Authentication Based on Logical Time

Anand Natrajan

Technical Report No. CS-97-23
May 10, 1996

Contact: anand@virginia.edu

Web: ftp://ftp.cs.virginia.edu/pub/techreports/CS-97-23.ps.Z

Authentication Based on Logical Time

Anand Natrajan
anand@virginia.edu

1 Introduction

Timestamp-based authentication protocols rely on real-time synchronisation between the principals involved. This synchronisation, which involves all concerned principals agreeing on the notion of their time, is often difficult to achieve, and hence nonce-based protocols were developed. However, the principals in a timestamp-based authentication protocol can be made to synchronise to logical time. Efficient logical time systems can guarantee that processors (or principals in this case) agree on a logical time. Using this property, new protocols can be developed for various communication paradigms. We show one such protocol for interactive communication between two parties using a trusted authentication server.

2 Authentication Issues

In this section we present a brief background of traditional authentication issues. In particular, we wish to highlight the evolutionary progress of protocols. We follow standard notation for displaying messages:

- $A \rightarrow B \mid \ldots$ denotes a message from principal A to principal B
- $\{X\}_K$ denotes a message X encrypted under the key K
- N_{AB} denotes a nonce generated by principal A and given to principal B
- K_{AB} denotes a key shared by principals A and B

In addition, we make the following assumptions for the sake of discussion:

- A and B are the principals in the scenario wishing to communicate, with A initiating.
- S is the trusted authentication server, which is the authoritative source of information regarding the keys that are utilised by the principals on the network.
- I is the intruder capable of reading unencrypted messages, inserting, removing or replaying any and all messages, but incapable of decrypting messages within a reasonable amount of time.

2.1 Needham-Schroeder protocols

Needham and Schroeder present security protocols addressing three types of network communications: interactive communication, one-way communication and signed communication that can be authenticated by a third party [Need78]. Figure 1 shows the Needham-Schroeder protocol for interactive communication between A and B using secret keys*. The sequence of messages is as below:

$A \rightarrow S \mid N_{AS}, A, B$
$S \rightarrow A \mid \{N_{AS}, B, K_{AB}, \{A, K_{AB}\}_{K_{BS}}\}_{K_{AS}}$
$A \rightarrow B \mid \{A, K_{AB}\}_{K_{BS}}$
$B \rightarrow A \mid \{N_{BA}\}_{K_{AB}}$
$A \rightarrow B \mid \{f(N_{BA})\}_{K_{AB}}$

Similar figures can be drawn for the other protocols proposed by the authors. However, we will concentrate on this protocol for the rest of this paper, unless otherwise mentioned.

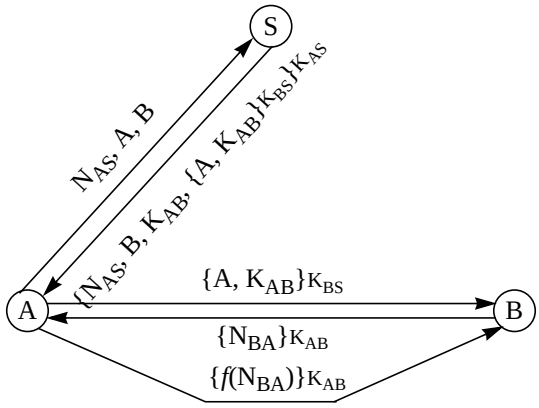


FIGURE 1: Needham-Schroeder protocol

* The order of the fields in the messages may have been changed for the sake of continuity in the discussion. It is obvious that these changes in order do not affect the protocol.

