A Fast Protocol for Authenticated Stream Delivery

J. R. M. Monteiro *C. L. Lucchesi*
R. Dahab

Relatório Técnico IC–99-20

Setembro de 1999

A Fast Protocol for Authenticated Stream Delivery

J. R. M. Monteiro* C. L. Lucchesi† R. Dahab‡

Abstract

We present a fast, synchronous cryptographic protocol by means of which Alice can send Bob a stream $\mathcal{B}$, with assurance of non-repudiation of reception of $\mathcal{B}$ or any prefix of it by Bob. Bob, on the other hand, can prove to any third party the origin of $\mathcal{B}$ or any prefix of it. While the protocol's synchronous nature may prevent its use in some broadcast-type applications, its speed and mutual non-repudiation capabilities make it well suited for several e-commerce applications.

1 Introduction

1.1 The Problem

Alice wants to send Bob a stream $\mathcal{B} = b_1 b_2 \cdots b_n$ of *data blocks*. During the execution of the protocol, Alice and Bob want to have mutual assurance of each other's identity. Moreover, (i) Bob wants to verify the authenticity of each block sent by Alice, before the next block is sent and (ii) Alice wants to verify that Bob has received a block before sending him the next block.

In addition, Alice wants to have non-repudiation ability, i.e., she would like to be able to prove to a third party that she has indeed sent Bob a particular sequence of blocks, and also that Bob has received such data on a specified occasion, identified by a timestamp.

Transmission of streams require fast authentication and signature verification schemes [1]. Furthermore, $\mathcal{B}$ may not be entirely known in advance, perhaps not even the length n of $\mathcal{B}$ may be known beforehand, which precludes the signing of $\mathcal{B}$ at once. Examples of streams are video and audio data, applets and live feeds from various sources, such as stock markets.

1.2 Cryptographic Tools and Notation

We make use of cryptographic hash functions, i.e., one-way, collision-resistant hash functions, and of traditional public-key signature schemes. Definitions and several examples of these are found in [3] and [4].

Alice and Bob are denoted A and B throughout the text. Notation $A \to B : x$ means the transmission of x from A to B, and $\overline{x}$ is the corresponding data as received by B. In normal circumstances, both A and B want x to be equal to $\overline{x}$. In any case, x and $\overline{x}$ will denote some original data and its alleged counterpart after transmission by either part, A or B. When describing

*monteiro@dcc.unicamp.br – CEPESC and IC-UNICAMP – Partially supported by CNPq, Brasil.

†lucchesi@dcc.unicamp.br – IC-UNICAMP – Supported by grants from CNPq and FAPESP. Member of PRONEX 107/97 (MCT/FINEP).

‡rdahab@dcc.unicamp.br – IC-UNICAMP – Member of PRONEX 107/97 (MCT/FINEP).

a sequence of data transmissions, and after reception of $\overline{x}$, we may denote it again by x, implying that x has been correctly received.

We write $h(x)$ to denote the hash function in use. We denote by $[x]_{S_A}$ the public-key signature of A on x using A's private key S_A. Accordingly, $[\overline{[x]_{S_A}}]_{P_A} = x$ denotes the correct verification of signature $[x]_{S_A}$. Details of such schemes, such as redundancy addition, etc, will be omitted here.

2 Related Work

Our work is inspired on that of Gennaro and Rohatgi [1], and of Goldwasser, Micali and Rivest [2]. Both use *one-time signature schemes* [3], in which a public-key is used to sign at most one message. Such schemes are usually much faster than traditional signature schemes, thus preferred in applications where low complexity is a requirement, such as on-line algorithms, restricted memory devices, etc.

In one algorithm proposed in [2] A sends B blocks $b_1, b_2, \cdots, b_n$ "packaged" into *data packets* $p_1, p_2, \cdots, p_n$. Each packet p_i contains b_i plus some extra information, to be used in authentication and signature verification procedures. Packets p_i are:

$$p_i := \langle b_i, [P_A^i]_{S_A}, [b_i]_{S_A^i} \rangle, \quad 1 \leq i \leq n,$$

where S_A^i, P_A^i are A's ith private and public-key respectively. In this scheme, for each packet p_i, we need a conventional scheme to verify the signature $[P_A^i]_{S_A}$. This is costly when compared to the verification of the one-time signature $[b_i]_{S_A^i}$. Even though the generation of each $[P_A^i]_{S_A}$ could be done off-line, its verification cannot. Gennaro and Rohatgi's scheme improves this situation. Their packets p_i are:

$$\begin{cases} p_0 := \langle [P_A^0]_{S_A} \rangle, \\ p_i := \langle b_i, P_A^i, [h(b_i, P_A^i)]_{S_A^{i-1}} \rangle, & \text{if } 1 \leq i \leq n. \end{cases}$$

