A Collusion Problem and its Solution *

Steven H. Low †
Dept. of Electrical & Electronic Engineering
University of Melbourne
Parkville, Vic. 3052
Australia
slow@ee.mu.oz.au

Nicholas F. Maxemchuk
AT&T Research
Murray Hill, NJ 07974
USA
nfm@research.att.com

*Partial and preliminary results have been presented in Low and Maxemchuk (1996), and in Low and Maxemchuk (1997). This paper is to appear in the journal Information and Computation.

†I'm grateful to AT&T Bell Laboratories and the Department of Electrical & Electronic Engineering, Hong Kong University of Science & Technology, where part of this work was done, and to Australian Research Council Grant S49711288 for partial financial support.

Abstract

Consider a group of colluders each with certain knowledge such as identity of some other colluders, some cryptographic keys, and some data, possibly multiply encrypted. Two colluders can combine their knowledge if their current knowledge satisfies certain condition. Their cryptographic keys can help decrypt each other's encrypted data, expanding their knowledge and revealing more collusion opportunities, and the process of collusion continues. The question we address is whether it is possible for them to uncover a target set of unencrypted data. In this paper we formulate the collusion problem and provide an algorithm that determines whether a collusion problem has a solution and if so, computes one. A solution is a specific way by which the colluders can uncover the hidden information. The solution generated by our algorithm is generally not one that involves the minimum number of colluders. We show however that to find such a solution is NP-complete. Complex communications protocols employing cryptographic building blocks are being developed to transfer information among some users and hide from others. The algorithm presented here can be applied to determine whether and how a subset of protocol users can discover during or after the protocol's execution the information that is to be hidden from them.

List of Symbols

k	lowercase "kay"
ϵ	lowercase epsilon
Σ	uppercase sigma
σ	lowercase sigma
Δ	uppercase delta
δ	lowercase delta
ρ	lowercase rho
λ	lowercase lambda
Θ	uppercase theta
ϕ	lowercase phi (also denotes null set)
$\cup$	union
$\subseteq$	subset or equal
$\not\subseteq$	not a subset of or equal to
$\supseteq$	superset or equal
$\not\supseteq$	not a superset of or equal to
$\in$	in (a set)
$\notin$	not in (a set)
$\sum$	summation
$\mathcal{X}$	uppercase "X" in calligraphy font
$\mathcal{Y}$	uppercase "Y" in calligraphy font

1 Introduction

Consider a group of colluders each knowing certain information such as identity of some other colluders, some cryptographic keys, and some data, possibly multiply encrypted. Two colluders can combine their knowledge provided their current knowledge satisfies certain condition. A colluder's cryptographic key can help decrypt the encrypted data in the other colluder's possession, expanding its knowledge. This may reveal even more decryption keys and new collusion opportunities, and the process of collusion continues. The question we address is whether it is possible for them to uncover a target set of unencrypted data.

This is motivated by the recent proliferation of complex communications protocols employing cryptographic building blocks that are are being developed not only to communicate, but also to protect privacy, e.g., in broadband networks (Pfitzmann and Waidner (1987), Pfitzmann, Pfitzmann and Waidner (1991)), in mobile networks (Federrath, Jerichow and Pfitzmann (1996)), in electronic commerce (Dukach (1992), and Low, Maxemchuk and Paul (1996)), and in health insurance systems (Maxemchuk and Low (1995)). The credit card protocol of Low, Maxemchuk and Paul (1996), for instance, uses 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.

One may think of a cryptographic protocol as defining a process by which some information is transferred among some users and hidden from others. Solution of the collusion problem presented here can be applied to determine whether, and how, it is possible for a subset of users to discover the information that is to be hidden from them after or during a protocol's execution. An example is given in Low, Maxemchuk and Paul (1996).

It is not always possible for two users to collude. In order to collude they might have to share a unique piece of information pertaining to the protocol run. This may be a unique message that has been exchanged during that protocol run, or a unique piece of data. See §2.2 below for motivation of this requirement. We stress however that our formulation includes as a special case the situation where two users can collude as long as one knows the other, even if they share no unique message or data.

In Low and Maxemchuk (1996) we first introduced a formal model for collusion analysis that consists of two phases in sequence: the protocol execution phase followed by a collusion phase. The protocol phase is modeled as a transition system and the collusion phase is modeled as another related transition system. We presented an algorithm there that completely solves the special case where two users can collude only if they shared a unique message that is exchanged during the protocol phase. That algorithm is extended in Low and Maxemchuk (1997) to solve the general case where users can collude on unique data as well as on unique messages.

In this paper we simplify our original formulation, and extend and provide proofs for these results. In the current formulation, the protocol phase is eliminated from the formal model and its effect can be summarized by the initial state of the transition system modeling the collusion phase.

In §2, we present our model and formulate the collusion problem as a reachability analysis on a large transition system, exhaustive search on which is impractical. The state of the transition system represents the colluders' knowledge and a transition represents a two-party collusion that expands the receiver's knowledge. A solution to the collusion problem, called a collusion path, is a specific way by which the colluders can uncover the hidden information. We show that the existence of a solution, 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 treat the special case where collusion is allowed only between two colluders sharing unique messages. We prove a closed form expression specifying a condition under which a solution exists and clarify the simple structure of collusion paths. The result establishes that for a collusion problem to have a solution it is necessary and sufficient that a subset of colluders exists such that the decrypted union of their *initial* knowledge contain the hidden information and that they share among them unique messages. This characterization leads to the algorithm that checks whether the condition is satisfied and if so computes a collusion path.

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

The collusion path computed by our algorithm is not necessarily one that involves the minimum number of colluders. More generally suppose that a cost is incurred when a pair of users collude and that the cost of a collusion path is the sum of collusion costs associated with each pair of colluders on that path. We show in §5 that to find a collusion path that involves the minimum number of colluders is NP-complete. This implies that the least cost collusion problem is NP-hard.

Cryptographic protocols are notoriously hard to design and their correctness is harder to prove (Simmons (1994)). Numerous cryptographic protocols have been published and later found to contain security flaws, see, e.g., Needham and Schroeder (1978), Denning and Sacco (1981), Needham and Schroeder (1987), Tatebayashi, Matsuzaki and Newman (1989), Simmons (1985), Meadows (1991), Moore (1988), Burrows, Abadi and Needham (1990). These often subtle failures do not require eroding the integrity of the underlying cryptoalgorithm and hence are weaknesses of the protocols. They clearly demonstrate the need for formal methods to verify cryptographic properties of protocols, such as the algebraic method of Dolev and Yao (1983), and Dolev, Even and Karp (1982) to analyze the security of a class of public key protocols, the logic of Burrows, Abadi and Needham (1990), and Gong, Needham and Yahalom (1990) to verify authentication protocols, and the state machine models of Millen (1984), Kemmerer (1989), Meadows (1991), and Kemmerer, Meadows, and Millen (1994) to specify and automatically verify cryptographic protocols.

There are two important differences between our work and earlier work on formal analysis of cryptographic protocols. First, previous work (Dolev and Yao (1983), Dolev, Even and Karp (1982), Burrows, Abadi and Needham (1990), Gong, Needham and Yahalom (1990), and Kemmerer, Meadows, and Millen (1994)) mostly verifies the security of a protocol, i.e., whether it fulfills its intended function. Given a secure protocol, we are concerned with how easy it is for a subset of protocol users to discover a target set of information through collusion. Second, as explained in §2 a collusion problem is defined on a transition system, and hence can in principle be solved by exhaustive reachability search, as done in Millen (1984), Kemmerer (1989), and Meadows (1991). By exploiting the special structure of the collusion problem, however, our algorithm avoids searching the state space of the transition system, which can have $2^{|U|(|U|-1)}$ reachable states, and works on a graph with $|U|$ nodes, where $|U|$ is the number of colluders. This reduction in time complexity is important as large multiparty cryptographic protocols become common. We note that our algorithm supplements, and can be incorporated into, existing protocol analysis tools such as those in Millen (1984), Kemmerer (1989), and Meadows (1991).

2 Model and Problem Formulation

In this section we first present our formal model and formulate the collusion problem. Then we remark on an possible application that motivated our work.

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 or events, and the partial function $\delta : Q \times \Sigma \to Q$ is a transition 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 $\rho_1 \cdot \rho_2$.

The following is a list of symbols used in the sequel and their meaning for easy reference:

U	A set of colluders.

