Scenarios and Covert channels: another game...

Loïc Hélouët [1]

IRISA/INRIA, Campus de Beaulieu, 35042 Rennes Cedex, France

Marc Zeitoun [2]

LIAFA, Case 7014, 2 place Jussieu 75251 Paris Cedex 05, France

Aldric Degorre [3]

ENS Cachan, Antenne de Bretagne, Campus de Ker Lann, av. R. Schuman, 35 170 Bruz, France

Abstract

Covert channels are information leaks in systems that use resources to transfer secretly a message. They are a threat for security, performance, but also for a system's profitability. This paper proposes a new approach to detect covert channels from scenario models of protocols. The problem of finding covert channels in scenarios is first modeled as a game, in which a pair of malicious users $\{S, R\}$ is trying to transfer information while the rest of the protocol tries to prevent it. The messages transferred are encoded by behavioral choices at some precise moments, and decoded by a transducer whose input vocabulary is an observation of the system. We then characterize the presence of a covert channel as the existence of a winning strategy for $\{S, R\}$ and of a decoder.

Key words: Scenarios, games, covert channels.

1 Introduction

A covert channel is an information flow that violates a system's security policy. The term covert channel has first been introduced by [19]. Covert channels are considered as a threat for information systems, for several reasons. The first obvious reason is of course a security issue, as covert channels can be used to pass information secretly. Covert channels can also be an economical threat. They can be used to transmit information (very often at a low rate) using an existing system without paying for the service provided. Furthermore, they are often based on an obfuscated use of resources or functionalities, which heavily impacts the performances of a system.

[1] Email: loic.helouet@irisa.fr
[2] Email: mz@liafa.jussieu.fr
[3] Email: aldric.degorre@eleves.bretagne.ens-cachan.fr

Covert channels are often differentiated in terms of "storage channels" and "timing channels". A storage channel is a channel that uses a resource of a system to write information that can be read by a third party. A timing channel is a channel that modifies a system's response time in a way that can be observed by a third party. Information leaks are also characterized by their bandwidth, ie the number of bits transmitted through a covert channel per second. In [23], it is considered that no system should allow a rate greater than 100 bits per second, and for applications with high security requirements, this rate should not exceed 1 bit per second. Several recommendations [9,23] have defined policies to handle covert channels. It is recommended [23] to systematically try to detect covert channel with formal techniques. When a covert channel is discovered, it is then required to document it with scenarios of use, and to compute its bandwidth. It is generally admitted that closing all covert channels is impossible [22]. Very often closing means suppressing accesses to all resources used for information passing. It would then resume to avoiding all shared resources, message acknowledgments in protocols, and even internal clocks in computers! A more sensible solution is to add noise on a known covert channel (for example by accessing randomly the resources used) to limit its bandwidth. Another solution is to monitor covert channels use, hence allowing to take appropriate decisions when illegal information flows are detected.

A characterization of covert channels has first been proposed as a confinement problem in information systems. The confinement notion in information systems defines for each user the services he can access and the users he is allowed to communicate with. Covert channels can be used to transfer information outside this confinement zone. Security models for this vision of systems are defined in [4,3], and several approaches to detect indirect information flows have been proposed: shared resources matrices [18], axiomatic approaches [1]... More recently, the existence of covert channels has been defined through non-interference properties [12,10] as follows: there is non-interference between a set of users U and a set of users U' if and only if "what U does has no effect on what U' can see" [12]. Non-interference is also questioned [30] as the transfer of a single bit of information causes a non-interference violation. Several approaches to non-interference have been proposed, through typing [34] (a system contains an interference if it can not be correctly typed), or using process algebra [20]. Very often, non-interference approaches classify data and processes of a system according to two security levels, high and low. High-level processes can access any data in the system and communicate with all other high processes, while low-level processes are not allowed to read information from the high level, either directly or indirectly.

Note also that confinement and non-interference both assume that illegal information flows occur when the communicating parties are not allowed to send information to each other. However, illegal information flows may exist over legal information transfers. This kind of covert channel is often called "legitimate channel" [21]. A classical example is watermarking: it is possible to add covert information to a legal information flow just by altering some bits. However, the search for covert channels does not usually address the contents of legal data, which is more a steganography problem. Another possibility is to hide data in some useless parts of protocol frames to pass covert information in a legal data flow [28]. This kind of covert channel is frequent, but can be easily detected and closed. Another possibility in protocols is to use **the way information is passed** from a participant to another to encode information. This is not a confinement or non-interference problem, as the information flow is allowed between participants. This is not either a steganography problem, as the message transferred is not altered. Such a situation can be considered as a "protocol-based" covert channel. In fact, the protocol itself is used as a resource in more classical covert channels to store data. Detecting such covert channels is far more difficult, as they involve several messages in the protocol. Of course, protocol-based covert channels detection addresses the problem of illegal information flow in a less generic way than non-interference, as it makes an initial assumption on how data is encoded. However, data passing is not defined as the detection of an altered behavior, as participants to a protocol-based covert channel all behave normally.

For a more complete bibliography on covert channels, interested readers are referred to [31]. This paper proposes a scenario framework to detect potential "protocol based" covert channels. Using scenarios has several advantages: first, scenarios are often the first information one can obtain about a system's behavior. They are also used to describe systems requirements. Detecting covert channels at early stages of a system's development can help modifying the systems when modifications can still be done at reasonable cost. A second reason for the use of scenarios is that several recommendations ask to document covert channels use with such models. Studying covert channels directly from scenarios provides immediately examples to document an illegal information flow.

In addition to these considerations, scenarios can be used as a partial knowledge of a system's behavior. Indeed, it is often very hard to obtain a model of a system. Scenarios are good candidates to capture most common behaviors of a system. Then, from this model, a covert channel analysis can be performed. It may be argued that scenarios model only a part of a system, and hence may miss some illegal flows. The study we propose in this paper does not claim to be exhaustive, and it is generally admitted that a gap always exists between a model and its implementation. This gap may be due to implementation details, or to simplifications in the model. Consequently, a model-based study can miss some covert channels, or a contrario exhibit unrealistic scenarios. Our scenario based approach has the same drawback, and only reveals "potential covert channels", the existence of which should be tested on a real implementation of a protocol. However, one should keep in mind that scenarios are intended to represent in an abstract way executions that will be present in an implementation (if they are considered as a set of requirements) or that exist in an implementation (if they represent output traces of an implemented system). For this reason, we think that a covert channel identified on scenarios has many chances to be usable in an implementation.

The approach proposed is to start from a scenario description of a system. From this description, a covert channel is modeled as a game in which a pair of corrupted users $\{S, R\}$ try to send information while the rest of the protocol is trying to prevent this information flow. A covert channel exists if $\{S, R\}$ has a winning strategy in the game, and if R can decode the message that is transmitted with a transducer built from the strategy. The main contribution of the paper is an algorithm that partitions the game's arena into two subsets of vertices: one subset Y from which information encoding is possible, and its complementary X, from which information encoding is not ensured. If Y is not empty, then it is shown that a winning strategy to encode information of arbitrary size exists from any vertex of Y.