In this case, only one conventional signature is needed, in packet p_0

Our work differs from that of Gennaro and Rohatgi's in one fundamental aspect: ours is a demand-response protocol, in which Alice, the stream server, only sends data block b_i after receiving from Bob, the client, a signed acknowledgement of the correct reception of b_{i-1}. Thus, our protocol is forcedly synchronous, which may prevent its use in some real-time applications, in which the demand-response model is unacceptable. However, like in Gennaro and Rohatgi's work, we do provide non-repudiation ability to Bob regarding Alice, and the authentication procedure on Bob's side is very fast. Additionally, we provide non-repudiation ability to Alice with regard to Bob, a feature not present in Gennaro and Rohatgi's scheme. That broadens the scope of applications of our method, to situations in which Alice requires a proof of reception from Bob. Many instances of e-commerce exhibit that need. Another characteristic, common in most client-server applications and exploited by our protocol, is the workload asymmetry between Alice and Bob. Naturally, the bulk of the transmission lies with Alice. Accordingly, her cryptographic overhead is limited to one hash computation and one public-key signature verification per block. Bob's packets, however, which are quite short, are signed with a public-key algorithm; in addition, he has to perform two hash computations per block.

3 The Protocol

In our protocol, user A sends p_{i+1} only after receiving from B an *acknowledgement packet* q_i, which indicates the acceptance of p_i by B.

For presentation purposes, we will assume that the length n of the sequence $\mathcal{B}$ is known in advance by both A and B. This is not mandatory: the protocol may be extended indefinitely, may the need for sending further blocks arise. For this, it suffices to start another sequence of n blocks. In other words, we can send an arbitrarily long stream, whose length may not be known in advance, in finite bursts of n blocks.

The actual exchange of packets that defines the protocol is described in Section 3.4. The following sections introduce the terms used there.

3.1 Pre-Computation

Before transmission begins, A pre-computes a *hash sequence* $V = v_1, v_2, \cdots, v_{n+1}$, where v_{n+1} is a random number and $v_i := h(v_{i+1})$, for $1 \leq i \leq n$. It is Alice's responsibility to ensure that v_{n+1} is "fresh", i.e., that it has not been used in previous exchanges with any other users.

3.2 Computation During the Protocol

As the protocol progresses, A computes an *authentication sequence* $X = x_0, x_1, \cdots, x_n$ as follows:

$$x_i := \begin{cases} h(t), & i = 0, \\ h(b_i, v_{i+1}, x_{i-1}), & 1 \leq i \leq n, \end{cases} \tag{1}$$

where t is a timestamp, required as an assurance against malicious replay.

3.3 Description of the Packets

The sequence of data packets $P = (p_i)$ sent by A includes two extra packets, p_0 and p_{n+1}:

$$p_i := \begin{cases} \langle [v_1, t]_{S_A} \rangle, & i = 0, \\ \langle b_i, v_i, x_i \rangle, & 1 \leq i \leq n, \\ \langle v_{n+1} \rangle, & i = n+1, \end{cases} \tag{2}$$

where t is the timestamp in (1).

The acknowledgment stream $Q = (q_i)$, sent by B to A, has length $n+1$. Its contents are much simpler and shorter than those of P:

$$q_i := \begin{cases} \langle [p_0]_{S_B} \rangle, & i = 0, \\ \langle [x_i]_{S_B} \rangle, & 1 \leq i \leq n. \end{cases} \tag{3}$$

3.4 The Packet Exchange

The packets are *accepted* alternately by Bob and Alice, as the exchange progresses, as follows:

1. Upon reception of packet $\overline{p_0} = \overline{\langle [v_1, t]_{S_A} \rangle}$, Bob accepts it if the signature of A in $\overline{p_0}$ is correctly verified and the value of t is "fresh". Having accepted packet p_0, Bob sends packet q_0 to Alice.

2. Upon reception of packet $\overline{q_0} = \overline{\langle [p_0]_{S_B} \rangle}$, Alice accepts it if $[\overline{q_0}]_{P_B} = p_0$. Having accepted packet q_0, Alice sends packet p_1 to Bob.

3. Upon reception of packet $\overline{p_1} = \overline{\langle b_1, v_1, x_1 \rangle}$, Bob accepts it if $\overline{v_1}$ is equal to the first term v_1 of pair $[p_0]_{P_A}$. Having accepted packet p_1, Bob sends packet q_1 to Alice.