D	A finite set of data.
K	A finite set of cryptographic keys.
L	A subset of $U \cup D \cup K$.
I	Information set $K^*(U \cup D \cup K)$.
Δ	Decryption function $\Delta : 2^I \to 2^I$.
N	Set of unique message identifier.
W	Knowledge set $2^N \times 2^I$.
$\Theta = (W^{\lvert U \rvert}, \Sigma, \delta)$	Collusion (transition) system.
$w(0)$	Initial state of Θ.
$w(\rho)$	State of Θ after ρ is followed starting from $w(0)$.
$w_c(\rho) \in W$	Knowledge of colluder $c \in U$ when Θ is in state $w(\rho)$.
$G(\rho)$	Labeled graph corresponding to path ρ.
$\rho(G)$	Path corresponding to labeled graph G.
$A(y; \rho)$	Set of all timed ancestors of y in $G(\rho)$.
$\bar{A}(y; \rho)$	$\bar{A}(y; \rho) \cup \{y\}$.
$F = (U, E(w(0))$	Graph where there is an edge $(u, v) \in E(w(0))$ iff $u \sim v$.

2.2 Collusion Model

Consider a group of colluders, who have been involved in a protocol execution. 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 other colluders, a set of data, possibly multiply encrypted, and a set of cryptographic keys. The colluders' objective is to collectively discover a certain set of information. 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 quadruple (U, D, K, L) where

1. U is a set of colluders;
2. D is a finite set of data;
3. K is a finite set of cryptographic keys, including the identity key ϵ;
4. $L \subseteq U \cup D \cup K$ is a set of information that determines whether two colluders can collude; see condition (6) below.

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

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

For example, if $d \in U \cup D \cup K$ 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 through the cancellation rule $k^{-1}\ k = k\ k^{-1} = \epsilon$, the identity key, as the removal of encryption from a piece of encrypted data. For instance $\Delta(\{k(d)\} \cup \{k^{-1}\}) = \{d, k^{-1}\}$. In general $\Delta(A)$ 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 $k_n^{-1} \notin \Delta(A)$. The precise specification of Δ is straightforward but cumbersome. In fact it is not important for our purpose, as we only need the following property of Δ: for any A, A' in I,

$$\Delta(\ \Delta(A) \cup A'\ ) \ = \ \Delta(A \cup A') \tag{1}$$

i.e., the order in which a user receives and decrypts information is immaterial. The final combined and reduced information is the same. 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 $u \in U$.

As colluders in U collude by exchanging messages, their knowledge is modified. This evolution is modeled by a transition system $\Theta = (W^{|U|}, \Sigma, \delta)$. Here, a state $w = (w_u, u \in U)$ in $W^{|U|}$ is the knowledge of all colluders. An event $\sigma = (s, r)$ in $\Sigma := U \times U$ 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 *partial* function δ describes the transformation of receiver's knowledge as a result of the message exchange. When the current state is $w = (w_u, u \in U)$ and the next event is $\sigma = (s, r)$, then the next state $w' := \delta(w, \sigma)$, *if defined*, is:

$$\begin{aligned} w'_y &= w_y \quad \text{if } y \neq r & (2)\\ w'_r.N &= w_r.N \cup w_s.N & (3)\\ w'_r.I &= \Delta(w_r.I \cup w_s.I) & (4) \end{aligned}$$

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

For each state $w = (w_u, u \in U)$ 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:

$$\begin{aligned} w_s.N \cap w_r.N &\neq \phi & (5)\\ w_s.I \cap w_r.I \cap L &\neq \phi. & (6) \end{aligned}$$

The conditions say that for s and r to collude, s must know r and they must either share a message (condition (5)) or a piece of data in L (condition (6)). Note that if $L = U$ then since $u \in w_u$ for all $u, r \in w_s.I$ implies condition (6). Then the collusion prerequisites (5–6) reduce to the special case in which collusion is allowed as long as the sender s knows the receiver r.

To motivate the requirements (5–6) consider as an example an intermediary cx that forwards a piece of data to its recipient r in order to hide the identity of its sender s from r (Chaum (1981)):

1. $s \longrightarrow cx : k_{cx}(r, k_r(d))$
2. $cx \longrightarrow r : k_r(d)$

In the above s encrypts the (encrypted) data $k_r(d)$ and the recipient's identity r with a key k_{cx} that can only be decrypted by the intermediary cx and sends them to cx (message 1). The intermediary cx then forwards the encrypted data to r (message 2), thus hiding the identity of the sender s from r. Variants of this simple protocol have been the building blocks of large cryptographic protocols to provide privacy in broadband networks (Pfitzmann and Waidner (1987), Pfitzmann, Pfitzmann and Waidner (1991)), in credit card transactions (Low, Maxemchuk and Paul (1996)), and in mobile networks (Federrath, Jerichow and Pfitzmann (1996)), where traffic volumes are high. After the above steps are carried out, cx knows $w_{cx} := (\{\text{message 1, message 2}\}, \{s, cx, r, k_{cx}^{-1}, k_r(d)\})$ and r knows $w_r := (\{\text{message 2}\}, \{cx, r, k_r^{-1}, d\})$. For r to discover s, r must learn the information in w_{cx}. In a large system however cx may have forwarded a large number of messages to the same recipient r in a short period of time and they have collected a large number of w_{cx} and w_r, corresponding to different protocol runs. Moreover the larger protocol of which the above is only a part can be implemented on a datagram network so that messages from different protocol runs may be interleaved at cx. Hence to combine the information in w_{cx} and w_r of the *same* protocol run, cx and r must share a unique piece of information pertaining to that protocol run. The unique message that is exchanged between cx and r can be used to pair up w_{cx} and w_r that belong to the same protocol run. This is modeled by condition (5). Sometimes two colluders can combine their knowledge pertaining to a particular protocol run if they share a piece of data. For instance two banks may have the unique social security number of a customer and hence can combine their knowledge about the customer. This is modeled by condition (6).

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

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 piece of data in L that the sender and that user share, 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 colluders and how their knowledge evolves as collusion proceeds.

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

2.3 Problem formulation

The collusion problem is to determine if the colluders can combine their knowledge, by passing messages, and extract the hidden information. Suppose we have a collusion system $\Theta = (W^{|U|}, \Sigma, \delta)$.

Collusion problem

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

We call ρ in the definition of the collusion problem a *collusion path*. It specifies which colluders should transmit to which other colluders and when.

The collusion problem is simply a reachability analysis on the state machine Θ. Given $w(0)$, however, the set of reachable states in Θ is a subset of all possible combinations of the colluders' initial knowledge, and contains up to $2^{|U|(|U|-1)}$ states. For example, for the simple protocol analysed in Low, Maxemchuk and Paul (1996), $|U| = 8$ 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 develop a solution that avoids exploring Θ.

For the rest of this paper, we make the following natural assumption 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 message (condition (5)), then they must know each other in $w(0)$ and hence (v, u) must also be enabled, i.e., $w(0)$ satisfies the condition:

$$\begin{aligned} & w_u(0).N \cap w_v(0).N \neq \phi \\ \Rightarrow \quad & v \in w_u(0).I \quad \text{and} \quad u \in w_v(0).I \end{aligned} \tag{7}$$

The motivation behind this assumption is that, in our setting, if u and v share a message, then they must have directly exchanged that message during protocol execution before collusion is carried out. Clearly the source of this exchange knows the destination. In almost all communication protocols the identity of the source is also included in the header of the message for error and flow control. Hence the source and destination of a direct message exchange always know the identity of each other after the exchange.

Henceforth fix an environment (U, D, K, L), an initial state $w(0)$ and a target information set T.

2.4 Example application

We now describe an example application that motivated this work. A cryptographic protocol is designed in Low, Maxemchuk and Paul (1996) to implement anonymous credit cards in which a typical transaction involves the cardholder, the store where she makes a purchase, the cardholder's bank, the store's bank, and several intermediaries. At the end of a transaction no single participant, except the cardholder, knows both her identity and what she purchases. The question one might ask is which subsets of these participants (except the cardholder herself) must collude to associate the cardholder's identity and her purchase.

This question can be broken down into several subproblems, each involving a different subset of participants. Each subset defines an environment (U, D, K, L) and a collusion problem, where the set U of colluders is the subset of participants under study and the target information T is cardholder's identity and her purchase. These collusion problems can be solved using the algorithm presented in the sequel and exhibit the potential vulnerabilities of the protocol to privacy protection; see Low, Maxemchuk and Paul (1996). In fact, by Theorem 4 below, we do not need to consider all possible subsets of participants, but only those subsets whose initial information, when combined, contains T.

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 (5)). It illustrates the structure of the problem and is useful to the solution of the general case, presented in the next section.