This approach is close to a definition of non-interference for scenarios, but presents several advantages. First, covert channels are not characterized as the possibility to send a single bit, as often in non-interference frameworks, but rather as the possibility to encode and decode a message of arbitrary size. In addition to this, using strategy with memory allows for the introduction of some intelligence in attackers' behaviors. Another interest of our approach is that it does not need to consider security levels or a distinction between high and low security levels, which allows to consider legitimate channels.

The paper is structured as follows. Section 2 defines the scenario notation we use in the paper. Section 3 gives an example of how information can be encoded using a protocol's functionalities. Section 4 recalls some basic definitions for game theory that will be used to characterize covert channels. Section 5 characterizes potential covert channel presence in scenarios, and gives an algorithm to detect them. Section 6 shows how to build a decoder for a given covert channel, and section 7 concludes this work. Due to a lack of space, an example of covert channel and proofs of propositions and theorems are included in an appendix.

2 Scenarios basics

Scenario languages have met a great interest this last decade. Several languages have been proposed recently: Message Sequence Charts [17], Live Sequence Charts [14], UML Sequence diagrams [24]... All these languages share a common view of behavior representation, namely basic chronograms that are then composed by means of several operators. Scenarios describe possible executions as causal dependencies between occurrences of events in a system. Of course, each language has its own semantics subtleties, but basically, scenarios can be defined as compositions of partial orders. The idea to compose partial orders to describe systems behaviors is not new [25]. However, the recognition of scenarios as useful languages is a recent phenomenon. Before 1992, scenarios were only considered for output traces for distributed systems. After the definition of the language MSC'96 [17,27,26,29] and of UML sequence diagrams [24], scenarios have been used to capture requirements. However, as many graphical languages, scenarios can not be used to design exhaustively a system. They are more dedicated to the representation of abstract and incomplete behaviors. An usual trend is to model typical executions of a system, or a contrario exceptional cases by means of scenarios, hence covering a large subset of possible behaviors. So, even if incomplete, scenarios are still relevant to detect properties of a system.

For the rest of the paper, we have considered Message Sequence Charts. However, the approach proposed by this paper certainly adapts to other scenario formalisms. Message sequence charts is composed of two kind of diagrams. Basic chronograms (called basic Message Sequence Charts -or simply bMSC for short) are composed by means of High-level Message Sequence Charts, roughly speaking bMSC-labeled automata.

Definition 2.1 A *bMSC* is a tuple $M = (E, \leq, A, I, \alpha, \phi)$ where E is a set of events, $\leq$ is a partial ordering between events, A is a set of action names, I is a set of processes called instances, $\alpha : E \to A$ is a function associating an action name to each event, and $\phi : E \to I$ is a function associating a locality to each event. For a bMSC M, we will define as $Min(M)$ the set of minimal events for the causal order relation, that is, the set of events with no causal predecessor.

Partial orders alone are not sufficient to model interesting behaviors, and some operators such as choice, loops, and sequence have been proposed. Sequential composition mainly consists in merging two bMSCs along their common instance axis. This is defined more formally in the following definition.

Definition 2.2 The *sequential composition* of two bMSCs M_1 and M_2 is the bMSC $M_1 \circ M_2 = (E_1 \uplus E_2, \leq_{1\circ 2}, A_1 \cup A_2, I_1 \cup I_2, \alpha_1 \cup \alpha_2, \phi_1 \cup \phi_2)$, where $\leq_{1\circ 2} = \big(\leq_1 \uplus \leq_2 \uplus \{(e_1, e_2) \in E_1 \times E_2 \mid \phi(e_1) = \phi(e_2)\}\big)^*$.

The approach proposed in the paper mainly focuses on the relations between events executed by an instance called the sender and events executed by another instance called the receiver of a covert channel, and by the causal relationships between them. Hence, we will need the notion of projection on an instance, which consists in hiding all events of other instances in a bMSC.

Definition 2.3 The *projection* of a bMSC $M = (E, \leq, A, I, \alpha, \phi)$ on an instance $i \in I$ is the bMSC $\pi_i(M) = (E' = \phi^{-1}(i), \leq \cap E' \times E', A, \{i\}, \phi_{|E'})$. As we assume a total ordering on instance axes, we will often denote the projection of M on instance $i \in I$ by the word $w_{\pi_i}(M) = e_1 \cdots e_n$ such that $\{e_1, \ldots, e_n\} = \phi^{-1}(i)$ and $\forall p < q, e_p < e_q$.

As already mentioned, bMSCs alone are not expressive enough to design interesting behaviors. They have to be extended with operators. A common and widely accepted way of composing bMSC is called High-level Message Sequence Charts [29], a kind of bMSC automata.

Definition 2.4 A *HMSC* is a tuple $H = (N, \longrightarrow, \mathcal{B}, n_0)$, where N is a set of nodes, n_0 is a specific node called the "initial node", $\mathcal{B}$ is a set of bMSCs, $\longrightarrow \subseteq N \times \mathcal{B} \times N$ is a set of transitions. In addition, we sometimes distinguish *sink nodes*, which are nodes without successors.

A *path* in a HMSC is a word $p = t_1 \cdot t_2 \cdots t_k \in \longrightarrow^*$. One associates to p the bMSC $O_p = l(t_1) \circ \cdots \circ l(t_k)$, the sequential composition of the labels $l(t_i)$ of transitions t_i in p.

Definition 2.5 A *choice node* in a HMSC H is a node n with at least two outgoing transitions. A choice node n is *local* [5,15] if and only if there is a unique instance $i \in I$ such that for all paths p leaving n, $\phi(min(O_p)) = \{i\}$. When a choice n is local, we will say that n *is controlled* by i.

In this paper, we only consider local HMSCs, *ie*, whose choices are all local. Such HMSCs have been frequently considered and enjoy nice algorithmic properties [11]. Algorithms to detect the locality property can be found in [5,15]. For the rest of the paper, we will also need some basic notions on graphs, that are recalled below.

Let $G = (V, E)$ be a graph. A *subgraph* G' of G is a pair $G' = (V', E')$ such that $V' \subseteq V$ and $E' \subseteq E \cap V' \times V'$. Considering HMSCs as graphs we can define a sub-HMSC of a HMSC $H = (N, \longrightarrow, \mathcal{B}, n_0)$ as a HMSC $H' = (N', \longrightarrow', \mathcal{B}', n_0')$ where $N' \subseteq N$, $\mathcal{B}' \subseteq \mathcal{B}$, and $\longrightarrow' \subseteq \longrightarrow \cap N' \times \mathcal{B}' \times N'$. The complete subgraph on a set of nodes N' will be called the *restriction* of G to N', and noted $G_{|N'}$.

