Detecting Key-Dependencies

Tage Stabell–Kulø[1], Arne Helme[1]*, and Gianluca Dini[2]

[1] Department of Computer Science, University of Tromsø, Norway
{tage,arne}@acm.org
[2] Dipartimento di Ingegneria della Informazione, University of Pisa, Italy
gianluca@iet.unipi.it

Abstract. The confidentiality of encrypted data depends on how well the key under which it was encrypted is maintained. If a session key was exchanged encrypted under a long-term key, exposure of the long-term key may reveal the session key and hence the data encrypted with it. The problem of key-dependencies between keys can be mapped onto connectivity of a graph, and the resulting graph can be inspected. This article presents a structured method (an algorithm) with which key-dependencies can be detected and analysed. Several well-known protocols are examined, and it is shown that they are vulnerable to certain attacks exploiting key-dependencies. Protocols which are free from this defect do exist. That is, when a session is terminated it is properly closed.

1 Introduction

In principle, any message that flows through a communication network can be recorded by eavesdroppers. Recording a message implies that the contents of the message can be revealed at any later time, even after both the sender and the intended receiver of the message have destroyed it. The contents of a message ceases to exist when no copy of the message exists in the system. It is obvious that two communicating partners are unable to enforce the extinction of messages exchanged between them.

Distribution of session keys among communication partners is a task that is accomplished using an authentication protocol. A closer look at authentication protocols reveals, not surprisingly, that many are constructed such that the session key is conveyed to the parties by means of messages. If the session key has been sent in a message, encrypted using some long-term key, then the session key does not cease to exist before the long-term key is destroyed. The term *dependency* will be used to describe the relationship that comes into existence between keys when one secret key is sent encrypted by another secret key, e.g., when the session key is encrypted by a long-term key. The effect of key-dependency is that the long-term secrecy of the session depends on the secrecy of the long-term key. It also influences the quality of the session key. The longer a long-term key is in use, the higher the risk of compromise, and the session key is exposed to

* Funded by the GDD-II project of the Research Council of Norway (project number 1119400/431)

C. Boyd and E. Dawson (Eds.): ACISP'98, LNCS 1438, pp. 356–365, 1998.

the same risk through the dependency. When a key-dependency arises from a protocol the assumption that a key *is* secret is transformed into an assumption that the key will *remain* secret. Thus, the protocol alters the assumptions, or, the way by which the assumptions are used alters them. This property is called *forward secrecy* [5].

When a session key *depends* on a long-term key the session is not *closed* before both the long-term key and the session key is destroyed. A session is not closed before the *only* way to obtain access to the data is by means of cryptanalysis. In other words, key-dependency is a security problem since it provides potential attackers with options.

As an example, consider the Kerberos protocol [11] outlined below:

Message 1 $A \rightarrow S : A, B$
Message 2 $S \rightarrow A : \{T_S, K_{AB}, B, \{T_S, K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$
Message 3 $A \rightarrow B : \{T_S, K_{AB}, A\}_{K_{BS}}, \{A, T_A\}_{K_{AB}}$
Message 4 $B \rightarrow A : \{T_A + 1\}_{K_{AB}}$

In the protocol description, A and B are the two principals that want to communicate, S is a server trusted by A and B to provide proofs on user/key bindings, K_{XY} is the secret key shared between principals X and Y and T_X is a time stamp made by X. The notation is adopted from [3]. The protocol description is slightly simplified, see [11] for more details.

In the protocol, the session key K_{AB} is sent in messages, encrypted with both K_{AS} and K_{BS}, in Message 2 and 3, respectively. When a short-term key (K_{AB}) is encrypted with a long-term key (in fact two keys, both K_{AS} and K_{BS}), a dependency is created between the short- and long-term keys. The implication is that the session based on K_{AB} is not properly closed before all the three keys K_{AB}, K_{AS} and K_{BS} have been discarded. The secrecy of the session depends on the long-term secrecy of the keys K_{AS} and K_{BS} and to properly close a session in the Kerberos system, both the keys K_{AS} and K_{BS} must be destroyed. The long-term privacy of A and B thus rests on the honesty of S as the protocol is in progress (e.g., S discards K_{AB} as soon as Message 2 has been sent) and the management of S after the protocol is terminated.

This paper presents an algorithm for analysing protocols for dependencies. Armed with it, designers and users of authentication protocols can analyse protocols in order to obtain a better understanding of the side effects of running them. Basically, the algorithm maps the dependencies onto connectivity in a graph. The resulting graph can be inspected to determine key-dependencies.

