the one-time signature schemes. Still, the scheme generates 300 bytes for each signature.

Signature chain. Gennaro and Rohatgi [2] introduced techniques to sign off-line and on-line digital streams. The first packet of an off-line stream is signed and the hash of each packet is embedded in the *next* packet. The on-line scheme signs the initial packet and embeds the public key of a one-time signature in each packet, which is used to sign the subsequent packet. Although an elegant solution, it does not tolerate packet losses and it incurs high communication overhead.

Perrig *et al.* [11], [10] proposed TESLA and EMSS for efficient and secure multicast. TESLA embeds the signature of packet p_i and the key to verify packet p_{i-1} in packet p_i. The key of packet p_i is sent in packet p_{i+1}. The adversary will see the key but it is too late to forge the signature. TESLA requires *strict ordering* of packets by the sender, which cannot be guaranteed in distributed *many-to-one* streaming. Furthermore, if the supplying clients generate keys and sign the digests as in TESLA, they may not be acceptable to other clients because clients are not assumed to be trustworthy.

Signature tree. Wong and Lam [16] studied data authenticity and integrity for lossy multicast streams. They proposed using Merkle signature tree to sign multicast streams. In their scheme, the root is signed to amortize one signature over multiple messages. Each packet contains the digests of all nodes necessary to compute the digest of the root and the signature of the root. As a result, the space requirement is rather high: 200 bytes in each packet using 1024-bit RSA for a tree of 16 packets. The

protocol we propose also uses Merkle tree. However, we significantly reduce the overhead by sending the digests of one subtree *before* sending any data. Thus, a packet does not need to carry all the digests that are required to verify its integrity.

Park *et al.* [8] proposed *SAIDA* that leverages erasure codes to amortize a single signature operation over multiple packets. In SAIDA, a block—a series of contiguous packets—of a packets carries the encoded digests and signature of the block. The signature and digests are recoverable, if the receiver gets any $b \leq a$ packets. This digest encoding is robust against bursty packet losses to a certain level. To reduce overhead, FEC is used to encode only digests, not data.

Both signature tree and SAIDA are designed for multicast where the sender signs packets and the receivers trust the sender. In our protocol, a receiver does not have to trust the suppliers. Unlike signature tree and SAIDA, we choose not to use digital signature to further reduce overhead.

Erasure codes and homomorphic hash function. Krohn *et al.* [4] use homomorphic hash function to verify erasure codes during a many-to-one bulk file transfer session. A client can verify each block on the fly while downloading a large file such as Linux ISO from multiple suppliers. Thus, the client does not have to wait until the end of the transfer to verify the entire file. While effective for file downloading, this verification scheme is not applicable to real-time media streaming because the client has to wait until the end of the file transfer to decode all blocks.

III. Protocol Description

In our protocol, we assume the existence of a trusted authority denoted as the *Authentication Server* (or "server") S_0. To request a distributed streaming session, a client will first authenticate itself with S_0 and obtain a point of reference to be used for data integrity verification during the streaming session. In our protocol, the reference data is only 20 bytes long for each requesting client. We first define the distributed media streaming model and then describe our protocol.

Streaming model. Client P_0 requests a media streaming session to be served by a set of supplying clients $\mathbb{P} = \{P_1, P_2, \ldots, P_m\}$. The set of suppliers is determined by a certain lookup and selection mechanism. The requested media file has a size of F bytes and is divided into a set of M *blocks* $\mathbb{B} = \{b_1, b_2, \ldots, b_M\}$. Each block consists of l *packets*. We denote a block as $b_i = \{p_{i1}, p_{i2}, \ldots, p_{il}\}$ where p_{ij} is the j-th packet of i-th block. Finally, a sequence of blocks is referred to as a *group*. During a streaming session, different suppliers transmit different packets of each block to the client, which will re-construct the block. Details on packet assignment to suppliers can be found in [3].

Tree-based Forward Digest Protocol (TFDP). As in *Tree Chaining* proposed by Wong and Lam for multicast flows [16], we use Merkle tree to design TFDP. Merkle tree [7] generates one-time signature using one-way function tree and hash function. Each message to be signed corresponds to a node in the tree. Each node consists of verification parameters that are used to sign a message and to authenticate the verification parameters of subsequent nodes. The root of the tree is the public key for signature verification. We note that the idea of constructing one-way function tree has also been applied to key management in secure group communications [12], [6]. In fact, such tree construction is common in a number of application scenarios, but for different purposes: Rafaeli and Hutchison [12] as well as McGrew and Sherman [6] use it for key management, Merkle [7] uses it to generate signatures, while we (TFDP) as well as Wong and Lam (Tree Chaining) use it to authenticate data streams.

