How to Sign Digital Streams

Rosario Gennaro * Pankaj Rohatgi *

February 24, 1998

Abstract

We present a new efficient paradigm for signing digital streams. The problem of signing digital streams to prove their authenticity is substantially different from the problem of signing regular messages. Traditional signature schemes are message oriented and require the receiver to process the entire message before being able to authenticate its signature. However, a stream is a potentially very long (or infinite) sequence of bits that the sender sends to the receiver and the receiver is required to consumes the received bits at more or less the input rate and without excessive delay. Therefore it is infeasible for the receiver to obtain the entire stream before authenticating and consuming it. Examples of streams include digitized video and audio files, data feeds and applets.

We present two solutions to the problem of authenticating digital streams. The first one is for the case of a finite stream which is entirely known to the sender (say a movie). We use this constraint to devise an extremely efficient solution. The second case is for a (potentially infinite) stream which is not known in advance to the sender (for example a live broadcast). We present proofs of security of our constructions. Our techniques also have applications in other areas, for example, efficient authentication of long files when communication is at a cost and signature based filtering at a proxy server.

1 Introduction

Digital Signatures (see [6, 18]) are the cryptographic answer to the problem of information authenticity. When a recipient receives digitally signed information and she is able to verify the digital signature then she can be certain that the information she received is exactly the same as what the sender (identified by his public key) has signed. Moreover, this guarantee is *non-repudiable*, i.e., the entity identified by the public key cannot later deny having signed the information. Thus, the recipient can hold the signer responsible for the content she receives.[1]

However current digital signature technology was designed to ensure message authentication and its straightforward application does not yield a satisfactory solution when applied

---

*I.B.M. T.J.Watson Research Center, P.O.Box 704, Yorktown Heights, NY 10598, U.S.A. Email: `rosario,rohatgi@watson.ibm.com` A preliminary version of this paper appeared in the proceedings of CRYPTO'97.

[1] This distinguishes digital signatures from *message authentication codes* (MAC) which allow the receiver to have confidence on the identity of the sender, but not to prove to someone else this fact, i.e. MAC's are repudiable.

to information resources which are not message-like. In this paper we discuss one such type of resource: *streams.* We point out shortcomings in several approaches (some of them used in practice) to tackle the problem of signing streams and then present our solution which does not have such shortcomings.

1.1 Streams Defined

A stream is a potentially very long (infinite) sequence of bits that the sender sends to the receiver. The stream is usually sent at a rate which is negotiated between the sender and receiver or there may be a demand-response protocol in which the receiver repeatedly sends requests for additional (finite) amount of data. The main features of streams which distinguish them from messages is that the receiver must consume the data it receives at more or less the input rate, i.e., it can't buffer large amounts of unconsumed data. In fact in many applications the receiver stores relatively very small amounts of the stream. In some cases the sender itself may not store the entire sequence, i.e., it may not store the information it has already sent out and it may not know anything about the stream much beyond what it has sent out.

There are many examples of digital streams. Common examples include *digitized video* and *audio* which is now routinely transported over the Internet and also to television viewers via various means, e.g., via direct broadcast satellites and very shortly via cable, wireless cable, telephone lines etc. This includes both pre-recorded and stored audio/video programming as well as live feeds. Apart from audio/video, there are also *data feeds* (e.g., news feeds, stock market quotes etc) which are best modeled as a stream. The Internet and the emerging interactive TV industry also provides another example of an information resource which is best modeled as a stream, i.e., *applets.* Most non-trivial applets are actually very large programs which are organized into several modules. The consumer's machine first downloads and executes the startup module and as the program proceeds, additional modules are downloaded and executed. Also, modules which are no longer in use may be discarded by the consumer machine. This structure of applets is forced by two factors. Firstly the amount of storage available on the consumer machine may be limited (e.g., in the emerging interactive TV industry set-top boxes have to be cheap and therefore resource limited) . Secondly (in the case of the Internet), the bandwidth available to download code may be limited and applets must be designed to start executing as soon as possible. Also it is quite likely that some of the more sophisticated applets may have data-rich components generated on the fly by the applet server. Therefore applets fit very nicely into the demand/response streams paradigm.

Given the above description, it is clear that message oriented signature schemes cannot be directly used to sign streams since the receiver cannot be expected to receive the entire stream before verifying the signature. If a stream is infinitely long (e.g., the 24-hours news channel), then it is impossible for the receiver to receive the entire stream and even if a stream is finite but long the receiver would have to violate the constraint that the stream needs to be consumed at roughly the input rate and without delay.

1.2 Previous Solutions and their Shortcomings

Up to the authors' knowledge there has been no proposed specific solution to the problem of signing digital streams in the crypto literature. One can envision several possible solutions, some of them are actually proposed to be used in practice.

One type of solution splits the stream in blocks. The sender signs each individual block and the receiver loads an entire block and verifies its signature before consuming it. This solution also works if the stream is infinite. However this solution forces the sender to generate a signature for each block of the stream and the receiver to verify a signature for each block. With today's signature schemes either one or both of these operations can be very expensive computationally. Which in turns means that the operations of signing and verifying can create a bottleneck to the transmission rate of the stream.

Another type of solution works for only finite streams. In this case, once again the stream is split into blocks. Instead of signing each block, the sender creates a table listing cryptographic hashes of each of the blocks and signs this table. When the receiver asks for the authenticated stream, the sender first sends the signed table followed by the stream. The receiver first receives and stores this table and verifies the signature on it. If the signature matches then the receiver has the authenticated cryptographic hash of each of blocks in the stream and thus each block can be verified when it arrives. The problem with this solution is that it requires the storage and maintenance of a potentially very large table on the receiver's end. In many realistic scenarios the receiver buffer is very limited compared to the size of the stream, (e.g., in MPEG a typical movie may be 20 GBytes whereas the receiver buffer is only required to be around 250Kbytes). Therefore the hash table can itself become fairly large (e.g., 50000 entries in this case or 800Kbytes for the MD5 hash function) and it may not be possible to store the hash table itself. Also, the hash table itself needs to be transmitted first and if it is too large then there will be a significant delay before the first piece of the stream is received and consumed. To address the problem of large tables one can also come up with a hybrid scheme in which the stream is split in consecutive pieces and each piece is preceded by a small signed table of contents.[2]

The above solution can be further modified by using an authentication tree: the blocks are placed as the leaves of a binary tree and each internal node takes as a value the hash of its children (see [13].) This way the sender needs to sign and send only the root of this tree. However in order to authenticate each following block the sender has to send the whole authentication path (i.e. the nodes on the path from the root to the block, plus their siblings) to the receiver. This means that if the stream has k blocks, the authentication information associated with each block will be $O(\log k)$.

As we will see briefly our solution eliminates all these shortcomings. The basic idea works for both infinite and finite streams, only one expensive digital signature is ever computed, there are no big tables to store, and the size of the authentication information associated with each block does not depend on the size of the stream.

NON-REPUDIATION. Notice that if the receiver were only interested in establishing the identity of the sender, a solution based on MAC would suffice. Indeed once the sender and receiver share a secret key, the stream could be authenticated block by block using a MAC computation on it. Since MAC's are usually faster than signatures to compute and verify, this solution would not incur the computational cost associated with the similar signature-based solution described above.

