Programming Techniques and Data Structures

Ellis Horowitz Editor

How to Expose an Eavesdropper

RONALD L. RIVEST and ADI SHAMIR

ABSTRACT: We present a new protocol for establishing secure communications over an insecure communications channel in the absence of trusted third parties or authenticated keys. The protocol is an improvement over the simpler protocol in which the communicating parties exchanged their public encryption keys and used them to encrypt messages. It forces a potential eavesdropper—if he wants to understand the messages—to reveal his existence by modifying and seriously garbling the communication.

1. INTRODUCTION

Public-key cryptosystems [1, 2] with central directories that authenticate and distribute the keys give their users a high degree of protection. However, for large, loosely organized and continuously changing networks (telephones, home computers, electronic mail, etc.), a central directory is almost impossible to maintain, and the communicating parties have to rely on local, insecure directories or they have to exchange their public keys themselves. The purpose of this paper is to suggest a new communications protocol that protects the network members against eavesdroppers even in this case.

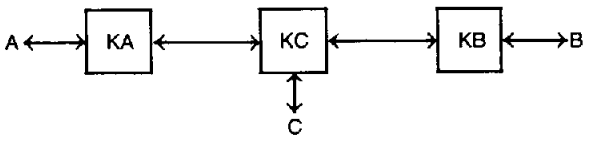
An application we have in mind is one in which two company executives who can recognize each other's voice but who do not have each other's key want to communicate via a scrambled telephone line. All the key exchanges and encryption/decryption parts of the protocol are handled automatically, and the two executives are aware only of each other's unscrambled voice.

This research was partially supported by NSF Grant No. MCS-8006938.

2. THE EAVESDROPPER SCENARIO

Consider the following eavesdropper scenario. We define an eavesdropper to be someone who wants to monitor the communication between two parties without tampering with the data and without exposing his existence. He may modify the ciphertext stream in any manner whatsoever (deleting, delaying, substituting, or inserting ciphertexts) as long as he does not change the cleartexts received by the communicating parties. Note that, in the context of a public-key cryptosystem, a successful eavesdropper must actively participate in the key-exchange protocol; but, if he wants to monitor the communications for a long period of time, he would have to try to behave as transparently as possible, since any trace he leaves in the cleartexts is likely to arouse suspicion.

A well-known and serious problem with unauthenticated public-key exchange protocols is that the communication between the two parties, *A* and *B*, can be transparently monitored by an eavesdropper, *C*, who inserts into the communication line an encryption/decryption device as follows:



When A wants to communicate with B, C replaces both the public key, KA, that A sends to B and the public key, KB, that B sends to A by his own public key, KC (or by a pair of keys, KC' and KC'', if the keys contain an identifying prefix). Whenever A sends an encrypted message $E_{KC}(MA)$ to B, C intercepts it, decrypts it in order to read MA, and then reencrypts it as $E_{KB}(MA)$ before sending it to B. Messages, MB, sent by B to A are handled in a similar way.

The communicating parties can try to trap C by sending their public keys again for verification as part of the cleartexts they exchange. If C is not allowed to change such messages, he may get into trouble. To avoid this technical difficulty, we allow eavesdroppers to change all the key-related portions of messages and assume that they are clever enough to detect them in real time.

One can almost prove that when all the communication lines between A and B are controlled by C, this cryptanalytic attack cannot be foiled. If A and B cannot authenticate the keys they receive, KC looks just as legal as KA and KB. Since all of C's actions are transparent, A and B cannot possibly distinguish between a scenario in which C exists and a scenario in which C does not exist. Yet, we claim that a simple change in the communications protocol can dramatically reduce the danger posed by eavesdroppers.

We note that no such protocol can be perfect since it is conceivable that C could pretend to be B sufficiently well that A would have no means of determining that he was talking to C rather than B. This possibility is of particular concern when A and B are merely machines. However, the net effect of the protocol to be proposed is that any authentication provided by A's *a priori* knowledge of B's communication patterns, knowledge or voice is used to expose the would-be eavesdropper. This is a feature that the ordinary "exchange of public keys" protocols does not possess, since there, C can successfully eavesdrop without any *a priori* knowledge about A and B.

