

Analysis of Key-dependencies in Cryptographic Protocols*

Tage Stabell-Kulø
Department of Computer Science
University of Tromsø
Norway

Arne Helme
Department of Computer Science
University of Twente
The Netherlands

Gianluca Dini
Dipartimento di Ingegneria della Informazione
University of Pisa
Italy

Draft of November 19, 1997

Abstract

The confidentiality of encrypted data depends on how well the key under which it was encrypted is maintained. Keys cannot be exposed safely before the data encrypted with it has ceased to exist. If a session key was exchanged encrypted under a long-term key, then exposure of the long-term key may reveal the session key (if a copy of the message containing the key still exists somewhere) and hence the data encrypted with it. Keeping track of key-dependencies is therefore very important. 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.

*Submitted for publication to the 1998 IEEE Symposium on Security and Privacy

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 terminates it is properly closed (assuming the participants are honest).

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 that the contents of messages exchanged between them cease to exist.

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 in such way that the session key is conveyed to the parties by means of messages. If the session key has been sent in a message, probably 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 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* [Diffie et al., 1992].

Consider the Kerberos protocol [Steiner et al., 1988] 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 [Burrows et al., 1990]. The protocol description is slightly simplified, see [Steiner et al., 1988] for more details.

In the protocol, the session key K_{AB} is sent in messages, encrypted with both K_{AS} and K_{BS} , in Messages 2 and 3, respectively. When, as here, 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. Thus, 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 discarded. 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 it. 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 comes a presentation of a method to analyse protocols for key-dependencies. It 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, is the conclusions presented.

2 Analysing dependencies

This section outlines a structured method to detect and analyse key-dependencies in key-distribution protocols. The central idea is to model the problem of locating key-dependencies between keys as determining the connectivity of a directed graph. More precisely, a graph describing a key distribution protocol contains nodes representing either data or transformations. Edges leading to datum nodes represent dependencies while edges leading to transformation

nodes represent input to the transformation. The graph can be inspected in order to detect key-dependencies that render sessions open.

Modeling key-dependencies as edges of a graph is closely related to the methods described in [Gong et al., 1993], 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. This article uses a similar approach to detect key-dependency properties in authentication and key-distribution protocols.

The set of nodes in the key-dependency graph \mathcal{G} is defined as follows:

- N1.** Graph \mathcal{G} has one node 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 \mathcal{G} contains nodes for m, x and y . Moreover, if a conventional cryptosystem and the message $m = \{x, y\}_k$ are considered, then \mathcal{G} contains one node for the message m itself, one node for each message component (i.e., one for x and one for y), and one node for the key k . Similarly, if a public-key cryptosystem and the message $m = \{x, y\}_k$ are considered, then graph \mathcal{G} contains one node for each message component, x, y , one node for the message m itself, and one node for the private key k^{-1} (the decryption key corresponding to the public encryption key k).
- N2.** The graph \mathcal{G} has one node for the computation that a principal has to perform in order to obtain the key (or other material) on the material received through messages, in its cleartext form, or local information¹. Moreover, the graph contains one node for each argument of the computation and one for the result. For instance, if the computation $x = f(x_1, \dots, x_n)$ is considered, then graph \mathcal{G} contains one node for f , one node for x and one node for each $x_i, i = 1, \dots, n$.

Notice that by **N1**, if a graph is built from two messages containing the same datum, e.g., the messages $m_1 = \langle a, x \rangle, m_2 = \langle b, x \rangle$, the resulting graph will have five nodes (m_1, m_2, a, b, x) as x is one datum transmitted twice.

The set of arcs in \mathcal{G} is defined as follows

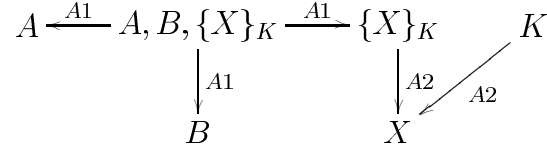
- A1.** Let m be a message with n components, $m = \langle m_1, \dots, m_n \rangle$. Then, graph \mathcal{G} contains one arc from m to each $m_i, i = 1, \dots, n$.
- A2.** Shared key encryption $m = \{x\}_k$ is characterised by a pair of arcs, the one from m to x and the other from k to x . Public-key encryption can

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

be characterised similarly: if $m = \{x\}_k$ is considered, then graph \mathcal{G} contains one arc from m to x and one arc from k^{-1} to x .