---

[2] This is the case now (Java Developer Kit 1.1) for large signed java applets which are distributed as a collection of Java archives (JAR) where each archive has a signed table of hashes of contents and the archives are loaded in the order given in the HTML page in which the applet is embedded.

However a MAC-based approach would not enjoy the non-repudiation property. We stress that we require such property for our solution. Also in order for this property to be meaningful in the context of streams we need to require that *each* prefix of the stream to be non-repudiable. That is, if the stream is $\mathcal{B} = B_1, B_2, \ldots$ where each B_i is a block, we require that each prefix $\mathcal{B}_i = B_1 \ldots B_i$ is non-repudiable. This rules out a solution in which the sender just attaches a MAC to each block and then signs the whole stream at the end.

This is to prevent the sender from interrupting the transmission of the stream before the non-repudiability property is achieved. Also it is a guarantee for the receiver. Consider indeed the following scenario: the receiver notices that the applets she is downloading are producing damages to her machine. She interrupts the transfer in order to limit the damage, but at the same time she still wants some proof to bring to court that the substream downloaded so far did indeed come from the sender.

1.3 Our solution in a nutshell

Our solution makes some reasonable/practical assumptions about the nature of the streams being authenticated. First of all we assume that it is possible for the sender to embed authentication information in the stream. This is usually the case, see Section 7 to see how to do this in most real-world situations like MPEG video/audio. We also assume that the receiver has a "small" buffer in which it can first authenticate the received bits before consuming them. Finally we assume that the receiver has processing power or hardware that can compute a small number of fast cryptographic checksums faster than the incoming stream rate while still being able to play the stream in real-time.

The basic idea of our solution is to divide the stream into blocks and embed some authentication information in the stream itself. The authentication information of the i^{th} block will be used to authenticate the $(i+1)^{st}$ block. This way the signer needs to sign just the first block and then the properties of this single signature will "propagate" to the rest of the stream through the authentication information. Of course the key problem is to perform the authentication of the internal blocks fast. We distinguish two cases.

In the first scenario the stream is finite and is known in its entirety to the signer in advance. This is not a very limiting requirement since it covers most of the Internet applications (digital movies, digital sounds, applets). In this case we will show that a *single* hash computation will suffice to authenticate the internal blocks. The idea is to embed in the current block a hash of the following block (which in turns includes the hash of the following one and so on...)

The second case is for (potentially infinite) streams which are not known in advance to the signer (for example live feeds, like sports event broadcasting and chat rooms). In this case our solution is less optimal as it requires several hash computations to authenticate a block (although depending on the embedding mechanism these hash computations can be amortized over the length of the block). The size of the embedded authentication information is also an issue in this case. The idea here is to use fast 1-time signature schemes (introduced in [11, 12]) to authenticate the internal blocks. So block i will contain a 1-time public key and also the 1-time signature of itself with respect to the key contained in block $i-1$. This signature authenticates not only the stream block but also the 1-time key attached to it.

1.4 Related Work

The "chaining" technique of embedding the hash of the following block in the current block can be seen as a variation of the Merkle-Damgård meta-method to construct hash functions based on a simpler compression function [14, 5]. The novelty here is that we exploit the structure of the construction to allow fast authentication of single blocks in sequential order. It can also be seen as a weak construction of *accumulators* as introduced in [2]. An accumulator for k blocks $B_1, \ldots, B_k$ is a single value ACC that allows a signer to quickly authenticate any of the blocks in any particular order. Accumulators based on the RSA assumption were proposed in [2]. In our case we have a much faster construction based on collision–free hash functions, since we exploit the property that the blocks must be authenticated in a specific order.

Mixing "regular" signatures with 1-time signatures, for the purpose of improving efficiency is discussed in [8]. However in that paper the focus is in making the signing operation of a message M efficient by dividing it in two parts. An off-line part in which the signer signs a 1-time public key with his long-lived secret key even before the messages M is known. Then when M has to be sent the signer computes a 1-time signature of M with the authenticated 1-time public key and sends out M tagged with the 1-time public key and the two signatures. Notice that the receiver must compute two signature verifications: one on the long-lived key and one on the 1-time key. In our scheme we need to make both signing and verification extremely fast, and indeed in our case each block (except for the first) is signed (and hence verified) only once with a 1-time key.

We also use the idea to of using old keys in order to authenticate new keys. This has appeared in several places but always for long-lived keys. Examples include [1, 16, 19] where this technique is used to build provably secure signature schemes. We stress that the results in [1, 16, 19] are mostly of theoretical interest and do not yield practical schemes. Our on-line solution somehow mixes these two ideas in a novel way, by using the chaining technique with 1-time keys, embedding the keys inside the stream flow so that old keys can authenticate at the same time *both* the new keys and the current stream block.

2 Preliminaries

In the following we denote with n the security parameter. We say that a function $\epsilon(n)$ is *negligible* if for all c, there exists an n_0 such that, for all $n > n_0$, $\epsilon(n) < 1/n^c$.

COLLISION-RESISTANT HASH FUNCTIONS. Let $\mathcal{H}$ be a family of functions that map arbitrarily long binary strings into elements of a fixed domain D. We say that $\mathcal{H}$ is a *collision-resistant* family of hash functions if any polynomial time algorithm who is given as input the values $H(x_i)$ on several adaptively chosen values x_i and a random element $H \in \mathcal{H}$, finds a collision, i.e. a pair (x, y) such that $x \neq y$ and $H(x) = H(y)$, only with negligible probability. MD5 [17] and SHA-1 [15] are conjectured collision-resistant hash functions.

Signature Schemes. A *signature scheme* is a triplet (G, S, V) of probabilistic polynomial-time algorithms satisfying the following properties:

- G is the *key generator* algorithm. On input 1^n it outputs a pair $(SK, PK) \in \{0,1\}^{2n}$. SK is called the secret (signing) key and PK is called the public (verification) key.
- S is the *signing* algorithm. On input a message M and the secret key SK, it outputs a signature σ.
- V is the *verification* algorithm. For every $(PK, SK) = G(1^n)$ and $\sigma = S(SK, M)$, it holds that $V(PK, \sigma, M) = 1$.

In [9] security for signature schemes is defined in several variants. The strongest variant is called "existential unforgeability against adaptively chosen message attack". That is, we require that no efficient algorithm will be able to produce a valid signed message, even after seeing several signed messages of its choice.

Stream Signatures. We define a *stream* to be a (possibly infinite) sequence of *blocks* $\mathcal{B} = B_1, B_2, \ldots$ where each $B_i \in \{0,1\}^c$ for some constant[3] c.

We distinguish two cases. In the first case we assume that the stream is finite and known to the sender in advance. We call this the *off-line* case. Conversely in the *on-line* case the signer must process one (or a few) block at the time with no knowledge of the future part of the stream.