4. For $1 \leq i \leq n$, upon reception of packet $\overline{q_i} = \overline{\langle [x_i]_{S_B} \rangle}$, Alice accepts it if $[\overline{q_i}]_{P_B} = x_i$. Having accepted packet q_i, Alice sends packet p_{i+1} to Bob.

5. For $0 < i < n$, upon reception of packet $\overline{p_{i+1}} = \overline{\langle b_{i+1}, v_{i+1}, x_{i+1} \rangle}$, Bob accepts it if the following properties hold:

   (a) $h(\overline{v_{i+1}}) = v_i$, and
   (b) $h(b_i, \overline{v_{i+1}}, x_{i-1}) = x_i$.

   Having accepted packet p_{i+1}, Bob sends packet q_{i+1} to Alice.

6. Bob accepts packet $\overline{p_{n+1}} = \overline{\langle v_{n+1} \rangle}$ if the following properties hold:

   (a) $h(\overline{v_{n+1}}) = v_n$, and
   (b) $h(b_n, \overline{v_{n+1}}, x_{n-1}) = x_n$.

Figure 1 shows the exchange of packets during the execution of the protocol.

Step	**Action**
$0 \leq i \leq n$	$A \rightarrow B : p_i$;
	B checks whether $\overline{p_i}$ is accepted; if not, INTERRUPT;
	$B \rightarrow A : q_i$;
	A checks whether $\overline{q_i}$ is accepted; if not, INTERRUPT.
$n+1$	$A \rightarrow B : p_{n+1}$;
	B checks whether $\overline{p_{n+1}}$ is accepted; if not, INTERRUPT.

Figure 1: Steps of the protocol

4 Analysis of the Protocol

In this section we are concerned with the correctness of the protocol in two aspects: (i) externally induced forgery or error and (ii) attempts at forgery when Alice or Bob are dishonest.

The first case is discussed in section 4.1. We show that the protocol correctly detects transmission errors or any attempts at forgery of packets by third parties.

The second aspect concerns the non-repudiation properties of the protocol, and is discussed in section 4.2. We show (i) that Alice sent a particular sequence of packets to Bob; (ii) that Bob received the alleged packets; and (iii) that the sequence is fresh.

It is thus understood that any allegation by Bob will be considered valid unless Alice is able to prove the contents, reception and freshness of the packets.

4.1 Detection of Errors or Third Party Interferences

In this section we are concerned with the detection of transmission errors or malicious tampering of the packets by a third party.

We say that a packet r is *good* if it satisfies the following properties:

(i) packet r is not a malicious replay of a previous packet, in this instance of the protocol or in some previous instance,

(ii) packet r was sent by its legitimate originator, and

(iii) packet r has been received intact by the addressee.

Theorem 1 *For $0 \leq r \leq n$, if Bob accepts packet p_{r+1} then each of the $2r+2$ packets p_i, q_i $(0 \leq i \leq r)$ is good and the value v_{r+1} in packet p_{r+1} is authentic and fresh. Moreover, if $r = n$ packet p_{n+1} is also good.*

Proof: by induction on r.

Case 1 *r=0.*

Bob accepts packet $\overline{p_0} = \overline{\langle [v_1, t]_{S_A} \rangle}$ only if the signature of Alice in $\overline{p_0}$ is correctly verified and the value of t is fresh. Thus, packet p_0 has been received undisturbed and Alice is indeed its originator. Also, by checking the freshness of t, Bob knows that p_0 is not a replay of a packet previously sent by Alice.

In order to accept packet $\overline{q_0} = \overline{[[p_0]_{S_B}]_{P_B}}$, Alice checks the validity of the signature of Bob in $\overline{q_0}$ and also whether equality $[\overline{q_0}]_{P_B} = p_0$ holds. This assures Alice that q_0 was sent by Bob and also that q_0 was received correctly. Moreover, it also assures Alice that Bob accepted packet p_0. Thus, p_0 is fresh, whence so too is q_0. We conclude that p_0 and q_0 are both good.

In order to accept packet $\overline{p_1} = \overline{\langle b_1, v_1, x_1 \rangle}$, Bob checks whether $\overline{v_1}$ is equal to v_1, which is authentic and fresh. Therefore, the value $\overline{v_1}$ in packet $\overline{p_1}$ is authentic and fresh.

Case 2 $r > 0$.

By induction hypothesis, packets $p_0, q_0, \cdots, p_{r-1}, q_{r-1}$ are good. Also, the value $\overline{v_r}$ in packet $\overline{p_r}$ is fresh and authentic.

