Reprinted from INFORMATION AND CONTROL Vol. 55, Nos. 1–3, October/November/December 1982
Printed in Belgium

On the Security of Ping-Pong Protocols

D. DOLEV*

Institute of Mathematics and Computer Science, Hebrew University, Jerusalem

S. EVEN†

Computer Science Department, Technion, Haifa, Israel

AND

R. M. KARP‡

Computer Science Division, EECS, University of California, Berkeley, Berkeley, California 94720

Consider the class of protocols, for two participants, in which the initiator applies a sequence of operators to a message M and sends it to the other participant; in each step, one of the participants applies a sequence of operators to the message received last, and sends it back. This "ping-pong" action continues several times, using sequences of operators as specified by the protocol. The set of operators may include public-key encryptions and decryptions. An $O(n^3)$ algorithm which determines the security of a given protocol (of length n) is presented. This is an improvement of the algorithm of Dolev and Yao (*IEEE Trans. Inform. Theory* **IT-30** (2) (1983), 198–208).

I. INTRODUCTION

The use of public-key encryption, (Diffie and Hellman, 1976; Rivest *et al.*, 1978) for secure network communication has received considerable attention. Such systems are effective against a "passive" eavesdropper, namely, one who merely taps the communication line and tries to decipher the intercepted message. However, as pointed out by Needham and Schroeder (1978), an improperly designed protocol can be vulnerable to "active" sabotage.

* Part of the research was done while the author visited IBM Research Lab, San Jose, California.

† Part of the research was done while the author visited the EECS Department, U.C. Berkeley. Supported by NSF Grants MCS 79-15762 and MCS 82-04506 and the Fund for the Promotion of Research at the Technion.

‡ Research supported by NSF Grant MCS 81-05217.

0019-9958/82 $2.00

The saboteur may be a legitimate user in the network. He can intercept and alter messages, impersonate other users or initiate instances of the protocol between himself and other users, in order to use their responses. It is possible that through such complex manipulations he can read messages, which are supposed to be protected, without cracking the cryptographic systems in use.

In view of this danger it is desirable to have a formal model for discussing security issues in a precise manner, and to investigate the existence of efficient algorithms for checking the security of protocols.

Dolev and Yao (1983) investigated the security of what we call here "ping-pong protocols." These protocols involve two participants, the sender S and the receiver R. Let M be a message generated by S. First, S applies a sequence of operators to M and sends it to R. Next, R applies a sequence of operators to the message received, and sends the result back to R. In each step, the participant applies a sequence of operators to the last message received, and sends it back. The number of times this is done, as well as the sequences of operators used, is defined by the protocol.

Dolev and Yao considered the security of two such families of protocols, assuming only few limitations on the behavior of the saboteur. Their second and more general family of protocols is extended here to allow more operators, and an $O(n^3)$ time algorithm for checking the security of protocols is presented. This improves the algorithm of Dolev and Yao, which is $O(n^8)$ time.

We briefly recall the essence of public-key systems (see Diffie and Hellman, 1976 or Rivest *et al.*, 1978 for more details). Every user X has an encryption function E_X and a decryption function D_X. Both are mappings from $\{0, 1\}^*$ into $\{0, 1\}^*$. There is a public directory containing all (X, E_X) pairs, while the decryption function D_X is known only to X. The main requirements on E_X, D_X are:

(1) $E_X D_X = D_X E_X = \lambda$, where λ is the identity function, and

(2) knowledge of $E_X(M)$ does not reveal anything about the value M.

Before we attempt any formal definitions of protocols or security let us consider several simple examples of protocols, and discuss informally their security.