Definition 1 *An* off-line stream signature scheme *is a triplet (G, S, V) of probabilistic polynomial-time algorithms satisfying the following properties:*

- *G is the* key generator *algorithm. On input 1^n it outputs a pair $(SK, PK) \in \{0,1\}^{2n}$. SK is called the secret (signing) key and PK is called the public (verification) key.*
- *S is the* signing *algorithm. On input a finite stream $\mathcal{B} = B_1, \ldots, B_k$ and the secret key SK algorithm S outputs a new stream $\mathcal{B}' = B'_1, \ldots, B'_k$ where $B'_i = (B_i, A_i)$.*
- *V is the* verification *algorithm. For every $(PK, SK) = G(1^n)$ and $\mathcal{B}' = S(SK, \mathcal{B})$, it holds that $V(PK, B'_1, \ldots, B'_i) = 1$ for $1 \leq i \leq k$.*

Notice that we modeled the off-line property by the fact that the signing algorithm is given the whole stream in advance. Yet the verifier is required to authenticate *each* prefix of the scheme without needing to see the rest of the stream. As it will become clear in the following our algorithms will not require the off-line verifier to store the whole past stream either.

Definition 2 *An* on-line stream signature scheme *is a triplet (G, S, V) of probabilistic polynomial-time algorithms satisfying the following properties:*

- *G is the* key generator *algorithm. On input 1^n it outputs a pair $(SK, PK) \in \{0,1\}^{2n}$. SK is called the secret (signing) key and PK is called the public (verification) key.*

---

[3] The assumption that the blocks have all the same size is not really necessary. We just make it for clarity of presentation.

- *S is the* signing *algorithm. Given a (possibly infinite) stream $\mathcal{B} = B_1, \ldots$, algorithm S with input the secret key SK process each block one at the time, i.e.,*

$$S(SK, B_1, \ldots, B_i) = B'_i = (B_i, A_i)$$

- *V is the* verification *algorithm. For every $(PK, SK) = G(1^n)$ and $B'_1, B'_2, \ldots$ such that $B'_i = S(SK, B_1, \ldots, B_i)$ for all i, it holds that $V(PK, B'_1, \ldots, B'_i) = 1$ for all i.*

Notice that in the on-line definition we have the signer process each block "on the fly" so knowledge of future blocks is not needed. In this case also the definition seems to requires knowledge of all past blocks for both signer and verifier, however this does not have to be the case (indeed in our solution some past blocks may be discarded).

The above definitions say nothing about security. In order to define security for stream signing we use the same notions of security introduced in [9]. That is, we require that no efficient algorithm will be able to produce a valid signed stream, even after seeing several signed streams. However notice that given our definition of signed streams, a prefix of a valid signed stream is itself a valid signed stream. So the forger can present a "different" signed stream by just taking a prefix of the ones seen before. However this hardly constitutes forgery, so we rule it out in the definition. With $\mathcal{B}^{(1)} \subseteq \mathcal{B}^{(2)}$ we denote the fact that $\mathcal{B}^{(1)}$ is a prefix of $\mathcal{B}^{(2)}$.

Definition 3 *We say that an off-line (resp. on-line) stream signature scheme (G, S, V) is* secure *if any probabilistic polynomial time algorithm F, who is given as input the public key PK and adaptively chosen signed streams $\mathcal{B}'^{(j)}$, outputs a new previously unseen valid signed stream $\mathcal{B}' \not\subseteq \mathcal{B}'^{(j)}$ $\forall j$ only with negligible probability.*

For signed streams we slightly abuse the notation: when we write $\mathcal{B}'^{(1)} \not\subseteq \mathcal{B}'^{(2)}$ we mean that not only $\mathcal{B}'^{(1)}$ is not a prefix of $\mathcal{B}'^{(2)}$ but also the underlying "basic" unsigned streams are in the same relationship, i.e. $\mathcal{B}^{(1)} \not\subseteq \mathcal{B}^{(2)}$.

This is the definition of existential unforgeability against adaptive chosen message attack, the strongest of the notions presented in [9]. Following [9] weaker variants can be defined.

3 The Off-Line Solution

In this case we assume that the sender knows the entire stream in advance. (e.g., music/movie broadcast). Assume for simplicity that the stream is such that it is possible to reserve 20 bytes of extra authentication information in a block of size c.

The stream is logically divided into blocks of size c. The receiver has a buffer of size c. The receiver first receives the signature on the 20 byte hash (e.g., SHA-1) of the first block. After verification of the signature the receiver knows what the hash of the first block should be and then starts receiving the full stream and starts computing its hash block by block. When the receiver receives the first block, it checks its hash against what the signature was verified upon. If it matches, it plays the block otherwise it rejects it and stops playing the stream. How are other blocks authenticated? The key point is that the first block contains the 20 byte hash of the second block, the second block contains the 20 byte hash of the third block and so on... Thus, after the first signature check, there are just hashes to be checked for every subsequent block.

In more detail, let (G, S, V) be a regular signature scheme. The sender has a pair of secret-public key $(SK, PK) = G(1^n)$ of such signature scheme. Also let H be a collision-resistant cryptographic hash function. If the original stream is

$$\mathcal{B} = B_1, B_2, \ldots, B_k$$

and the resulting signed stream is

$$\mathcal{B}' = B'_0, B'_1, B'_2, \ldots, B'_k$$

the processing is done *backwards* on the original stream as follows:

$$B'_k = < B_k, 00 \ldots 0 >$$

$$B'_i = < B_i, H(B'_{i+1}) > \text{ for } i = 1, \ldots, k-1$$

$$B'_0 = < H(B'_1, k), S(SK, H(B'_1, k)) >$$

Notice that on the sender side, computing the signature and embedding the hashes requires a single *backwards* pass on the stream, hence the restriction that the stream is fully known in advance. Notice also that the first block B'_0 of the signed stream contains an encoding of the length of the stream (k).

The receiver verifies the signed stream as follows: on receiving $B'_0 = < B, A_0 >$ she checks that

$$V(PK, A_0, B) = 1$$

and extracts the length k in blocks of the stream. Then on receiving $B'_i = < B_i, A_i >$ (for $1 \leq i \leq k$) the receiver accepts B_i if

$$H(B'_i) = A_{i-1}$$

Thus the receiver has to compute a single public-key operation at the beginning, and then only one hash evaluation per block. Notice that no big table is needed in memory.

4 The On-Line Solution

In this case the sender does not know the entire stream in advance (e.g, live broadcast). In this scenario it is important that also the operation of signing (and not just verification) be fast, since the sender himself is bound to produce an authenticated stream at a potentially high rate.

ONE-TIME SIGNATURES. In the following we will use a special kind of signature scheme introduced in [11, 12]. These are signatures which are much faster to compute and verify than regular signatures since they are based on one-way functions and do not require a trapdoor function. Conjectured known one-way functions (as DES or SHA-1) are much more efficient then the known conjectured trapdoor functions as RSA. However these schemes cannot be used to sign an arbitrary number of messages but only a prefixed number of them (usually one). Several other 1-time schemes have been proposed [8, 3, 4]; in Section 6 we discuss possible instantiations for our purpose.

In this case also the stream is split into blocks. Initially the sender sends a signed public key for a 1-time signature scheme. Then he sends the first block along with a 1-time signature on its hash based on the 1-time public key sent in the previous block. The first block also contains a new 1-time public key to be used to verify the signature on the 2nd block and this structure is repeated in all the blocks.