We solve this special case in two steps. In §3.1 we prove in Theorem 4 below that the collusion problem has a solution if and only if there is 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. This means, in particular, that if the target information in T is distributed among clusters of users who have not communicated during protocol execution, then no user can discover the entire T regardless of what knowledge each user has. This theorem is proved through a sequence of lemmas.

Based on this characterization we present in §3.2 an algorithm that checks whether the condition is satisfied, and if so, computes a collusion path.

3.1 Solution characterization

The structure of a collusion path on the transition system Θ can be better exhibited in terms of a *collusion graph* we now explain. Consider a path $\rho = (s_1, r_1) \cdots (s_n, r_n)$. We may also use ρ to refer to the set of events $\rho = \{(s_t, r_t), t = 1, \ldots, n\}$ in the path, or the set of colluders $\rho = \{s_t, r_t, t = 1, \ldots, n\}$; the meaning should be clear from the context. Hence by '$(s, r) \in \rho$' and '$c \in \rho$', we mean $(s, r) = (s_t, r_t)$ and $c = s_t$ or r_t, respectively, for some t. As noted above all paths are assumed to start from the given initial state $w(0)$ unless otherwise specified. For any path ρ in Θ starting from $w(0)$, $w(\rho)$ denotes Θ's state after the transitions in ρ have been made.

A path in Θ can be equivalently specified by a *labeled graph* $G = (V, E)$ where V

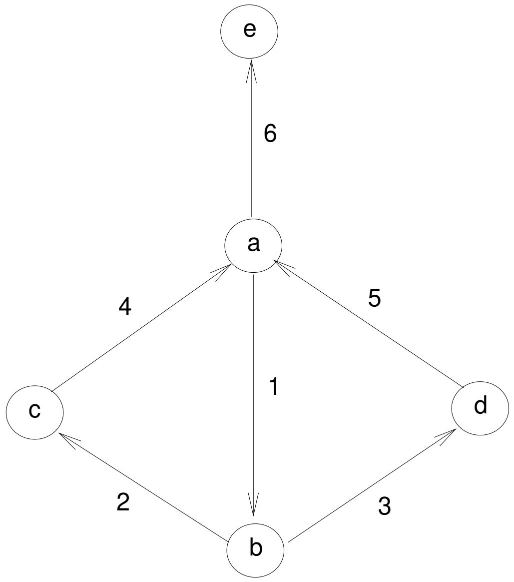

Figure 1: Labeled graph G. $\rho(G) = (a,b)(b,c)(b,d)(c,a)(d,a)(a,e)$. An example timed path is $(a,b,1)$ $(b,c,2)$ $(c,a,4)$. Timed ancestors of a are $A(a;\rho(G)) = \{a,b,c,d\}$.

are the nodes and $E = \{(u,v,t) \mid u,v \in V,\ t \in \{1,\ldots,|E|\}\}$ is a set of directed edges from node u to node v labeled by t. The nodes represent colluders involved in the path and the edges represent messages among the colluders. The label on an edge indicates its relative transmission time. It is an important consideration because some events are not enabled until other messages have been transmitted. There can be multiple edges between two nodes in the same direction with different labels, corresponding to the same sender-receiver pair appearing multiple times in the path. Let $\rho(G)$ be the path in Θ defined by a sequentially labeled graph G such that each edge $(u,v,t) \in E$ corresponds to the event (u,v) and is the t-th event in $\rho(G)$. Similarly, let $G(\rho) = (V(\rho), E(\rho))$ be the unique labeled graph defined by ρ in Θ, such that $V(\rho) = \{c \in \rho\}$ and $E(\rho) = \{(u,v,t) \mid (u,v) \text{ is the } t\text{-th event in } \rho\}$.

Definition 2 *A labeled graph G is a* valid *graph if $\rho(G)$ is a path in Θ; it is a* collusion graph *if $\rho(G)$ is a collusion path.*

Consider a path $\rho = (s_1, r_1) \cdots (s_n, r_n)$ with n events and the associated labeled graph $G(\rho) = (V, E)$. A *timed path* (with respect to ρ) from node x to node y is a directed path in $G(\rho)$

$$(x, u_1, t_1) \cdots (u_{k-1}, y, t_k)$$

starting at x and terminating at y such that $1 \leq t_1 < \cdots < t_k \leq n$. If there is a timed path from x to y, we call x a *timed ancestor* of y. Let $A(y;\rho)$ denote the set of all timed ancestors of node y, with respect to ρ. An example is given in Figure 1.

The following useful lemma explains how to compute a colluder's knowledge after a path ρ is followed. It says that a colluder c knows in state $w(\rho)$ the decrypted union of the *initial* information of all, and only, its timed ancestors and c. To simplify notation define functions $f_1 : 2^U \to 2^N$ and $f_2 : 2^U \to 2^I$ by: for any subset $A \subseteq U$ of colluders,

$$\begin{aligned} f_1(A) &= \cup_{u\in A}\, w_u(0).N && (8)\\ f_2(A) &= \Delta(\cup_{u\in A}\, w_u(0).I) && (9) \end{aligned}$$

i.e., $f_1(A)$ is the union of message identifiers colluders in A initially have and $f_2(A)$ is the decrypted union of their initial information. Note that $(f_1(A), f_2(A)) \in W$. Define $\bar{A}(c;\rho) = A(c;\rho) \cup \{c\}$.

Lemma 1 *The knowledge of any colluder c in state $w(\rho)$ to which a path ρ leads is given by*

$$w_c(\rho) = (f_1(\bar{A}(c;\rho)), f_2(\bar{A}(c;\rho))) \qquad (10)$$

Note that (10) express each colluder's final knowledge, and hence the final state $w(\rho)$, in terms of the initial state $w(0)$.

Proof. Let $\rho = (s_1, r_1)(s_2, r_2)\cdots(s_n, r_n)$ consists of n events. We prove the lemma by induction on n.

Let $n = 1$. If $c = s_1$ then $\bar{A}(c;\rho) = \{c\}$, and hence $w_c(\rho) = w_c(0) = (f_1(\bar{A}(c;\rho)), f_2(\bar{A}(c;\rho)))$ by update rule (2). If $c = r_1$ then $\bar{A}(c;\rho) = \{s_1, c\}$, and hence $w_c(\rho) = (f_1(\bar{A}(c;\rho)), f_2(\bar{A}(c;\rho)))$ by update rule (2-3).

Assume the theorem holds for $n = k$. Consider $n = k+1$. Let $\rho^k = (s_1, r_1)(s_2, r_2)\cdots(s_k, r_k)$ consists of the first k events of ρ, i.e., $\rho = \rho^k \cdot (s_{k+1}, r_{k+1})$. By induction hypothesis $w_c(\rho^k) = (f_1(\bar{A}(c;\rho^k)), f_2(\bar{A}(c;\rho^k)))$.

If $c \neq r_{k+1}$ then $\bar{A}(c;\rho) = \bar{A}(c;\rho^k)$ and hence by update rule (2),

$$\begin{aligned} w_c(\rho) &= w_c(\rho^k)\\ &= (f_1(\bar{A}(c;\rho^k)), f_2(\bar{A}(c;\rho^k)))\\ &= (f_1(\bar{A}(c;\rho)), f_2(\bar{A}(c;\rho))). \end{aligned}$$

If $c = r_{k+1}$ then $\bar{A}(c;\rho) = \bar{A}(s_{k+1};\rho^k) \cup \bar{A}(r_{k+1};\rho^k)$ and hence by update rule (3-4),

$$\begin{aligned} w_c(\rho).N &= w_{s_{k+1}}(\rho^k).N \cup w_{r_{k+1}}(\rho^k).N\\ &= f_1(\bar{A}(s_{k+1};\rho^k)) \cup f_1(\bar{A}(r_{k+1};\rho^k))\\ &= f_1(\bar{A}(c;\rho)) \end{aligned}$$

and

$$\begin{aligned} w_c(\rho).I &= \Delta(w_{s_{k+1}}(\rho^k).I \cup w_{r_{k+1}}(\rho^k).I)\\ &= \Delta(f_2(\bar{A}(s_{k+1};\rho^k)) \cup f_2(\bar{A}(r_{k+1};\rho^k)))\\ &= f_2(\bar{A}(s_{k+1};\rho^k) \cup \bar{A}(r_{k+1};\rho^k)) \;=\; f_2(\bar{A}(c;\rho)) \end{aligned}$$

where we have used (1) twice in the second last equality. Hence the theorem holds for $n = k + 1$, and this completes the proof. $\square$

Minimal collusion paths, those on which no event can be omitted in order to complete the path in the same relative order, have especially simple structure. Denote by $\rho - (s_t, r_t)$ the sequence

$$(s_1, r_1) \cdots (s_{t-1}, r_{t-1})(s_{t+1}, r_{t+1}) \cdots (s_n, r_n)$$

obtained by deleting (s_t, r_t) from $\rho = (s_1, r_1) \cdots (s_n, r_n)$. Note that ρ being a path in Θ does not imply that $\rho - (s_t, r_t)$ is also a path.