EXAMPLE 1. Consider the following protocol:

(1) $(X, E_Y(M), Y)$,

(2) $(Y, E_X(M), X)$,

which simply means this: X wants to send M to Y and get an echo in order to verify that M has reached Y. He computes $E_Y(M)$, using Y's public

encryption key and sends via the network $(X, E_Y(M), Y)$, which stands for "X sends to Y the message $E_Y(M)$." Clearly, no one but Y can apply D_Y to $E_Y(M)$, in order to recover M. After doing so, Y computes $E_X(M)$ and sends $(Y, E_X(M), X)$. When X gets it he can compare the echo, $D_X E_X(M)$, with the original M in order to verify that M has indeed reached Y.

This innocent-looking protocol is insecure. A saboteur Z may intercept $(X, E_Y(M), Y)$ and replace it by $(Z, E_Y(M), Y)$. Y will get M, and respond, according to the protocol by sending $(Y, E_Z(M), Z)$. Z can now read M by applying his secret key, D_Z. He can then even produce the echo $(Y, E_X(M), X)$ and send it over to the satisfied and unsuspecting X. Clearly, this works only if M does not include information about the original sender's identity. Indeed, this observation leads to the technique of name-appending:

EXAMPLE 2. (1) $(X, E_Y(MX), Y)$,
(2) $(Y, E_X(M), X)$.

The word MX is formed by appending to M the name X. Now, after Y applies D_Y to $E_Y(MX)$ to get MX, he checks whether the suffix of the string matches the declared name of the sender, i.e., X. If it does not, he knows that someone has meddled with the message and simply terminates his participation in this instance of the protocol. Otherwise, he computes $E_X(M)$ and sends $(Y, E_X(M), X)$.

This protocol is indeed secure. A formal way to prove it will be shown in Section 3.

One may be led to believe that name-appending is the cure for all evils, but consider this seemingly "even safer" protocol:

EXAMPLE 3. (1) $(X, E_Y(E_Y(M)\, X), Y)$,
(2) $(Y, E_X(M), X)$.

Here the name X is appended to $E_Y(M)$ instead of to M itself. This protocol is insecure!

One can use the algorithm of Section 3 to verify that this protocol is insecure. There are two natural ways to crack this protocol. One method is as follows: Z sends $(Z, E_Y(E_Y(E_Y(M)\, X)), Y)$, receives $(Y, E_Z(E_Y(M)\, X), Z)$, sends $(Z, E_Y(E_Y(M)\, Z), Y)$ and receives $(Y, E_z(M), Z)$. Another method: Z sends $(Z, E_X(E_X(M)\, Z), X)$ and receives $(X, E_Z(M), Z)$.

In addition to the operator E_X and D_X, we have used in the last two examples two more operators, which we shall denote i_X and d_X. If X is a string (name of user X) and M is a string (message) then $i_X(M) = MX$. Let S be a string, $d_X(S)$ is defined as follows. If X is a suffix of S, i.e., $S = MX$,

then $d_X(S) = M$; else, $d_X(S)$ is undefined, which means that the participation in this instance of the protocol is terminated. Clearly,

$$d_X i_X = \lambda$$

but $i_X d_X(S)$ is not even defined (unless $S = MX$ for some M).

Let us also define an operator d, which is simply the removal of the appended user name. This is easy to do, if, for example, all names use exactly the same number of bits. Therefore, it is natural to assume that a saboteur can perform d. Thus, for every user name X,

$$di_X = \lambda$$

but again $i_x d(S) = S$ only if $S = MX$.

In general we shall assume that there is a set of operators Σ which can be used by the participants in the network. Some operators may have a user name subscript (such as E_X, D_X, i_X, and d_X in our examples). The subset of operators, which user X can perform will be denoted by Σ_X and will be called X's *vocabulary*. The vocabularies of all users are similar in the sense that if one replaces the index X by Y, and Y by X, in Σ_X, the result is Σ_Y.

Also, there will be a given set of *cancellation rules* of the form $\sigma\tau = \lambda$, where σ and τ are elements of Σ. If both σ and τ are indexed then the indices are the same. The cancellation rules are similar for all users. Thus, if one or both operators are indexed then the same cancellation rule holds for every user name index.

