An Algorithm to Compute Collusion Paths

Steven H. Low

Department of EEE
University of Melbourne
VIC 3052 Australia
slow@ee.mu.oz.au

Nicholas F. Maxemchuk

AT&T Research
Murray Hill
NJ 07974 USA
nfm@research.att.com

Abstract

In earlier work we have formulated a collusion problem that determines whether it is possible for a set of colluders to collectively discover a target set of information, starting from their initial knowledge, and have presented a complete solution for a special case of the problem. In this paper we present an algorithm that solves the general case. Given a collusion problem the algorithm determines whether it has a solution, and if it does, computes one. A solution to the collusion problem is a method with which the colluders can uncover the hidden information. Communications protocols that employ cryptographic techniques are increasingly used to protect privacy as well as to communicate. A cryptographic protocol defines a process by which information is transferred among some users while hidden from others. The algorithm presented here can be used to determine whether a subset of protocol users can discover, during or after the protocol's execution, the information that is designed to be hidden from them.

1 Introduction

Complex communications protocols employing cryptographic building blocks are being developed not only to communicate, but also to protect privacy, e.g., in broadband networks [11, 10], in mobile networks [3], in electronic commerce [2, 5, 6], and in health insurance systems [9]. A cryptographic protocol defines a process by which some information is transferred among some users and hidden from others. The collusion problem determines whether it is possible for a subset of users to discover, through collusion, the information that is to be hidden from them after or during a protocol's execution. A solution to the problem is a specific way with which the colluders can uncover the hidden information.

It is not always possible for two users to collude. In order to collude they must share a common, unique piece of information pertaining to the protocol run. This may be necessary because, in practice, the sender and the receiver, such as banks in the anonymous credit card context [5, 6], may exchange a large number of messages corresponding to different protocol runs within a short time interval. It may not be sufficient, for example, for two banks to know that both have been involved in some transaction, because they may have been involved in a great many transactions within a short time interval. In order to combine their knowledge pertaining to a particular protocol run, they need to share a unique link pertaining to that run. There are two possible such links: 1) a unique message that is exchanged during that protocol run, or 2) a unique piece of data, e.g, two banks may have the unique social security number of a customer and hence can combine their knowledge about the customer. We stress however that our formulation includes as a special case in which two users can collude as long as one knows the other, even if they share no unique information.

The collusion problem is motivated by our earlier work [5] and [9] in which we designed multiparty communications protocols to protect privacy in credit card and health-care systems. In the credit card system, for instance, the protocol uses standard cryptographic techniques to hide different pieces of transaction information from different parties involved in the transaction so that at the end of a credit card transaction, no single party except the cardholder can associate the cardholder's identity with where or what she purchases. Moreover it takes many parties to collude in order to compromise the cardholder's privacy.

0-8186-7780-5/97 $10.00 © 1997 IEEE

In [8, 4] we first introduced a formal model for cryptographic protocols and their collusion analysis. The collusion problem is completely solved for the special case where two users can collude only if they have exchanged a message during the protocol's execution. An algorithm is presented in [8], and proved in [4], that determines whether a collusion problem has a solution and if so, computes one. In [7] we provide an algorithm that determines the existence of a solution for the general case where collusion can occur on unique data as well as on unique messages. In this paper we refine the algorithm of [7] so that it not only determines the existence of a solution, but also computes one when it exists.

We present our model and problem statement in §2. Roughly, the collusion problem determines whether it is possible for a subset of users to discover a target set of information, through a sequence of message exchanges among the colluders that satisfy the above collusion prerequisites. A solution to the problem, called a *collusion path*, is a specific sequence of message exchanges that uncovers the target set of information. The problem is formulated as a reachability analysis on a large transition system, exhaustive search on which is infeasible. It turns out, however, that the existence of a solution to the collusion problem, as well as the construction of a collusion path, can be determined by examining just the initial knowledge of all the colluders. This eliminates the need to explore the large transition system.