A *strongly connected component* of G is a subset V' of V such for all $v_1, v_2 \in V'$ there is a path from v_1 to v_2. For a given graph, a decomposition into strongly connected components can be performed in linear time using Tarjan's algorithm [33]. Let us call $[v]$ the strongly connected component containing $v \in V$. A *quotient graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ can be computed from G, where $\mathcal{V}$ is the set of connected components of G, and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V} = \{([v_1], [v_2]) \mid \exists (v_1, v_2) \in E \wedge [v_1] \neq [v_2]\}$. Since $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is an acyclic graph, one can define the *depth* of a strongly connected component c as $d(c) = max\{|p|$ such that p is an acyclic path leading to c in $\mathcal{G}\}$.

3 Sending information with decisions nodes and transducers

Let us consider a simple communication protocol that transfers data using short or long data packets, that are chosen according to the network's congestion. Now, consider a network in which two users, *Sender* and *Receiver* are allowed to send data to each other, using this protocol. By choosing long or short data packets, Sender and Receiver can encode 0 and 1, and hence add information over a legal data flow. Clearly, this is not a steganography problem, as the data transferred is not altered. The situation can not either be considered as violating a confinement or a non interference property, as Sender and Receiver are allowed to communicate. Of course, congestion adds noise to the covert channel, which becomes inefficient if the network gets saturated.

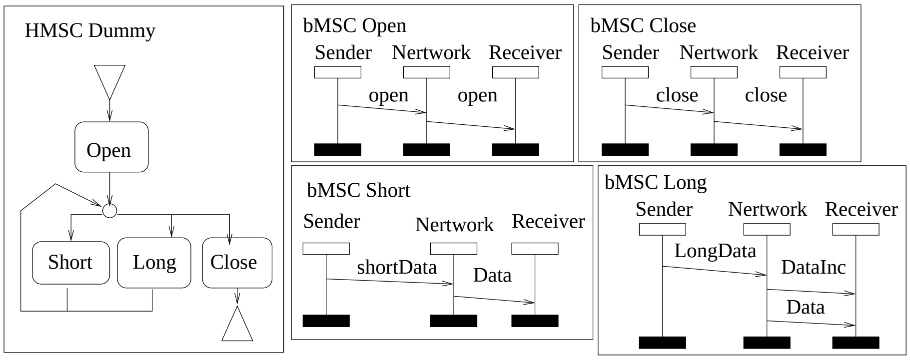

Fig. 1. The DummyIP protocol

A HMSC depicting the main functionalities of our protocol is given in Figure 1. Once a session is opened, a user can send short data packets, which are forwarded as data packets to the receiver, or long data packets, which are split into two kinds of packets: $DataInc$, meaning incomplete $Data$ packets, followed by $Data$ Packets. From this description of our protocol, one can see that instance sender can choose to emit short data packets to encode value 0, and long data packets to encode 1. The receiving instance can then decode a message by observing the respective order between $Data$ and $DataInc$ packets. Note that information transfer is only possible if Sender and Receiver have agreed on a protocol for sending information. Note also that as a message can be of arbitrary length, one needs to be able to perform an arbitrary number of decisions for encoding it. Hence, for our protocol, one should not consider session closing as an encoding possibility. Note also that being able to perform two decisions is not sufficient to transmit information. The choices must have different observable consequences for the receiver. Suppose that $DataInc$ packet are replaced by *Data* packets. Then, upon reception of 3 data packets, it would become impossible for a receiver to be sure whether message "0.1" or message "1.0" was sent. Hence, despite the two possible decisions, the protocol can not be used to transfer reliably covert data.

So far, we have mentioned that covert information was encoded by decisions of a sending instance, and decoded by a receiving instance. In fact, the receiving instance must observe what happens in the system, and deduce the choices performed by the sender. Such a decoder can be formally defined as a finite state transducer [2,6], that takes as input the observation of the system, and outputs a sequence of decisions, *ie*, the decoded message.

Definition 3.1 A *Transducer* is a tuple $T = (Q, \Sigma_{in}, \Sigma_{out}, \delta, Q_I, F)$ where Q is a set of states, Σ_{in} is an input alphabet, Σ_{out} is an output alphabet, $\delta \subseteq Q \times \Sigma_{in}^* \times \Sigma_{out}^* \times Q$, $Q_I \in Q$ is a set of initial states of T, $F \subseteq Q$ is a set of final or accepting states.

A transducer can be considered as a machine that reads letters from Σ_{in} and produces outputs in Σ_{out}^*. The outputs of a transducer T for a word $w \in \Sigma_{in}^*$ will be denoted $T(w)$. A transition of a transducer can be considered as a rewriting step. Note that transitions can contain empty words either on input or on output side, *ie*, transitions of the kind $(q, \epsilon, w_{out}, q')$ and $(q, w_{in}, \epsilon, q')$ are allowed.

Definition 3.2 A transducer is *functional* if and only if $|T(w)| \leq 1$ for all $w \in \Sigma_{in}^*$. Note that functionality is different from the notion of determinism: a non deterministic transducer can be functional, and a deterministic transducer can be non functional. The transducer of Figure 2-a is not deterministic, as two transitions labeled by $a/0$ leave state 0. However, it is functional, as the output for $a.b$ is 0.1, even if two different paths of the transducer accept $a.b$. The transducer Figure 2-b is deterministic, but not functional, as the word abc generates two different outputs, 0.1.1 and 1.0.1. In fact, functionality concerns the output produced for a given word, and not the paths that allow this rewriting. Checking this property is decidable. [6,32] has proved that for a transducer with m states, deciding whether a transducer is functional resumes to verifying that for all pairs of states and all words of size $\leq 2m$ the output generated by the transducer was unique. While this procedure is exponential, there is also a quadratic time algorithm [8] for checking this property.

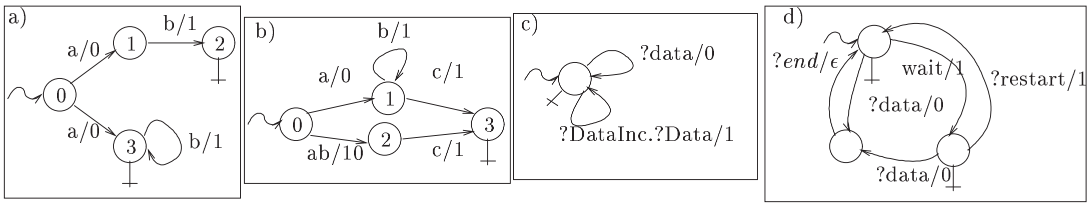

Fig. 2. Transducers examples

A simple strategy to encode data is to perform choices that ensure that the protocol will eventually get back to the same decision point. This was proposed as a first covert channel identification procedure in [16]. The main idea behind this definition of covert channels is that corrupted users can exploit iterations in protocol's behavior to get back to states from which a decision can be taken by the sender in the covert channel, and which causal consequences can be observed by the receiver.