3. THE "INTERLOCK" PROTOCOL

After A and B have exchanged their public keys, they exchange a pair of data blocks, MA and MB, as follows:

1. A encrypts MA under KB but sends B only the first half of the bits of the resulting ciphertext $E_{KB}(MA)$.
2. B encrypts MB under KA and sends A the first half of $E_{KA}(MB)$.
3. A sends B the second half $E_{KB}(MA)$.
4. B sends A the second half of $E_{KA}(MB)$.
5. A and B concatenate the two halves of $E_{KA}(MB)$ and $E_{KB}(MA)$, respectively, and use their secret decryption keys to read the messages.

Each side performs a step in this protocol only after he receives the information sent by the other side in the previous step.

Assuming that the opponent, C, succeeded in replacing KA and KB by KC, let us examine his situation after Step 1 has been executed. He has, at his disposal, the first half of $E_{KC}(MA)$, but this is not enough to read MA. He must send B something, otherwise, the communication will be terminated and he will discover nothing about MA and MB. If he sends B the same half ciphertext he received from A, B will later try to decrypt it with the "wrong" key and will get garbage. C is thus forced to behave nontransparently and to invent a new message, MA' (which probably has nothing to do with MA), to encrypt it under KB, and to send the first half of $E_{KB}(MA')$ to B. Similarly, he is forced to replace the unknown MB by another message MB' in Step 2. By the time he discovers the true values of MA and MB in Steps 3 and 4, it is too late to change MA' and MB', since he is already committed to the first halves of their ciphertexts. Therefore, any attempt by C to read MA and MB will either garble or completely change the communication between A and B.

Instead of transmitting the two halves of the ciphertext separately as proposed here, other two-part methods could be used as long as the transmission of the first part effectively commits the sender to the final cleartext although the cleartext cannot be computed without the use of the second half as well. Furthermore, the first part must depend on the recipient's public key in such a manner that an enemy could not modify the first part so as to depend on his own public key instead. For example, the first part could be a "cryptographic checksum" or "one-way function" of the ciphertext, and the second part could be the ciphertext itself. However, more bits are exchanged by the parties in this variant than in the original interlock protocol.

4. GENERALIZATIONS

If A and B want to exchange n blocks of information, they can repeat the interlock protocol for each pair of blocks. A sophisticated opponent can try to use the following delaying technique:

1. During the first cycle through the protocol, C sends A and B dummy messages MB' and MA', and records the actual messages MB_1 and MA_1.
2. During the ith cycle, C sends A and B the properly translated versions of the ciphertexts of MB_{i-1} and MA_{i-1}, and records the new messages, MB_i and MA_i.
3. The last pair of blocks, MB_n and MA_n, are lost since A and B do not expect any more messages after the nth cycle.

While this mode of attack reduces the interference of C with the communication between A and B, it can be detected by the communicating parties since the messages they send and the messages they receive are out of phase: If A poses a question to B during cycle i, B receives it only during cycle $i + 1$, and the response he sends during cycle $i + 2$ is received by A only during cycle $i + 3$. Assuming that each message contains both a question and an answer to the previous question, it is easy to show that C cannot interleave his exchanges with A and B in a way that will look transparent to

both of them, and the delays he is forced to introduce are likely to arouse suspicion.

The interlock protocol requires that, at the beginning of each cycle, both A and B are ready to transmit meaningful messages (a "full-duplex" communication pattern). More common is a "half-duplex" mode in which the parties alternate in the transmission of messages, and where each message may depend upon the message just received from the other party. In this mode of operation, it is harder to expose an eavesdropper since a message sent by one party must become decipherable before any meaningful response is expected from the recipient of the message. The only way an eavesdropper can be detected in this case is by timing the delay between questions and answers. If a question takes t seconds to transmit (at a fixed Baud rate which cannot be changed by the eavesdropper), and if the eavesdropper can start translating it only when the transmission is complete, (e.g., when the ciphertext is superencrypted by a temporary key which is revealed at the end of the transmission), the the translation adds at least t seconds to the transmission delay. This extra delay can be detected if the question must be answered not sooner than s seconds and not later than $s + t$ seconds after it has been posed, for some fixed but arbitrary s.