In §3 we review the special case where collusion is allowed only between two colluders sharing unique messages. We present a closed form expression specifying the condition under which a solution exists and clarify the simple structure of collusion paths.

The solution to the special case illustrates the structure of the problem and leads to the solution for the general case when collusion is allowed between colluders sharing unique data as well. This is explained in §4. We present an algorithm that determines whether a collusion problem has a solution and if it does, computes one. The algorithm includes the solution for the special case as its first step.

2 Model and Problem Formulation

In this section we present our model and formulate the collusion problem.

2.1 Notation

We use $(y_j, j \in J)$ to denote a vector with components y_j, j spanning the index set J; the jth component y_j is sometimes denoted $y.j$. For any set A, $|A|$ denotes the number of elements in A, 2^A denotes the collection of subsets of A, and A^* denotes its Kleene closure.

If d is a piece of data and k is a cryptographic key, then $k(d)$ denotes the encryption of d with k. A string $k_n \cdots k_1$ represents successive application of keys $k_1, \ldots, k_n$ in order. We use ϵ to denote the identity key: for any piece of data d, $\epsilon(d) = d$.

For any key k, k^{-1} denotes its inverse with the cancellation rule $k^{-1}\, k = k\, k^{-1} = \epsilon$. For example, The keys k and k^{-1} are identical in secret-key cryptosystems, but not in public-key cryptosystems. When we refer to a string γ of keys, we always assume that γ is in reduced form that cannot be further simplified by application of the cancellation rule.

A *transition system* is a triple $\Theta = (Q, \Sigma, \delta)$, where Q is a set of states, Σ is a finite set of transition labels, and $\delta : Q \times \Sigma \to Q$ is a transition (partial) function. For example, a finite state machine is a special transition system in which Q is finite. A *path* is a sequence of transitions. A concatenation of two paths ρ_1 and ρ_2 is denoted $\rho_1 \cdot \rho_2$.

2.2 Collusion Model

Consider a set of users we call colluders. Each colluder initially has a set of knowledge that includes a subset of messages that have been exchanged during a protocol run, the identity of some protocol users that have been involved in that protocol run, a set of data, possibly multiply encrypted, and a set of cryptographic keys. The colluders obtain this set of initial knowledge possibly by participating in the protocol run, or stealing messages over the communication channel; it is not important for our purpose how they obtain their initial knowledge.

The colluders' objective is to collectively discover a certain set of information, e.g., a cardholder's identity and what she purchases in the context of anonymous credit cards in [6]. A colluder first finds another colluder with which it can collude and then sends it its complete knowledge, including cryptographic keys in its possession. The recipient attempts to decrypt the combined knowledge with the keys it now has, and the process continues. We are interested in whether

a given set of colluders can discover a target set of information. We now make these notions precise.

Collusion is carried out in an *environment* described by the 5-tuple (U_p, D, K, U_c, L) where

1. U_p is a finite set of protocol users;
2. D is a finite set of data;
3. K is a finite set of cryptographic keys, including the identity key ϵ;
4. $U_c \subseteq U_p$ is a set of colluders;
5. $L \subseteq U_p \cup D \cup K$ is a set of information that determines whether two colluders can collude; see condition (2) below.

Define the *information set* as the set of every possible encryption and clear text combination of every piece of information in the system:

$$I \;:=\; K^*(U_p \cup D \cup K)$$

For example, if $d \in U_p \cup D \cup K$ and $k_i \in K$, then d, $k_1(d)$, $k_2k_1(d)$, $k_1k_2k_1(d)$ are all in I.

Decryption is a function $\Delta : 2^I \to 2^I$ that is defined by the cancellation rule $k^{-1}\, k = k\, k^{-1} = \epsilon$, the identity key. For instance $\Delta(\{k(d)\} \cup \{k^{-1}\}) = \{d, k^{-1}\}$. $\Delta(A)$ represents the decryption of a set $A \subseteq I$ of information by the keys included in A such that if $k_n \cdots k_1(d) \in \Delta(A)$ then $k_n^{-1} \notin \Delta(A)$. Whenever we refer to a subset of I we always assume that it is in this reduced form.