Needham and Schroeder avoid using timestamps in their protocol because it presupposes a network-wide reliable source of time. However, Denning and Sacco point out that the Needham-Schroeder protocols are not secure when the session keys are compromised [Denn81]. They propose a solution with fewer messages based on timestamps that would prevent replays of messages if a key is compromised. T is the timestamp of the corresponding message in their protocol (Figure 2). The sequence of messages follows:

$A \rightarrow S \mid A, B$

$S \rightarrow A \mid \{T, B, K_{AB}, \{T, A, K_{AB}\}_{K_{BS}}\}_{K_{AS}}$

$A \rightarrow B \mid \{T, A, K_{AB}\}_{K_{BS}}$

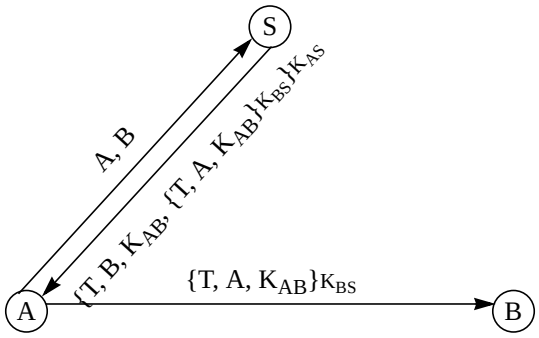


FIGURE 2: Denning-Sacco modification

Principals receiving messages are expected to check the timestamp of the message with their own times, factor in normal discrepancies between their clocks and the server's clock (Δt_1), and account for expected network delay times (Δt_2). The authors suggest that A and B can verify that their messages are not replays by checking that $|Clock - T| < \Delta t_1 + \Delta t_2$[†]. These measures are estimates and cannot be expected to always hold in a real system. Hence, if a message arrives slightly later than expected, B cannot distinguish between an intruder attack and an abnormal delay.

Accordingly, Needham and Schroeder suggest a modification to their original protocol, shown in Figure 3, that eliminates the timestamp [Need87]. The steps of the protocol follow:

$A \rightarrow B \mid A$

$B \rightarrow A \mid \{N_{BA}, A\}_{K_{BS}}$

$A \rightarrow S \mid N_{AS}, A, B, \{N_{BA}, A\}_{K_{BS}}$

$S \rightarrow A \mid \{N_{AS}, B, K_{AB}, \{N_{BA}, A, K_{AB}\}_{K_{BS}}\}_{K_{AS}}$

$A \rightarrow B \mid \{N_{BA}, A, K_{AB}\}_{K_{BS}}$

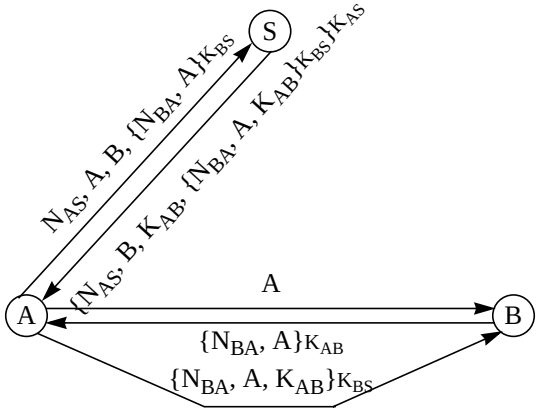


FIGURE 3: Modified Needham-Schroeder protocol

It is noteworthy that the above protocols require double encryption. Otway and Rees remove that requirement, thus making their protocol efficient as well as symmetric [Otway87]. The Otway-Rees protocol, shown in Figure 4, also reduces the number of messages to four. Assuming C to be a conversation identifier between A and B, the steps in the protocol are:

$A \rightarrow B \mid C, A, B, \{N_{AS}, C, A, B\}_{K_{AS}}$

$B \rightarrow S \mid C, A, B, \{N_{AS}, C, A, B\}_{K_{AS}}, \{N_{BS}, C, A, B\}_{K_{BS}}$

$S \rightarrow B \mid C, \{N_{AS}, K_{AB}\}_{K_{AS}}, \{N_{BS}, K_{AB}\}_{K_{BS}}$

$B \rightarrow A \mid C, \{N_{AS}, K_{AB}\}_{K_{AS}}$

Notice that this protocol makes extensive use of the keys shared between the principals and S. These keys, as opposed to the session key K_{AB}, are expected to be long-standing. It is thus slightly inadvisable to encode too many messages with them as that gives an intruder more data to work on while attempting to crack these keys.