The rest of the article is structured as follows. First, in Section 2 a method to analyse protocols for key-dependencies is presented. The method consists of an algorithm which can be applied to a protocol description to produce a graph, and a description of how the resulting graph should be interpreted. Then, in Section 3, several well known protocols are analysed, both to demonstrate the usefulness of the method and to show the protocols' properties in respect to key-dependencies. Section 4 contains the discussion and an outline of future work. At the end, in Section 5, the conclusions are presented.

2 Analysing dependencies

This section outlines a structured method to detect and analyse key-dependencies in key-distribution protocols. The idea is to model the problem of locating key-dependencies as determining the connectivity of a directed graph, and the graph can be inspected in order to detect key-dependencies that render sessions open. More precisely, a key distribution protocol is represented as a directed graph. In such a graph $\mathcal{G}$, the members of the set of vertices $V(\mathcal{G})$ represents either data—stemming from the receipt of a message, generated locally or the result of a decryption—or transformations such as decryption. An element (x, y) of the set of edges $E(\mathcal{G})$ represents the fact that y is derived from x. For example, if y is the "result" of decrypting x, then $(x, y) \in E(\mathcal{G})$. The graph is then interpreted according so certain rules, and the interpretation reveals information about the protocol. Vertices are drawn as nodes containing a string identifying the data or transformation. Edges are drawn as arches.

Modelling key-dependencies as a graph is closely related to the methods described in [7], where a graph is built to detect the weakest (shortest) path between passwords that can be guessed (or text that can be verified) and a session key. A similar approach is used here to detect key-dependency properties in authentication and key-distribution protocols.

The set $V(\mathcal{G})$ of vertices in the key-dependency graph $\mathcal{G}$ is defined as follows:

V1. The set $V(\mathcal{G})$ has one element for each message, for each message component, and for each key necessary to decrypt the message. For instance, if message $m = \langle x, y \rangle$ is considered, then $m, x, y \in V(\mathcal{G})$. Moreover, if a conventional cryptosystem is used and the message $m = \{x, y\}_k$ is considered, then $V(\mathcal{G})$ contains one element for the message m itself, one element for each message component (i.e., $m, x, y \in V(\mathcal{G})$), and one element for the key k. Similarly, if a public-key cryptosystem is used and the message $m = \{x, y\}_k$ is considered, then $m, x, y \in V(\mathcal{G})$ as above, and $k^{-1} \in V(\mathcal{G})$, where k^{-1} is the decryption key corresponding to the public encryption key k.

V2. The set $V(\mathcal{G})$ has one element for the computation a principal has to perform in order to obtain the key (or other material) on the material received through messages, in its clear-text form, or local information.[1] Moreover, the set of vertices contains one element for each argument of the computation and one for the result. For instance, if the computation $y = f(x_1, \ldots, x_n)$ is considered, then $f, y, x_i \in V(\mathcal{G}), i = 1, \ldots, n$.

Notice that by **V1**, when two messages containing the same datum, e.g., the messages $m_1 = \langle a, x \rangle, m_2 = \langle b, x \rangle$, the resulting set of vertices will have five elements (m_1, m_2, a, b, x) as x is one datum transmitted twice.

The set of edges $E(\mathcal{G})$ is defined as follows

E1. Let m be a message with n components, $m = \langle m_1, \ldots, m_n \rangle$. Then, $(m, m_i) \in E(\mathcal{G}), i = 1, \ldots, n$.

[1] This computation is of course different from the computation that a principal has to perform in order to build up a message.

E2. Let $m = \{x\}_k$ be a message where x is encrypted with shared-key encryption. A pair of elements are added to $E(\mathcal{G})$, namely (m, x) and (k, x). Public-key encryption can be characterized similarly: if $m = \{x\}_k$ is considered, then the edges (m, x) and (k^{-1}, x) are added.

E3. If a computation $y = f(x_1, \ldots, x_n)$ is considered, then $(f, y), (x_i, f) \in E(\mathcal{G})$.

For instance the message $\langle A, B, \{X\}_K \rangle$, where K is a shared key, yields the following graph. Each arch is labelled with the rule that applies to it.

$$\begin{tikzcd}
A & A, B, \{X\}_K \arrow[l, "E1"'] \arrow[r, "E1"] \arrow[d, "E1"] & \{X\}_K \arrow[d, "E2"] & K \arrow[dl, "E2"] \\
 & B & X &
\end{tikzcd}$$

After a graph has been constructed according to the rules **V1**–**V2** and **E1**–**E3**, it is reduced using the following rules.