- A3.** If a computation $x = f(x_0, x_1, \dots, x_n)$ is considered, then graph \mathcal{G} contains a set of arcs as follows: one arc from every x_i to f , ($i = 0, \dots, n$), and one arc from f to x .

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



After the graph has been constructed according to the rules **N1–N2** and **A1–A3**, it is reduced using the following rules.

- R1.** For each distinct path from any node corresponding to a long-term key to the session key, mark all nodes on the path.
- R2.** Remove all unmarked nodes.

The result is that information about key-dependencies is transformed to edges in the 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

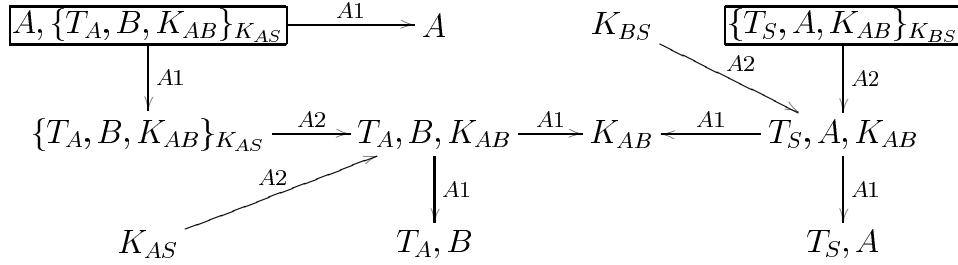
In this section five well-known protocols are analysed by means of the method described in the previous section. As will be shown, these protocols give rise to a varying degrees of key-dependencies.

3.1 Wide-Mouthed-Frog Protocol

First the Wide-Mouthed-Frog protocol [Burrows et al., 1990], 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.

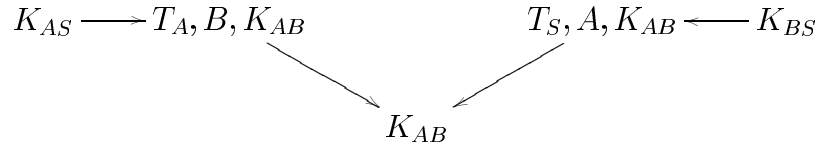
$$\begin{array}{l}
 \text{Message 1 } A \rightarrow S : A, \{T_A, B, K_{AB}\}_{K_{AS}} \\
 \text{Message 2 } S \rightarrow B : \{T_S, A, K_{AB}\}_{K_{BS}}
 \end{array}$$

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



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

The long-term keys are K_{AS} and K_{BS} . Applying the rules **R1–R2** yields the following graph:



The graph shows that key K_{AB} depends on either of two other keys, K_{AS} and K_{BS} . Thus, knowing *either* of the latter two 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. However, both keys are known to S , which implies that A and B do not control when the session will be closed.

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 [Lampson et al., 1992]. 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 [Lampson et al., 1992] for a complete description.

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

The session key K_{AB} is then found as a hash of J_A and J_B . Building the graph, removing the nodes that are marked public, and let $h()$ indicate the

hash function, produces the following graph:

$$\begin{array}{ccccccc}
 K_A^{-1} & \longrightarrow & J_A & \longrightarrow & h(J_A, J_B) & \longleftarrow & J_B \longleftarrow K_B^{-1} \\
 & & & & \downarrow & & \\
 & & & & K_{AB} & &
 \end{array}$$

When inspecting this graph, it becomes clear that the session key depends on two values, which again depend on two different keys. In particular, a single edge, not two as in the Wide-Mouthed-Frog protocol, is leading to the session key. In order to find K_{AB} one must find *both* J 's and they each depend on different keys. 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.

We can conclude that the node-to-node channel protocol achieves a better result than the Wide-Mouthed-Frog protocol. The main reason is way public-key cryptography is used. The public key in a public-private key-pair gives rise to a one-way channel leading to the holder of the private key. The one-way property implies that if A sends a datum to B through the channel represented by B 's public key, and A dutifully discards the datum, there is no way to regain the datum without B 's participation. This is used in the protocol by sending parts of the key on the unrelated channels. This can also be seen as a separation of the issues of authentication and conveying a secret.

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 tools² 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:

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

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

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 [CCITT, 1991]. P is the 46 bytes called “pre-master-secret”, the function H is MD5 [Rivest, 1992], 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.