In Tree Chaining, a file is divided into a number of blocks, with each block containing a set of packets. A signature tree is then constructed for each block with all packets of the block as leaves. The root of the tree is signed by a digital signature. In TFDP, however, we do not sign the root of every subtree that belongs to each block. Instead, we only compute digests to build a Merkle tree. The main difference between Tree Chaining and TFDP is that the former is designed for multicast flows (one-to-many) where each packet carries the necessary information for its authentication and thus incurs high overhead, while the latter is designed for many-to-one streaming where the overhead is amortized over a group of data blocks.

In TFDP, server S_0 generates the Merkle tree for a media file. The leaves of the tree are packets in the file. Each non-leaf node of the tree represents the digest of its children. The server enforces that a supplier keeps a minimum number of digests, so that the overhead of sending digests is amortized over a group of data blocks. During a streaming session, N_{min} digests are transmitted before transmitting the media data blocks. A higher N_{min} will reduce the block verification overhead. However, it will incur longer delay in the streaming session.

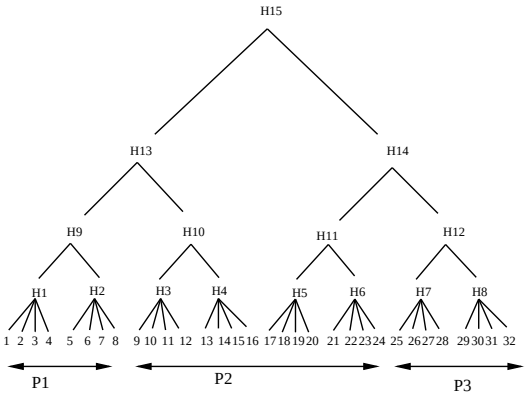


Fig. 1. A Merkle tree with 32 packets that belong to 8 blocks. P_1, P_2, and P_3 are suppliers. P_1 will provide digests H_1, H_2, H_{10}, and H_{14} that are required to verify the first two blocks. Similarly, P_2 will provide digests for the next four blocks and P_3 will provide digests for the last two blocks.

Figure 1 shows a simple example with 32 packets that belong to 8 blocks. Each non-leaf node H_i represents the digest of its children. P_1, P_2, and P_3 are suppliers involved in the same streaming session. Let P_1 be assigned to provide digests for the first two blocks, P_2 for the next four blocks, and P_3 for the last two blocks. In this example, the digests provided by P_1 are H_1, H_2, H_{10}, and H_{14}. P_0 then computes H_9 from H_1 and H_2, H_{13} from H_9 and H_{10}, and H_{15} from H_{13} and H_{14}, and compares H_{15} against the digest supplied by server S_0. If there is a match, the *belief* in H_{15} is transferred to all digests provided by P_1 because of the property of collision-free hash function. During the streaming session, media data sent by the three suppliers can be verified block by block using H_1 and H_2 provided by P_1. P_2 and P_3 operate in a similar fashion. We now describe the steps of TFDP:

- *Step 1:* Client P_0 authenticates itself with server S_0 by sending $E_{S_0}(D_{P_0}(M_0))$, where the request message M_0 is signed by P_0 for non-repudiation. Then, it is encrypted with the public key of S_0.

- *Step 2:* The server sends $E_{P_0}(D_{S_0}(H_{root}, T))$ to P_0, where the digest of the root of the Merkle tree is H_{root} which is encrypted with the public key of P_0 and signed by the private key of the server. T is a timestamped ticket that needs to be presented to each supplier to prevent a user from using the same ticket beyond a specific time period.
- *Step 3:* P_0 tells supplier P_i in $\mathbb{P}$ to provide the digests needed to verify N_{min} blocks. If more than one supplier has all the digests, P_0 randomly selects one of them.
- *Step 4:* Supplier P_i provides to P_0 all required digests to verify N_{min} data blocks.
- *Step 5:* If the root digest computed by P_0 matches the root digest obtained from the server, P_0 will trust the digests provided by P_i and use them for data integrity verification during the streaming session. Otherwise, the protocol goes to Step 3 and obtains digests from a different supplier.
- *Step 6:* P_0 signals the suppliers in $\mathbb{P}$ to send media data.
- *Step 7:* The suppliers start the streaming session. Using the digests obtained in Step 4, P_0 verifies every media data block re-constructed from packets transmitted by the suppliers. Once the streaming of N_{min} blocks is done, the protocol goes to Step 3 to repeat the process for the next N_{min} blocks.