The *knowledge set* is the combination of the messages and information:

$$W := 2^N \times 2^I$$

where N is the set of unique message identifiers and I is the information set. An element $w = (w.N, w.I)$ of W represents a user's knowledge. It has two components: the first component $w.N \subseteq N$ represents all the messages the user has seen, and the second component $w.I \subseteq I$ represents all the information the user knows. As noted above $w.I$ is in reduced form. User u's knowledge is denoted $w_u \in W$. We naturally assume that $u \in w_u.I$ for all $u \in U_p$.

As colluders in U_c collude by exchanging messages, their knowledge is modified. This evolution is modeled by a transition system $\Theta = (W^{|U_c|}, \Sigma, \delta)$. Here, a state $w = (w_u, u \in U_c)$ in $W^{|U_c|}$ is the knowledge of all colluders. An event $\sigma = (s, r)$ in $\Sigma := U_c \times U_c$ describes the transfer of the sender's complete knowledge w_s to the receiver r to attempt to extract the hidden information at the receiver. The transition function δ describes the transformation of colluders' knowledge as a result of the message exchange.

For each state $w = (w_u, u \in U_c)$ and event $\sigma = (s, r)$, the next state $\delta(w, \sigma)$ is defined if and only if $r \in w_s.I$ and *at least* one of the following conditions is satisfied:

$$w_s.N \cap w_r.N \;\neq\; \phi \qquad (1)$$

$$w_s.I \cap w_r.I \cap L \;\neq\; \phi. \qquad (2)$$

The conditions, motivated in §1, says that for s and r to collude, s must know r and they must either share a unique message (condition (1)) or a unique piece of data in L (condition 2)).

Note that if $L = U_c$ then since $u \in w_u$ for all u, $r \in w_s.I$ implies condition (2). Then the collusion prerequisites reduce to the special case in which collusion is allowed as long as the sender s knows the receiver r.

We call an event $\sigma = (s, r)$ *enabled in state* w if the next state $\delta(w, \sigma)$ is defined; we often say that σ is *enabled* when the state from which the transition is made is understood. A path is *valid* if every transition on the path is enabled.

When the current state is $w = (w_u, u \in U_c)$ and an enabled transition $\sigma = (s, r)$ is made, the next state $w' := \delta(w, \sigma)$ is:

$$\begin{aligned} w'_y &= w_y \quad \text{if } y \neq r \\ w'_r.N &= w_r.N \cup w_s.N \\ w'_r.I &= \Delta(w_r.I \cup w_s.I) \end{aligned}$$

i.e., the receiver's knowledge is expanded to include that of the sender.

Note that the set of users that can collude can increase as users collude and information is combined. For instance, a sender can have encrypted information that includes the identity of a user and a unique piece of data that the sender and that user shares, and a receiver may have the key to decrypt that information. After the sender transfers its information to the receiver, the receiver can collude with the user that was hidden in the encrypted information.

We summarize our model in the following definition. The transition system Θ describes all the possible sequences of message exchanges among the col-

luders and how their knowledge evolves as collusion proceeds.

Definition 1 *Given an environment* (U_p, D, K, U_c, L), *a collusion system is the (unique) transition system* $\Theta = (W^{|U_c|}, \Sigma, \delta)$ *defined above.*

2.3 Problem formulation

The collusion problem is to determine if a subset of users can combine their information, by passing messages, and extract the hidden information after or during a protocol's execution. Suppose we have a collusion system $\Theta = (W^{|U_c|}, \Sigma, \delta)$.

Collusion problem

Given an initial state $w(0) \in W^{|U_c|}$ and a target set of unencrypted information $T \subseteq U_p \cup D \cup K$, does there exist a valid path ρ in Θ that starts in $w(0)$ and terminates in a state $w(\rho)$ in which a colluder $c \in U_c$ knows T, i.e., $w_c(\rho).I \supseteq T$?