In our examples,

$$\Sigma = \{d\} \cup \{E_X, D_X, i_X, d_X \mid X \text{ is a user name}\},$$

$$\Sigma_X = \{d, D_X\} \cup \{E_Y, i_Y, d_Y \mid Y \text{ is a user name}\},$$

and the cancellation rules are

$$E_X D_X = \lambda,$$

$$D_X E_X = \lambda,$$

$$d_X i_X = \lambda,$$

and

$$di_X = \lambda.$$

Note that if $a, b, c \in \Sigma$, $ab = \lambda$ and $bc = \lambda$ then $a = c$. This follows from the fact that members of Σ are operators: Let $w \in \{0, 1\}^*$. $abc(w) = a(bc(w)) = a(w)$, since $bc = \lambda$, but on the other hand, $abc(w) = ab(c(w)) = c(w)$. Thus $a = c$.

Given a string $\alpha \in \Sigma^*$, one may repeatedly apply cancellation rules until

no cancellation rule is applicable any more. By the previous paragraph, the reduction process has the Church–Rosser property (Rosen, 1973), and thus the end result is unique. Let us denote this *reduced form* of α by $\bar{\alpha}$.

An underlying assumption in our analysis is that the set Σ is free from any relations other than those implied by the cancellation rules. That is, two strings of operators, α and β are equivalent if and only if both have the same reduced form.

II. Ping-Pong Protocols and Security

Definition. A ping-pong protocol $P(S, R)$ is a sequence $\Gamma = (\alpha_1, \alpha_2, ..., \alpha_l)$ of operator-words, such that if i is odd then $\alpha_i \in \Sigma_S^*$ and if it is even then $\alpha_i \in \Sigma_R^*$.

The structure of the protocol is similar for every ordered pair of (different) users. Thus, if in $P(V, W)$, we replace the index V by X, and W by Y, we get $P(X, Y)$. We assume that for every two users X and Y, $P(X, Y)$ may be initiated, i.e., there are no restrictions, imposed by the network or the users, on communication via P.

In Example 1, $\alpha_1[S, R] = E_R$, $\alpha_2[S, R] = E_S D_R$ and $l = 2$. In Example 2, $\alpha_1[S, R] = E_R i_S$, $\alpha_2[S, R] = E_S d_S D_R$ and again $l = 2$. In both examples, $\alpha_1(M)$ is sent by S to R and $\alpha_2\alpha_1(M)$ is sent by R, back to S.

In general the interpretation is as follows: S invents a message-word $M \in \{0, 1\}^*$. He applies α_1 to it and sends it to R; i.e., the first step is $(S, \alpha_1(M), R)$. Next $(R, \alpha_2\alpha_1(M), S)$, etc. If l is odd the last step is $(S, \alpha_l\alpha_{l-1} \cdots \alpha_1(M), R)$ and if l is even, then it is $(R, \alpha_l\alpha_{l-1} \cdots \alpha_1(M), S)$.

In this paper, we are not concerned with the purpose of using P. Instead, we are interested only in the question of whether a saboteur (or a group of them) can extract M.

Thus, we assume that some user S has invented a message M, chosen a user R and initiated $P(S, R)$ on M. We assume that neither S, nor R, is a saboteur. We have to define what are the actions which the saboteur(s) can take.

We shall assume that for every $1 \leqslant i \leqslant l$, for every two different users X and Y and for every $W \in \{0, 1\}^*$ the saboteur can effect $\alpha_i[X, Y]$ on W. We shall explain, shortly, why we make this assumption, but if one believes that this is too conservative one may restrict the saboteur actions, and as long as these restrictions are symmetric (not username dependent), an $O(n^3)$ algorithm for checking security still exists. For example, one could assume that a saboteur cannot effect $\alpha_1[S, R]$ if he is not S, or that he cannot effect $\alpha_i[S, R]$, $1 \leqslant i \leqslant l$, if he is not S.