R1. Find all vertices that represent a long-term key.

R2. For each distinct path in $\mathcal{G}$ where the initial vertex represents a long-term key and the terminating vertex represents the session key, mark all vertices along the path.

R3. Remove from $V(\mathcal{G})$ all unmarked elements.

R4. Remove from $E(\mathcal{G})$ any element which one (or both) endpoints are no longer in $V(\mathcal{G})$.

In the resulting reduced graph, key-dependencies are represented as edges. Intuitively, the graph shows possible weak links in the chain of keys involved in a system (in so far as a decryption key can be called a weak link).

We interpret the resulting graph as follows:

I1. For all adjacent vertices representing data, the (contents of the) terminal vertex depends on the (contents of the) initial vertex. We denote this dependency by *or-dependency.*

I2. A vertex (f) representing a transformation depends on the union of all vertices where there exist an edge so that f is the terminating vertex. We denote this dependency by *and-dependency.*

In addition, dependency has the property of being transitive. We say that a key-dependency exists in the protocol if there is at least one path in the reduced graph

Below, five protocols are analysed, both to demonstrate that the algorithm indeed captures key dependencies, and to evaluate the protocols for key-dependencies.

3 Examples

In this section five well-known protocols are analysed by means of the method described in the previous section. As will be shown, these protocols give rise to a varying degrees of key-dependencies.

3.1 Wide-Mouthed-Frog Protocol

First the Wide-Mouthed-Frog protocol [3], a relatively simple protocol which involves three parties. In this protocol, the two parties A and B each have a secret key, shared with the authentication server S, K_{AS} and K_{BS} respectively. This protocol consists of only two messages.

$$\begin{array}{l} \text{Message 1 } A \rightarrow S : A, \{T_A, B, K_{AB}\}_{K_{AS}} \\ \text{Message 2 } S \rightarrow B : \{T_S, A, K_{AB}\}_{K_{BS}} \end{array}$$

When following the procedure outlined above, the following graph is obtained.

$$\begin{tikzcd}
\boxed{A, \{T_A, B, K_{AB}\}_{K_{AS}}} \arrow[r, "E1"] \arrow[d, "E1"] & A & K_{BS} \arrow[dr, "E2"] & \boxed{\{T_S, A, K_{AB}\}_{K_{BS}}} \arrow[d, "E2"] \\
\{T_A, B, K_{AB}\}_{K_{AS}} \arrow[r, "E2"] & T_A, B, K_{AB} \arrow[r, "E1"] \arrow[d, "E1"] & K_{AB} & T_S, A, K_{AB} \arrow[l, "E1"'] \arrow[d, "E1"] \\
K_{AS} \arrow[ur, "E2"] & T_A, B & & T_S, A
\end{tikzcd}$$

The two messages that were sent have been framed for clarity. In addition, each arch is labelled according to the rule that applies to it.

The long-term keys are K_{AS} and K_{BS}. Applying the rules **R2**–**R3** yields the following graph:

$$\begin{tikzcd}
K_{AS} \arrow[r] & T_A, B, K_{AB} \arrow[dr] & & T_S, A, K_{AB} \arrow[dl] & K_{BS} \arrow[l] \\
& & K_{AB} & &
\end{tikzcd}$$

By **I1** the key K_{AB} or-depends on T_A, B, K_{AB} and T_S, A, K_{AB}. These depends, again by **I1**, on two nodes containing long-term encryption keys. Recalling that dependency is transitive, we interpret the graph to imply that K_{AB} depends on either one of two other keys, K_{AS} and K_{BS}. That is, knowing either K_{AS} *or* K_{BS} will make it possible to recover K_{AB}, provided that the attacker has a recording of the protocol and the session. Consequently, in order to close a session based on K_{AB}, both K_{AS} and K_{BS} must be discarded (in addition to K_{AB}). However, both keys are known to S, which implies that A and B does not control the closing of the session.