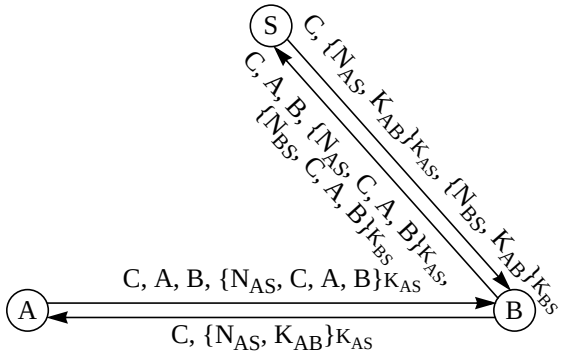


FIGURE 4: Otway-Rees protocol

[†] Incidentally, this equation itself is flawed in the case of B. Notice that the third protocol message originates from S, but reaches B *via* A. Also, A has to extract this third message from the second message, a decryption operation that requires some non-zero time. Therefore, for the third message, B's freshness check would have to be $|Clock - T| < \Delta t_1 + 2\Delta t_2 + t_3$, where t_3 represents B's estimate of A's processing speed!

3 Logical Time Systems

We now digress briefly from authentication protocols to present a background on logical time systems. Logical time was first proposed by Lamport in [Lamp78]. Lamport suggested the use of the "happened-before" relationship, which forms a partial ordering over events in a distributed system. Assume a system composed of a collection of processes, each of which is a sequence of events. The "happens-before" relationship, denoted by "$\rightarrow$", is defined as:

- If a and b are events on the same process, and a occurs before b, then $a \rightarrow b$
- If a is the send of a message and b is the corresponding receive, then $a \rightarrow b$
- The transitive closure over the above two conditions

This partial order can be extended into a total order by defining an arbitrary total order for the processes that breaks ties between events for which the "happens-before" relationship is undecidable. Virtual time, proposed by Jefferson in [Jeff85], is another logical time system. In this, every event is labelled with a clock value from a totally ordered virtual time scale in accordance with Lamport's clock conditions ([Lamp78]). Virtual time is a global temporal coordinate system imposed on a distributed computation. Jefferson visualised virtual time as a one-dimensional quantity, but later researchers have proposed vector ([Matt89]) and matrix ([Sarin87]) times. A crucial difference between Lamportian and virtual time systems is in the assignment of times to events. In the former, events are assigned times as they occur, whereas in the latter, the times assigned to events are pre-determined and fixed.

3.1 Isotach Systems

Isotach time is an extension of Lamport's logical time. An isotach network uses isotach time to reduce overhead arising from the need for synchronisation among multiple processors in a parallel system ([Reyn89] [Will91]). With isotach time, a processor can control the time at which the messages that it sends are executed by the receiving processor. Isotach logical times are lexicographically ordered n-tuples of integers, commonly of the form (*pulse*, *pid*, *rank*), where *pulse* denotes the progression in the time perceived by the entire network, *pid* is the identifier of the process that issued the message timestamped with isotach time, and *rank* is the issue rank of the message, i.e., *rank* = *r* if the message is the r^{th} message issued by the pid^{th} process. Isotach networks maintain the isotach invariant: a message is received exactly *d* pulses after it is sent, where *d* is the logical distance the message travels. Since a processing element (PE) can control the logical time of receipt of any message it sends in an isotach system, the isotach network gives the power to enforce properties like atomicity and sequential consistency.