Let us denote the saboteur by Z. If $X = Z$, then Z has no difficulty to get $\alpha_1(W)$, since $\alpha_1 \in \Sigma_Z^*$. If $X \neq Z$ (and X is not one of the collaborating

saboteurs) Z may be able to convince X to initiate $P(X, Y)$ on W. By tapping the message $(X, \alpha_1(W), Y)$, Z will get $\alpha_1(W)$.

In order to effect $\alpha_i[X, Y]$ on W, for $1 < i \leqslant l$, Z can wait for $P(X, Y)$ to occur (or somehow convince X to initiate it), wait for the $(i-1)$th message, $(X, \alpha_{i-1}\alpha_{i-2} \cdots \alpha_1(M), Y)$—assuming i is even, intercept it and replace it with (X, W, Y). Now, Y responds with $(Y, \alpha_i(W), X)$, as expected of him (assuming $\alpha_i(W)$ is defined) and sends it through the network, where Z can tap it. It follows that the language of operator-words which a single saboteur Z can effect (on any $W \in \{0, 1\}^*$) is

$$\Delta = [\Sigma_Z \cup \{\alpha_i[X, Y] \mid 1 \leqslant i \leqslant l, X \text{ and } Y \text{ are different users}\}]^*.$$

DEFINITION. Let $\alpha_1[S, R]$ be the first operator-word of $P(S, R)$ and $Z \notin \{S, R\}$. P is *insecure* if there exists an operator-word $\gamma \in \Delta$ such that $\overline{\gamma\alpha_1} = \lambda$.

Observe that it is not necessary to consider $\alpha_i\alpha_{i-1} \cdots \alpha_1(M)$, for $1 < i \leqslant l$, which is also heard over the network. For if a $\gamma \in \Delta$ exists which satisfies $\overline{\gamma\alpha_i\alpha_{i-1} \cdots \alpha_1} = \lambda$ then there is a $\gamma' \in \Delta$ (in fact $\gamma' = \gamma\alpha_i\alpha_{i-1} \cdots \alpha_2$ will do) for which $\overline{\gamma'\alpha_1} = \lambda$.

In the definition of security given above, we called P insecure if for some ordered pair of users (not including saboteurs), a $\gamma \in \Delta$ exists for which $\overline{\gamma\alpha_1} = \lambda$. In fact, such a γ exists for one pair (S, R) if and only if it exists for every set of users. This follows immediately from the fact that change of names of users does not change the pattern of cancellations. Thus, in what follows we shall restrict our attention to a fixed pair of users, (S, R), free of saboteurs, and only consider the question of whether for $\alpha_1[S, R]$ a $\gamma \in \Delta$ exists which satisfies $\overline{\gamma\alpha_1} = \lambda$.

One may wonder why we have defined Δ to include Σ_Z, but have not allowed a set of saboteurs $\{Z_1, Z_2, \ldots, Z_m\}$ and put $\bigcup_{i=1}^{m} \Sigma_{Z_i}$ in Δ instead. Let us show that this is not necessary, since whatever a set of saboteurs can do, a single saboteur can do also.

Assume $\gamma = \delta_1\beta_1\delta_2\beta_2 \cdots \beta_k\delta_{k+1}$, where for every $1 \leqslant j \leqslant k+1$,

$$\delta_j \in \left[\bigcup_{p=1}^{m} \Sigma_{Z_p}\right]^*$$

and for every $1 \leqslant j \leqslant k$, β_j is some $\alpha_i[X, Y]$, where $X \neq Y$ and

$$\overline{\gamma \cdot \alpha_1[S, R]} = \lambda.$$

We may assume that if $\beta_j = \alpha_i[X, Y]$ then the performer of α_i (X for odd i, Y for even i) is not a saboteur, for otherwise $\alpha_i \in [\bigcup \Sigma_{Z_p}]^*$ and there is no need to single out β_j, which can be absorbed in δ_j.

If we now replace saboteur Z_p by Z, for all p, in γ, the cancellation will

still occur, and all α_i's used (for β_j's) will still be legitimate, since its two users will be different. Thus, if a γ exists for a set of saboteurs, it exists for one.