Definition 3 *A collusion path $\rho = \{(s_t, r_t), t = 1, \ldots, n\}$ is* minimal *if for all $j \in \{1, \ldots, n\}$, $\rho - (s_j, r_j)$ is not a collusion path. A collusion graph G is* minimal *if $\rho(G)$ is a minimal collusion path.*

For a minimal collusion path ρ, either $\rho - (s_t, r_t)$ is not a path or it does not lead to the target information, i.e., $T \not\subseteq w_c(\rho - (s_t, r_t)).I$ for all $c \in \rho$. A minimal collusion path is not necessarily a collusion path with the minimum length. For instance, there may be a single user, which is not on the path, which has all of the information in T.

The next lemma says that on a minimal collusion path, every colluder is an ancestor of the last recipient, and only the last recipient knows the target information set.

Lemma 2 *Suppose $\rho = (s_1, r_1) \cdots (s_n, r_n)$ is a minimal collusion path. Then*

(i) no $c \in \rho$ except r_n satisfies $T \subseteq w_c(\rho).I$;

(ii) $\bar{A}(r_n; \rho) = \rho$, i.e., $u \in \bar{A}(r_n; \rho)$ if and only if $u \in \rho$.

Proof. Since ρ is a collusion path there is some $c \in \rho$ with $w_c(\rho).I \supseteq T$. Suppose $c \neq r_n$ satisfies this property. Then $w_c(\rho - (s_n, r_n)).I = w_c(\rho).I \supseteq T$, contradicting the minimality of ρ. Hence only r_n knows T in state $w(\rho)$.

Let $G(\rho) = (V, E)$, $V = \rho$, be ρ's collusion graph. The second claim asserts that $V = \bar{A}(r_n; \rho)$. By definition of $\bar{A}(r_n; \rho)$ we have $V \supseteq \bar{A}(r_n; \rho)$. Hence we only need to show that $V \subseteq \bar{A}(r_n; \rho)$.

Consider the subgraph $G' = (V', E')$ of $G(\rho)$ induced by $V' := \bar{A}(r_n; \rho) \subseteq V$, where $E' = \{(s, r, t) \mid (s, r, t) \text{ is on a timed path in } G(\rho) \text{ from some } u \in \bar{A}(r_n; \rho) \text{ to } r_n\}$. We need to show that $V' = V$. Suppose not, i.e., V' and E' are strict subsets of V and E, respectively. We will construct a new collusion path ρ' by removing some events from ρ and keeping the relative order of the remaining events, hence contradicting the minimality of ρ.

The collusion path ρ is of the form

$$\rho = \lambda_1(s'_1, r'_1)\lambda_2(s'_2, r'_2) \cdots \lambda_k(s'_k, r'_k)$$

where $r'_k = r_n$, $s'_t, r'_t \in \bar{A}(r_n; \rho)$, and λ_i is a sequence of events not in E'. Consider the sequence

$$\rho' = (s'_1, r'_1) \cdots (s'_k, r'_k)$$

obtained from ρ by removing all edges not in E'. If ρ' is a path, then since $\bar{A}(r_n;\rho') = \bar{A}(r_n;\rho)$, by Lemma 1,

$$\begin{aligned} w_{r_n}(\rho').I &= f_2(\bar{A}(r_n;\rho')) \\ &= f_2(\bar{A}(r_n;\rho)) \\ &= w_{r_n}(\rho).I \ \supseteq \ T \end{aligned}$$

i.e., ρ' is a collusion path. We hence only need to show that ρ' is a path.

Now the first even (s'_1, r'_1) of ρ' is enabled if $r'_1 \in w_{s'_1}(0).I$ and $w_{r'_1}(0).N \cap w_{s'_1}(0).N \neq \phi$. By construction colluders in λ_1 cannot be timed ancestors of s'_1 or r'_1, for otherwise they will be in $\bar{A}(r_n;\rho)$. Hence, since $\bar{A}(s'_1;\lambda_1) = \{s'_1\}$ and $\bar{A}(r'_1;\lambda_1) = \{r'_1\}$, we have

$$\begin{aligned} w_{s'_1}(\lambda_1) &= (f_1(\bar{A}(s'_1;\lambda_1)), f_2(\bar{A}(s'_1;\lambda_1))) &= w_{s'_1}(0) \\ w_{r'_1}(\lambda_1) &= (f_1(\bar{A}(r'_1;\lambda_1)), f_2(\bar{A}(r'_1;\lambda_1))) &= w_{r'_1}(0) \end{aligned}$$

Therefore $\lambda_1(s'_1, r'_1)$ being a path implies that (s'_1, r'_1) is also a path. Repeating the argument shows that the knowledge of s'_t and r'_t do not depend on transitions in $\lambda_1, \ldots, \lambda_t$, and hence since ρ is a path, so is ρ'. Hence ρ' is a collusion path, contradicting the minimality of ρ. $\Box$

Lemmas 1 and 2 imply that $w_{r_n}(\rho).I$, which contains the target information T, is the decrypted union of the initial information $w_u(0)$ in ρ. Hence, to solve the collusion problem, it is necessary and sufficient to find a set of colluders whose *initial* information in state $w(0)$, when combined, yields T and to find a way for all of them to communicate their information to the same colluder. We now show that such a set of colluders must share unique messages among them in the initial state $w(0)$. In the next subsection we will show how these colluders can combine their information.

Define the symmetric and reflexive relation $\sim$ on the set U of colluders:

$$u \sim v \quad \text{iff} \quad w_u(0).N \cap w_v(0).N \ \neq \ \phi. \tag{11}$$

By assumption (7), $u \in w_v(0).I$ and $v \in w_u(0).I$. That is, $u \sim v$ if and only if they can collude in the initial state $w(0)$. This relation can be represented by an undirected graph $F = (U, E(w(0)))$ with all colluders as its nodes. There is an edge (u, v) in $E(w(0))$ if and only if $u \sim v$. F thus describes all the events that are initially enabled by condition (5). The next lemma says that it is not possible for two colluders in different connected components of F to collude.

Lemma 3 *Given any path $\rho = (s_1, r_1) \cdots (s_n, r_n)$, $s_t \sim r_t$ for $t = 1, \ldots, n$. Hence if ρ is a minimal collusion path then the set $\rho = \bar{A}(r_n;\rho)$ of colluders are all in the same connected component of F.*

Proof. Let $\rho^t = (s_1, r_1) \cdots (s_t, r_t)$ be the first t events of ρ, and $w(\rho^t)$ be the state of Θ after ρ^t is followed starting from $w(0)$. We prove by induction on t that all timed ancestors $\bar{A}(r_t;\rho^t)$ of receiver r_t belong to the same connected component of F. This then implies the first assertion of the lemma since $s_t \in \bar{A}(r_t;\rho^t)$.

By construction of F s_1 and r_1 are in the same connected component. Suppose the assertion holds for $t = 1, \ldots, k$.

For $t = k+1$ consider the state $w(\rho^k)$ before the transition (s_{k+1}, r_{k+1}) is made. If $s_{k+1} \neq r_j$ for $j = 1, \ldots, k$, i.e., s_{k+1} has not been a receiver, then $\bar{A}(s_{k+1}; \rho^k) = \{s_{k+1}\}$. Otherwise let J be the largest $j \leq k$ such that $s_{k+1} = r_j$. Then $\bar{A}(s_{k+1}; \rho^k) = \bar{A}(r_J; \rho^J)$. Similarly let J' be the largest $j \leq k$ such that $r_{k+1} = r_j$. Then $\bar{A}(r_{k+1}; \rho^k) = \bar{A}(r_{J'}; \rho^{J'})$. If no such J' exists then $\bar{A}(r_{k+1}; \rho^k) = \{r_{k+1}\}$. Since (s_{k+1}, r_{k+1}) is enabled in state $w(\rho^k)$, s_{k+1} and r_{k+1} must share a message:

$$w_{s_{k+1}}(\rho^k).N \ \cap \ w_{r_{k+1}}(\rho^k).N \ \neq \ \phi.$$

By (10) and (8) there must be an $u \in \bar{A}(s_{k+1}; \rho^k)$ and an $v \in \bar{A}(r_{k+1}; \rho^k)$ that share a message *in the initial state* $w(0)$. That is u and v satisfy (11) and hence are in the same connected component of F. But then the induction hypothesis implies that $\bar{A}(s_{k+1}; \rho^k)$ and $\bar{A}(r_{k+1}; \rho^k)$ are all in the same connected component. This completes the induction.