We call the valid path ρ in the definition of the collusion problem a *collusion path.*

Though the collusion problem is simply a reachability analysis on the state machine Θ, given $w(0)$, the reachable set contains up to $2^{|U_c|(|U_c|-1)}$ states. For example, for the simple protocol analysed in [6], $|U_c| = 8$ and the reachable set contains up to 10^{17} states. It is hence impractical to do an exhaustive search. In the next two sections we present a solution that avoids searching on Θ.

For the rest of this paper, we make the following natural assumptions on the initial state $w(0)$ of the collusion problem. We assume that in state $w(0)$, if (u, v) is enabled because u and v share a unique message (condition (1)), then they must know each other in $w(0)$ and hence (v, u) must also be enabled, i.e., $w(0)$ satisfies the condition:

$$\begin{aligned} & w_u(0).N \cap w_v(0).N \neq \phi \\ \Rightarrow \quad & v \in w_u(0).I \text{ and } u \in w_v(0).I \end{aligned} \tag{3}$$

3 Special case: $L = \phi$

In this section we consider the special case in which $L = \phi$, i.e., two users can collude only if they share a unique message that was exchanged during the protocol run (condition (1)). We present an algorithm that completely solves this special case. Given a collusion problem it determines whether it has a solution and if it does, computes one. The algorithm is a minor adaptation of the one derived in [8]. The adaptation makes it useful for solving the general case, presented in the next section. Results of this section are proved in [4].

For the rest of this paper, fix an environment (U_p, D, K, U_c, L), an initial state $w(0)$ and a target information set T. In this section $L = \phi$ so that condition (2) is always violated. It is shown in [8, 4] that, to solve the collusion problem, it is necessary and sufficient to find a set of colluders who have exchanged messages among themselves during protocol execution and whose *initial* information in state $w(0)$, when combined, contains T.

Define $F = (U_c, E(w(0)))$ as a graph, depending on the initial state $w(0)$, that describes all the events that are initially enabled by condition (1). F contains all colluders as its nodes. There is an edge (u, v) in $E(w(0))$ if and only if (u, v) is enabled in the initial state $w(0)$. According to assumption (3), (u, v) is enabled in state $w(0)$ if and only if (v, u) is also enabled. Hence we can take F to be an undirected graph with each edge representing two directed edges in opposite directions.

Results of [8, 4] suggest Algorithm 1 below to solve the collusion problem. It first constructs the graph F. Then it finds each connected component $F' = (V, E)$ of F using breadth-first search while, at the same time, constructing a candidate collusion path with the special structure described by [8, Theorem 3]. The candidate path has visited every node in the connected component when the search on the connected component is complete. Hence the last recipient on the path knows the combined knowledge $\Delta(\cup_{u \in V} w_u(0).I)$ of every colluder in the connected component. This combined knowledge is then checked to see if it contains the target information set T. A collusion path is found if it does; otherwise the search is repeated on a different connected component of F. When all connected components have been searched without producing a collusion path, it is proved in [4] that none exists.

The algorithm maintains several data structures. The adjacent lists *Adj*[v] represent the graph F. The variable *discovered*[v] stores the status of a node v in F. It is initialized to be NO and becomes YES after node v is discovered in the breadth-first search. Q is a queue of nodes and HEAD[Q] is the node at the head of the queue. A node is appended to the end of Q when

it is first discovered and removed from the queue when all its neighbors have been discovered. The variable ρ stores the path under construction and the variable *last* stores the last node visited by ρ. The algorithm extends ρ, whenever possible, by a directed edge from *last* to a newly discovered node.