Our next goal is to restrict Δ even further, in order to simplify the security decision problem. Let us show that if a $\gamma \in \Delta$ exists, for which $\overline{\gamma \cdot \alpha_1[S, R]} = \lambda$, then the same statement holds for

$$\Delta' = \{\Sigma_Z \cup \{\alpha_i[X, Y] \mid 1 \leqslant i \leqslant l, X \neq Y \text{ and } \{X, Y\} \subset R, S, Z\}\}^*.$$

If we replace each user $U \notin \{R, S\}$ who appears in γ by Z, the cancellation pattern is maintained while each α_i either remains legitimate (with two different users X, Y, $\{X, Y\} \subset \{R, S, Z\}$) or becomes an operator-word in Σ_Z^*. This proves that we can replace Δ by Δ' in the security decision problem.

III. An Algorithm for Checking Protocol Security

Construct a nondeterministic finite state automaton A, as follows:

(1) State 0 is the (unique) initial state and state 1 is the (unique) accepting state. The (input) alphabet is $\Sigma = \Sigma_Z \cup \Sigma_S \cup \Sigma_R$.

(2) There is a directed path from state 0 to state 1 whose (input) labels correspond to $\alpha_1[S, R]$.

(3) For every input letter (operator) $\sigma \in \Sigma_Z$, there is a self-loop from 0 to 0, labelled σ.

(4) For every $\alpha_i[X, Y]$, $1 \leqslant i \leqslant l$ and $\{X, Y\} \subset \{R, S, Z\}$ there is a loop from 0 to 0 whose edges are labelled, in sequence, by the letters of α_i.

Consider the protocol of Example 2. We have seen that $\alpha_1 \ [X, Y] = E_Y i_X$ and $\alpha_2[X, Y] = E_X d_X D_Y$. Thus, the automaton A is as shown in Fig. 1. The self-loop labeled $\sigma \in \Sigma_Z$ represents 11 self-loops, i.e., one for each member of Σ_Z, where

$$\Sigma_Z = \{E_R, E_S, E_Z, D_Z, i_R, i_S, i_Z, d_R, d_S, d_Z, d\}.$$

The security question, therefore, translates into the following: Is there no accepting path, in A, whose corresponding input word w satisfies $\bar{w} = \lambda$? The protocol is secure if and only if no such collapsing word is accepted by A.

In our example, the loops whose intermediate states are 11 to 20, are all superfluous, since they correspond to words in Σ_Z^*. Thus, A can be simplified, as shown in Fig. 2.

Let us assume that the (simplified) automaton A has been constructed and that its set of states is $S = \{0, 1, ..., s\}$. We say that a directed path, p, in A

