The Scope of a Logic of Authentication

Michael Burrows* Martín Abadi* Roger Needham†

May 13, 1994

*Digital Equipment Corporation, Systems Research Center
†Computer Laboratory, University of Cambridge

SRC Research Report 39 was originally published on February 28, 1989, and revised on February 22, 1990. This is an appendix to the revised version. The main body of the revised version is available separately.

1 A Logic of Authentication

Authentication protocols are both essential to security in distributed systems and notoriously obscure. A typical authentication protocol involves only two or three parties (called principals) and a few messages between them (see [1, 10, 13, 14, 15, 16, 19] for example). The intended outcome is that the parties obtain a guarantee that they are dealing with each other. In addition, the parties sometimes acquire a new shared secret, such as a session key. The exchanges are brief, but many of the authentication protocols found in the literature contain serious security flaws ([1, 10, 14, 19] for example).

The logic of authentication described in the first part of this report has helped us in analyzing certain aspects of authentication protocols. Our logic consists of a simple set of inference rules, in a notation designed specifically for the study of authentication protocols.[1] The goal of the logic is to describe the beliefs of trustworthy parties involved in authentication, and the evolution of these beliefs while the principals communicate. This approach has proved useful. We have met with some success in explaining protocols and in uncovering subtle assumptions and dangerous bugs. On occasion, we have also found redundancies, such as unnecessary message parts and unnecessary encryption. More recently, others have used the logic, sometimes with a similar result [4, 17].

The logic does not attempt to answer every important question about every conceivable authentication protocol. In this note, we wish to clarify what the logic captures and what it does not capture, and where there is room for other formal or informal techniques. We introduce and discuss the logic through an example that relies on public-key cryptography [18].

In our example, a principal A obtains the public key of another principal B from a server S. First, A requests a certificate containing B's public key, from a trusted server S. In an informal notation, this message might be written:

$$A \rightarrow S: \ A, B$$

The server replies with the desired certificate, which includes B's name, a key K_b, a timestamp T, and which is encrypted under the server's secret key K_s^{-1}:

$$S \rightarrow A: \ S, \{K_b, B, T\}_{K_s^{-1}}$$

[1] We use slightly different notation in other works on the logic. The notation here is more verbose, but more readable.

The analysis of a protocol consists of three phases:

- reformulating the protocol in our notation;
- spelling out the implicit assumptions;
- applying the inference rules of the logic to the assumptions and to the messages.

We will go through these phases in our example.

First, we transform the protocol into an idealized form, where each message becomes an expression in the logic. Intuitively, the expression reflects the logical meaning of the message. For example, S says that K_b is B's public key (written $\stackrel{K_b}{\mapsto} B$) in the second message, and hence the second message becomes:

$$S \rightarrow A: \; \{\stackrel{K_b}{\mapsto} B, T\}_{K_s^{-1}}$$

We have removed a cleartext field from the second message. Similarly, the first message, from A to S, can be omitted altogether from the idealized form of the exchange. Unencrypted parts are simply "hints" designed to help with the timely progress of the protocol, and would not contribute to our proofs. Therefore, the idealized protocol proceeds as if S sends B's public key to A spontaneously.

The transformation from an informal description of a protocol to a description in our notation is carried out by hand, but we have found that it is usually straightforward. Nevertheless, this transformation is not always possible: some protocols rely on techniques that cannot be represented adequately in the logic. However, the class of protocols treated is large enough to be useful.

We have not yet been able to make rigorous the correspondence between formulas in idealized protocols and bits transmitted. Hence, our derivations provide no guarantees about actual implementations of the protocols. It might be desirable and possible (at least in principle) to prove properties of authentication protocols expressed in conventional programming languages.

In the second phase of the analysis, we write assumptions. The assumptions that underly this message exchange are fairly obvious.

- The server's public key must be known to A, otherwise the message is of no use to A. We say that A believes that K_s is S's public key, and write it A **believes** $\stackrel{K_s}{\mapsto} S$.

- A must believe that the timestamp is a recent one if it is to accept S's message as timely (and not a replay by an intruder). In this case we say that A believes that T is fresh, and write it A **believes fresh**(T). What it is about T that causes A to believe it fresh is outside the province of the logic.

  We divide time into two epochs, the present (the current run) and the past; a fresh expression is one that was never uttered in the past. The distinction of present and past suffices for our purposes.

- S must believe that K_b really is B's public key, for S's message to be honest and not a deliberate lie. Although we allow for the possibility of attackers, we take all the principals mentioned in the protocol to be honest.