3.2 Node-to-node channel

Consider the protocol to set up a node-to-node channel between the two nodes A and B in a distributed system [8]. The essence is that both A and B invent a random number, the numbers are exchanged, and the session key is constructed as a function of them. A and B are assumed to have public keys K_A and K_B, known to the other party, and both are competent to invent good random numbers. The protocol is slightly simplified, see [8] for a complete description.

$$\begin{array}{l} \text{Message 1 } A \rightarrow B : \{J_A\}_{K_B} \\ \text{Message 2 } B \rightarrow A : \{J_B\}_{K_A} \end{array}$$

The session key K_{AB} is then found as a hash of J_A and J_B. Building the graph and reducing it, let $h()$ indicate the hash function, gives the following graph:

$$\begin{tikzcd} K_A^{-1} \arrow[r] & J_A \arrow[r] & h(J_A, J_B) \arrow[d] & J_B \arrow[l] & K_B^{-1} \arrow[l] \\ & & K_{AB} & & \end{tikzcd}$$

We notice that **I1** does not apply to this graph. By **I2** and transitivity, K_{AB} and-depends on K_A^{-1} *and* K_B^{-1}. Thus, to decrypt the session protected by K_{AB} both K_A^{-1} and K_B^{-1} (assuming both J's are discarded) need to be compromised.

3.3 SSL 3.0

SSL is a protocol designed to be used in a variety of circumstances and with a variety of security environments, and with a variety of cryptographic tools[2] This protocol is widely used, in particular by Web-browsers. SSL can be used in settings where both the client and server have public keys and mutual authentication is desired. With some simplifications (for example, only one method for hashing), the protocol can be described as follows:

$$\begin{array}{ll} \text{Message 1 } C \rightarrow S : & C, N_C, T_C \\ \text{Message 2 } S \rightarrow C : & N_S, T_S, K_S, \{N_C\}_{K_S^{-1}} \\ \text{Message 3 } C \rightarrow S : & K_C, \{P\}_{K_S}, \{H(M + H(Z + M))\}_{K_C^{-1}}, \\ & H(M + H(Y_C + M)) \\ \text{Message 4 } S \rightarrow C : & H(M + H(Y_S + M)) \end{array}$$

In the protocol description, T_S and T_C are the time stamps, N_S and N_C are 28-byte nonces, and K_S and K_C are the public keys of the server and client, respectively. The keys are sent together with X.509 certificates making claims on the user-key binding [4]. P is the 46 bytes called "pre-master-secret", the function H is MD5 [10], M is the master-secret derived from the pre-master-secret by combining the pre-master-secret with N_C and N_S plus some padding, and hashing the result. Z is the concatenation of Message 1 and Message 2, Y_C is the concatenation of Z and the number 1129074260, Y_S is the concatenation of Z, Message 3 and the number 1397904978. In essence, the parties sign each others nonces.

[2] A detailed description of SSL is available at URL:http://home.netscape.com/eng/-ssl3/ssl-toc.html.

When processed according to the graph reduction rules, the following is obtained:

$$\begin{tikzcd} K_S^{-1} \arrow[r] & P \arrow[r] & \mathrm{M} \end{tikzcd}$$

Inspection of the reduced graph reveals that the secrecy of the master secret depends solely on the secrecy of K_S^{-1}, which again implies that the client is unable to close the session based on the master secret. Although SSL is based upon public-key cryptography, its behavior with respect to key dependencies is weaker that the Wide-Mouthed-Frog protocol. In the latter, the "users" have the possibility to close the session by changing the key they share with the server. In SSL, this is not possible. The analysis of SSL also demonstrates that the use of public keys is not a panacea.

3.4 Demonstration Protocol

In [7], quite a few protocols are described, and in the following, the *Demonstration Protocol* is studied in more detail. It consists of eight messages sent between two principals A and B and a security server S. The last three messages form an exchange of nonces for verification, and are left out of the protocol description:

$$\begin{aligned}
&\text{Message 1 } A \rightarrow S : \{A, B, na_1, na_2, \{ta\}_{K_{AS}}\}_{K_S} \\
&\text{Message 2 } S \rightarrow B : A, B \\
&\text{Message 3 } B \rightarrow S : \{B, A, nb_1, nb_2, \{tb\}_{K_{BS}}\}_{K_S} \\
&\text{Message 4 } S \rightarrow A : \{na_1, k \oplus na_2\}_{K_{AS}} \\
&\text{Message 5 } S \rightarrow B : \{nb_1, k \oplus nb_2\}_{K_{BS}}
\end{aligned}$$

The symbol $\oplus$ denotes the bit-wise exclusive-or operation, the datums prefixed by n's are nonces, the key K_S is the public key of S, the keys K_{AS} and K_{BS} are shared between A (and B) and S, and k is the session key. The protocol is slightly simplified—confounders are left out—see [7] for the details. In the graph, the nodes denoted with $\otimes$ represent a computation as described by **V2**. In this protocol, the computation is in fact bit-wise exclusive-or, but regarding it as a general computation does not alter the graph.