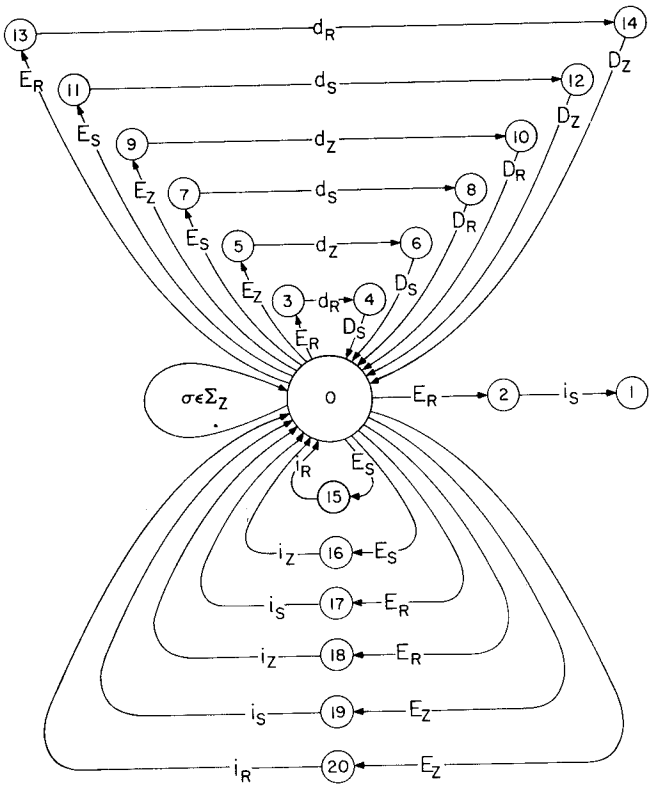

FIGURE 1

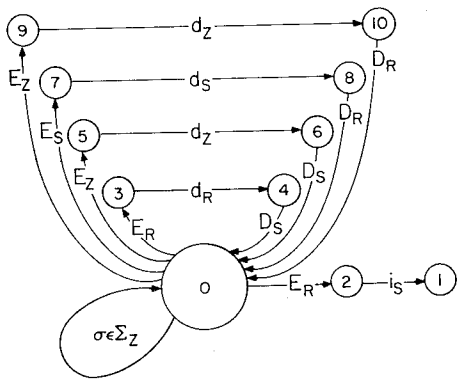

FIGURE 2

collapses, if its corresponding word w collapses, i.e., $\bar{w} = \lambda$. Define the *collapsing relation* $C \subseteq S \times S$ as follows: $(i,j) \in C$ if there is a directed path from i to j, in A, which collapses. The security question is therefore reduced to the question of whether $(0, 1) \in C$. The protocol P is secure if and only if $(0, 1) \notin C$.

In what follows, $i \to^{\sigma} j$ stands for an edge from state i to state j, labelled σ. Q is a queue of pairs of states. Our algorithm for constructing C is as follows:

(0) $C \leftarrow \{(i, i) \mid 0 \leqslant i \leqslant s\}$, $Q \leftarrow C$. [*Comment*: Each new pair of C enters Q once]

while $Q \neq \emptyset$, *do*

(1) Delete the first pair, (i,j), from Q.

(2) *If* $(j, k) \in C$ and $(i, k) \notin C$ *then* put (i, k) in C and in Q.

(3) *If* $(k, i) \in C$ and $(k,j) \notin C$ *then* put (k,j) in C in Q.

(4) *If* $k \to^{\sigma} i$ and $j \to^{\tau} l$ and $\sigma\tau = \lambda$ [is one of the cancellation rules] and $(k, l) \notin C$ *then* put (k, l) in C and in Q. *od*

The algorithm terminates, since there can be at most $(s + 1)^2$ pairs in C and each can cause the loop to occur one; the number of operations in each pass of steps (1)–(4) is easily seen to be finite. We shall shortly examine the time complexity questions more closely.

THEOREM 1. *The algorithm generates the collapsing relation C of automaton A.*

Proof. For every $(i,j) \in C$ let $l(i,j)$ be the length of a shortest collapsing path from i to j. It is easy to see that each (i,j) which is put in C by the algorithm belongs there. We prove that if $(i,j) \in C$ then the algorithm will put it into C, by induction on $l(i,j)$. If $l(i,j) = 0$ (i.e., $i = j$) then (i,j) is put in C in step (0).

Assume now that all $(i,j) \in C$ for which $l(i,j) < L$ have been put in C; let us prove that if $l(i,j) = L$ then it is put in C also. Let w be the word which corresponds to some shortest collapsing path for (i,j). Let σ be the first letter of w. Eventually, in the process of collapsing w, σ is cancelled with some τ via a cancellation rule $\sigma\tau = \lambda$. Thus, $w = \sigma w_1 \tau w_2$.

If $w_2 = \lambda$, then for $\sigma\tau = \lambda$ to happen, w_1 must vanish first. Thus, $\bar{w}_1 = \lambda$. Now if w_1 corresponds to a (p, q) path, since $l(p, q) = L - 2$, by the inductive hypothesis (p, q) has been put in C and in Q. When it leaves Q, the pair (i,j) is discovered via step (4), if it has not been generated earlier.