One mode of operation in which the existence of an eavesdropper cannot be exposed is a one-way communication in which A wants to send B a message but does not expect (or cannot receive) any response. It is clear that without having authenticated information about A's key or exact transmission time, the translation of the ciphertext by C cannot be detected.

REFERENCES

1. Diffie, W. and Hellman, M. New directions in cryptography. *IEEE Trans. Inf. Theory*, (Nov. 1976).
2. Rivest, R., Shamir, A. and Adelman, L. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM 21*, (Feb. 1978).

CR Categories and Subject Descriptors: C.2.0 [**Computer-Communication Networks**]: General—*security and protection*
General Term: Security
Additional Key Words and Phrases: cryptography, cryptographic protocols

Received 4/82; revised 10/83; accepted 11/83

Author's Present Address: Ronald L. Rivest, Laboratory for Computer Science, Massachusetts Institute of Technology, 545 Technology Square, Cambridge, MA 02139; Adi Shamir, Applied Mathematics, The Weizmann Institute of Science, Rehovot, Israel.

Abstracts from Other ACM Publications

ACM Transactions on Programming Languages and Systems April Issue

Using Time Instead of Timeout for Fault-Tolerant Distributed Systems

Leslie Lamport

A general method is described for implementing a distributed system with any desired degree of fault-tolerance. Instead of relying upon explicit timeouts, processes execute a simple clock-driven algorithm. Reliable clock synchronization and a solution to the Byzantine generals problem are assumed.

For Correspondence: L. Lamport, Computer Science Laboratory, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025.

"Hoare Logic" of CSP, and All That

Leslie Lamport and Fred B. Schneider

Generalized Hoare Logic is a formal logical system for deriving invariance properties of programs. It provides a uniform way to describe a variety of methods for reasoning about concurrent programs, including noninterference, satisfaction, and cooperation proofs. We describe a simple meta-rule of the Generalized Hoare Logic—the Decomposition Principle—and show how all these methods can be derived using it.

For Correspondence: L. Lamport, Computer Science Laboratory, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025.

Communicating Sequential Processes for Centralized and Distributed Operating System Design

M. Elizabeth C. Hull and R. M. McKeag

How the notation of Communicating Sequential Processes may be used in the design of an operating system is demonstrated. Furthermore, how such an approach assists in the design and development of a system distributed over a network of computers is shown. The technique uses a well-defined design methodology.

For Correspondence: M.E.C. Hull, School of Computer Science, Ulster Polytechnic, Shore Road, Newtownabbey, Co Antrim BT37 0QB, North Ireland.

Global Data Flow Analysis Problems Arising in Locally Least-Cost Error Recovery

Roland Backhouse

Locally least-cost error recovery is a technique for recovering from syntax errors by editing the input string at the point of error detection. A scheme for its implementation is recursive descent parsers, which in principle embodies a process of passing a parameter to *each* procedure in the parser for *each* terminal symbol in the grammar, has been suggested. For this scheme to be practical it is vital that as much parameterization as possible is eliminated from the recursive descent parser. This optimization problem and how it may be split into three separate global data flow analysis problems—classifying terminal symbols and the so-called min and max follow cost problems—are discussed. The max follow cost problem is a particularly difficult one to solve. The application of Gaussian elimination to its solution is shown by expressing it as a continuous data flow problem, and is also related to an "idiosyncratic" data flow problem arising in the optimization of very high level languages. Classifying terminal symbols is also difficult since the problem is unsolvable in general. However, for the class of LL(1) grammars, the problem is shown to be expressible as a distributive data flow problem and so may be solved using, say, Gauss-Seidel iteration.

For Correspondence: R. Backhouse, Dept. of Computer Science, University of Essex, Wivenhoe Park, Colchester CO4 3SQ, U.K.