If ρ is a minimal collusion path then, by Lemma 2(ii), $\rho = \bar{A}(r_n; \rho)$. Then the above induction shows that colluders in ρ are in the same connected component. $\square$

We hence have the following characterization of when the collusion problem has a solution.

Theorem 4 *The collusion problem has a solution if and only if there is a connected component $C = (V, E)$ of the undirected graph F such that $f_2(V) \supseteq T$.*

Proof. If there is such a connected component C of F then, since the edges connecting them are enabled in $w(0)$ and remain enabled as collusion proceeds, any path that visits every node in the connected component C is a collusion path with the last recipient of the path knowing T by Lemma 2. (For a construction of such a path see Theorem 4 below.)

Conversely suppose there is a collusion path ρ. We can assume that it is minimal for otherwise we can make it minimal by removing redundant events from ρ and keeping the remaining events in the same relative order. Lemma 3 then implies that ρ is a connected component of F. Lemmas 1 and 2 imply that $f_2(\rho) \ = \ \Delta(\cup_{u \in \rho} w_u(0).I) \ \supseteq \ T$. $\square$

3.2 Algorithm

Theorem 4 specifies a condition under which a collusion graph exists. In general a collusion graph can take the form of any directed graph. The next result clarifies the simple structure of collusion paths and leads to our algorithm. It says that every colluder, except the first and the last, receive, decrypt, and forward exactly once.

Theorem 5 *The collusion problem has a solution if and only if there is a collusion path with the simple structure*

$$\rho \ = \ (u_0, u_1)(u_1, u_2) \cdots (u_{n-1}, u_n)$$

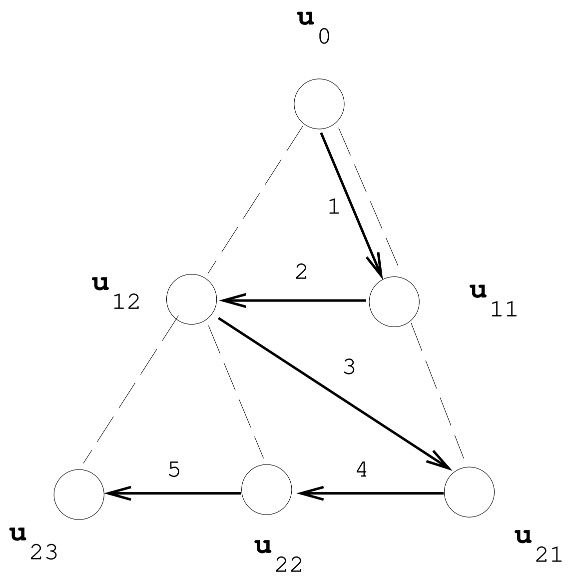

Figure 2: Construction of collusion graph G. Dashed line indicates the spanning tree G'; solid line is G.

where u_i are all distinct.

Proof. The sufficiency follows from the definition of the collusion problem. For necessity, suppose the collusion problem has a solution. By Theorem 4, there is a connected component $C = (V, E)$ of graph F in which all edges in E are enabled in $w(0)$ and $f_2(V) = \Delta(\cup_{u \in V}\ w_u(0).I) \supseteq T$. We will construct a valid graph $G = (V, E')$ that consists of a simple path[1] that visits every node in V exactly once. By Lemma 1, the last recipient knows $f_2(V)$ and hence T in the final state, i.e., G is indeed a collusion graph. Moreover, the corresponding collusion path $\rho(G)$ has the structure given in the theorem.

To construct $G = (V, E')$, we only need to specify the set E' of edges. Let u_0 be any node in the connected component C. We will construct G to be a simple path that starts from u_0 and visits every node in V exactly once. Indeed it visits them in the same order as in a breadth-first search on C starting from u_0, but possibly with new edges not in C. Specifically, let G' be a spanning tree of C rooted at u_0. Let nodes in G' that are one hop away from u_0 be $u_{11}, u_{12}, \ldots u_{1k_1}$, those that are two hops away from u_0 be $u_{21}, u_{22}, \ldots u_{2k_2}$, and so on. Then the graph G, specified as a path, is

$$(u_0, u_{11}, 1)\ (u_{11}, u_{12}, 2)\ \cdots\ (u_{1(k_1-1)}, u_{1k_1}, k_1)\ \ (u_{1k_1}, u_{21}, k_1+1)\ (u_{21}, u_{22}, k_1+2)\ \cdots$$

until all nodes in C are visited. This is illustrated in Figure 2. We need to show that G is a valid graph.

The first edge $(u_0, u_{11}, 1)$ is valid by the choice of u_0 and u_{11}. For edges $(u_{1i}, u_{1(i+1)}, i+1)$, note that u_0 and each $u_{1(i+1)}$ share a unique message in $w(0)$ by definition of C. After

[1]A *simple path* is a graph that contains no loop.

the first i events, u_{1i}'s knowledge includes that of u_0. Hence u_{1i} knows $u_{1(i+1)}$ and shares a unique message with $u_{1(i+1)}$ (even though $u_{1(i+1)}$ does not necessarily know u_{1i}). Thus the edge $(u_{1i}, u_{1(i+1)}, i+1)$ is valid. For the edge $(u_{1k_1}, u_{21}, k_1+1)$, note that u_{21} shares a unique message with some u_{1i}. Since u_{1k_1}'s knowledge includes that of u_{1i} after the first k_1 events, u_{1k_1} knows u_{21} (not necessarily vice versa) and shares a unique message with u_{21}. Hence the edge $(u_{1k_1}, u_{21}, k_1+1)$ is valid. Other edges are valid following a similar argument. This completes the proof. $\square$

Theorems 4 and 5 suggest the following algorithm to solve the collusion problem. It first constructs the graph F that specifies all events that are initially enabled. Then it finds each connected component of F using breadth-first search while, at the same time, constructing a candidate collusion path that visits every node in the connected component. The path has the same structure as in Theorem 5 and the construction follows that in its proof. Theorem 4 then guarantees that the collusion problem has a solution if and only if such a collusion path can be found.

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 it is discovered by 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 *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.

Finally the algorithm initializes 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 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 $F = (U, E)$ from $w(0).N$
2. For each $v \in U$, *discovered*$[v] \leftarrow$ NO; *GROUP* $\leftarrow \phi$
3. For each $s \in U$
   If *discovered*$[s] =$ NO, then SEARCH(s)
4. Return (*GROUP*)

SEARCH(s)

1. *discovered*$[s] \leftarrow$ YES

2. $\rho \leftarrow$ NIL; $\quad last \leftarrow s$; $\quad Q \leftarrow \{s\}$
3. While $Q \neq \phi$

   $u \leftarrow$ HEAD$[Q]$

   For each $v \in Adj[u]$

   If $discovered[v] =$ NO

   then $\rho \leftarrow \rho \cdot (last, v)$; $\quad last \leftarrow v$

   $discovered[v] =$ YES; append v to the end of Q

   Remove u from Q
4. If $\rho =$ NIL, then $\rho \leftarrow s$;

   $i \leftarrow f_2(\rho) = \Delta(\cup_{u \in \rho} w_u(0).I)$; append (ρ, i) to $GROUP$
5. If $i \supseteq T$, then Return(ρ)

In Algorithm 1, the function SEARCH(s) is an adaptation of a breadth-first search (Cormen, Leiserson and Rivest (1993)). It finds a connected component of F that contains the node s. The function SEARCH(s) 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 (see proof of Theorem 5 for correctness). If the connected component consists of a single node, so does ρ. It has visited every node in the connected component when the entire connected component has been scanned. It then stores in i the combined initial information of all nodes on the path ρ. According to Lemmas 1 and 2 this represents the knowledge of the last recipient on the path when ρ is followed. The algorithm adds the pair (ρ, i) to the list $GROUP$. If i contains T then the algorithm stops and returns the collusion path ρ. 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 Theorem 4 guarantees that none exists, in which case it returns $GROUP$. We summarize.

Theorem 6 *If the collusion problem has a solution, Algorithm 1 returns a collusion path ρ; otherwise, it returns GROUP.*

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 data in L as well.