More in detail: let us denote with (G, S, V) a regular signature scheme and with (g, s, v) a 1-time signature scheme. With H we still denote a collision-resistant hash function. The sender has long-lived keys $(SK, PK) = G(1^n)$. Let

$$\mathcal{B} = B_1, B_2, \ldots$$

be the original stream (notice that in this case we are not assuming the stream to be finite) and

$$\mathcal{B}' = B'_0, B'_1, B'_2, \ldots$$

the signed stream constructed as follows. For each $i \geq 1$ let us denote with $(sk_i, pk_i) = g(1^n)$ the output of an independent run of algorithm g. Then

$$B'_0 =< pk_0, S(SK, pk_0) >$$

(public keys of 1-time signature schemes are usually short so they need not to be hashed before signing)

$$B'_i =< B_i, pk_i, s(sk_{i-1}, H(B_i, pk_i)) > \text{ for } i \geq 1$$

Notice that apart from a regular signature on the first block, all the following signatures are 1-time ones, thus much faster to compute (including the key generation, which however does not have to be done on the fly.)

The receiver verifies the signed stream as follows. On receiving $B'_0 =< pk_0, A_0 >$ she checks that

$$V(PK, A_0, pk_0) = 1$$

and then on receiving $B'_i =< B_i, pk_{i+1}, A_i >$ she checks that

$$v(pk_{i-1}, A_i, H(B_i, pk_i)) = 1$$

whenever one of these checks fails, the receiver stops playing the stream. Thus the receiver has to compute a single public-key operation at the beginning, and then only one 1-time signature verification per block.

5 Proofs of Security

We are able to prove the security of our stream signature schemes according to the definitions presented in Section 2, provided that the underlying components used to build the schemes are secure on their own.

THE OFF-LINE CASE. Let us denote with $(\mathcal{G}_{off}, \mathcal{S}_{off}, \mathcal{V}_{off})$ the off-line stream signature scheme described in Section 3. With (G, S, V) let us denote the "regular" signature scheme and with H the hash function used in the construction. The following holds.

Theorem 1 *If (G, S, V) is a secure signature scheme and H is a collision-resistant hash function then the resulting stream signature scheme $(\mathcal{G}_{off}, \mathcal{S}_{off}, \mathcal{V}_{off})$ is secure.*

Proof

Assume the thesis is false, i.e. that there is an algorithm $\mathcal{F}$ that succeeds in an existential forgery against $(\mathcal{G}_{off}, \mathcal{S}_{off}, \mathcal{V}_{off})$ using an adaptive chosen message attack with non-negligible probability ϵ. That is, $\mathcal{F}$ runs on input PK, adaptively asks for the signed versions of ℓ streams $\mathcal{B}^{(1)}, \ldots, \mathcal{B}^{(\ell)}$ where $\mathcal{B}^{(j)} = B_1^{(j)} \ldots B_{k_j}^{(j)}$ and receives them, let them be $\mathcal{B}'^{(1)}, \ldots, \mathcal{B}'^{(\ell)}$. Then $\mathcal{F}$ outputs a valid signed stream $\mathcal{B}'$ where $\mathcal{B}' = B'_0 \ldots B'_k$ which is not a prefix of any of the previous ones, i.e. $\mathcal{B}' \not\subseteq \mathcal{B}'^{(j)}$ $\forall j = 1, \ldots, \ell$. One of the following two cases must be true.

Case 1: *With probability at least $\epsilon/2$, $\mathcal{F}$ outputs a signed stream whose first block coincides with the first block of one of the signed streams it asked for before.* That is there exists a j such that $B'_0 = B'^{(j)}_0$. We show in this case how $\mathcal{F}$ can be used to build an algorithm $\mathcal{F}_c$ which finds collisions for H.

$\mathcal{F}_c$ runs as follows. It first runs $\mathcal{G}$ to obtain a pair of public and secret key (PK, SK). Then it runs $\mathcal{F}$ and uses SK to sign all the requests $\mathcal{B}^{(j)}$ for $j = 1, \ldots, \ell$ that $\mathcal{F}$ makes. Now with probability at least $\epsilon/2$ $\mathcal{F}$ outputs $\mathcal{B}' \not\subseteq \mathcal{B}'^{(j)}$ for all j, but such that there exists j such that $B'_0 = B'^{(j)}_0$. Let k be the length of the stream $\mathcal{B}'$ and k_j the length of the stream $\mathcal{B}'^{(j)}$.

If $k \neq k_j$ then $(B'_1, k) \neq (B'^{(j)}_1, k_j)$. But since $B'_0 = B'^{(j)}_0$ it must be that $H(B'_1, k) = H(B'^{(j)}_1, k_j)$. So $\mathcal{F}_c$ can output the pair $(B'_1, k), (B'^{(j)}_1, k_j)$ as a collision for H.

If $k = k_j$, then since $\mathcal{B}' \not\subseteq \mathcal{B}'^{(j)}$ there must exist an $0 < \alpha \leq k$ such that $B'_\alpha \neq B'^{(j)}_\alpha$ while $B'_\beta = B'^{(j)}_\beta$ for all $0 \leq \beta < \alpha$. That is we have that $B'_{\alpha-1} = B'^{(j)}_{\alpha-1}$, which (by our construction of signed streams) implies that $H(B'_\alpha) = H(B'^{(j)}_\alpha)$. So $\mathcal{F}_c$ outputs the pair $B'_\alpha, B'^{(j)}_\alpha$ as a collision for H.

Case 2: *With probability at least $\epsilon/2$, $\mathcal{F}$ outputs a signed stream whose first block is different from the first block of any of the signed streams it asked for before.* That is, for all j we have $B'_0 \neq B'^{(j)}_0$. We show in this case how $\mathcal{F}$ can be used to build an algorithm $\mathcal{F}_s$ which forges signatures for the regular signature scheme (G, S, V).

$\mathcal{F}_s$ runs on input a public key PK of the signature scheme (G, S, V). It is also allowed to query a signature oracle to get signed messages of its choice. $\mathcal{F}_s$ starts by running $\mathcal{F}$. When the latter requests a signature on a stream $\mathcal{B}^{(j)} = B_1^{(j)} \ldots B_{k_j}^{(j)}$ $\mathcal{F}_s$ first prepares the last k_j blocks of the signed stream $B'^{(j)}_1, \ldots, B'^{(j)}_{k_j}$ and then it uses its allowed queries to the signature oracle to compute $B'^{(j)}_0 =< H(B'^{(j)}_1), S(SK, H(B'^{(j)}_1)) >$.

Now with probability at least $\epsilon/2$ $\mathcal{F}$ outputs $\mathcal{B}' = B'_0 \ldots B'_k$ such that $B'_0 \neq B'^{(j)}_0$ for all j. This in turn means that B'_0 is a message/signature pair that $\mathcal{F}_s$ has not asked before to the signature oracle. $\mathcal{F}_s$ stops outputting such pair as the forged signature.

Since $\epsilon/2$ is a non-negligible probability, either Case 1 or Case 2 contradicts our hypothesis, so the thesis must be true.

■

THE ON-LINE CASE. Let us denote with $(\mathcal{G}_{on}, \mathcal{S}_{on}, \mathcal{V}_{on})$ the on-line stream signature scheme described in Section 4. With (G, S, V) let us denote the "regular" signature scheme, with

(g, s, v) the one-time signature scheme and with H the hash function used in the construction. The following holds.