The algorithm is adapted from [8] to also initialize a list *GROUP* of pairs (ρ, i), where ρ denotes a path or a set of colluders on the path, depending on context, and $i \in I$ is the combined information of colluders in ρ. *GROUP* is empty on entry of the algorithm; when returned it contains a pair (ρ, i) for each connected component of F, with the interpretation that ρ is a valid path that visits every colluder in the connected component and i is the information of the last recipient if ρ is followed. The list *GROUP* will be used to solve the general case in the next section.

Algorithm 1

Input: An initial state $w(0)$ and a target information set T.

Output: A collusion path ρ if the collusion problem has a solution; *GROUP* otherwise.

1. Construct graph $F = (U_c, E)$ from $w(0).N$ and store it in $Adj[v]$
2. For each $v \in U_c$, $discovered[v] \leftarrow$ NO; $GROUP \leftarrow \phi$
3. For each $s \in U_c$
   If $discovered[s]$ = NO, then SEARCH(s)
4. Return ($GROUP$)

SEARCH(s)

1. $discovered[s] \leftarrow$ YES
2. $\rho \leftarrow$ NIL; $last \leftarrow s$; $Q \leftarrow \{s\}$
3. While $Q \neq \phi$
   $u \leftarrow$ HEAD$[Q]$
   For each $v \in Adj[u]$
   If $discovered[v]$ = NO
   then $\rho \leftarrow \rho \cdot (last, v)$; $last \leftarrow v$
   $discovered[v]$ = YES;
   append v to the end of Q
   Remove u from Q
4. $i \leftarrow \Delta(\cup_{u \in \rho}\, w_u(0).I)$; append (ρ, i) to $GROUP$
5. If $i \supseteq T$, then Return(ρ)

In the algorithm, the function SEARCH(s) is an adaptation of a breadth-first search [1]. It finds a connected components of F that contains the node s. It also constructs a path ρ while it scans the connected component, by adding an edge from the last node of the current ρ to a new node when the new node is first discovered. For a proof that the constructed path ρ is valid, see [4]. It has visited every node in the connected component when the entire connected component has been scanned. The algorithm then adds the pair (ρ, i) to the data structure *GROUP*, where i is the combined initial information of all nodes on the path ρ. If i contains T then the candidate path ρ is a collusion path and the algorithm returns. Otherwise, it repeats the search on a different connected component of F. If all connected components of F have been searched without producing a collusion path then the algorithm returns *GROUP*. We summarize.

Theorem 1 *The algorithm terminates. Moreover it returns a collusion path if the collusion problem has a solution and the linked list* GROUP *otherwise.*

4 General case: $L \supseteq \phi$

In the last section we show how to solve the collusion problem for the special case where collusion is allowed only on unique messages. In this section we solve the general case where collusion is allowed on unique data in $L \neq \phi$ as well.

For the special case the condition under which a solution exists can be described by a simple expression involving the initial knowledge of the colluders. This is no longer possible for the general case, but the solution can still be determined from just the initial state $w(0)$. We will provide an algorithm, Algorithm 2 below, that verifies if a collusion path exists and if so, computes one, without having to search the state machine Θ on which the collusion problem is defined.

Recall the graph F that describes all events that are initially enabled in $w(0)$. The algorithm starts by calling Algorithm 1 of the last section. If there is a connected component of F whose combined initial information contains the target information set T, then Algorithm 1 will identify it and returns a collusion path that involves only nodes in that connected component. Otherwise Algorithm 1 will have initialized the data structure *GROUP* to specify, for each connected component of F, a valid path through all nodes in the connected component and their combined initial information.