In order to accept packet $\overline{q_r} = \overline{[x_r]_{S_B}}$, Alice checks the validity of the signature of Bob in $\overline{q_r}$ and also whether equality $[\overline{q_r}]_{P_B} = x_r$ holds. This assures Alice that q_r was sent by Bob and also that q_r was received correctly. Moreover, it also assures Alice that the value $\overline{x_r}$ in packet $\overline{p_r}$ received by Bob is equal to x_r, which is fresh and authentic.

In order to accept packet $\overline{p_{r+1}}$, Bob checks whether the value $\overline{v_{r+1}}$ contained in that packet satisfies the equality $h(\overline{v_{r+1}}) = v_r$. Only Alice knows the value v_{r+1}, therefore that value is authentic and fresh, because v_r is authentic and fresh. Therefore, Bob knows that Alice accepted his packet q_r, which tells Bob that the value $\overline{x_r}$ in packet $\overline{p_r}$ is authentic and fresh. In order to accept packet $\overline{p_{r+1}}$, Bob also checks whether $h(\overline{b_r}, v_{r+1}, x_{r-1})$ is equal to x_r, thereby determining whether $\overline{b_r}$ is fresh and authentic. We conclude that packet p_r is good and $\overline{v_{r+1}}$ is authentic and fresh, if Bob accept packet p_{r+1}.

In addition, if $r = n$ then v_{r+1} is the only information in packet p_{r+1}, therefore packet p_{r+1} is good if $r = n$ and Bob accepts packet p_{r+1}. □

An immediate consequence of this theorem is that the protocol correctly detects transmission errors or any attempts at forgery of packets by third parties.

4.2 Non-Repudiation Properties of the Protocol

There is a variety of false claims that Alice or Bob could make after the completion of the protocol. However, it is understood that the burden of the proof is always Alice's, who is guilty until proven innocent.

Theorem 2 *If Bob accepts packet p_{r+1} $(0 \leq r \leq n)$ then Alice can prove that sequence $b_1, \cdots, b_r$ was sent to Bob, that Bob received it intact, and that it is not a replay.*

Proof: Alice has packet q_0, which is packet p_0 signed by Bob with his private key. This implies that Bob accepted packet p_0. Thus, Bob accepted both t and v_1. Alice can then show that sequence $v_1, \cdots, v_{r+1}$ satisfies the recurrence $v_i = h(v_{i+1})$, for $1 \leq i \leq r$; therefore that sequence was authentic and fresh.

Let $m := \min\{r+1, n\}$. Alice also has packets $q_i = [x_i]_{S_B}$ $(1 \leq i \leq m)$, therefore she can prove that Bob accepted sequence $x_0, \cdots, x_m$. Thus, each x_i was authentic and fresh. Therefore, a third party may check that $h(b_i, v_{i+1}, x_{i-1}) = x_i$ $(1 \leq i \leq r)$, thereby validating Alice's assertions. It is computationally infeasible for Alice to create false sequences that satisfy all the recurrences above. □

An immediate consequence of this result is that (i) if the protocol is not completed then Alice is able to deny any false allegations by Bob concerning that instance of the protocol, except possibly for the contents of the last packet that she sent to Bob, and (ii) if the protocol is completed then Alice is able to deny any false allegations by Bob concerning that instance of the protocol.

5 Conclusions

Our protocol provides a fast means by which Alice can send Bob a stream $\mathcal{B}$, with assurance of non-repudiation of reception of $\mathcal{B}$ or any prefix of it by Bob. Bob, on the other hand, can prove to any third party the origin of $\mathcal{B}$ or any prefix of it. A possible drawback is the synchronous character of the protocol, which prevents its use in broadcast-type applications, but it opens the possibility of its utilization in several e-commerce applications.

We make use of the asymmetry between Alice and Bob, both in workload and responsibility. The on-line overhead on Alice's side is one hash computation and one public-key signature verification per data block of $\mathcal{B}$. That of Bob's is one public-key signature generation and two hash computations per data block. The public-key signature overhead can be reduced, with the use of one-time schemes. Alice's responsibility as the sender and initiator of the exchange imposes a heavier burden of proof on her: any allegations by Bob are valid until countered by Alice.

References

[1] R. Gennaro and P. Rohatgi, *How to sign digital streams*, Lecture Notes on Computer Science, vol. 1294, p. 180, Springer-Verlag, New York, 1997.

[2] S. Goldwasser, S. Micali, and R. Rivest, *On-line/off-line digital signatures*, J. of Cryptology **9** (1996), no. 1, 35–67.

[3] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of applied cryptography*, CRC Press, 1997.

[4] D. R. Stinson, *Cryptography - theory and practice*, CRC Press, 1995.