Figure 1 is a binary tree if we exclude the leaves. Especially, a parent node of the leaves represents the digest of the block to which the packets belong. The size of a block needs to be chosen carefully to ensure that it does not introduce delay to collect all packets in it. If the tree is a d-ary tree, the height of the tree will be $\log_d \frac{F}{l}$, where F is the size of the media file. The number of extra digests required to verify each block depends on the height of the tree. TFDP requires $(d-1)\left\lceil \log_d \frac{F}{N_{min} l} \right\rceil$ digests to verify N_{min} blocks. It can be easily shown that the number of extra digests to verify N_{min} blocks is minimized when $d = 2$.

IV. Comparison and Evaluation

In [8], the authors show that SAIDA performs better than both EMSS [11] and Tree Chaining [16] in tolerating bursty packet losses. Therefore, we only compare TFDP with SAIDA and Tree Chaining. We compare the three protocols in terms of communication and computation overhead. The communication overhead is the extra bytes per packet client P_0 needs to receive from the suppliers and S_0 for data integrity verification. The computation overhead at P_0 is due to hash computation, signature verification, and FEC decoding.

Communication overhead. Tree Chaining requires P_0 to obtain the public key (usually 128 bytes long) of the server to verify the signature. P_0 needs to receive $l \log l$ digests for each block, where l is the size of a block in number of packets. Each packet carries one 1024-bit signature. Thus, for each block, P_0 needs to receive $20l \log l + 128l$ bytes. TFDP requires only one digest (20 bytes) from the server. However, it needs $1 + \frac{1}{N_{min}} \log(\frac{M}{N_{min}})$ extra digests for each block. The digest of each block is encoded using FEC. We define α, the overhead factor of FEC, as:

$$\alpha = \frac{\text{total packets sent per block}}{\text{packets required to re-construct the block}}. \quad (1)$$

Thus, the total communication overhead of TFDP is $20(l\alpha + 1 + \frac{1}{N_{min}} \log(\frac{M}{N_{min}}))$ bytes per block. SAIDA requires one signature per block, and it uses FEC. Thus, it incurs $(20l+128)\alpha$ bytes of overhead for each block. Table I summarizes the comparison results.

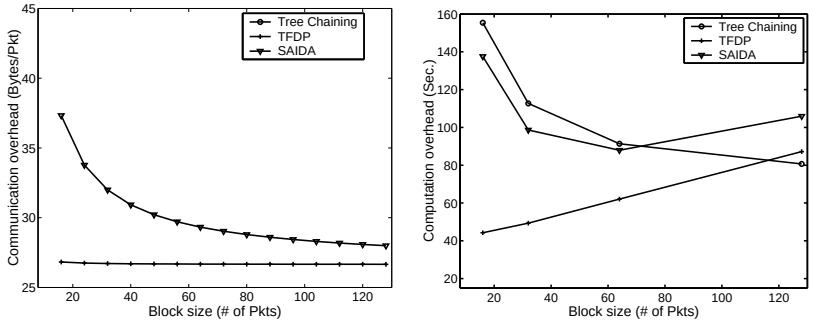
TABLE I

Comparison of data integrity verification protocols. M is the total number of blocks of a file, l is the size of one block in number of packets, α is the FEC overhead factor.

	Allow packet loss	Overhead: server $\rightarrow P_0$ (Bytes)	Overhead: $\mathbb{P} \rightarrow P_0$ (Bytes)	# of hash computation by server	# of hash computation by P_0	Sign by server	Verify signature by P_0	FEC decode by P_0
Tree Chaining	YES	0	$20Ml \log l + 128Ml$	$M(2l-1)$	$M(2l-1)$	M	M	—
SAIDA	YES	0	$(20l+128)M\alpha$	$M(l+1)$	$M(l+1)$	M	M	M
TFDP	YES	20	$20Ml\alpha + 20[M + \frac{M}{N_{min}} \log(\frac{M}{N_{min}})]$	$2M-1$	$Ml + M/N_{min}[(N_{min} -1) + \log(M/N_{min})]$	—	—	M

We demonstrate the communication overhead of the three protocols using the trace of *The Matrix*

movie. Figure 2(a) shows that the communication overhead of SAIDA and TFDP is both low, because FEC is used to encode digests and signatures. On the other hand, Tree Chaining does not use FEC and incurs much higher communication overhead (208 bytes per packet, for l=16, not shown in Figure 2(a)). TFDP incurs less overhead than SAIDA and Tree Chaining because TFDP verifies digests of every N_{min} data blocks as a group, which reduces the height of the Merkle tree from $\log M$ to $\log \frac{M}{N_{min}}$. The difference in communication overhead between TFDP and SAIDA narrows when the block size gets larger. However, a large block size can cause delay during a streaming session, because the client would need all packets in a block before it can re-construct the block.