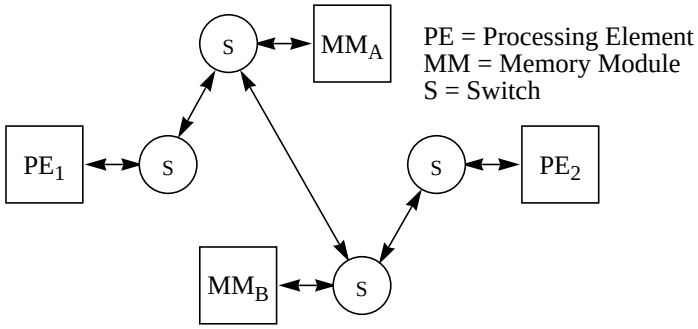
3.1.1 Sequential Consistency

In a sequentially consistent execution, the overall order of execution of operations is consistent with the order of execution implied by each individual process' sequential program [Lamp79]. In conventional systems, sequential consistency could be enforced by disallowing pipelining. Hence, a process would have to wait for information telling it of the execution of its last outstanding operation before it can issue its next operation. An isotach network imposes no such restrictions on pipelining. To enforce sequential consistency in isotach systems, a processor timestamps each *send* operation to be received in a pulse greater than or equal to the pulse in which the preceding operation was sent.

3.1.2 Atomicity

Conventional systems enforce atomicity with locks. However, the penalties associated with locks are: lock maintenance overhead, overly-restrictive access to shared objects, and the possibility of deadlock and livelock. Isotach systems use atomic actions to enforce atomicity. An atomic action issued by a process is a group of operations appearing to be executed indivisibly (without interleaving with other operations). Atomic actions can be **flat** or **structured**. Flat atomic actions have no data dependences among shared variables, whereas structured atomic actions have data dependences among the shared variables. A detailed discussion about atomic actions is inessential for our purposes. Interested readers may refer to [Will93] for a more complete discussion. However, in the interests of showing how processors can control the execute times of messages at their destination, we present an example using flat atomic actions.

A processor in an isotach system executes flat atomic actions by sending all operations in each atomic action to be received at the destinations in the same pulse. Consider the network in Figure 5. *A* and *B* are shared variables in memory modules MM_A and MM_B respectively. PE_1 and PE_2 are processing elements. Switches interconnect the elements. PE_1 atomically reads *A* and *B*, and PE_2 atomically writes *A* and *B*. PE_1 and PE_2 could execute their atomic actions asynchronously as follows: PE_1 sends the *read* on *A* one time pulse after it sends the *read* on *B*, so that both the *read*s reach their destinations in the same pulse. PE_2 sends its *write* on *B* one pulse after it sends the *write* on *A*, again to ensure that both the *write*s get to their destinations in the same pulse. By virtue of the isotach invariant, both operations in each atomic action are received in the same pulse. If all four operations happen to reach their destinations in the same time pulse, the executions will still be atomic because the operations in the same pulse will be executed in order of *pid* of the sender.



Operation	Sender	Send Time	Distance	Receive Time
read A	PE_1	(i+1, 1, …)	2	(i+3, 1, …)
read B	PE_1	(i, 1, …)	3	(i+3, 1, …)
write A	PE_2	(j, 2, …)	3	(j+3, 2, …)
write B	PE_2	(j+1, 2, …)	2	(j+3, 2, …)

FIGURE 5: Flat Atomic Actions on an Isotach Network

3.2 Other Logical Time Systems

In the interest of brevity we defer discussion on the suitability of other logical time systems to our protocol.

4 Authentication based on Logical Time

Previously-considered authentication protocols, with the exception of the Denning-Sacco protocol, were nonce-based because it was difficult to construct a system of processes that agreed on time. The Denning-Sacco protocol attempted to address this by accounting for possible skews between the clocks of various principals and message transmission delays. However, the problem of globally synchronised real-time clocks is hard. On the other hand, the problem of loosely-synchronised logical-time clocks has been solved by isotach and other systems. Therefore, a timestamp-based protocol employing logical time holds more promise than a similar protocol employing real-time. There are two aspects to the marriage between logical time systems and authentication. One aspect is proposing a new authentication protocol by exploiting the guarantees a logical time system offers. The other aspect is demonstrating the ease with which authentication can be embedded in a logical time system.