Theorem 2 *If (G, S, V) and (g, s, v) are secure signature schemes and H is a collision-resistant hash function then the resulting stream signature scheme $(\mathcal{G}_{on}, \mathcal{S}_{on}, \mathcal{V}_{on})$ is secure.*

Proof

Assume the thesis is false, i.e. that there is an algorithm $\mathcal{F}$ that succeeds in an existential forgery against $(\mathcal{G}_{on}, \mathcal{S}_{on}, \mathcal{V}_{on})$ using an adaptive chosen message attack with non-negligible probability ϵ. That is $\mathcal{F}$ runs on input PK, adaptively asks for the signed versions of ℓ streams[4] $\mathcal{B}^{(1)}, \ldots, \mathcal{B}^{(\ell)}$ where $\mathcal{B}^{(j)} = B_1^{(j)} \ldots$ and receives them, let them be $\mathcal{B}'^{(1)}, \ldots, \mathcal{B}'^{(\ell)}$. Then $\mathcal{F}$ outputs a valid signed stream $\mathcal{B}'$ where $\mathcal{B}' = B'_0, \ldots$ which is not a prefix of any of the previous ones, i.e. $\mathcal{B}' \not\subseteq \mathcal{B}'^{(j)}\ \forall j = 1, \ldots, \ell$. One of the following cases must be true.

Case 1: *With probability at least $\epsilon/2$, $\mathcal{F}$ outputs a signed stream whose first block is different from the first block of any of the signed streams it asked for before.* That is, for all j we have $B'_0 \neq B'^{(j)}_0$. We show in this case how $\mathcal{F}$ can be used to build an algorithm $\mathcal{F}_1$ which forges signatures for the regular signature scheme (G, S, V).

$\mathcal{F}_1$ runs on input a public key PK of the signature scheme (G, S, V). It is also allowed to query a signature oracle to get signed messages of its choice. $\mathcal{F}_1$ starts by running $\mathcal{F}$. When the latter requests a signature on a stream $\mathcal{B}^{(j)} = B_1^{(j)} \ldots$, $\mathcal{F}_s$ prepares the first block of the signed stream $B'^{(j)}_0$ by running g to get $(pk_0^{(j)}, sk_0^{(j)})$ and then queries the signature oracle to get a signature on pk_0. The remaining blocks of the stream can be easily prepared by $\mathcal{F}_1$ by running the 1-time key generator algorithm g as needed

Now with probability at least $\epsilon/2$ $\mathcal{F}$ outputs $\mathcal{B}' = B'_0 \ldots$ such that $B'_0 \neq B'^{(j)}_0$ for all j. This in turn means that the B'_0 is a message/signature pair for (G, S, V) that $\mathcal{F}_1$ has not asked before to the signature oracle. $\mathcal{F}_1$ stops outputting such pair as the forged signature.

Case 2: *With probability at least $\epsilon/2$, $\mathcal{F}$ outputs a signed stream whose first block coincides with the first block of one of the signed streams it asked for before.* That is there exists a j such that $B'_0 = B'^{(j)}_0$. However we also know that $\mathcal{B}' \not\subseteq \mathcal{B}'^{(j)}$. This implies that there exists an $\alpha > 0$ such that $B'_\alpha \neq B'^{(j)}_\alpha$ while $B'_{\alpha-1} = B'^{(j)}_{\alpha-1}$. Recall that, by our construction of signed streams, we have

$$B'_\alpha = < B_\alpha, pk_\alpha, s(sk_{\alpha-1}, H(B_\alpha, pk_\alpha)) >$$

$$B'^{(j)}_\alpha = < B_\alpha^{(j)}, pk_\alpha^{(j)}, s(sk_{\alpha-1}^{(j)}, H(B_\alpha^{(j)}, pk_\alpha^{(j)})) >$$

We know that $B_\alpha \neq B_\alpha^{(j)}$. One of this two subcases must be true.

Case 2a: *With probability at least $\epsilon/4$ the output of $\mathcal{F}$ is such that $H(B_\alpha, pk_\alpha) = H(B_\alpha^{(j)}, pk_\alpha^{(j)})$* We show how to construct an algorithm $\mathcal{F}_2$ that computes collisions for H.

$\mathcal{F}_2$ runs as follows. It first runs $\mathcal{G}$ to obtain a pair of public and secret key (PK, SK). Then it runs $\mathcal{F}$. When the latter asks to sign a specific stream, $\mathcal{F}_2$ uses SK to sign the first block and then generates "on the fly" one-time keys (using g) to sign all the other blocks of the given stream. Then $\mathcal{F}_2$ looks at the output of $\mathcal{F}$ which by assumption has the property that

---

[4] We may assume that the forger adaptively chooses the components of the stream, i.e. after seeing the i^{th} block signed it creates the $(i+1)^{st}$ block to be signed

$$H(B_\alpha, pk_\alpha) = H(B_\alpha^{(j)}, pk_\alpha^{(j)})$$

and since we know that $B_\alpha \neq B_\alpha^{(j)}$, $\mathcal{F}_2$ has found a collision.

Case 2b: *With probability at least $\epsilon/4$ the output of $\mathcal{F}$ is such that $H(B_\alpha, pk_\alpha) \neq H(B_\alpha^{(j)}, pk_\alpha^{(j)})$* We show how to construct an algorithm $\mathcal{F}_3$ that forges signatures in the 1-time scheme (g, s, v).

Let us denote with T an upper bound on the total number of stream blocks that $\mathcal{F}$ asks during its attempt at forgery. Without loss of generality let's assume that $\mathcal{F}$ always asks T blocks. Clearly T is polynomial in our security parameter n.

$\mathcal{F}_3$ works as follows. It runs on input a 1-time key pk and it can ask one query to get a single message signed by the corresponding secret key sk. Its goal is to output a different message and its signature under sk.

$\mathcal{F}_3$ runs as follows. It runs G to obtain a long-lived key pair PK, SK of (G, S, V). It then runs g several times in order to obtain several one-time key pairs. Finally it selects uniformly at random an integer i between 1 and $T - 1$.

$\mathcal{F}_3$ starts running $\mathcal{F}$. Whenever the latter asks for a signed stream, $\mathcal{F}_3$ can sign the first block since it knows SK and it uses the generated 1-time keys for the internal blocks. The exception is when $\mathcal{F}$ asks for the i^{th} block (sequentially). In this case $\mathcal{F}_3$ uses pk as the 1-time public key embedded in the i^{th} block. This means that when $\mathcal{F}$ asks for the $(i+1)^{st}$ block, $\mathcal{F}_3$ has query the signature oracle in order to compute the 1-time signature embedded in it.

At the end of this process, $\mathcal{F}$ stops outputting a signed stream with the properties outlined above. With probability $1/T$ the block $B'^{(j)}_{\alpha-1}$ is the one in which $\mathcal{F}_3$ used the target public key pk. This in turn means that the 1-time signature contained in the block B'_α outputted by $\mathcal{F}$ is valid under the key pk, yet it is on a different message than the one queried by $\mathcal{F}_3$.

So with probability $\epsilon/4T$, which is still non-negligible, the forger $\mathcal{F}_3$ succeeds.

■

Remark 1: In the bodies and proofs of the above theorems we meant security as "existential unforgeability against adaptively chosen message attack". However the theorems hold for any notion of security defined in [9], that is the stream signature scheme inherits the same kind of security of the underlying signature scheme(s) provided that the hash function is collision-resistant.