The condition under which a solution exists is no longer a simple expression as in Theorem 4 for the special case, but it can still be determined from just the initial state $w(0)$. We present Algorithm 2 below that verifies if a solution exists and if so, computes a collusion path. Indeed the algorithm uses Algorithm 1 of the last section as its first step. If a collusion path exists that involves only colluders who share unique messages in state $w(0)$ (i.e., in the same connected component of F), then Algorithm 1 will identify it. Otherwise any collusion path must involve two colluders that are in different components of F. Such collusion can occur only if these two colluders share a piece of data in L *when they collude.* As Algorithm 2 proceeds knowledge of different connected components of F is combined, whenever possible, by constructing a path that visits every node in these components. Algorithm 2 stops either when a collusion path is found or no further combination is possible (in which case the collusion problem has no solution). We now describe the algorithm in more detail.

Recall the graph F that describes all events that are initially enabled in $w(0)$. The algorithm starts by calling Algorithm 1 to identify a connected component of F whose combined initial information contains the target information set T. If it succeeds it returns a collusion path. Otherwise Algorithm 1 will have initialized the data structure *GROUP* to specify, for each connected component of F, a path through all nodes in the connected component and their combined initial information. To simplify exposition in what follows we assume that each connected component of F has more than a single node. The results can be easily extended to allow single-node components as well.

For any path ρ the variable $\text{TAIL}[\rho]$ represents the last recipient on ρ. At any time, an element (ρ, i) of *GROUP* identifies a group of colluders, a path ρ that visits every colluder in the group and the combined initial information $i = f_2(\rho) = \Delta\ (\cup_{u \in \rho} w_u(0).I)$ of the group. The last recipient $\text{TAIL}[\rho]$ knows 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 fewer groups, until either a collusion path is found or no further combination is possible. The collusion problem has no solution in the latter case. By combining we mean construction of a path, from the spanning paths of the two individual connected components, that visits every colluder in both components, as explained next.

Lemma 3 implies that two colluders in different connected components of F can collude only if they share a piece of data in L (condition (6)). The algorithm searches for two elements in *GROUP*, identified by (ρ_1, i_1) and (ρ_2, i_2), such that $\text{TAIL}[\rho_1]$ knows a colluder v in ρ_2 and shares a piece of data L with $\text{TAIL}[\rho_2]$. Note that $\text{TAIL}[\rho_1]$ may not share a piece of data in L with v to allow them to collude, nor may it know $\text{TAIL}[\rho_2]$ to collude with it. But by construction $\text{TAIL}[\rho_2]$ knows all colluders involved in ρ_2, in particular v. $\text{TAIL}[\rho_2]$ transfers all its knowledge to v. Then v and $\text{TAIL}[\rho_1]$ indeed share a piece of data in L and can collude. Hence

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

is a path in Θ. 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 3 and 4. If i_{12} contains T, then the algorithm stops and returns the collusion path ρ_{12}. 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.

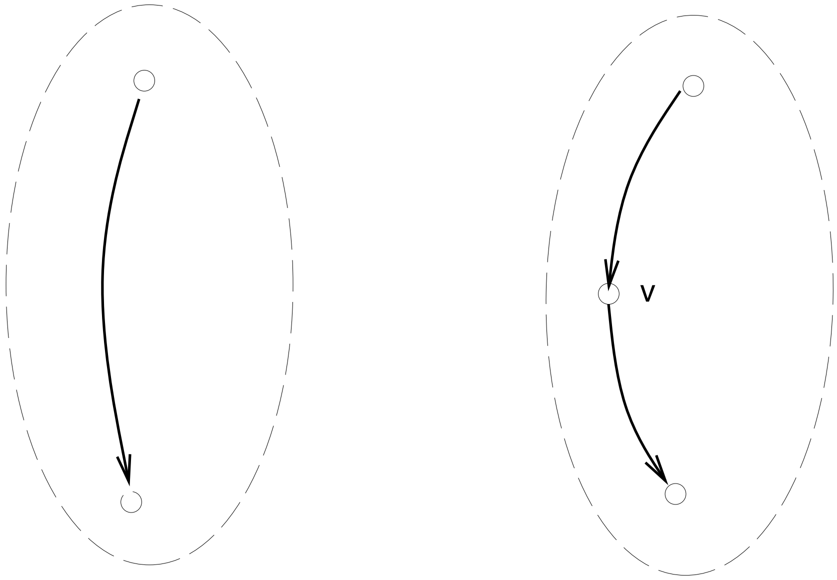

Figure 3: Elements (ρ_1, i_1) and (ρ_2, i_2) of $GROUP$

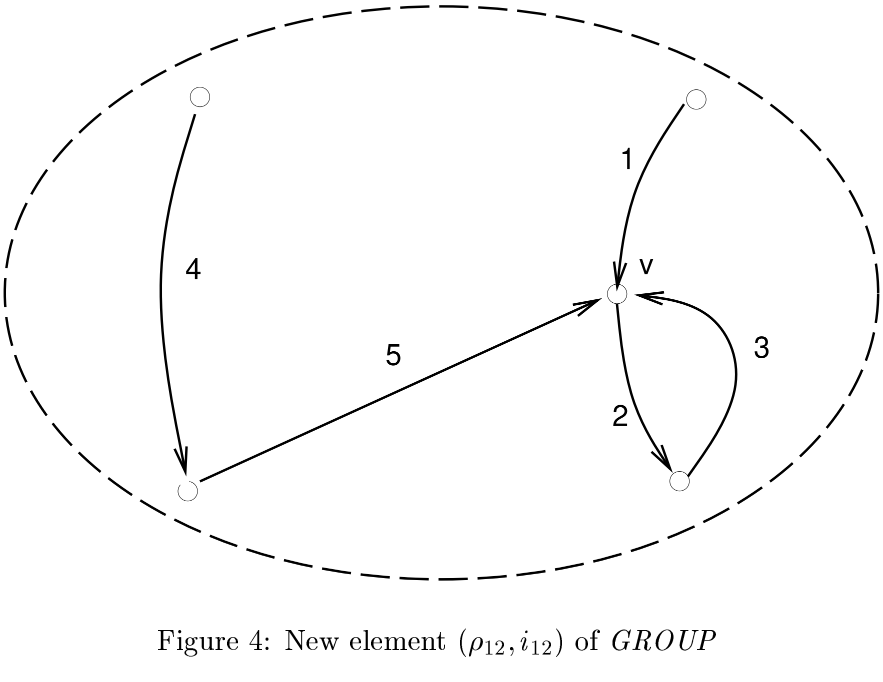

Figure 4: New element (ρ_{12}, i_{12}) of $GROUP$

We now present the algorithm. Recall that ρ denotes a path or the set of colluders involved in the path, depending on the context.

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 a collusion path ρ is found, then Return(ρ)
3. While there are distinct elements (ρ_1, i_1), (ρ_2, i_2) of $GROUP$ satisfying
   $i_1 \cap \rho_2 \neq \phi \quad$ and $\quad i_1 \cap i_2 \cap L \neq \phi$
   - 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.[2]

Theorem 7 *The algorithm terminates. Moreover it returns a collusion path if the collusion problem has a solution and* NIL *otherwise.*

Proof. If there is a connected component of F whose combined initial information contains T, then Algorithm 1 will return a collusion path and Algorithm 2 terminates in step 2. Otherwise Algorithm 1 will return the linked list $GROUP$ that contains one element for each connected component of F. Each time the 'while' loop in step 3 is entered, the linked list $GROUP$ is shortened by one. Hence Algorithm 2 must terminate, either inside the 'while' loop with a collusion path, or in step 4 with NIL. We are left to show that the collusion problem has a solution if and only if a collusion path is returned.

Suppose Algorithm 2 terminates in step 3 with a ρ_{12}. Then $i_{12} \supseteq T$. Hence to show that ρ_{12} is a collusion path, we will show by induction on n, the number of times the 'while' loop is entered, that ρ_{12} is a path in Θ and that its last recipient knows i_{12}.