If $w_2 \neq \lambda$ then both $\overline{\sigma w_1 \tau} = \lambda$ and $\bar{w}_2 = \lambda$. Let k be the state on the path between $\sigma w_1 \tau$ and w_2. Clearly both (i, k) and (k,j) are in C, and since $l(i, k) < L$ and $l(k,j) < L$, by the inductive hypothesis both have been put in

C and Q. When the last of them leaves Q, either through step (2) or step (3), (i,j) will be generated and put in C, if it has not been put in C earlier.

Q.E.D.

If we apply the algorithm to the automaton of Example 2 (Fig. 2), the final matrix, describing C, is as follows:

	0	1	2	3	4	5	6	7	8	9	10
0	1					1				1	
1		1									
2			1								
3	1			1		1				1	
4	1	1			1	1		1		1	
5	1					1				1	
6	1	1				1	1	1		1	
7	1	1				1		1		1	
8	1		1	1		1			1	1	
9	1					1				1	
10	1		1	1		1				1	1

Since the (0, 1) cell is empty (i.e., $(0, 1) \notin C$), the protocol of Example 2 is secure.

In the complexity analysis which follows, we assume the RAM model, and that the basic word-length is sufficient to accommodate all the operators. Thus, the test of whether $\sigma\tau = \lambda$ takes constant time.

THEOREM 2. *The time-complexity of the algorithm for constructing the collapsing relation of automaton A (of $s+1$ states) is $O(s^3 + s|\Sigma_Z|)$.*

Proof. Note that for all states $v \neq 0$, the in-degree, $d_{\text{in}}(v)$, is exactly 1, and the out degree, $d_{\text{out}}(v)$, is at most 1. For state 0, both $d_{\text{in}}(0)$ and $d_{\text{out}}(0)$ are bounded by $s + |\Sigma_Z|$. Now consider the loop (1)–(4) of the algorithm.

If $i \neq 0$, $j \neq 0$, then step (2) takes at most s steps, since all we have to do is compare the ith row of the matrix describing the current, C, with the jth row. A similar observation holds for step (3), using columns. For step (4) there is only one σ and one τ to check. Thus, in this case the loop takes time $O(s)$, and since the number of such pairs in C is bounded by s^2, the total time spent on such pairs is $O(s^3)$.

If $i = 0$ but $j \neq 0$, then steps (2) and (3) are still $O(s)$ time, while step (4) is $O(|\Sigma_Z| + s)$ since we have to check each incoming edge $k \to^{\sigma} 0$ against the $j \to^{\tau} l$ edge (assuming $j \neq 1$) to see if $\sigma\tau = \lambda$, and the number of incoming

edges ($k \to^{\sigma} 0$) is bounded by $|\Sigma_Z| + s$. Since the number of such pairs is s, the total time spent on such pairs is $O(s(|\Sigma_Z| + s))$. The case of $j = 0$ but $i \neq 0$ is similar.

Finally, if both $i = 0$ and $j = 0$, steps (2) and (3) are redundant, while step (4) takes $O(s(|\Sigma_Z| + s))$ time, since each incoming edge $i \to^{\sigma} 0$ $(i \neq 0)$ has to be checked against each of the $d_{\text{out}}(0)$ $(\leqslant |\Sigma_Z| + s)$ outgoing edges, and each $0 \to^{\tau} j$ $(j \neq 0)$ has to be checked against each of the $d_{\text{in}}(0)$ $(\leqslant |\Sigma_Z| + s)$ incoming edges, but there is no need to check a self-loop against a self-loop. Thus, the time spent in this case is also $O(s(|\Sigma_Z| + s))$. Q.E.D.