(a) Communication overhead (b) Computation overhead

Fig. 2. Overhead of Tree Chaining, TFDP, and SAIDA for *The Matrix* movie (size: 1.3 GB). The communication overhead is per packet while the computation overhead is for the entire file. Tree Chaining has 208 bytes overhead per packet (*not* shown in the figure).

Computation overhead. In Tree Chaining, M subtrees are created for M blocks, and each tree requires $2l - 1$ hash computation. SAIDA needs to decode digests for each packet and verify one signature for each block. Thus, the computation overhead of SAIDA is due to $M(l + 1)$ digests and M signatures. TFDP needs to compute extra digests for every N_{min} blocks. The number of extra digests is $M/N_{min}[(N_{min} - 1) + \log(M/N_{min})]$, and every block is verified during the streaming session. Thus, the total client-side computation overhead of TFDP is $Ml + M/N_{min}[(N_{min} - 1) + \log(M/N_{min})]$.

We use openSSL crypto library to calculate SHA-1 hash, RSA signature, and RSA verification. Cauchy-based Reed-Solomon code is used to encode digests. In Figure 2(b), we compare the computation overhead of the protocols using *The Matrix* movie. The computation overhead of Tree Chaining can be reduced by caching digests carried by previous packets. The cached digests are used to verify

upcoming packets of a block. SAIDA incurs higher computation overhead than TFDP because SAIDA has to verify the signature of every block, which is more computation-intensive than verifying digests.

V. Streaming Experiment Results

We have conducted both streaming simulations and real-world experiments. In our simulations, one client requests and receives streaming media from five suppliers. Like in SAIDA, we use the *two-state Markov loss model* to introduce bursty packet losses. The parameters of the Markov loss model are $Pr\{\text{no loss}\}$=0.95 and $Pr\{\text{loss}\}$=0.05. The model characterizes the loss of every underlying network link connecting the five suppliers and the client. We define data block verification rate as the fraction of data blocks that can be verified during a certain time interval.

The simulation results of SAIDA and TFDP are shown in Figure 3. In both protocols, the digests and signatures are FEC-encoded to tolerate 37.5% packet loss rate. We observe that due to bursty packet losses, some blocks cannot be verified. The reason why TFDP performs better is that TFDP incurs slightly less communication overhead than SAIDA, which requires RSA signature for each block.

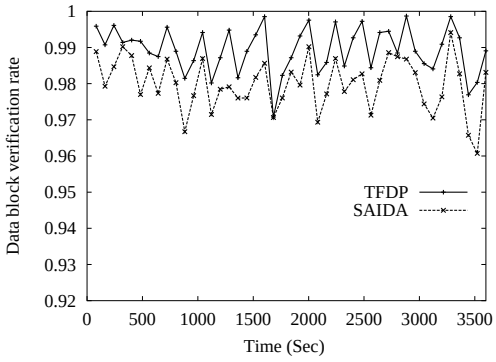


Fig. 3. Data block verification rate of SAIDA and TFDP (simulations).

We have also developed a distributed media streaming system called *PROMISE* [3]. The system monitors network dynamics, quality of connections from multiple suppliers to a receiver, as well as supplier availability, to maintain full media playback quality on the client side. Particularly, the set of suppliers in a streaming session may change dynamically, so that the fluctuation of network and supplier conditions will not affect the client-side aggregated media streaming rate. TFDP can be integrated

into PROMISE. We evaluate TFDP by conducting experiments in the wide-area *PlanetLab* testbed (www.planet-lab.org).

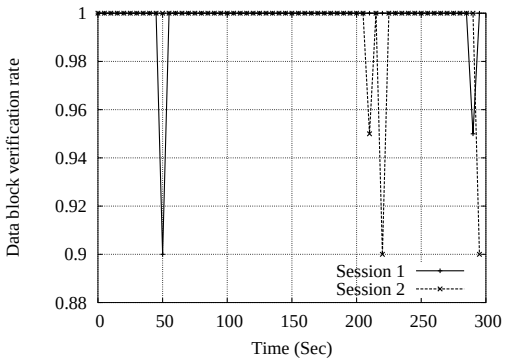


Fig. 4. Data block verification rate of TFDP (PlanetLab-based experiments).