Consider $n = 1$. As proved in the last section ρ_1 and ρ_2 are paths in Θ and their last recipients know i_1 and i_2, respectively. We show portion by portion that ρ_{12} is a path. To show that $\rho_2 \cdot (\text{TAIL}(\rho_2), v)$ is a path, note that $v \in \rho_2$ and hence $v \in w_{\text{TAIL}(\rho_2)}(\rho_2).I$. Moreover $\phi \neq w_v(\rho_2).N \subseteq w_{\text{TAIL}(\rho_2)}(\rho_2).N$ since v and $\text{TAIL}(\rho_2)$ are in the same connected component of F. Hence the event $(\text{TAIL}(\rho_2), v)$ is enabled in state $w(\rho_2)$,

[2] If both $\rho_1 = u$ and $\rho_2 = v$ consist of single nodes, then the assignment for ρ_{12} in step 3 should be modified to be $\rho_{12} \leftarrow (u, v)$. If only $\rho_1 = u$ is a single node, then $\rho_{12} \leftarrow \rho_2 \cdot (\text{TAIL}[\rho_2], v) \cdot (u, v)$. If only $\rho_2 = v$ is a single node, then $\rho_{12} \leftarrow \rho_1 \cdot (\text{TAIL}[\rho_1], v)$. The proof for these cases is simpler and omitted.

i.e., $\rho_2 \cdot (\text{TAIL}(\rho_2), v)$ is a path. Since ρ_1 is a path starting from state $w(0)$, so is it starting from state $w(\rho_2 \cdot (\text{TAIL}(\rho_2), v))$, i.e, $\rho_2 \cdot (\text{TAIL}(\rho_2), v) \cdot \rho_1$ is a path. Finally since $\text{TAIL}(\rho_1)$ knows i_1 and v knows i_2, the condition guarding the entry to the 'while' loop guarantees that the last event $(\text{TAIL}(\rho_1), v)$ is enabled in state $w(\rho_2 \cdot (\text{TAIL}(\rho_2), v) \cdot \rho_1)$. Hence ρ_{12} is a path; moreover its last recipient v knows $i_{12} = \Delta(i_1 \cup i_2)$.

When $n > 1$, $\text{TAIL}(\rho_2)$ and v may be in different connected components of F; yet the event $(\text{TAIL}(\rho_2), v)$ is still enabled because the set of message identifiers that $\text{TAIL}(\rho_2)$ has includes those that v has as $v \in \rho_2$. With this observation the same argument as for $n = 1$ goes through to show that ρ_{12} is a path and that its last recipient v knows $i_{12} = \Delta(i_1 \cup i_2)$. Hence when the algorithm terminates in step 3, ρ_{12} is a collusion path.

Conversely suppose a collusion path ρ exists; without loss of generality we may assume that ρ is minimal. We now show that a ρ_{12} can be constructed according to the recipe in step 3. Hence the algorithm cannot terminate with a NIL. The collusion path ρ must involve nodes that belong to different connected components of F, for otherwise, there is a connected component of F whose combined initial information contains T and Algorithm 1 would have returned a collusion path. Hence suppose the set of colluders in ρ belongs to two connected components of F; the argument can be easily extended to the case where ρ involves more than two connected components.

Let $\rho = (s_1, r_1)(s_1, r_1) \cdots (s_n, r_n)$. Since ρ is minimal only the last recipient r_n knows T by Lemma 2. Without loss of generality, suppose that the two connected components correspond to (ρ_1, i_1) and (ρ_2, i_2) of *GROUP* after Algorithm 1 terminates and before step 3 is entered for the first time. Suppose that in each of the first $k - 1$ events of ρ both the sender and the receiver belong to the same connected component of F, and (s_k, r_k) is the first event that crosses between the two components, i.e., $s_k \in \rho_1$ and $r_k \in \rho_2$, say. We claim that the path ρ_{12} so constructed from ρ_1 and ρ_2 in step 3 would have been a collusion path.

To see this, let ρ^{k-1} be the first $k - 1$ events of ρ. Since (s_k, r_k) is enabled after ρ^{k-1} has been followed, we must have

$$r_k \in w_{s_k}(\rho^{k-1}).I \tag{12}$$

$$\phi \neq w_{s_k}(\rho^{k-1}).I \cap w_{r_k}(\rho^{k-1}).I \cap L \tag{13}$$

But $s_k \in \rho_1$ and $r_k \in \rho_2$, and hence $i_1 = f_2(\rho_1) \supseteq w_{s_k}(\rho^{k-1}).I$ and $i_2 = f_2(\rho_2) \supseteq w_{r_k}(\rho^{k-1}).I$. Then (12) implies that $i_1 \cap \rho_2 \neq \phi$ and and (13) implies that $i_1 \cap i_2 \cap L \neq \phi$. Therefore ρ_{12} can be constructed as specified in step 3. Moreover $\rho_1 \cup \rho_2 \supseteq \rho$ by choice of ρ_1 and ρ_2, and hence $i_{12} = \Delta(i_1 \cup i_2) \supseteq f_2(\rho) \supseteq T$, i.e., ρ_{12} is indeed a collusion path.

Finally it is possible that the execution of the algorithm combines one of the two elements, say (ρ_1, i_1), with some other elements, instead of combining it with (ρ_2, i_2). In this case since information of colluders can only increase the new combined element of *GROUP* remains eligible to be combined with (ρ_2, i_2). Hence the algorithm must eventually terminate either with a collusion path that involves colluders in $\rho_1 \cup \rho_2$ or some other collusion path. This completes our proof. $\square$

5 Least Cost Collusion

Suppose a cost is incurred when a pair of users collude. In this section we present the negative result that the problem of determining a collusion path that incurs the least cost, when one exists, is NP-hard.

This problem is of interest for two reasons. First it is usually impossible to completely eliminate successful collusion. For instance if *every* participant colludes any information can be uncovered. Hence it might be more practical to design protocols that eliminate successful collusions that are inexpensive. Second cryptographic protocols are used to keep information apart in order to protect privacy. However it may sometimes be necessary to link information, e.g., for law enforcement purposes or to uncover an audit trail, by forcing protocol participants to collude. It is then desirable to determine a collusion path that incurs the least cost.

The algorithms presented earlier determine whether the collusion problem has a solution and produces a collusion path when it does, but the collusion path produced is generally not the least-cost one. For instance the algorithm in §3 constructs a collusion path that visits *every* node in a connected component of the graph F, even when a subset of the colluders in the connected component suffices. This is illustrated in Low, Maxemchuk and Paul (1996).

To narrow our problem, we will consider the special case considered in §3 where $L = \phi$. We argue that the algorithm there is polynomial and hence can be first used to determine a connected component of graph F from which a collusion path can be constructed, if any. We will show that, even for the special case where $L = \phi$, determining the least cost collusion is NP-hard with respect to this connected component.

We claim that the complexity of the algorithm in §3 is polynomial in $|U|+|T|+M+J$, where U is the set of colluders, T is the target information set, $M := |\cup_{u\in U} \ w_u(0).N|$, and $J := |\cup_{u\in U} \ w_u(0).I|$. The construction of graph F in line 1 of the algorithm takes $O(M^2)$ time. We make the *assumption* that the verification of the condition $\Delta(B) \ \supseteq \ T$ takes $O(|B||T|)$ time. Then lines 4 and 5 of $\mathsf{SEARCH}(s)$ takes $O(J|T|)$. Since $\mathsf{SEARCH}(s)$ is an adaptation of a breadth-first search, each call to $\mathsf{SEARCH}(s)$ takes $O(k_n^2 + J|T|)$ time (see Cormen, Leiserson and Rivest, Chapter 23 (1993)), where k_n is the number of nodes in the connected component of F that contains s when $\mathsf{SEARCH}(s)$ is invoked for the nth time. Since $\mathsf{SEARCH}(s)$ is called as many times as the number m of connected components of F, the algorithm takes time

$$O(M^2 + (k_1^2 + J|T|) + \cdots + (k_m^2 + J|T|))$$

where $\sum_{n=1}^m k_n = |U|$. Noting that $m \le |U|$ and $\sum_{n=1}^m k_n^2 \le |U|^2$, this expression is reduced to $O(M^2 + |U|^2 + J|T||U|)$.

Hence suppose we have already determined a connected component $G = (V, E)$ of the graph F using the polynomial algorithm in §3. Suppose that

$$f_2(V) \ = \ \Delta(\cup_{u\in V} \ w_u(0).I) \ \supseteq \ T \qquad (14)$$

and hence a collusion path can be determined from G. Suppose for each pair $(u,v) \in E$, collusion between them costs $c(u,v)$ in either direction. We are interested in the following decision problem. Recall that the environment for the collusion problem is $(U, D, K, L = \phi)$.

Least-cost Collusion

INSTANCE: A connected component $G = (V, E)$ of F, initial state $(w_u(0), u \in V)$ for nodes in V, target information set T, such that (14) is satisfied; an integer k.

QUESTION: Does there exist a collusion path ρ such that the total collusion cost $\sum_{(u,v)\in\rho} c(u,v)$ is less than k?