Let us denote by n the *length of the protocol P*, which is measured as

$$n = \sum_{i=1}^{l} |\alpha_i|,$$

where $|\alpha_i|$ is the length of the operator-word α_i. Thus, n is the total number of operators used in P. Since each word $\alpha_i[X, Y]$ generates exactly 6 loops in the automaton A (one for each choice of an ordered pair of users (X, Y) out of the set $\{S, R, Z\}$,) the number of states s of A is $O(n)$, while the number of self-loops is $|\Sigma_Z|$. If the operators (and cancellation rules) are fixed and are not part of the input of the security problem, then $|\Sigma_Z|$ and the table of cancellation rules is of constant size.

Thus, Theorem 2 implies, immediately, the following corollary:

COROLLARY 1. *For fixed vocabulary and cancellation rules, there exists a security checking algorithm of ping-pong protocols (of two users). Its time-complexity is $O(n^3)$, where n is the length of the protocol.*

In fact, one may allow the definition of the generic vocabulary and cancellation rules to be part of the input, and still maintain the $O(n^3)$ bound on the time-complexity. One only needs to incorporate the preparation of the cancellation rules in form of a table into the algorithm (in time $O(n^2)$). Thus,

COROLLARY 2. *For ping-pong protocols of two users there exists a security checking algorithm whose input is the generic cancellation rules and the protocol. Its time-complexity is $O(n^3)$, where n is the length of the input.*

EPILOGUE

Essentially, the problem we have solved in Section III is that of checking whether the intersection of a regular language and a certain context-free language is non-empty. Classically, if one is given a context-free language L, by a grammar G in *CNF*, and a regular language R, by a nondeterministic

automaton A, one constructs a new grammar G' which defines $L \cap R$, and then one can check in linear-time whether G' defines the empty language. If the description of G is of length m and A is of n states then G' comes out of size $O(n^3m)$. Thus, this leads to an $O(n^3)$-time, $O(n^3)$-space algorithm to solve the security problem, while our solution is $O(n^3)$-time, $O(n^2)$-space. In fact, our algorithm can be generalized to answer the question of whether $L \cap R$ is empty, in $O(n^3m)$-time. $O(n^2m)$-space.

Another issue is that of protocols for $k > 2$ users. If one assumes that for $P(U_1, U_2, ..., U_k)$ the saboteur can effect every α_i for k users, not necessarily distinct, then one saboteur is as powerful as many, and an $O(n^3)$ security checking algorithm similar to the one shown in Section III follows. However, it is natural to assume that this is not the case, since the user who is supposed to perform α_i, observing that not all k users are distinct, will become suspicious and will not cooperate.

Even and Goldreich have recently shown that there is an $O(k)$ bound on the number of "useful" saboteurs. Thus, for a fixed k an $O(n^3)$ security checking algorithm exists. However, if the number of users of P is part of the problem's intput this observation is not useful since the straightforward extension of the algorithm leads to an exponential blow up.

The problem of testing the security of protocols which are not of the ping-pong type remains wide open.

ACKNOWLEDGMENTS

The authors would like to thank Oded Goldreich and Michael A. Harrison for helpful discussions.

REFERENCES

DIFFIE, W. AND HELLMAN, M. E. (1976), New directions in cryptography, *IEEE Trans. Inform. Theory* **IT-22** (6), 644–654.

RIVEST, R. L., SHAMIR, A., AND ADLEMAN, L. (1978), A method for obtaining digital signatures and public-key cryptosystems, *Comm. ACM* **21** (2), 120–126.

NEEDHAM, R. M. AND SCHROEDER, M. D. (1978), Using encryption for authentication in large networks of computers, *Comm. ACM* **21** (12), 993–999.

DOLEV, D. AND YAO, A. C. (1983), On the security of public key protocols, *IEEE Trans. Inform. Theory* **IT-30** (2), 198–208.

ROSEN, B. K. (1973), Tree-manipulating systems and church–rosser theorems, *J. Assoc. Comput. Mach.* **20** (1), 160–187.

Printed by the St. Catherine Press Ltd., Tempelhof 41, Bruges, Belgium