Remark 2: The statements of the above theorems are valid not only in asymptotic terms, but have also a concrete interpretation which ultimately is reflected in the key lengths used in the various components in order to achieve the desired level of security of the full construction. It is not hard to see, by a close analysis of the proofs, that the results are pretty tight. That is, a forger for the stream signing scheme can be transformed into an attacker for one of the components (the hash function, the regular signature scheme and, a little less optimally, the 1-time signature scheme) which runs in about the same time, asks the same number of queries and has almost the same success probability. This is turns means that there is no major degradation in the level of security of the compound scheme and thus the basic components can be run with keys of ordinary length.

6 Implementation Issues

6.1 The Choice of the One-Time Signature Scheme

Several one-time schemes have been presented in the literature, see for example [11, 12, 8, 3, 4]. The main parameters of these schemes are signature length and verification time. In the solutions we know, these parameters impose conflicting requirements, i.e. if one wants a scheme with short signatures, verification time goes up, while schemes with longer signatures can have a much shorter verification time. In our on-line solution we would like to keep both parameters down. Indeed the verification should be fast enough to allow the receiver to consume the stream blocks at the same input rate she receives them. At the same time, since the signatures are embedded in the stream, it's important to keep them small so that they will not reduce the throughput rate of the original stream.

We first suggest a scheme which obtain a reasonable compromise. The scheme is based on a 1-way function F in a domain D. It also uses a collision resistant hash function H. The scheme allows signing of a single m-bit message. It is based on a combinations of ideas from [11, 20]. Here are the details of the scheme.

Key Generation. Choose $m + \log m$ elements in D, let them be $a_1, \ldots, a_{m+\log m}$. This is the secret key. The public key is

$$pk = H(F(a_1), \ldots, F(a_{m+\log m}))$$

Signing Algorithm. Let M be the message to be signed. Append to M the binary representation of the number of zero's in M's binary representation. Call M' the resulting binary string. The signature of M is $s_1, \ldots, s_{m+\log m}$ where $s_i = a_i$ if the i^{th} bit of M' is 1 otherwise $s_i = F(a_i)$.

Verification Algorithm. Check if

$$H(t_1, \ldots, t_{m+\log m}) = pk$$

where $t_i = s_i$ if i^{th} bit of M' is 0 otherwise $t_i = F(s_i)$.

Security. Intuitively this scheme is secure since it is not possible to change a 0 into a 1 in the binary representation of the message M without having to invert the function F. It is possible to change a 1 into a 0, but that will increase the number of 0's in the binary representation of M causing a bit to flip from 0 to 1 in the last $\log m$ bits of M', and so forcing the attacker to invert F anyway.

Parameters. This scheme has signature length $|D|(m + \log m)$ where $|D|$ is the number of bits required to represent elements of D. The receiver has to compute 1 hash computation of H plus on the average $\frac{m+\log m}{2}$ computations of F.

In practice we assume that F maps 64-bit long strings into 64-bit long strings. Since collision resistance is not required from F we believe this parameter to be sufficient. Conjectured good F's can be easily constructed from efficient block ciphers like DES or from fast hash functions like MD5 or SHA-1. [5] Similarly H can be instantiated to MD5 or SHA-1. In general we may assume m to be 128 or 160 if the message to be signed if first hashed using MD5 or SHA-1.

The SHA-1 implementation has then signatures which are 1344 bytes long. The receiver has to compute F around 84 times on the average. With MD5 the numbers become 1080 bytes and 68 respectively. When used in our off-line scheme one also has to add 16 bytes for the embedding of the public key in the stream.

Remark: Comparing the RSA signature scheme with verification exponent 3 with the above schemes, one could wonder if the verification algorithm is really more efficient (2 multiplications versus 84 hash computations). Typical estimates today are that an RSA verification is comparable to 100 hash computations. However we remind the reader that we are trying to improve *both* signature generation and verification. Indeed this scheme is used in the on-line case and as such both operations have to be performed on-line and thus efficiently. The improvement in signature generation is much more substantial.

Other schemes: The scheme above could be improved in the length of the signature by using Winternitz's idea [20]. The public key is composed by $m+1$ elements of D, let them be $a_0, a_1, \ldots, a_m$. The public key is $pk = H(F^m(a_0), F(a_1), \ldots, F(a_m))$. The signature on a message M is $s_0, s_1, \ldots, s_m$ where for $i \geq 1$, $s_i = a_i$ if the i^{th} bit of M' is 1 otherwise $s_i = F(a_i)$, while $s_0 = F^{\ell}(a_0)$ where ℓ is the number of 1's in M's binary representation. The verification of the signature takes the same amount of computation as the scheme described above. However the length of the signature is slightly shorter, i.e. $|D|(m+1)$ bits. For a SHA-1 implementation this means 1288 bytes. As noticed in [8] the security of this scheme is based on a somewhat stronger assumption on F. Namely F is assumed to be *non-quasi-invertible*, that is on input y it infeasible to find an index i and a value x such that $F^{i+1}(x) = F^i(y)$. Clearly if F is a one-way permutation the above notion is automatically satisfied.

This scheme can be further generalized as described in [8]. The message M is split in $\frac{m}{t}$ blocks of size t bits, let them be $M_1, \ldots, M_{\frac{m}{t}}$. The secret keys is composed of $\frac{m}{t}+1$ elements of D, let them be $a_0, a_1, \ldots, a_{\frac{m}{t}}$. The public key is $pk = H(F^{(2^t-1)\frac{m}{t}}(a_0), F^{2^t-1}(a_1), \ldots, F^{2^t-1}(a_{\frac{m}{t}}))$. The signature of a message M is computed by considering the integer value of the blocks M_i. The signature is composed by $\frac{m}{t}+1$ values $s_0, s_1, \ldots, s_{\frac{m}{t}}$ where, for $i \geq 1$, $s_i = F^{2^t-1-M_i}(x_i)$, while $s_0 = F^{\sum_i M_i}(x_0)$. In this case the signature length is $|D|(\frac{m}{t}+1)$ bits, but verification time goes up with the parameter t. Indeed the number of hash computations goes as $O(\frac{2^t m}{t})$. Also this scheme is based on the non-quasi-invertibility of F.

In general one has to look at the specific application and decide among the tradeoffs specified by the parameter t in order to decide if it is better to reduce the signature length or the computation time. See also Section 6.3 for other ways to deal with this issue.

---

[5] As a cautionary remark to prevent attacks where the attacker builds a large table of evaluations of F, in practice F could be made different for each signed stream (or for each large portion of the signed stream) by defining $F(x)$ to be $G(Salt||X)$ where G is a one-way 128 bit to 64 bit function, and the *Salt* is generated at random by the signer once for each stream or large pieces thereof.

6.2 Non-Repudiation

In case of a legal dispute over a content of a signed stream the receiver must bring to court some evidence. If the receiver saves the whole stream, then there is no problem. However in some cases, for example because of memory limitations, the receiver may be forced to discard the stream data after having consumed it. In these cases what should she save to protect herself in case of a legal dispute?