The Dummy IP example allows for a transfer of information from a single control point. The associated decoder is given by transducer of Figure 2-c. However, this kind of attack can be easily monitored as users iterate systematically the same behaviors. Furthermore, if encoding from a choice node implies an error scenario (for example requesting a missed packet), then the number of errors for this user will differ from all users, which may help detecting an attack. Furthermore, error scenarios are less likely to occur than others. Hence, covert information transfer might result in an highly unlikely scenario to occur.

With a minimal knowledge of a system, an attacker may however find some more elaborated strategies, for which monitoring becomes more difficult. These strategies consist in moving the systems towards multiple decision points where information transmission is always possible. Consider the example of Figure 3. Let us decide that encoding 0 at choice node n_1 can be performed by choosing scenario *Data*, and 1 by scenario *Wait*. Following the encoding strategy defined previously, one can decide to choose systematically to get back to decision node n_1 by executing scenario *Restart* after choosing scenario *Wait*, hence allowing another bit transmission. However, at node n_2, executing *Restart* or *Resume* is another encoding possibility. It is also very easy to define the decoder for this more elaborated strategy (it is given Figure 2-d). So, a covert channel can be implemented if the communicating parties agree on a set of decision nodes that will be used to encode information, and on which behavior must be executed to encode a bit in each decision node. In fact, these encoding strategies can be considered as a game between a pair Sender/Receiver, and the rest of the protocol. The attackers win if they can transmit any message of arbitrary length, and the protocol wins if he can prevent messages from being passed.

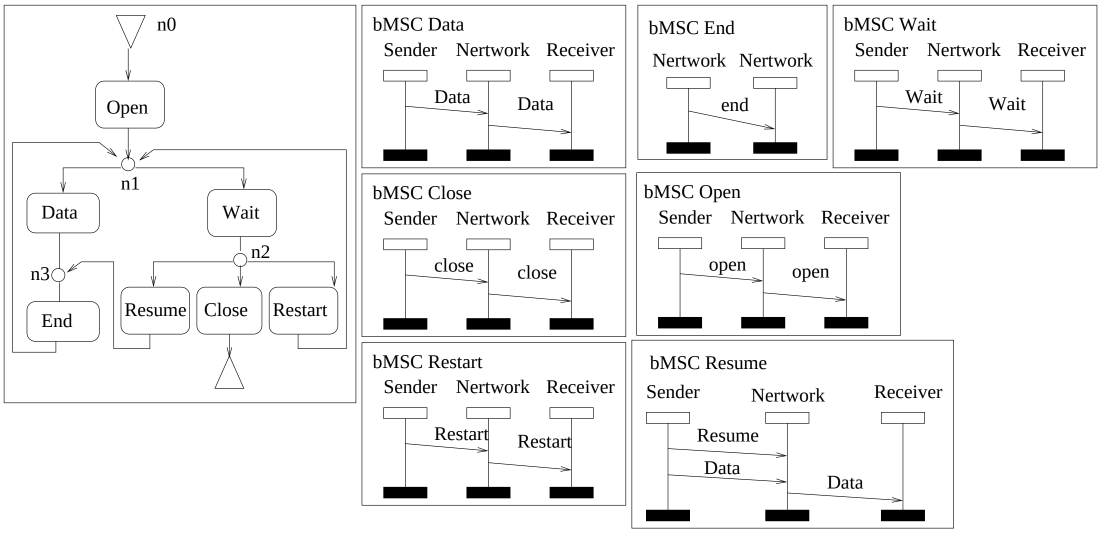

Fig. 3. Protocol containing a covert channel involving two decision points

4 Games

This section recalls some basic notions of game theory. Most of this material is given with more details in [13]. To avoid the multiplication of formalisms, we have slightly adapted the traditional definition of arenas to include labels on edges. One can consider covert information passing as a two players game where the attacker wins if he transfers information to its peer, and the protocol wins if it can prevent this information from being reliably passed. As information to be passed between covert parties is of unbounded size, message passing will be tightly related to infinite behaviors of HMSCs. In addition to this, as covert information transfer have to use infinitely often nodes that allow information encoding, our covert game will be best described as a Muller game.

An *arena* is a tuple $\mathcal{A} = (V_0, V_1, \Sigma, E)$ where V_0 and V_1 are sets of vertices, Σ is an alphabet, and $E \subseteq (V_0 \cup V_1) \times \Sigma^* \times (V_0 \cup V_1)$ is a set of labeled edges. We note $V = V_0 \cup V_1$, and for an edge $e = (v, w, v') \in E$, we call v the origin of e and v' the goal of e. Arenas are used to model games with two players 0 and 1. An arena is represented by a graph, with round and square vertices respectively associated to 0 and 1 vertices (see Figure 6 for an example). Edges represent "moves": a move consists in passing from one vertex to another. In round vertices, player 0 chooses the next move, and in square vertices, player 1 chooses the next move. A *play* in an arena is a maximal path in this arena. If a play Π is infinite, we denote by $\mathit{Inf}(\Pi)$ the set of vertices that are visited infinitely often. A Muller game is a pair $G = (\mathcal{A}, \mathcal{F})$, where $\mathcal{A}$ is an arena, and $\mathcal{F} \subseteq 2^{V_0 \cup V_1}$ is a Muller condition. Player 0 is the *winner* of a play Π if:

- $\Pi = e_0 \cdots e_l$ is finite and the goal of e_l is a 1-vertex from which player 1 cannot move.
- Π is an infinite game, and $\mathit{Inf}(\Pi) \notin \mathcal{F}$.

Otherwise, player 1 wins the play. Let A be an arena and let $\sigma = \{0, 1\}$. Let $f_\sigma : V^* V_\sigma \longrightarrow 2^E$ be a partial function. A play prefix $\Pi = e_0 e_1 \cdots e_l$, with each move of the form $e_i = (v_i, w, v_{i+1}) \in E$, *conforms to* f_σ if for all $0 \leq i \leq l$ such that $v_i \in V_\sigma$, $f(v_0 \cdots v_i)$ is defined and $v_{i+1} \in f(v_0 \ldots v_i)$. A play Π conforms to a function f_σ if and only if all its finite prefixes conform to f_σ.

A function f_σ is a *strategy* for player σ on $U \subseteq V$ if f_σ is defined for any prefix of a game that conforms to f_σ, starts from a vertex $u \in U$, and does not end in a sink state for player σ. Let $G = (A, \mathcal{F})$ be a game, and f_σ be a strategy on U for player σ. We say that f_σ is a *winning strategy* for player σ on U if all plays starting from U and that conform to f_σ are winning plays for σ. A player σ *wins a game* G on $U \subseteq V$ if and only if he has a winning strategy on U. A winning strategy f_σ is *maximal* if it is maximal among all winning strategies for the inclusion relation.