The algorithm produces the following graph:

$$\begin{tikzcd}
K_{AS} \arrow[d] & na_2 \arrow[d] & K_S^{-1} \arrow[l] \arrow[r] & nb_2 \arrow[d] & K_{BS} \arrow[d] \\
k \oplus na_2 \arrow[r] & \otimes \arrow[r] & k & \otimes \arrow[l] & k \oplus na_2 \arrow[l]
\end{tikzcd}$$

The two arches leading to k are or-dependencies, implying that k depends on two sets of keys while, by **I2**, the arches to the transformations induce and-dependencies implying that the key depends on all keys. However, the secret key K_S^{-1} is a member of both sets. Furthermore, all keys represented in the graph (K_S, K_{AS} and K_{BS}) are known to S and none of them are discarded after Message 5 has been sent (after which S no longer takes part in the session). On the other hand, compromise based on K_{AS} (or K_{BS}) alone is not enough.

Compared to the Wide-Mouthed-Frog protocol, the outcome is better since compromise of one of the user's key is not enough to endanger the privacy of the session. The outcome is better than that of SSL, in that if A and B both change the key they share with S, the session is closed, while in SSL the session key depends on K_S^{-1} alone.

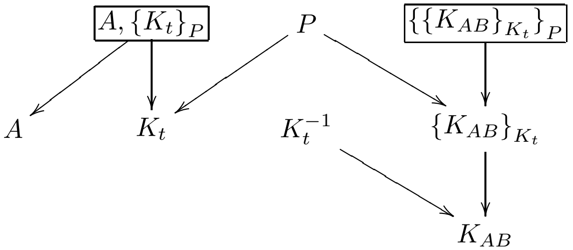
3.5 Encrypted Key Exchange

From the previous examples, it is clear that key-dependencies arise when session keys are encrypted with long-term keys. Using a fresh, temporary public key avoids the key-dependency issue. As an example, the Encrypted Key Exchange [2] is described.

Let A and B be the two parties, P a shared secret, K_t a public key with K_t^{-1} as the secret counterpart, and K_{AB} a session key. The protocol consists of five messages, of which the last three are for mutual verification of the key; they are left out. A creates the temporary public key pair (K_t, K_t^{-1}) and sends K_t to principal B. B creates the session key K_{AB} and uses K_t to securely send K_{AB} to A. The first two messages are:

$$\begin{array}{l} \text{Message 1 } A \rightarrow B : A, \{K_t\}_P \\ \text{Message 2 } B \rightarrow A : \{\{K_{AB}\}_{K_t}\}_P \end{array}$$

The protocol gives rise to the following graph:



Reduction results in the following graph:

$$P \longrightarrow \{K_{ab}\}_{K_t} \longrightarrow K_{AB}$$

First, notice that that the secret, temporary key K_t^{-1} is not included in the graph since it is not a long-term key. Second, inspection of the graph reveals that holding key P is not sufficient to obtain the K_{AB} because it does not come to depend on the key K_t^{-1}. The graph, in its reduced form, captures this fact by depicting a path from P to K_{AB} which contains encrypted material whose decryption key is not depicted. In other words, the graph captures the essence in the protocol, that shared and public key encryption complement each other.

4 Discussion and future work

The analysis of the five protocols in the previous section reveals a clear relationship between the use of shared-key encryption and key-dependencies. Without a pre-arranged secret channel it is hard for participants to verify that a session key indeed is correct [6]. This becomes evident in protocols based on shared keys, where the session key *must* be exchanged over the shared channel as there is no other alternative. In such settings, key-dependency is inevitable. This can be argued for as follows: Assume two peer principals wanting to communicate and exchange a session key by means of a security server. Based on the messages sent, B must decide on the same session key as A. This is only possible if A can assume that B's actions are deterministically based on the input (the messages sent by A and S). If C knows the algorithms that B follows, and can read the channel to and from B, C will be able to achieve the same result as B. Thus, it is impossible to avoid key-dependency in a system solely depending on shared key encryption. The above analysis verifies this.