- Finally, A must trust S to provide B's key. More precisely, if A believes that S believes that K_b is B's key, then A believes that K_b is B's key. We say that A believes that S has jurisdiction over $\stackrel{K_b}{\mapsto} B$, and write A **believes** S **controls** $\stackrel{K_b}{\mapsto} B$.

Once the protocol is idealized and the assumptions are made explicit, we check that all the principals communicate honestly in the course of the exchange. This means that the sender of each message believes in the truth of its contents, either on the basis of initial assumptions or as a result of valid inference from previously received messages. In the example, the honesty of S follows from the assumption that S **believes** $\stackrel{K_b}{\mapsto} B$.

We also use the assumptions and the messages to reason about the final state achieved by the protocol. We consider only stable formulas, those that once true remain true for the duration of a protocol run. Therefore, all the assumptions about the initial state still hold as post-conditions to the messages. (The manipulation of pre-conditions and post-conditions is inspired by Hoare logic [7].) In addition, it also holds as a post-condition that A sees S's message, which we write A **sees** $\{\stackrel{K_b}{\mapsto} B, T\}_{K_s^{-1}}$.

Hence we proceed to apply the rules of inference of the logic to the assumptions and to the post-condition A **sees** $\{\stackrel{K_b}{\mapsto} B, T\}_{K_s^{-1}}$. These rules simply account for how A ascertains that K_b is B's public key.

The first relevant rule is

$$\frac{P \textbf{ believes } \stackrel{K}{\mapsto} Q, \quad P \textbf{ sees } \{X\}_{K^{-1}}}{P \textbf{ believes } Q \textbf{ said } X}$$

which expresses that if P believes that K is Q's public key and it receives a message $\{X\}_{K^{-1}}$ under the inverse of K, then P believes that Q once said the contents of the message. We derive A **believes** S **said** $(\stackrel{K_b}{\mapsto} B, T)$.

Another rule enables us to deduce A **believes fresh**$(\stackrel{K_b}{\mapsto} B, T)$ from the assumption A **believes fresh**(T):

$$\frac{P \textbf{ believes } \textbf{fresh}(Y)}{P \textbf{ believes } \textbf{fresh}(X, Y)}$$

A third rule,

$$\frac{P \textbf{ believes } \textbf{fresh}(X), \quad P \textbf{ believes } Q \textbf{ said } X}{P \textbf{ believes } Q \textbf{ believes } X}$$

then yields that A **believes** S **believes** $(\stackrel{K_b}{\mapsto} B, T)$. This rule expresses the check that a message belongs to the current run of the protocol. If P believes that X is fresh and Q has once said X, then P believes that Q has said X during the current run, and hence that Q believes X at present.

Immediately, we can derive A **believes** S **believes** $\stackrel{K_b}{\mapsto} B$. Notice a formal oddity here: we can also derive A **believes** S **believes** T, where a timestamp is treated as a statement—a "believable" expression. This seems less odd if we view T as a statement such as "it is 5PM," rather than as the corresponding term "5PM." A reformulation of the syntax of the logic could avoid the oddity altogether.

We complete the proof with the rule:

$$\frac{P \textbf{ believes } Q \textbf{ controls } X, \quad P \textbf{ believes } Q \textbf{ believes } X}{P \textbf{ believes } X}$$

which expresses that if P believes that Q believes X and P trusts Q on X then P believes X as well. Immediately, we obtain A **believes** $\stackrel{K_b}{\mapsto} B$.

Thus, we have followed the process of obtaining a public key step by step. The derivation may have seemed particularly trivial, but in fact similar derivations, with the same rules, do account for many of the authentication protocols found in the literature.

2 The Scope of the Logic

The purpose of the logic is to describe the beliefs of principals in the course of authentication, and this purpose affects our choice of constructs and rules.

For example, the logic has a notation for saying that K_b is a public key for B, but there is no rule for checking that K_b is long enough, as this sort of check does not typically happen in the course of authentication. Nor is there a way to show that a cryptosystem is secure and that it is used properly. (Voydock and Kent [20] and Moore [11] discuss inappropriate use of cryptosystems.)

Nevertheless, a few basic assumptions about encryption and how it is used underly the rules of the logic, and they are worth keeping in mind:

- Each encrypted unit is integral; it cannot be altered or adapted by a principal not knowing the appropriate key(s), without the encrypted part being reduced to nonsense. For example, an encrypted message $\{S, T\}_K$ may not be fabricated from $\{S\}_K$, $\{T\}_K$, $\{S, U\}_K$, and $\{V, T\}_K$ except by decryption and re-encryption, with knowledge of K.