At this stage, the relation between games and covert channel may not appear clearly. Let us try to gather some ideas that will be used hereafter to define covert channel strategies. Nodes of HMSCs will be considered as vertices of an arena. The tricky point is to associate vertices to a player and to define winning strategies in the so defined game. Covert channel users can be considered as player 1 in this game, and the rest of the protocol as player 0. First, as covert channel users may want to transfer unbounded amount of information, the protocol should never reach a sink node. Hence, sink nodes in a HMSC will be associated to player 1 . Second, winning plays for player 0 (the protocol) will be plays in which information transmission is impossible for player 1.

Defining information encoding as a game strategy may seem surprising as the protocol is not playing. In fact, the protocol's initial purpose is not to counter any cover channel, but to provide a service, transport information, etc. Therefore, an attacker can be seen as playing against a protocol implementation, but a protocol is not playing against the attacker. However, even if the protocol plays random moves, it may be sufficient to prevent information passing. We will hence try to exhibit a strategy for corrupted users when the protocol may play the best move by chance. Note however that it is possible to win a game by playing randomly... Clearly, the encoding example described in previous section is a simplistic game with only one state.

5 Potential Covert Channels

In this section, we propose an algorithm to find a transmission strategy using a complete protocol. A pair sender/receiver will win a game if it has a strategy to transmit information. Consequently, when the protocol is in a deadlocked state, no information can be transmitted, and $\{S, R\}$ lose the game. $\{S, R\}$ also lose when the protocol remains in a loop in which no information can be transmitted. The main idea behind the algorithm is to partition the set of choice nodes of a HMSC into winning and losing nodes for a player, *ie*, find nodes from which a player has a strategy to win the game. Intuitively, a "good" move for covert channel users will be a move after which player 1 will eventually be able to encode data and keep the control of the protocol, or a move that will force the opponent to perform a move for which we still have a winning strategy.

Let X be a set of nodes in an arena and σ be a player. Recall that the σ-attractor of X is the set of nodes from which player σ can force its opponent to move to a node of X. It is defined by $Att_\sigma(X) = \bigcup_k Att^k_\sigma(X)$, where:

$$Att^0_\sigma(X) = X \qquad Att^{n+1}_\sigma(X) = Att^n_\sigma(X) \cup \{m \in V_\sigma \mid \exists n \in Att^n_\sigma(X), \exists e = (m, w, n) \in E\}$$
$$\cup \{m \in V_{1-\sigma} \mid e = (m, w, n) \in E \Rightarrow n \in Att^n_\sigma(X)\}$$

The notion of attractor will be useful to capture the notion of vivacity needed to transfer messages of unbounded size. The definitions proposed so far will be used mainly to detect who can control a part or another of a protocol. However, controlling a protocol is not sufficient to make sure that data can be transmitted. For the simple cycle-based strategy described Section 3, data encoding was possible if different choices had different observable consequences. As we are now defining more elaborated strategies, we have to define the possibility of passing data in a more general way.

Definition 5.1 Let H be a HMSC and R be an instance of H. For a given transition $t = (n, M, n')$ of a H, we will define as $\lambda_R(t) = \alpha(w_{\pi_R}(M))$ the *observation of* t *on* R. The definition can be extended to any path p of H writing $\lambda_R(t \cdot p) = \lambda_R(t) \cdot \lambda_R(p)$.

As already mentioned, the receiver in a covert channel tries to deduce the actions performed by the sender from its observation of the running system. Roughly speaking, λ_R defines the sequences of events observed when a scenario is executed.

Definition 5.2 Let H be a HMSC, C be a set of nodes of H, R be an instance, and p be a path of C. We define as $\Lambda_R(C, p) = \{\lambda_R(p.s) \mid p.s \text{ is a path of } H_{|C}\}$, the set of words generated by paths starting with a prefix p. One can see $\Lambda_R(C, p)$ as the possible observations on R when path p is imposed as a start. Note that while the trace language of a HMSC is in general not regular, its projection on a given instance is regular (hence so is $\Lambda_R(C, p)$).

In this definition, the path p should be interpreted as a sequence of choices performed by the sender, and $\Lambda_R(C,p)$ as the set of possible visible consequences (from the receiver's point of view) when only transitions of the connected component C are chosen.

Definition 5.3 Let D be a strongly connected component of a HMSC H. We will say that a choice node m *encodes no information* in a strongly connected component D and write $E(D,m)$ iff either S does not control m, or for all $t=(m,b_1,n_1)$, $t'=(m,b_2,n_2)$ such that $n_1,n_2 \in D$ we have $(\epsilon \cup \Lambda_R(D,t)) \cap \Lambda_R(D,t') \neq \emptyset$ or $(\epsilon \cup \Lambda_R(D,t')) \cap \Lambda_R(D,t) \neq \emptyset$

More informally, $E(D,m)$ holds iff it is impossible for R to differentiate two choices of S starting from m. Furthermore, when $E(D,m)$ holds, then it is possible to loop forever on vertex m without transferring information. Note that $E(D,m)$ can be due to some kind of non-determinism, but also to the emptiness of observable actions for a receiver. When $E(D,m)$ holds for all nodes of D, then the strongly connected component D cannot be used to transmit data. If a protocol can force a pair sender/receiver to stay in a strongly connected component D where no event is observable or the sequence of events observed is always the same independently from the choices of the sender, then no data transmission is possible in D. Note that $E(\{n\},n)$ holds trivially when n is a sink node.

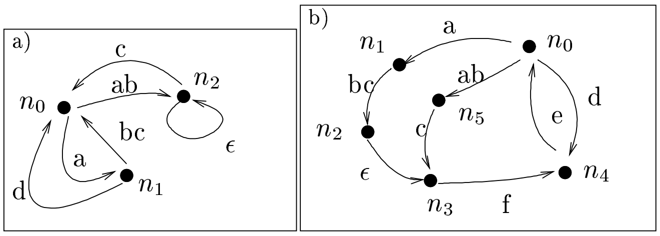

Fig. 4. Property E

Consider the graph in Figure 4-a, depicting a strongly connected component $D=\{n_0,n_1,n_2\}$ of a HMSC. Nodes are HMSC nodes, and transitions from one node to another are labeled by $\lambda_R(t)$. $E(D,n_0)$ holds, as the language observed by R after transition (n_0,a,n_1) is $\{abc;ad\}^*$, while that observed after transition (n_0,ab,n_2) is $\{abc\}.\{abc;ad\}^*$. $E(D,n_2)$ also holds, as n_2 contains a non observable transition (labeled by ϵ). However, if n_1 is controlled by the sending instance, $E(D,n_1)$ does not hold, as for the two transitions leaving n_1, the observable consequences generated form disjoint languages. Hence, D can be used to encode information. If we consider the strongly connected component $D'=\{n_0,n_1,n_2,n_3,n_4,n_5\}$ in Figure 4-b, $E(D',x)$ trivially holds for $x \in \{n_1,n_2,n_3,n_4,n_5\}$, as these nodes only have a single outgoing transition. If n_0 is controlled by the sending instance, $E(D',n_0)$ does not hold, and starting with a transition labeled with a or with d is sufficient to generate observable information, even if the output $\{abcfe\}$ can be generated via two different choices.