By considering how dependencies arise it becomes evident that to improve the situation with shared keys there must not exist a path in the graph from long-term keys to the session key(s). That is, a cryptographic channel must exist that is not transmitted. Today, public key cryptography is the most common choice for such channels, but more exotic possibilities are possible, see for example [9]. However, as became evident in the analysis of SSL, protocols using public keys can also give rise to key-dependencies.

In systems based on public key cryptography and where a trusted third party is used to ease authentication, one can separate the issues of authentication from the exchange of a session key. In particular, one can leave it to the users to handle the latter. This approach, for more or less this reason, is taken in [8, Footnote 10] (and in [12]). When the server is not engaged in issuing the session key, no key-dependency arises on some key known to the server. It is, however, regarded as good engineering practice to involve a server in this process, see [3] and [1, example 11.2].

Although the protocols analysed in this article were not designed with forward secrecy in mind, it is still important to point out that key-dependency vulnerabilities do exist in them. The design of SSL, for example, is considered sound for authentication purposes, but as shown in this article, can be vulnerable to attacks exploiting key-dependencies.

As it stands, the analysis must be carried out "by hand". Among the future lines of work is an effort to parse a protocol description to build the graph directly. Also, processing—as in the Note-to-Node protocol—to obtain keys needs attention as it is not captured in the messages that are sent.

5 Conclusions

As computers are used in an ever larger part of life, the importance of forward secrecy becomes paramount. Key-dependencies have implications on a protocols' forward secrecy, and a tool to analyse protocols has merits.

In this paper, a structured method for analysing authentication protocols for key-dependencies has been presented, and is usefulness has been demonstrated.

Acknowledgments

Funding was received from the Commission of European Communities, Esprit Project 20422, “Moby Dick, The Mobile Digital Companion”. Dini was partly supported by the Moby Dick project and partly by Ministero dell’Universita e della Ricerca Scientifica e Tecnologica, Italy. Funding has also been received from the Norwegian Research Council through several projects. General funding has been received from the GDD project (project number 112577/431), Helme is supported by the GDD-II project (project number 1119400/431), and Stabell-Kulø received support for a sabbatical in Pisa (project number 110911). The support is gratefully acknowledged.

Alberto Bartoli, Feico Dillemma, Terje Fallmyr and Sape Mullender provided us with feedback that improved the presentation.

References

1. Martín Abadi and Roger Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, January 1996.
2. Steve Bellovin and Michael Merritt. Encrypted key exchange: passord-based protocols secure against dictionary attacks. In *Proceedings of the 1992 IEEE Computer Society Conference on Research in Security and Privacy*, pages 72–84. IEEE Computer Society, 1992.
3. Michael Burrows, Martín Abadi, and Roger Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, February 1990.
4. CCITT. Information Technology — Open Systems Interconnection — The Directory: Authentication Framework. CCITT Recommodation X.509, ISO/IEC 9594-8, December 1991.
5. W. Diffie, P.C. van Oorschot, and M.J. Weiner. Authentication and Authenticated Key Exchanges. In *Designs, Codes and Cryptography 2*, pages 107–125, 1992.
6. Li Gong. Increasing availability and security of an authentication service. *IEEE Journal on Selected Areas in Communications*, 11(5):657–62, June June 1993.
7. Li Gong, T. Mark A. Lomas, Roger M. Needham, and Jerome H. Saltzer. Protecting poorly chosen secrets from guessing attacks. *IEEE Journal on Selected Areas in Communications*, 11(5):648–56, June 1993.
8. Butler Lampson, Martín Abadi, Michael Burrows, and Edward Wobber. Authentication in distribued systems: theory and practice. *ACM Transactions on Computer Systems*, 10(4):265–310, November 1992.
9. Ralph C. Merkle. Secure Communication over Insecure Channels. *Communications of the ACM*, 21(4):294–299, April 1978.
10. Ronald Rivest. RFC 1321: The MD5 Message-Digest Algorithm, April 1992.
11. J. G. Steiner, B. G. Neumann, and J. I. Schiller. Kerberos: An Authentication System for Open Network Systems. In *Proc. of the Winter 1988 Usenix Conference*, pages 191–201, February 1988.
12. Edward Wobber, Martín Abadi, Mike Burrows, and Butler Lampson. Authentication in the Taos operating system. *ACM Transactions on Computer Systems*, 12(1):3–32, February 1994.