- An encrypted message is verifiable. That is, when a principal applies a decryption algorithm D to a ciphertext C using key K, it will be able to judge whether the result is or is not the plaintext used to generate C in the first place, and thus whether it has been decrypted using the "right" key. The decision will necessarily be based on prior knowledge, total or partial, of the expected plaintext.

  Gong, Needham, and Yahalom [6] have recently proposed a way of bringing verifiability within the scope of the logic. This may be useful, since it is not always obvious whether an author has overlooked the question of verifiability or has assumed it is handled at a lower level.

- A principal can recognize his own encrypted messages, and will not mistakenly attribute them to another principal. This is easily achieved through direction bits, for example. In the logic, we make $\{X\}_K$ an abbreviation for the longer expression $\{X\}_K$ *from* P; we use the "from" field in side conditions to some rules.

  The assumption is more important in the shared-key world than in the public-key world, because it is uncommon for a principal to encrypt with his own public key in authentication protocols.

The foregoing are assumed properties of encryption. Another issue concerns the purposes of encryption. Encryption may be used to preserve secrecy, to maintain the integrity of a compound message, or simply to demonstrate knowledge of a key by using it. Our logic does not capture these distinctions.

A great deal of what is interesting in an authentication protocol does not depend on the cryptosystem chosen, but rather on the structure of the message exchange. Many of the bugs in published authentication protocols do not come from a poor cryptosystem, but rather from mistakes in the higher-level design. Often the formal analysis of a protocol has suggested the existence of these high-level bugs, and how to construct attacks. In order to illustrate how some bugs manifest themselves in proofs, let us consider a few flawed variants of our example:

- B's name is omitted: If B's name is not mentioned in the encrypted message from S, there is nothing to tie B to this message, and there is a danger of misinterpreting a message about some other principal C (perhaps an adversary) as a message about B. Notice that it does not suffice to include B's name in clear next to the encrypted packet, as cleartext can be modified by adversaries. In our analysis, we would not be entitled to idealize S's message as we have, since we cannot argue that S believes that K_b is B's key when S sends $\{K_b, T\}_{K_s^{-1}}$.

- S is not an authority: If it is not assumed that A trusts S to provide B's public key, the formal analysis would proceed exactly as above, but the last step would be blocked, as the last rule of inference would not be applicable. Thus, under the weaker assumptions, the protocol yields only A **believes** S **believes** $\stackrel{K_b}{\mapsto} B$.

- A has no key for S: If A does not know that K_s is S's public key, the protocol is totally useless, as A cannot read the message, or cannot recognize it as coming from S. We obtain A **sees** $\{\stackrel{K_b}{\mapsto} B, T\}_{K_s^{-1}}$, but cannot proceed further.

- S's secret key can be discovered: For the protocol to work, it is essential that S's secret key remain secret. Otherwise, A has no reason to trust a message signed with S's secret key, and in fact the message could come from an attacker. If we add a step where S broadcasts his secret key, the formal analysis still goes through, and we obtain that A believes that K_b is B's public key. When we trace the origin of this ridiculous conclusion, we find that the protocol no longer justifies the assumption A **believes** $\stackrel{K_s}{\mapsto} S$. The bad assumption naturally leads to a vacuous proof. Once this assumption is removed, the analysis becomes correct again, though quite uninteresting.

We make no formal attempt to detect incongruities between assumptions and messages. This might be possible, using existing tools to guard against release of secrets (see [2, 5, 9, 10, 12] for example).

- T is not fresh: If the message includes an old timestamp, A should not accept the message as fresh. Old messages can be replays, and could provide A with a compromised key for B [3]. In the formal analysis, the proof would be blocked at A **believes** S **said** $(\stackrel{K_b}{\mapsto} B, T)$.

- T is omitted: The danger is exactly the same as when T is not fresh. In the formal analysis, the idealized protocol would not include T, and the proof would finish with A **believes** S **said** $\stackrel{K_b}{\mapsto} B$.

If no bug becomes apparent, then the protocol is correct, in the sense that if all initial assumptions are justified then all conclusions are justified as well. More precisely, define knowledge as truth in all possible states (as in [8]); our notion of belief is a rudimentary approximation to knowledge, and it is simple to see that if all initial beliefs are knowledge then all final beliefs are knowledge and, in particular, they are true.