Now that zones where data transmission is possible are identified, let us compute the winning zones for a HMSC. For the sending instance, winning zones are zones where data transmission is possible without losing control of the channel, and for the protocol, winning zones are zones from which infinite data transmission is impossible. Obviously, sink nodes are winning zones for the protocol. In a similar way, all connected components controlled by the protocol are also winning for this player. Finally, a zone that is not entirely controlled by the protocol but does not allow for data transmission is also a winning zone for the protocol. The 0-attractor of these 3 cases is also a winning subset for the protocol (where 0 is the player representing the protocol).

The covert channel game is an iterated reachability game. In fact, corrupted users want the protocol to pass infinitely often in nodes where information encoding is possible. Let $H = (N, \longrightarrow, \mathcal{B}, l)$ be a HMSC. The arena associated to H is defined as $\mathcal{A}_H = (N_0, N_1, \Sigma, E)$, where $N_0 = \{n \in N \mid n \text{ choice node} \wedge \text{neither } S \text{ nor } R \text{ control } n\}$, $N_1 = N - N_0$, $\Sigma = \lambda_R(\mathcal{B})$, $E = \{(n, \lambda_R(b), n')) \mid (n, b, n') \in \longrightarrow\}$. As stated before, we want to define a covert channel as a Muller game allowing infinite encoding, that is, whose infinite plays can traverse infinitely often encoding nodes. We first partition the arena in two sets, Y from which an infinite encoding will always be possible, and X from which no encoding or only bounded encoding will be possible.

algorithm: Partition($\mathcal{A}_H$)
  $X = \emptyset$ (Winning set for the protocol)
  $Y = \emptyset$ (Winning set for the pair Sender/Receiver)
  $CC = Tarjan(H)$ `/* computes the strongly connected components of H */`
  $Stack$ = Stack with connected components ordered by depth (deepest on top)
  while $Stack \neq \perp$ **do**
    $D = POP(Stack)$
    if $D \cap Y \neq \emptyset$ **then** `Case (1)`
      $D := D - Y$
      if $D \neq \emptyset$ **then**
        $PUSH(Tarjan(D))$
      end if
    else
      if $\exists m \in D, \neg E(D, m)$ **then**
        if $D \subseteq Att_{RS}(\{m\} \cup Y)$ **then** `Case (2)`
          $Y = Att_{RS}(\{m\} \cup Y)$
        else `Case (3)`
          $PUSH(Tarjan(D \cap Att_{RS}(\{m\} \cup Y)))$
          $PUSH(Tarjan(D - Att_{RS}(\{m\} \cup Y)))$
        end if
      else `Case (4)`
        `/* here,` $D \cap Y = \emptyset$ `and` $E(D, m)$ `for all` $m \in D$ `*/`
        $X = X \cup D$
      end if
    end if
  end while

The algorithm computes a set of nodes X from which the protocol has a strategy to prevent S and R from passing information of arbitrary length, and Y, the complement of X. If Y is not empty, then there is potentially a covert channel from S to R in the protocol. This algorithm studies successively connected components in the arena associated to a scenario description. For each connected component, several cases depicted Figure 5 may occur. The important invariant of the algorithm is that at all stages Y is the $\{S, R\}$-attractor of the encoding states found so far. Note also that any transition leaving a connected component leads to a vertex located in a deeper component. Hence, the classification of nodes of a connected component as X or Y node at depth n can rely on classification of nodes of all components at depth $n+1$. Every time a component D is studied, it is either stable with respect to the set Y computed so far, and can be added to X or Y, or unstable, and is split into sub-components.

In case 1, $D \cap Y$ is not empty. Nothing can be deduced yet for D, and the study must be refined for all connected components of $D - Y$.

In case 2, $\exists m \in D, \neg E(D, m)$ and $D \subseteq Att_{RS}(\{m\} \cup Y)$. Hence, from any node of D, S and R can force the protocol to get back to m, or move towards a node of Y. So, D can be added to Y, and as we want Y to be closed by attraction, $Y = Att_{RS}(\{m\} \cup Y)$.

In case 3, $\exists m \in D, \neg E(D, m)$, as in case 2, but $D \not\subseteq Att_{RS}(\{m\} \cup Y)$. So there are some nodes of D from which it is still possible for the protocol to avoid m or all nodes of Y. So, we have to refine the search in connected components of $D \cap Att_{RS}(\{m\} \cup Y)$ and $D - Att_{RS}(\{m\} \cup Y)$.

In case 4, D is a connected component, and does not contain nodes allowing information transfer. Furthermore, as $D \cap Y$ is empty, it is impossible to leave D to reach a part of the arena where data transfer is possible. Hence, D can be added to X.

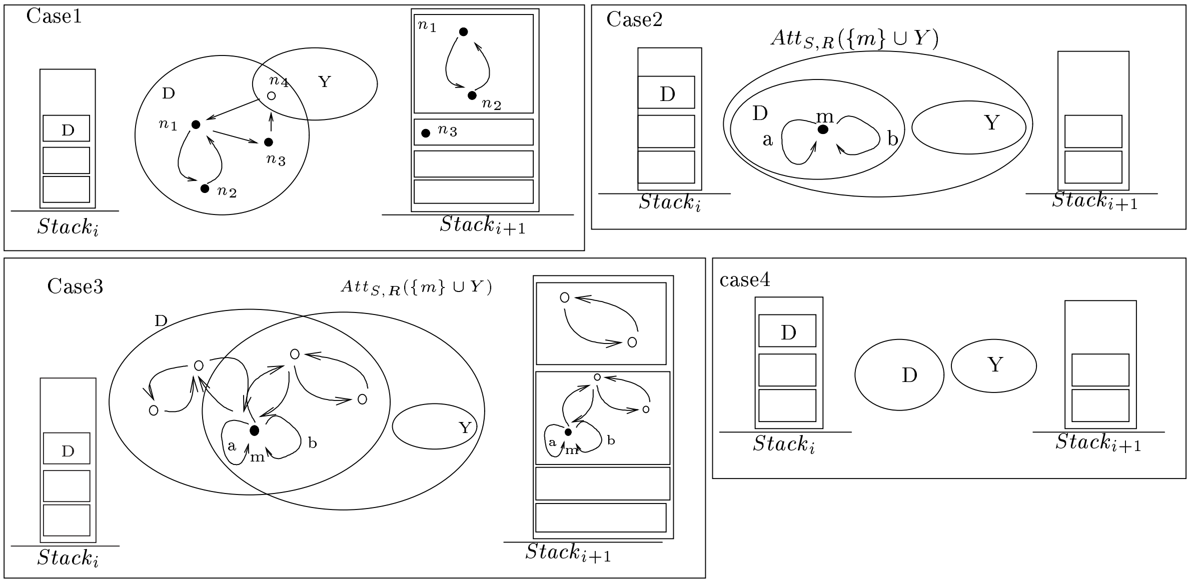

Fig. 5. Different configurations of the algorithm

Let us apply this algorithm on the example of Figure 6. Strongly connected components CC_i of this arena are identified by dashed rectangles. Nodes controlled by the pair Sender/receiver are symbolized by squares, and nodes controlled by the rest of the protocol are symbolized by circles. Sink nodes are supposed to be controlled by the pair sender/receiver. They are obviously winning nodes for the protocol, as it is the attacker's turn to move, and no move is allowed from these nodes.