In the off-line solution, assuming the last block of the signed stream always has a special reserved value for the hash-chaining field, (say all 0's) she needs to save only the first signed block. Indeed this proves that she received something from the sender. Now we could conceivably move the burden of proof to the sender to reconstruct the whole stream that matches that first block and ends with the last block which has the reserved value for the hash-chaining field.

Similarly in the on-line solution, at a minimum the receiver needs to save the first signed block and all 1-time signatures and have the sender reconstruct the stream. However in practice, this may still be too much to save. For practical applications, we suggest the following scheme. The on-line stream could be broken up into a sequence of chunks, each chunk representing a logical unit, e.g., a TV programme or a live broadcast of a game or even programming sent over some fixed sized time interval. The idea is that once a logical unit is decided, an upper bound on the total number of blocks in it can be estimated and all the one-time keys needed for a chunk of that size could be precomputed by the sender. In addition, by using the off-line stream signing technique, the sender can compute a single hash value which when signed can be used to authenticate each of these one-time public keys if they were to be sent as a stream of keys, one key in each stream block. The new on-line scheme works as follows: Initally the sender sends a regular digital signature on the hash value which can be used to authenticate a stream of one-time public keys. The stream of one-time keys is then embedded in the actual on-line data stream and the data stream itself is authenticated by one-time signatures based on the one-time keys where each one-time signature is on the running hash of all the data sent so far on the stream. This way the receiver only needs to store the initial regular digital signature and the last one-time signature that was received and verified. For non-repudiation purposes, based on the initial regular digital signature, the sender can be forced to disclose the entire stream of one-time public keys, and, based on the last valid one-time signature stored by the receiver, the sender can be forced to produce a data stream with the same hash as what is signed by the last one-time signature stored by the receiver.

6.3 Hybrid Schemes

In the on-line scheme, the length of the embedded authentication information is of concern as it could cut into the throughput of the stream. In order to reduce it hybrid schemes can be considered. In this case we assume that some asynchrony between the sender and receiver is acceptable.

Suppose the sender can process a group (say 20) of stream blocks at a time before sending them. With a pipelined process this would only add an initial delay before the stream gets transmitted. The sender will sign with a one-time key only 1 block out of 20. The 20 blocks in between these two signed blocks will be authenticated using the off-line scheme. This way the long 1-time signatures and the the verification time can be amortized over the 20 blocks.

A useful feature of our proposed 1-time signature scheme is that it allows the verification of (the hash of) a message bit by bit. This allows us to actually "spread out" the signature bits and the verification time among the 20 blocks. Indeed if we assume that the receiver is allowed to play at most 20 blocks of unauthenticated information before stopping if tampering is detected we can do the following. We can distribute the signature bits among the 20 blocks and verify the hash of the first block bit by bit as the signature bits arrive. This maintains the stream rate stable since we do not have long signatures sent in a single block and verification now takes 3-4 hash computations per block, on *every* block.

It is also possible to remove the constraint on playing 20 blocks of unauthenticated information before tampering is detected. This requires a simple modification to our on-line scheme. Instead of embedding in block B_i its own 1-time signature, we embed the signature of the next block B_{i+1}. This means that in the on-line case blocks have to be processed two at a time now. When this modification is applied to the hybrid scheme, the signature bits in the current 20 blocks are used to authenticate the following 20 blocks so unauthenticated information is never played. However this means that now the sender has to process 40 blocks at a time in the hybrid scheme.

6.4 Probabilistic one-time signatures.

The length and the computation time of a 1-time signature are determined by the length of the message being signed. Typically the message is first hashed, thus the range of the collision-resistant hash function is the crucial parameter here. However in order to make sure that the function is a strong collision-resistant one, it is necessary to assume a long range. MD5 with 128-bits is considered borderline. SHA-1 seems to give more security with a 160-bits range.

Is it possible to use a weak collision resistant hash function to hash messages instead? In the presence of a chosen message attack this is not possible as an attacker could find two message x and y that hash to the same value and by obtaining a signature on x would then obtain a signature on y. It is however possible to foil this particular attacks by first randomizing the message being signed.

Consider the following approach. Let (G, S, V) be a secure signature scheme (against adaptive chosen message attack) on a message of size b, but we want to sign messages of arbitrary length. Let H_k be a family of weak collision resistant hash functions whose range is $(b - |k|)$-bit strings. Consider the following new signature scheme on messages of arbitrary size. On input a message M, choose a random k, compute $h = H_k(M)$ and output k together with a signature on the pair k, h. We claim that this new signature scheme is also secure against adaptive chosen message attack. This is because the receiver cannot use the fact that H_k is a weak collision hash function, because she does not know which function is going to be used.

The only issue here is non-repudiation as the signer can find collisions and when it is challenged with a signed message M he can present another message M' that has the same signature. However this is not really a problem as one can hold the signer responsible for *any* signed message as only he can find collisions.

Notice that the family H_k can be easily built out of regular hash functions via "keying" techniques. For example in iterative constructions of H, k could be used as the IV. In practice we suggest to use the first 80 bits of SHA-1 keyed via the IV with k.

The above technique does not have a particular impact with typical signature schemes as reducing the range of the hash function is not an issue there. But with 1-time signatures this allows for some savings in the length of the signature and the computation time. For example with typical lengths $|k|$ would be say 60 bits and a weak collision hash function would be around 80 bits in range (to obtain a level of security comparable to the 160-bit strong collision-resistant functions). So the probabilistic hashing would return a value of 140 bits which implies a savings of almost 15% in length and computation time. We stress that this a general result for all 1-time signature schemes.

When applied to our stream signature scheme that improvement itself is already valuable. But in our specific case we can improve even more as we can use the hybrid approach once again. Indeed we don't have to use a different k for each block. The signer could keep k fixed for a limited amount of time. This time window should be small enough to prevent an attacker from finding collisions for the hash function H_k which is kept fixed during this time window. If H_k is a good hash function it should take roughly 2^{40} steps to find such collisions. Thus a small window of a few seconds should not pose security problems. If the rate of the stream is high enough during this time window we will sign several blocks. Say for simplicity that we sign 20 blocks with the same k. This will allow us to spread the bits of k on 20 blocks which means that now the probabilistic hashing returns an 83-bit value per block, thus achieving almost a 50% efficiency improvement in signatures length and computation time!

7 Applications

MPEG VIDEO AND AUDIO. In the case of MPEG video and audio, there are several methods for embedding authentication data. Firstly, the Video Elementary stream has a USER-DATA section where arbitrary user defined information can be placed. Secondly, the MPEG system layer allows for an elementary data stream to be multiplexed synchronously with the packetized audio and video streams. One such elementary stream could carry the authentication information. Thirdly, techniques borrowed from digital watermarking can be used to embed information in the audio and video itself at the cost of slight quality degradation. In the case of MPEG video since each frame is fairly large, (hundreds of kilobits) and the receiver is required to have a buffer of at least 1.8Mbits, both the off-line as well as the on-line solutions can be deployed without compromising picture quality. In the case of audio however, in the extreme case the bit rate could be very low (e.g., 32Kbits/s) and each audio frame could be small (approx. 1000 bytes) and the receiver's audio buffer may be tiny (¡ 2 Kbytes). In such extreme cases the on-line method, which requires around 1000 bytes of authentication information per block cannot be used without seriously cutting into audio quality. For these extreme cases, the best on-line strategy would be either to send the authentication information via a separate but multiplexed MPEG data stream. For regular MPEG audio, if the receiver has a reasonably sized buffer (say 32K) then by having a large audio block (say 20K) our on-line scheme would a server-introduced delay of approximately 5-6 seconds and a 5% quality degradation. If the receiver buffer is small but not tiny (say 3 K) a hybrid scheme would work: as an example of a scheme that can be built one could use groups of 33 hash–chained blocks of length 1000 bytes each; this would typically result in a 5% degradation and a server initial delay in the 20 second range.