There is much room for improvement here. For example, we do not take into account complexity-theoretic assumptions or provide probabilistic guarantees. A strong semantics for the logic might remedy this, and at the same time it should provide a more compelling and robust notion of belief than those we have considered. Mark Tuttle and one of us (M.A.) have started work on an improved semantics.

Finally, we should point out that knowledge and belief must not be confused, and that the usual semantics of knowledge is not adequate for the belief operator. As knowledge is often defined, principals can know only true facts. It follows that if P knows that Q knows X, then P knows X. In contrast, one of our rules says that if P believes that Q believes X, then P believes X, but only under an additional hypothesis: P must believe that Q has jurisdiction over X. The additional hypothesis expresses the trust that P puts in Q, and should not be suppressed. The delegations in a distributed system are often founded on subtle trust relations, which it is fruitful to make explicit.

References

[1] CCITT. CCITT Blue Book, Volume VIII, Fascicle VIII.8, Recommendation X.509: The Directory—Authentication Framework. Geneva, 1989.

[2] R.A. DeMillo, N.A. Lynch, and M.J. Merritt. Cryptographic Protocols, *Proceedings of the Fourteenth ACM Symposium on the Theory of Computing*, 1982, pp. 383–400.

[3] D.E. Denning and G.M. Sacco. Timestamps in Key Distribution Protocols, *CACM* Vol. 24, No. 8, August 1981, pp. 533–536.

[4] D. Davis and R. Swick. Workstation Services and Kerberos Authentication at Project Athena, manuscript, 1989.

[5] D. Dolev and A.C. Yao. On the Security of Public Key Protocols, *IEEE Transactions on Information Theory* Vol. IT-29, No. 2, March 1983, pp. 198–208.

[6] L. Gong, R.M. Needham, and R. Yahalom. Reasoning about Belief in Cryptographic Protocols, manuscript, 1989.

[7] C.A.R. Hoare. An Axiomatic Basis for Computer Programming, *CACM* Vol. 12, No. 10, October 1969, pp. 576–580.

[8] J.Y. Halpern and Y.O. Moses. Knowledge and Common Knowledge in a Distributed Environment. *Proceedings of the Third ACM Conference on the Principles of Distributed Computing*, 1984, pp. 480–490.

[9] R.A. Kemmerer. Analyzing Encryption Protocols Using Formal Verification Techniques, *IEEE Journal on Selected Areas in Communications* Vol. 7, No. 4, May 1989, pp. 448–457.

[10] W.P. Lu and M.K. Sundareshan. Secure Communication in Internet Environments: A Hierarchical Key Management Scheme for End-To-End Encryption, *IEEE Transactions on Communications* Vol. 37, No. 10, October 1989, pp. 1014–1023.

[11] J.H. Moore. Protocol Failures in Cryptosystems. *Proceedings of the IEEE* Vol. 76, No. 5, May 1988, pp. 594–602.

[12] J.K. Millen, S.C. Clark, and S.B. Freedman. The Interrogator: Protocol Security Analysis, *IEEE Transactions on Software Engineering* Vol. SE-13, No. 2, March 1987, pp. 274–288.

[13] S.P. Miller, C. Neuman, J.I. Schiller, and J.H. Saltzer. Kerberos Authentication and Authorization System. *Project Athena Technical Plan* Section E.2.1, MIT, July 1987.

[14] R.M. Needham and M.D. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *CACM* Vol. 21, No. 12, December 1978, pp. 993–999.

[15] R.M. Needham and M.D. Schroeder. Authentication Revisited. *Operating Systems Review* Vol. 21, No. 1, January 1987, p. 7.

[16] D. Otway and O. Rees. Efficient and Timely Mutual Authentication. *Operating Systems Review* Vol. 21, No. 1, January 1987, pp. 8–10.

[17] Racal Research Ltd. Private communication from John Walker, 1989.

[18] R.L. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-key Cryptosystems, *Communications of the ACM* Vol. 21, No. 2, February 1978, pp. 120-126.

[19] M. Satyanarayanan, Integrating Security in a Large Distributed System, *ACM Transactions on Computer Systems* Vol. 15, No 3, August 1989, pp 247-280.

[20] V.L. Voydock and S.T. Kent. Security Mechanisms in High-Level Network Protocols, *Computing Surveys* Vol. 15, No. 2, 1983, pp. 135–171.