We first start with $CC_7.CC_5.CC_6.CC_4.CC_3.CC_8.CC_2.CC_1$ in the stack, and with $X =$

$\emptyset, Y = \emptyset$. The first five steps of the algorithm are trivial, since $E(CC_i, n)$ holds for all nodes $n \in CC_i$ and $i \in 3..7$. (Observe indeed that no choice node of these CC_i is controlled by $\{S, R\}$.) Then, case 4 applies, and it just consists in adding vertices of these connected components to X. After step 5, we hence have $X = \{5, 6, 7, 8, 9, 10, 11, 12\}$. Step six pops CC_8, which is of different nature, as it contains node 13, and $\neg E(CC_8, 13)$. Here, case 3 applies, we have to separate CC_8 into two connected components, $CC_{8,1} = CC_8 \cap Att_{RS}(\{13\}) = \{13, 15\}$ and $CC_{8,1} = CC_8 - Att_{RS}(\{13\}) = \{14, 16\}$. Step 7 and 8 will then add $CC_{8,1}$ to Y as it conforms to case 2, and $CC_{8,2}$ to X (case4). Similarly, CC_2 will be added to Y and CC_1 to X. The algorithm terminates with $X = \{1, 5, 6, 7, 8, 9, 10, 11, 12, 14, 16\}$ and $Y = \{2, 3, 4, 13, 15\}$.

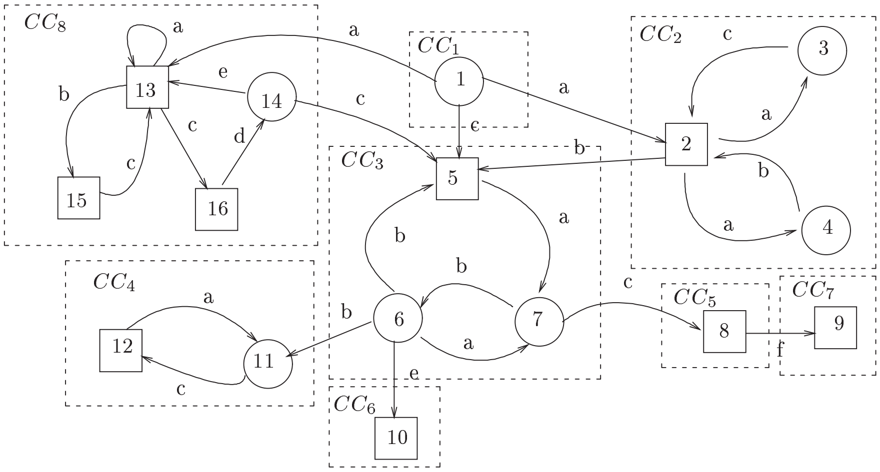

Fig. 6. Computing X and Y

Now that we have computed a set from which a node enabling information encoding can be reached in a finite number of steps, we also have to define winning subsets (in the Muller sense). Staying in Y is not sufficient to ensure that information is transmitted: Y is not accurate enough. We cannot either require that all nodes y such that $\neg E(D, y)$ for some connected component D appear infinitely often, as the game can be stuck in a peculiar connected component and remain a winning game for the pair $\{S, R\}$. Hence, we can define the winning condition of our covert channel game as the Muller condition $Win(Y) = 2^Y - \{P \in 2^Y \mid \forall n \in P, E(P, n)\}$. A node $y \in Y$ such that $\exists W \in Win(Y)$ and $\neg E(W, y)$ will be called an *encoding node*. The winning subsets of $Win(Y)$ define the parts of the protocol that can be used by the sending instance to create a covert channel. For an attacker, staying in a restricted part of the protocol behaviors (typically a small strongly connected component of the HMSC graph) can be sufficient to transfer information. It is convenient, as the sender needs fewer memory to implement a covert channel[4], but makes a channel more vulnerable to monitoring applications that compare users behaviors to profiles of honest users. Hence, a "good" strategy for a sender is to use the larger possible part of the protocol to send information while mimicking the behavior of honest users.

[4] The simple Büchi condition $Win(Y) = Y - \{y | \forall W \subseteq Y, E(W, y)\}$ is sufficient to exhibit strategies which conform plays traverse some encoding nodes infinitely often. However, a Muller condition allows for strategies using larger parts of the specification (that can not always be obtained by Büchi conditions). With such strategies, covert channel use are more likely to be similar to honest behaviors.

Proposition 5.4 *Let $GH = (\mathcal{A}_H, Win(Y))$ be the Muller game associated to H, with $Y \neq \emptyset$. From GH, one can compute a maximal strategy f_Y for nodes controlled by S and R such that:*

i) any infinite run in Y that conforms to f_Y passes infinitely often through a set of encoding nodes.

ii) For all encoding node y and all $v_0, \dots v_l$ such that $v_0 \cdots v_l.y$ is a path, $f_Y(v_0 \cdots v_l.y)$ contains at least two transitions t_1,t_2 such that $\Lambda_R(Y, t_1) \cap \Lambda_R(Y, t_2) = \emptyset$.

proof: If $Y \neq \emptyset$, then from the construction algorithm, $\forall y \in Y$, $\exists D, \exists y' \in D$, such that $\neg E(D, y')$ and $y \in Att_{RS}(y')$. So, for all $y \in Y$, there is a (positional) strategy leading from y to an encoding node. Hence, there exists a strategy f that leads from any node to an encoding node, and from an encoding node to another encoding node. As the number of encoding nodes is finite, any play that conforms to this strategy passes infinitely often through a set of encoding nodes. Note however that this strategy f is not necessarily maximal.

Let us build the parity automata P_H associated to our Muller game. Let us call K the size of P_H. The states of our parity automata will be of the form $v_1 \dots v_k.y$, with $k < K$. We can observe that a strategy for this parity automata is in fact a sub-graph of P_H. We can also note that a strategy in P_H will be a winning strategy if it does not allow passing infinitely often through a set of nodes $\bigcup_{i \in I} v_{1_i} \dots v_{k_i}.y_i$ that does not contain encoding nodes. Hence, a strategy f will be a winning strategy iff the subgraph of P_H associated to f does not contain a "silent cycle", *ie*, a cycle on a set of states $\bigcup_{i \in I} v_{1_i} \dots v_{k_i}.y_i$ that does not contain encoding nodes.

So, from a winning strategy f, one can add a chain leading from a node to another if it does not create silent cycles. The new strategy computed will be bigger (in terms of transitions allowed) than f.