At any time, an element (ρ, i) of *GROUP* identifies a group of colluders, a *valid* path ρ that visits every

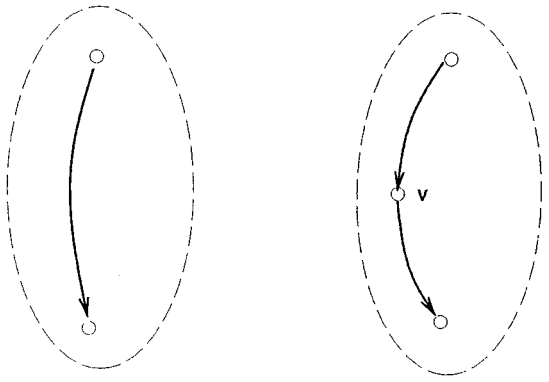

Figure 1: Elements (ρ_1, i_1) and (ρ_2, i_2) of *GROUP*

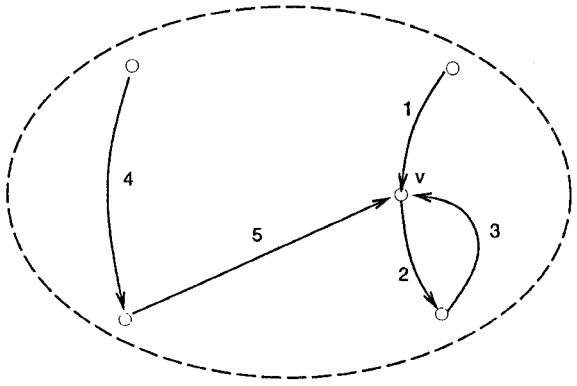

Figure 2: New element (ρ_{12}, i_{12}) of *GROUP*

colluder in the group and the combined initial information $i = \Delta\,(\cup_{u\in\rho}\, w_u(0).I)$ of the group. The last recipient of ρ, denoted TAIL$[\rho]$, knows the combined information i when ρ is followed. Immediately after Algorithm 1 returns without finding a collusion path, each element of *GROUP* corresponds to a connected component of F. As Algorithm 2 proceeds these elements are combined to form bigger and few groups, until either a collusion path is found or no further combination is possible. The collusion problem has no solution in the latter case.

Two colluders in different connected components of F can collude only if they share a unique piece of information in L (condition (2)). The algorithm searches for two elements in *GROUP*, identified by (ρ_1, i_1) and (ρ_2, i_2), such that TAIL$[\rho_1]$ knows a colluder v in ρ_2 and shares a unique piece of information with TAIL$[\rho_2]$. Note that TAIL$[\rho_1]$ may not share a unique piece of information with v to allow them to collude, nor may it know TAIL$[\rho_2]$ to collude with it. But TAIL$[\rho_2]$ certainly knows v since $v \in \rho_2$ and hence can transfer all its knowledge to v. Then v and TAIL$[\rho_1]$ indeed share a unique piece of information and can collude. In summary the path

$$\rho_{12} \;:=\; \rho_2 \cdot (\text{TAIL}[\rho_2], v) \cdot \rho_1 \cdot (\text{TAIL}[\rho_1], v)$$

is valid. Moreover its last recipient v knows the combined information $i_{12} := \Delta(i_1 \cup i_2)$. The construction of ρ_{12} from ρ_1 and ρ_2 is illustrated in Figures 1 and 2.

If i_{12} contains T, then the corresponding candidate path ρ_{12} is a collusion path, and the algorithm returns. Otherwise, the two elements (ρ_1, i_1) and (ρ_2, i_2) are removed from *GROUP* and the new element (ρ_{12}, i_{12}) is added, and the search repeats. When no further elements in *GROUP* can combine their information the algorithm concludes that the collusion problem has no solution and returns NIL.

Algorithm 2

Input: An initial state $w(0)$ and a target information set T.

Output: A collusion path ρ if the collusion problem has a solution; NIL otherwise.

1. Execute Algorithm 1
2. If collusion path ρ is found, Return(ρ)
3. While there are elements (ρ_1, i_1), (ρ_2, i_2) of *GROUP* satisfying $i_1 \cap \rho_2 \neq \phi$ and $i_1 \cap i_2 \cap L \neq \phi$
   Let $v \in i_1 \cap \rho_2$
   $\rho_{12} \leftarrow \rho_2 \cdot (\text{TAIL}[\rho_2], v) \cdot \rho_1 \cdot (\text{TAIL}[\rho_1], v)$
   $i_{12} \leftarrow \Delta(i_1 \cup i_2)$
   If $i_{12} \supseteq T$ then Return (ρ_{12})
   Remove (ρ_1, i_1), (ρ_2, i_2) from *GROUP*
   Add (ρ_{12}, i_{12}) to *GROUP*
4. Return (NIL).