4.1 Protocol

Our new protocol is succinctly summed up by Figure 6. We list the protocol steps below:

$A \rightarrow S \mid t, A, B$

$S \rightarrow A \mid \{t+1, A, B, K_{AB}\}K_{AS}$

$S \rightarrow B \mid \{t+1, A, B, K_{AB}\}K_{BS}$

In words, at logical time *t*, A expresses to S an intent to communicate with B. (In the context of an isotach-based system, A ensures that S *executes* this message at time *t*.) S responds at time *t+1*[‡] by sending out a valid session key for A and B. The key is sent to A encoded under K_{AS}, and to B encoded under K_{BS}. Since these keys are shared only between the involved principals, an intruder I cannot draw any meaning from the encrypted messages.

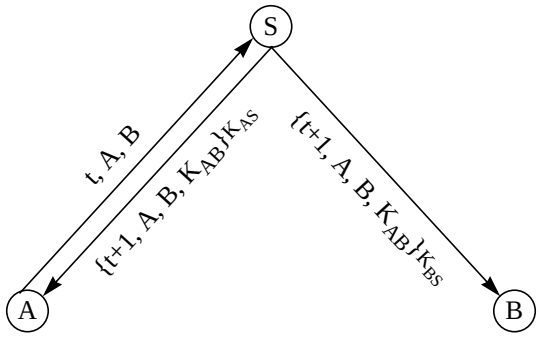


FIGURE 6: Logical Time-Based protocol

4.2 Resistance to Intruder Attacks

In isotach-based logical time systems, the logical time associated with a message is inserted by the system itself.[+] However, we will assume the intruder capable of bypassing this feature and forging times. The intruder I could modify unencrypted messages, and remove or insert any and all messages. Removing the first message causes a denial of service, an attack that is not addressed well by most protocols, including ours. I could modify the timestamp of the first message to t', but this would cause A to become suspicious when it sees K_{AB} arrive at $t'+1$. Of course, if $t'<t$, S would get a message from its past, causing it to become suspicious because in an isotach system a process can get messages only in its present or immediate future. I could change any of the principals in the first message, resulting in denial of service. With the messages timestamped $t+1$, the worst I could do is remove one or both messages. Clearly, a replay attack will not work because both A and B can detect the discrepancy this would cause in the progression of logical time. Also, I could not forge any of these messages because it does not possess K_{AS} or K_{BS}. If I removes the message sent to B, then when A begins to communicate with B, presumably at $t+2$, B would not understand that message, not being in possession of K_{AB}. Since the only reason such a scenario could have occurred is due to an intruder attack, B could raise a warning. Lastly, if I removes the message sent to A, A could progress its logical time and assume S's non-response to be a denial of service caused by an intruder attack.

4.3 Formal Analysis

We now formally analyse our protocol using BAN Logic. BAN is a simple doxastic logic that describes the beliefs of trustworthy parties involved in communication and the evolution of these beliefs as a result of communication. An involved description of BAN is in [Burr90]. The idealised protocol is as below:

$A \rightarrow S \mid t, A, B$
$S \rightarrow A \mid \{t+1, A \overset{K_{AB}}{\leftrightarrow} B\}K_{AS}$
$S \rightarrow B \mid \{t+1, A \overset{K_{AB}}{\leftrightarrow} B\}K_{BS}$

To analyse our protocol, we first give the following hypotheses:

A **believes** $A \overset{K_{AS}}{\leftrightarrow} S$,	B **believes** $B \overset{K_{BS}}{\leftrightarrow} S$,
S **believes** $A \overset{K_{AS}}{\leftrightarrow} S$,	S **believes** $B \overset{K_{BS}}{\leftrightarrow} S$,
A **believes** (S **controls** $A \overset{K}{\leftrightarrow} B$),	B **believes** (S **controls** $A \overset{K}{\leftrightarrow} B$),
A **believes fresh**(t+1),	B **believes fresh**(t+1),
S **believes** $A \overset{K_{AB}}{\leftrightarrow} B$	

The steps of the proof are:

A receives the message directed to it. By annotation,	A **sees** $\{t+1, A \overset{K_{AB}}{\leftrightarrow} B\}K_{AS}$
Since we have the hypothesis	A **believes** $A \overset{K_{AS}}{\leftrightarrow} S$,
the message-meaning rule for shared keys applies and yields	A **believes** S **said** $(t+1, A \overset{K_{AB}}{\leftrightarrow} B)$
We also have the hypothesis	A **believes fresh**(t+1)
The nonce-verification rule applies and yields	A **believes** S **believes** $(t+1, A \overset{K_{AB}}{\leftrightarrow} B)$
We break the conjunction to yield	A **believes** S **believes** $A \overset{K_{AB}}{\leftrightarrow} B$
Then, we instantiate K to K_{AB} in the hypothesis	A **believes** S **controls** $A \overset{K}{\leftrightarrow} B$
to get the more concrete	A **believes** S **controls** $A \overset{K_{AB}}{\leftrightarrow} B$
Finally, the jurisdiction rule applies and yields	A **believes** $A \overset{K_{AB}}{\leftrightarrow} B$ . . . . . (1)
Similar reasoning applied to B yields	B **believes** $A \overset{K_{AB}}{\leftrightarrow} B$ . . . . . (2)

An isotach-based server is expected to ensure both A and B receive K_{AB} in the same logical time. Thus, when A and B receive their messages, it is perfectly reasonable for them to believe that the other party would also have received a similar message from S. Note that neither A nor B *knows* whether the other has received the key or not, but both of them *believe* that the other does. Thus, one could proceed in the proof as follows:

B receives the message directed to it. By knowledge of server,	B **believes** A **sees** $\{t+1, A \overset{K_{AB}}{\leftrightarrow} B\}K_{AS}$
Applying the same transformations as in the first 8 steps,	B **believes** A **believes** $A \overset{K_{AB}}{\leftrightarrow} B$ . . . . (3)
Performing the same reasoning on A's behalf,	A **believes** B **believes** $A \overset{K_{AB}}{\leftrightarrow} B$ . . . . (4)

---

‡ In general, the time at which S responds could be $t+p$, where p is known beforehand. p could be a measure of the logical distance between the principals and the time S is likely to take before producing K_{AB}.

+ The isotach discussion of this requires a lengthy digression into the Switch Interface Unit (SIU), which we will postpone for the moment. It suffices to say that the application may not know about the logical time at which its own processor is. Since this does not detract from our proposal, we blur the distinction between the process (principal) and its processor-switch pair.

As noted in [Burr90], the four guarantees, labelled 1 through 4, make our protocol stronger than most others. The caveats to BAN Logic are well-known ([Ness90] [Boyd93]), and it is recommended that other logics also be used to test the protocol.

4.4 Benefits of Applying Logical Time to Authentication

As we have seen in earlier sections, assuming a logical time system underlying the actions of principals enables us to formulate a new protocol. The advantages of this protocol over some existing protocols are:

- **Fewer messages**: Our protocol employs only three messages, one of which is unencrypted. Fewer messages implies easier analysis and also increased efficiency due to fewer encryption opportunities. In addition, our messages are very short.
- **No double encryption**: The two encrypted messages in our protocol are encrypted just once. This is feasible because there is no exchange between the principals desiring to communicate until actual data transmission.
- **Low usage of shared keys**: Our protocol utilises the keys shared between the server and principals just once. Unlike session keys that are expected to be short-lived, shared keys are expected to be long-lasting. However, using them often gives a potential intruder more data to work on in attempting to crack them. Therefore, sparse usage of shared keys is a significant advantage.
- **Mutating keys**: Principals A and B could change their session key K_{AB} at pre-determined times. This could be done by the server S handing out a list of keys with associated logical times instead of just the one key during the authentication sequence. A and B (and S) will be loosely synchronised as long as they stay on the logical time network. Therefore, assuming S hands out the same list of key-time pairs to both A and B, all concerned parties can be sure that the session keys will mutate predictably at the correct times.
- **Symmetric**: Our protocol is symmetric in its treatment of the principals.
- **Optional continuance**: A, B and S must adhere to the logical time system in order that A and B achieve mutual authentication. However, after receiving the session key and sending the first message at time *t+2*, both A and B need not adhere to the logical time system. In effect, A and B could "plug in" and "plug out" of the logical time system whenever they wish to achieve mutual authentication. Thus, the overhead of logical time remains only for authentication purposes.
- **Scalable communication**: Insufficient literature exists on how existing protocols scale in the face of communication topologies that are different from the standard two-party case shown here. A communication topology that involves three parties sharing a common key, thus indulging in some sort of conferencing, is not far-fetched. Intuitively, our protocol scales better than existing protocols in terms of the number of messages that need to be sent in order for the key to be distributed and mutual authentication achieved.