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

$$K_S^{-1} \longrightarrow P \longrightarrow M$$

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 behaviour with respect to key dependencies is weaker than 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

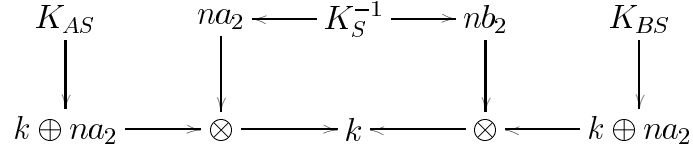
In [Gong et al., 1993], quite a few protocols are described, and in the following, the *Demonstration Protocol* is studied in more detail. It consists of eight messages sent between two principals A and B and a security server S . The last three messages form an exchange of nonces for verification, and 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 [Gong et al., 1993] for the details. In the graph, the nodes denoted with \otimes represent a

computation as required by **N2**. 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:



Observe that there are two edges leading to k , indicating that k depends on two sets of keys. However, the secret key K_S^{-1} is a member of both sets. The outcome from the node-to-node protocol is better in this respect. Furthermore, all endpoints leading to k in the graph (K_S, K_{AS} and K_{BS}) are known to S and neither 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_A (or K_B) 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.

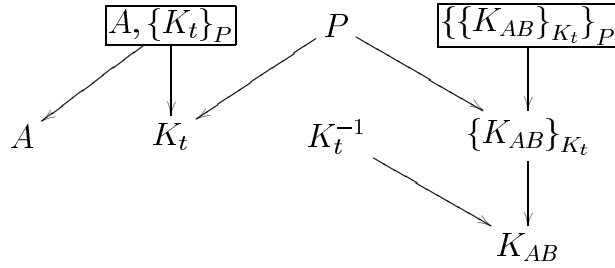
3.5 Encrypted Key Exchange

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 [Bellare and Merritt, 1992] 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. 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 the secret, temporary, key 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 comes to depend also on the key K_t^{-1} , which is temporary. 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 [Gong, 1993]. 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, a key-dependency is inevitable. This can be argued for as follows: Assume two peer principals wanting to communicate, and exchange a session key, by the 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 an unencrypted 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 [Merkle, 1978]). However, as became evident in the analysis of SSL, public keys also gives rise to key-dependencies if not applied with care.

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 [Lampson et al., 1992, Footnote 10] (and in [Wobber et al., 1994]). 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 [Burrows et al., 1990] and [Abadi and Needham, 1996, 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 has implications on a protocols’ forward secrecy, and a tool to analyse protocols has its merits.

In this paper, a structured method for analysing authentication protocols for key-dependencies has been presented, and its 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 111400/431), 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

- [Abadi and Needham, 1996] Abadi, M. and Needham, R. (1996). Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15.
- [Bellovin and Merritt, 1992] Bellovin, S. and Merritt, M. (1992). 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.
- [Burrows et al., 1990] Burrows, M., Abadi, M., and Needham, R. (1990). A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36.
- [CCITT, 1991] CCITT (1991). Information Technology — Open Systems Interconnection — The Directory: Authentication Framework. CCITT Recommendation X.509, ISO/IEC 9594-8.
- [Diffie et al., 1992] Diffie, W., van Oorschot, P., and Weiner, M. (1992). Authentication and Authenticated Key Exchanges. In *Designs, Codes and Cryptography 2*, pages 107–125.
- [Gong, 1993] Gong, L. (June 1993). Increasing availability and security of an authentication service. *IEEE Journal on Selected Areas in Communications*, 11(5):657–62.
- [Gong et al., 1993] Gong, L., Lomas, T. M. A., Needham, R. M., and Saltzer, J. H. (1993). Protecting poorly chosen secrets from guessing attacks. *IEEE Journal on Selected Areas in Communications*, 11(5):648–56.
- [Lampson et al., 1992] Lampson, B., Abadi, M., Burrows, M., and Wobber, E. (1992). Authentication in distributed systems: theory and practice. *ACM Transactions on Computer Systems*, 10(4):265–310.

- [Merkle, 1978] Merkle, R. C. (1978). Secure Communication over Insecure Channels. *Communications of the ACM*, 21(4):294–299.
- [Rivest, 1992] Rivest, R. (1992). RFC 1321: The MD5 Message-Digest Algorithm.
- [Steiner et al., 1988] Steiner, J. G., Neumann, B. G., and Schiller, J. I. (1988). Kerberos: An Authentication System for Open Network Systems. In *Proc. of the Winter 1988 Usenix Conference*, pages 191–201.
- [Wobber et al., 1994] Wobber, E., Abadi, M., Burrows, M., and Lampson, B. (1994). Authentication in the Taos operating system. *ACM Transactions on Computer Systems*, 12(1):3–32.