The correctness of the algorithm is guaranteed by the following theorem.

Theorem 2 *The algorithm terminates. Moreover it returns a collusion path if the collusion problem has a solution and* NIL *otherwise.*

We now briefly sketch why the theorem should hold. If there is a connected component of F whose combined initial information contains T, then Algorithm 1 in the previous section will identify a collusion path, and the algorithm will terminate in step 2. Otherwise, Algorithm 1 will return the list *GROUP* that contains one element for each connected component of F. Each time the 'while' loop is entered the list *GROUP* is shortened by one. Hence Algorithm 2 must terminate,

either inside the 'while' loop with a collusion path, or in step 4 without one. It is proved in [4] that it terminates inside the 'while' loop with a collusion path if and only if there exists one.

5 Conclusion

In earlier work [8] we have formulated a collusion problem which determines whether a set of colluders can, starting from their initial knowledge, collectively discover a target set of information, and have provided a complete solution for a special case. In this paper we have solved the general case with an algorithm that determines whether a collusion problem has a solution, and if it does, computes one. All results in [8] and this paper can be found in [4].

Communications protocols that employ cryptographic techniques are increasingly used to protect privacy as well as to communicate. A cryptographic protocol defines a process by which information is transferred among some users while hidden from others. The algorithm presented here can be applied to determine whether a subset of protocol users can collectively discover, during or after the protocol's execution, the information that is designed to be hidden from them; see [8, 6] for an example.

References

[1] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms.* The MIT Press, 1993.

[2] Semyon Dukach. SNPP: A Simple Network Payment Protocol. In *Proceedings of the Computer Security Applications Conference*, San Antonio, TX, November 1992.

[3] Hannes Federrath, Anja Jerichow, and Andreas Pfitzmann. Mixes in mobile communication systems: Location management with privacy. *Proc. Workshop on Information Hiding*, May 1996.

[4] S. H. Low and N. F. Maxemchuk. Collusion in cryptographic protocols. Technical report, University of Melbourne, Department of Electrical & Electronic Engineering, 1996.

[5] S. H. Low, N. F. Maxemchuk, and S. Paul. Anonymous credit cards. *Proceedings of the 2nd. ACM Conference on Computer and Communications Security*, November 2–4 1994.

[6] S. H. Low, N. F. Maxemchuk, and S. Paul. Anonymous credit cards and their collusion analysis. *IEEE/ACM Transactions on Networking*, 4(6):809–816, December 1996.

[7] Steven H. Low and Nicholas F. Maxemchuk. Collusion analysis of cryptographic protocols. In *Proc. of Globecom'96*, pages 1–5. London, UK, November 1996.

[8] Steven H. Low and Nicholas F. Maxemchuk. Modeling cryptographic protocols and their collusion analysis. In *Proc. of First International Workshop on Information Hiding*, Cambridge, U.K., May/June 1996. Ross Anderson, editor, Vol. 1174 of *Lecture Notes in Computer Science*, pages 169–186, Springer Verlag.

[9] N. F. Maxemchuk and S. H. Low. The use of communications networks to increase personal privacy. *Proceedings of Infocom'95*, pages 504–512, April 1995.

[10] A. Pfitzmann, B. Pfitzmann, and M. Waidner. ISDN-MIXes – Untraceable Communications with very Small Bandwidth Overhead. *Proc. IFIP/Sec'91*, pages 245–258, May 1991.

[11] A. Pfitzmann and M. Waidner. Networks without user observability. *Computers and Security*, 6(2):158–166, 1987.