4.5 Benefits of Applying Authentication to Logical Time Systems

We are aware of applications for logical time systems that could utilise authentication among principals. An example system would be a transaction-based system involving financial institutions, wherein each institution must authenticate itself to the other before initiating a transaction. Such applications also require atomicity and sequential consistency — features provided by isotach-like systems. Pending implementation of isotach systems, we can only speculate on the extent of applicability authentication will have for systems desirous of logical time.

5 Conclusions

We proposed a new authentication protocol based on logical time. Loosely-synchronised processors are possible under isotach systems. Using this property, principals in loose synchrony with an authentication server can achieve mutual authentication using fewer messages and no double encryption. We analysed our protocol informally and formally using BAN. We believe isotach-based applications can easily exploit authentication.

A more complete analysis of authentication protocols using logical time seems to be the direction of further research. In particular, we expect to formulate new protocols for public key systems as also for the other communication paradigms outlined in [Need78]. Another avenue of research would be in studying the suitability of other logical time systems such as Isis and Horus towards authentication.

6 References

[Boyd93] Boyd, C., Mao, W., *On a Limitation of BAN Logic*, Proceedings of ESORICS 92: 2nd European

Symposium on Research in Computer Security, Toulouse, France, November, 1992.

[Burr90] Burrows, M., Abadi, M., Needham, R. M., *A Logic of Authentication*, ACM Transactions on Computer Systems, Volume 8, Number 1, February 1990.

[Denn81] Denning, D. E., Sacco, G. M., *Timestamps in Key Distribution Protocols*, Communications of the ACM, Volume 24, Number 8, August 1981.

[Jeff85] Jefferson, D. R., *Virtual Time*, ACM Transactions on Programming Languages and Systems, Volume 7, Number 3, July 1985.

[Lamp78] Lamport, L., *Time, Clocks and the Ordering of Events in a Distributed System*, Communications of the ACM, Volume 21, Number 7, July 1978.

[Lamp79] Lamport L., *How to Make a Multiprocessor Computer That Correctly Executes Multiprocessor Programs*, IEEE Transactions on Computers, Volume 28, 1979.

[Matt89] Mattern, F., *Virtual Time and Global States of Distributed Systems*, Parallel and Distributed Algorithms, Elsevier Science Publishers B.V. (North-Holland), 1989.

[Need78] Needham, R. M., Schroeder, M. D., *Using Encryption for Authentication in Large Networks of Computers*, Communications of the ACM, Volume 21, Number 12, December 1978.

[Need87] Needham, R. M., Schroeder, M. D., *Authentication Revisited*, Operating Systems Review, January 1987.

[Ness90] Nessett, D. M., *A Critique of the Burrows, Abadi and Needham Logic*, Operating Systems Review, April 1990.

[Otway87] Otway, D., Rees, O., *Efficient and Timely Mutual Authentication*, Operating Systems Review, January 1987.

[Reyn89] Reynolds, P. F. Jr., Williams, C. C., Wagner, R. R., *Parallel Operations*, University of Virginia Computer Science Technical Report 89-16, December 1989.

[Sarin87] Sarin, S. K., Lynch, L., *Discarding obsolete information in a replicated database system*, IEEE Transactions on Software Engineering, Volume 13, Number 1, January 1987.

[Will91] Williams, C. C., Reynolds, P. F. Jr., *Combining Atomic Actions in a Recombining Network*, University of Virginia Computer Science Technical Report 91-33, November 1991.

[Will93] Williams, C. C., *Concurrency Control in Asynchronous Computations*, Ph.D. Thesis, University of Virginia, 1993.