JAVA. In the original version of java (JDK 1.0), for an applet coming from the network, first the startup class was loaded and then additional classes were (down) loaded by the class loader in a lazy fashion as and when the running applet first attempted to access them. Since our ideas apply not only to streams which are a linear sequence of blocks but in general to trees as well (where one block can invoke any of its children), based on our model, one way to sign java applets would be to sign the startup class and each downloaded class would have embedded in it the hashes of the additional classes that it downloads directly. However for code signing, Javasoft has adopted the multiple signature and hash table based approach in JDK1.1, where each applet is composed of one or several Java archives, each of which contains a signed table of hashes (the manifest) of its components. It is our belief that once java applets become really large and complex the shortcomings of this approach will become apparent: (1) the large size of the hash table in relation to the classes actually invoked during a run. This table has to fully extracted and authenticated before any class gets authenticated; (2) the computational cost of signing each of the manifests if an applet is composed of several archives; (3) accommodating classes or data resources which are generated on the fly by the application server based on a client request.

These could be addressed by using some of our techniques. Also the problem of how to sign audio/video streams will have to be considered in the future evolution of Java, since putting the hash of a large audio/video file in the manifest would not be acceptable.

BROADCAST APPLICATIONS. Our schemes (both the off-line and the on-line one) can be easily modified to fit in a broadcast scenario. Assume that the stream is being sent to a broadcast channel with multiple receivers who dynamically join or leave the channel. In this case a receiver who joins when the transmission is already started will not be able to authenticate the stream since she missed the first block that contained the signature. Both schemes however can be modified so that every once in a while apart from the regular chaining information, there will also be a regular digital signature on a block embedded in the stream. Receivers who are already verifying the stream via the chaining mechanism can ignore this signature whereas receivers tuned in at various time will rely on the first such signature they encounter to start their authentication chain. A different method to authenticate broadcasted streams, with weaker non-repudiation properties than ours, was proposed in [10].

LONG FILES WHEN COMMUNICATION IS AT COST. Our solution can be used also to authenticate long files in a way to reduce communication cost in case of tampering. Suppose that a receiver is downloading a long file from the Web. There is no "stream requirement" to consume the file as it is downloaded, so the receiver could easily receive the whole file and then check a signature at the end. However if the file has been tampered with, the user will be able to detect this fact only at the end. Since communication is at a cost (time spent online, bandwidth wasted etc) this is not a satisfactory solution. Using our solution the receiver can interrupt the transmission as soon as tampering is detected thus saving precious communication resources.

SIGNATURE BASED CONTENT-FILTERING AT PROXIES. Recently there has been interest in using digital signatures as a possible way to filter content admitted in by proxy servers through firewalls. Essentially when there is a firewall and one wishes to connect to an external server, then this connection can only be done via a proxy server. In essence one establishes a connection to a proxy and the proxy establishes a separate connection to the external server (if that is permitted). The proxy then simulates a connection between the internal machine and the external machine by copying data between the two connections. There has been some interest in modifying proxies so that they would only allow signed data to flow from the external server to the internal machine. However, since the proxy is only copying data as it arrives from the external connection into the internal connection and it cannot store all the incoming data before transferring it, the proxy cannot use a regular signature scheme for solving this problem. However, it is easy to see that in the proxy's view the data is a stream. Hence if there could be some standardized way to embed authentication data in such streams then techniques from this paper would prove useful in solving this problem.

8 Acknowledgments

We would like to thank Hugo Krawczyk for his advice, guidance and encouragement for this work. We would also like to thank Mihir Bellare, Ran Canetti and Mike Wiener for helpful discussions.

References

[1] M. Bellare, S. Micali. How to Sign Given any Trapdoor Permutation. *J. of the ACM*, 39(1):214–233, 1992.

[2] J. Benaloh, M. de Mare. One-Way Accumulators: A Decentralized Alternative to Digital Signatures. *Advances in Cryptology-EUROCRYPT'93.* LNCS, vol.765, pp.274–285, Springer–Verlag, 1994.

[3] D. Bleichenbacher, U. Maurer. Optimal Tree-Based One-time Digital Signature Schemes. *STACS'96*, LNCS, Vol. 1046, pp.363–374, Springer-Verlag.

[4] D. Bleichenbacher, U. Maurer. On the efficiency of one-time digital signatures. *Advances in Cryptology-ASYACRYPT'96*, to appear.

[5] I. Damgard. A Design Principle for Hash Functions. *Advances in Cryptology-Crypto'89.* LNCS, vol.435, Springer–Verlag, 1990.

[6] W. Diffie, M. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT-22(6):74–84, 1976.

[7] T. ElGamal. A Public-Key Cryptosystem and a Signature Scheme based on Discrete Logarithms. *IEEE Transactions on Information Theory*, IT-31(4):469–472, 1985.

[8] S. Even, O. Goldreich, S. Micali. On–Line/Off–Line Digital Signatures. *J. of Cryptology*, 9(1):35–67, 1996.

[9] S. Goldwasser, S. Micali, R. Rivest. A Digital Signature Scheme Secure Against Adaptive Chosen Message Attack. *SIAM J. Comp.* 17(2):281–308, 1988.

[10] G. Itkis. Asymmetric MACs. Rump talk at Crypto'96.

[11] L. Lamport. Constructing Digital Signatures from a One-Way Function. *Technical Report SRI Intl.* CSL 98, 1979.

[12] R. Merkle. A Digital Signature based on a Conventional Encryption Function. *Advances in Cryptology–Crypto'87.* LNCS, vol.293, pp. 369–378, Springer–Verlag, 1988.

[13] R. Merkle. A Certified Digital Signature. *Advances in Cryptology–Crypto'89.* LNCS, vol.435, pp. 218–238, Springer–Verlag, 1990.

[14] R. Merkle. One-way Hash Functions and DES. *Advances in Cryptology–Crypto'89.* LNCS, vol.435, Springer–Verlag, 1990.

[15] National Institute of Standard and Technology. Secure Hash Standard. NIST FIPS Pub 180-1, 1995.

[16] M. Naor, M. Yung. Universal One-Way Hash Functions and their Cryptographic Applications. *Proceedings of STOC 1989*, pp.33–43.

[17] R. Rivest. The MD5 Message Digest Algorithm. Internet Request for Comments. April 1992.

[18] R. Rivest, A. Shamir, L. Adleman. A Method for Obtaining Digital Signatures and Public Key Cryptosystems. *Comm. of the ACM*, 21(2):120–126, 1978.

[19] J. Rompel. One-Way Functions are Necessary and Sufficient for Secure Signatures. *Proceedings of STOC 1990*, pp.387–394.

[20] Winternitz. Personal communication to R. Merkle.