To show that the least-cost collusion problem is NP-hard, we consider a special case where $c(u,v) = c$, independent of (u,v). Then the total collusion cost $\sum_{(u,v)\in\rho} c(u,v) = c|\rho|$ where $|\rho|$ is the number of colluders in ρ. Note that we only need to consider collusion paths ρ of the form in Theorem 5. Hence the total cost is minimized if ρ is a collusion path that involves the minimum number of colluders.

Minimum Collusion Path

INSTANCE: A connected component $G = (V, E)$ of F, initial state $(w_u(0), u \in V)$ for nodes in V, target information set T, such that (14) is satisfied; an integer k.

QUESTION: Does there exist a *connected subgraph* $G' = (V', E')$ of G such that

$$f_2(V') \ = \ \Delta(\cup_{u\in V'} \, w_u(0).I) \ \supseteq \ T \tag{15}$$

and $|V'| \le k$?

In the definition of the minimum collusion path problem we only need to identify a connected subgraph of G with the smallest number of colluders. Application of the polynomial algorithm in §3 to the subgraph will then yield a collusion path that involves the minimum number of colluders. When collusion costs are equal it is also the least-cost path. Hence the minimum collusion path problem is indeed a special case of the least-cost collusion problem.

We now show that the minimum collusion path problem is NP-complete, by reducing the well-known NP-complete set-covering problem to it.

Theorem 8 *Suppose that the verification of the condition $\Delta(B) \supseteq T$ takes polynomial time. Then the minimum collusion path problem is NP-complete.*

Proof. We will first show that the minimum collusion path problem is in the class NP. Then we will show how to reduce any instance of the set-covering problem to an instance of the minimum collusion path problem. Since the minimum set-covering problem is NP-complete, the theorem will be proved.

Given a candidate subgraph $G' = (V', E')$ of G, we only need to verify condition (15) and that $|V'| \le k$. Both can be done in polynomial time under the assumption of the theorem. Hence the least-cost collusion problem can be verified in polynomial time and is therefore in NP.

An instance $(X, \mathcal{X}, l)$ of the set-covering problem (see Cormen, Leiserson and Rivest, Chapter 37 (1993)) consists of a finite set X and a family $\mathcal{X}$ of subsets of X, such that every element of X belongs to at least one subset in $\mathcal{X}$. The question is to find a subset $\mathcal{Y} \subseteq \mathcal{X}$ such that

$$X = \bigcup_{C \in \mathcal{Y}} C \tag{16}$$

and that $|\mathcal{Y}| \leq l$. Given an instance $(X, \mathcal{X}, l)$ of the set-covering problem, construct the following instance $(G, w(0), T, k)$ of the minimum collusion path. Let $G = (V, E)$ be a *fully connected* undirected graph that has a node v for every subset $X(v)$ of X in $\mathcal{X}$. For the initial state, let $w_v(0).I = X(v)$ for each $v \in V$, and let $w_v(0).N$ be such that the graph derived from $w(0)$ is fully connected (i.e., every pair (u, v) shares a unique message in state $w(0)$). Let $T = X$ and $k = l$. Treat the elements of X as unencrypted elements so that $\Delta(B) = B$ for any subset B of $\cup_{v \in V} w_v(0).I$. Then there is a solution $\mathcal{Y}$ to the set-covering problem such that (16) holds if and only if the subgraph $G' = (V', E')$ of G induced by $V' = \{v \mid X(v) \in \mathcal{Y}\} \subseteq V$ is a solution to the minimum collusion path problem. Hence the set-covering problem can be reduced to the minimum collusion path problem. Moreover the reduction takes polynomial time. Hence the minimum collusion path problem is NP-complete. □

Since the minimum collusion problem is a special case of the least cost collusion problem, we have the following corollary.

Corollary 9 *The least cost collusion problem is NP-hard.*

6 Conclusion

We have formulated a collusion problem that determines whether a group of colluders can collectively discover a target set of unencrypted information starting from their initial knowledge. We have designed a simple algorithm that determines whether a collusion problem has a solution and when it does, computes a collusion path.

We view a cryptographic protocol as defining a process by which information is transferred among some users and hidden from others. The algorithm presented here can be applied to determine whether a subset of protocol users can discover, through collusion, the information that is to be hidden from them during or after a protocol's execution. An example is given in Low, Maxemchuk and Paul (1996).

Our algorithm does not necessarily compute a collusion path that involves the minimum number of colluders. We have shown, however, that this problem is NP-complete. More generally, suppose there is a collusion cost associated with each pair of colluders. We have shown that the least-cost collusion problem is NP-hard.

Acknowledgement: We are grateful to anonymous referees for helpful suggestions that led to Lemma 3 and that improved the presentation of the paper.

References

Burrows, M., Abadi, M. and Needham, R. (1990), A logic of authentication, *ACM Transactions on Computer Systems*, 8(1):18–36.

Chaum, D. (1981), Untraceable electronic mail, return addresses, and digital pseudonyms, *Communications of the ACM*, 24(2):84–88.

Cormen, T., Leiserson, C. and Rivest, R. (1993), *Introduction to Algorithms*, The MIT Press.

Dolev, D., Even, S. and Karp, R. (1982), On the security of Ping-Pong protocols, *Information and Control*, 55:57–68.

Denning, D. and Sacco, G. (1981), Timestamps in key distribution protocols, *Communications of the ACM*, 24(8):533–536.

Dukach, S. (1992), SNPP: A Simple Network Payment Protocol, in *Proceedings of the Computer Security Applications Conference*, San Antonio, TX.

Dolev, D. and Yao, A. (1983), On the security of public key protocols, *IEEE Transactions on Information Theory*, IT-29(2):198–208.

Federrath, H., Jerichow, A. and Pfitzmann, A. (1996), Mixes in mobile communication systems: Location management with privacy, in *Proc. Workshop on Information Hiding*, Springer LNCS 1174, Ross Anderson (Ed.), 121–135.

Gong, L., Needham, R. and Yahalom, R. (1990), Reasoning about belief in cryptographic protocols, in *Proceedings of the 1990 IEEE Symposium on Security and Privacy*, pp. 234–248.

Kemmerer, R. (1989), Analyzing encryption protocols using formal verification techniques, *IEEE Journal on Selected Areas in Communications*, 7(4):448–457.

Kemmerer, R., Meadows, C. and Millen, J. (1994), Three systems for cryptographic protocol analysis, *Journal of Cryptology*, 7:79–130.

Low, S. and Maxemchuk, N. (1996), Modeling cryptographic protocols and their collusion analysis, in Ross Anderson, editor, *Proc. of First International Workshop on Information Hiding*, pp. 169–186, Springer Verlag, Vol. 1174 of Lecture Notes in Computer Science.

Low, S. and Maxemchuk, N. (1997), An algorithm to compute collusion paths, in *Proc. of Infocom'97.* Kobe, Japan.

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

Meadows, C. (1991), A system for the specification and analysis of key management protocols, in *Proceedings of the 1991 IEEE Symposium on Security and Privacy*, pp. 182–195.

Millen, J. (1984), The Interrogator: A Tool for Cryptographic Protocol Security, in *Proceedings of the 1984 IEEE Symposium on Security and Privacy*, pp. 134–141.

Maxemchuk, N. and Low, S. (1995), The use of communications networks to increase personal privacy, in *Proceedings of Infocom'95*, pp. 504–512.

Moore, J. (1988), Protocol failures in cryptosystems, *Proceedings of the IEEE*, 76(5):594–602.

Needham, R. and Schroeder, M. (1978), Using encryption for authentication in large networks of computers, *Communications of the ACM*, 21(12):993–999.

Needham, R. and Schroeder, M. (1987) Authentication revisited, *ACM Operating System Review*, 21(1):7–7.

Pfitzmann, A., Pfitzmann, B. and Waidner, M. (1991), ISDN-MIXes – Untraceable Communications with very Small Bandwidth Overhead, in *Proc. IFIP/Sec'91*, pp. 245–258.

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

Simmons, G. (1985), How to (selectively) broadcast a secret, in *Proceedings of the 1985 IEEE Symposium on Security and Privacy*, pp. 108–113.

Simmons, G. (1994), Proof of soundness (integrity) of cryptographic protocols, *Journal of Cryptology*, 7(2):69–77.

Tatebayashi, M., Matsuzaki, N. and Newman, D. (1989), Key distribution protocol for digital mobil communication systems, in G. Brassard, editor, *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pp. 324–333. Springer–Verlag, New York.