Figure 4 shows the results from two streaming sessions using TFDP in PlanetLab. Both sessions can tolerate up to 20% packet loss due to FEC. In Session 1, TFDP is able to verify almost all the blocks during the first 300 seconds of the session. At the 50th second, the network loss rate goes up to 40% and the block verification rate temporarily drops to 0.9. Session 2 experiences a few more glitches than Session 1. Still, in both sessions, the overall block verification rate achieved by TFDP remains high. The main reason is that PROMISE has *dynamic supplier switching* capability. The suppliers in a streaming session can be dynamically replaced when significant packet loss is experienced, in order to avoid the congested network links. Our experiments show that TFDP works well in the event of a dynamic supplier switch.

VI. Conclusion

We study the problem of data integrity verification in distributed media streaming sessions. We propose a simple and efficient Tree-based Forward Digest Protocol (TFDP) as our solution. TFDP incurs low communication and computation overhead, compared with existing data integrity verification protocols. More importantly, TFDP relieves the source server from supplying the digest of every data block in a streaming session. Instead, this load is distributed among the multiple supplying clients serving in the streaming session, enabling data integrity verification even if the suppliers are not trustworthy.

Both simulation and Internet experiments demonstrate the effectiveness and practicality of TFDP.

REFERENCES

[1] D. Boneh, G. Durfee, and M. Franklin. Lower bounds for multicast message authentication. In *proceedings of Eurocrypt '01*, Lecture Notes in Computer Science, Vol. 2045, pages 437–452. Springer-Verlag, 2001.

[2] R. Gennaro and P. Rohatgi. How to sign digital streams. Technical report, IBM T. J. Watson research center, 1997.

[3] M. Hefeeda, A. Habib, B. Botev, D. Xu, and B. Bhargava. PROMISE: Peer-to-peer media streaming using CollectCast. In *proceedings ACM Multimedia (MM '03)*, Nov. 2003.

[4] M. N. Krohn, M. J. Freedman, and D. Mazieres. On-the-fly verification of rateless erasure codes for efficient content distribution. In *proceedings IEEE Symposium on Security and Privacy*, Oakland, California, May, 2004.

[5] L. Lamport. Constructing digital signatures from a one-way function. Technical report, SRI-CSL-98, SRI International Computer Science Laboratory, Oct. 1979.

[6] D. A. McGrew and A. T. Sherman. Key establishment in large dynamic groups using one-way function trees. *IEEE Transactions on Software Engineering*, 29:444–458, May 2003.

[7] R. C. Merkle. A certified digital signature. In *Proceedings on Advances in cryptology*, pages 218–238, 1989.

[8] J. M. Park, E. Chong, and H. Siegel. Efficient multicast packet authentication using signature amortization. In *proceedings IEEE Symposium on Security and Privacy (S&P)*, May 2002.

[9] A. Perrig. The BiBa one-time signature and broadcast authentication protocol. In *proceedings ACM Conference on Computer and Communications Security (CCS '01)*, pages 28–37, Philadelphia, PA, Nov. 2001.

[10] A. Perrig, R. Canetti, D. Song, and D. Tygar. Efficient and secure source authentication for multicast. In *proceedings Network and Distributed System Security Symposium, (NDSS '01)*, San Diego, CA, Feb. 2001.

[11] A. Perrig, R. Canetti, J. D. Tygar, and D. X. Song. Efficient authentication and signing of multicast streams over lossy channels. In *proceedings IEEE Symposium on Security and Privacy (S&P '00)*, pages 56–73, Nov. 2000.

[12] S. Rafaeli and D. Hutchison. A survey of key management for secure group communication. *ACM Computing Survey*, 35(3):309–329, 2003.

[13] L. Reyzin and N. Reyzin. Better than BiBa: Short one-time signatures with fast signing and verifying. In *proceedings 7th Australian Conference ACSIP*, Sept 2002.

[14] R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signature and public key cryptosystems. *Communication of the ACM*, pages 120–126, Feb. 1978.

[15] P. Rohatgi. A compact and fast hybrid signature scheme for multicast packet. In *proceedings ACM Conference on Computer and Communications Security (CCS '01)*, pages 93–100, Nov. 1999.

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