After a certain number of additions, it will not be possible to add a single transition to the strategy without creating a silent cycle, and the strategy obtained will be maximal.

This proves point i) of proposition. Let us now prove point ii). Adding any transition from a state $s = v_1 \dots v_k.y$ where y is an encoding node to another state $s'v'_1 \dots v'_k.y'$ to a winning strategy f does not create silent cycles, as all the new cycles created pass through s, and are hence not silent. $\Box$

Note that maximal winning strategies are not unique. Note also that this proof just establishes the existence of a maximal winning strategy, but does not provides the most efficient algorithm to compute such strategy. In fact, to solve our problem, a strategy only have to remember the set of visited states, and not their order of appearance, which lets us suppose that the problem can be solved as the research of a positional strategy on an automata of size $n.2^n$ instead of the usual $n.!n$ for Muller games.

Intuitively, the existence of two transitions with different observable consequences means that a bit can be encoded. Finding a winning strategy for $\{S, R\}$ just indicates that there is a possible information transfer, not that this transfer is always decipherable by the receiver. For this, we have to build a transducer that observes the behaviors of the system and outputs a decoded message.

6 Building a transducer for message decoding

The message received by a receiving instance can be seen as a continuous flow of events, and its decoding will be correct only when this flow can be properly segmented to deduce the sequence of choices that have been performed by a sender. Let Y be a winning set computed from $H = (N, \longrightarrow, \mathcal{B}, n_0)$, P_H be the parity game associated to the Muller game on Y with winning condition $Win(Y)$. Let $f_Y : Y^* \longrightarrow 2^{\longrightarrow}$ be a maximal strategy computed from P_H. The transducer associated to f_Y is the transducer $T_{f_Y} = (Q, \Sigma_{in}, \Sigma_{out}, \delta, Q_I, F)$ where:

- $Q = Q_I = dom(f_Y)$, $\Sigma_{in} = \bigcup_{b \in \mathcal{B}} \lambda_R(b)$
- $\Sigma_{out} = \{\epsilon\} \cup \{t = (y, b, y') \mid \neg E(X, y) \text{ for some } X \in Win(Y)\}$
- $\delta = \{(w.y, \sigma_{in}, t, w'.y') \mid w.y \text{ and } w'.y' \text{ are vertices of } P_H, (w.y, w'.y) \text{ is an edge of } P_H,$
  $\exists t = (y, b, y') \in Y \times \mathcal{B} \times Y,$
  $\sigma_{in} = \lambda_R(b) \wedge t = (y, b, y') \in f_Y(w.y) \wedge \exists X \in Win(Y), \neg E(X, y)\}$
  $\cup\{(w.y, \sigma_{in}, \epsilon, w'.y') \mid w.y \text{ and } w'.y' \text{ are vertices of } P_H, (w.y, w'.y) \text{is an edge of } P_H,$
  $\exists t = (y, b, y') \in Y \times \mathcal{B} \times Y, \sigma_{in} = \lambda_R(b) \wedge \forall X \in 2^Y, E(X, y)\}$
- $F = \{w.y \in Dom(f_Y) \mid \neg E(X, y) \text{ for some } X \in Win(Y)\}$

The transducer T_{f_Y} reads observations of the receiving instance R, and outputs the sequence of choices at encoding nodes that may have engendered this observation (ie a list of transitions). When a transition is not leaving an encoding node, it does not participate in the covert transmission, and it is replaced by ϵ. The states of the transducer is the set of vertices computed for the parity game P_H, and the transitions are labeled by a pair σ_{in}/σ_{out} when a transition is allowed by the strategy f_Y. When a transition $t = (s, b, s')$ leaves an encoding vertex, σ_{out} is the name of the transition, otherwise $\sigma_{out} = \epsilon$. In both cases, σ_{in} represents the observation of the receiver ($\sigma_{in} = \lambda_R(b)$).

Definition 6.1 Let T be a set of transitions leaving nodes controlled by S, and such that $\forall t = (y, b, y')$, $\exists t' = (y, b', y'')$ originating from the same node. One can define a partition $\mathcal{P}$ of T into two subsets T_0 and T_1 such that for all $y, \exists t_0 = (y, b, y') \in T_0 \wedge \exists t_1 = (y, b, y'') \in T_1$.

For a given partition of a set T of transitions, let us define the *interpretation* $[\![\ ]\!]_{\mathcal{P}}$ as the function $[\![\ ]\!]_{\mathcal{P}} : X \cup \{\epsilon\} \longrightarrow \{T_0, T_1, \epsilon\}$ that associates the corresponding partition to each transition of T, ϵ to ϵ, and also ϵ to each transition of $X - T$.

Theorem 6.2 *Let H be a HMSC, and $\mathcal{A}_H$ be the associated arena. Let Y be the winning set computed on $\mathcal{A}_H$, and $Win(Y)$ be the winning conditions included in Y. Let f be a strategy for $Y, Win(Y)$ and T_{f_Y} be the transducer associated to Y and f. If T_{f_Y} is functional, then there exists an interpretation $[\![\ ]\!]_{\mathcal{P}}$ such that:*

$$\forall y \in Y, \forall m \in \{0,1\}^*, \exists p = t_1 \ldots t_k \textit{ with } t_1 = (y, b, y_1) \textit{ and } [\![T_{f_Y}(\lambda_R(p))]\!]_{\mathcal{P}} \equiv m$$

proof: by induction on the length of m.

For $m = 0$, for all y, there is a finite path p leading to a node y' such that $E(y')$. Hence, there are at least two transitions t_0, t_1 such that $\Lambda_R(Y, t1) \cap \Lambda_R(t_2) = \emptyset$. Hence, there is a partition $\mathcal{P} = \{T_0, T_1\}$ with $t_0 \in T_0$ and $t_1 \in T_1$, and then $[\![T_{f_Y}(\lambda_R(p.t_0))]\!]_{\mathcal{P}} \equiv 0$ and

$[\![T_{f_Y}(\lambda_R(p.t_1))]\!]_{\mathcal{P}} \equiv 1$.

Let us suppose that this property is satisfied for any message of size $\leq n$, and let us show that it is also true for a message of size $n+1$.

As m is of size $n+1$, m can be decomposed as $m = m'.\{0,1\}$, with $|m'| = n$. Hence, for any y there is a partition and a path $p = t_1 \ldots t_k$ such that $[\![T_{f_Y}(\lambda_R(p))]\!]_{\mathcal{P}} \equiv m$. Furthermore, the path p leads to a node y', for which there exists a path p_0 with $[\![T_{f_Y}(\lambda_R(p_0))]\!]_{\mathcal{P}} \equiv 0$, and a path p_1 with $[\![T_{f_Y}(\lambda_R(p_1))]\!]_{\mathcal{P}} \equiv 1$. Hence, for all y there is a path $p' = p_0$ or $p' = p_1$ from y such that $[\![T_{f_Y}(\lambda_R(p.p'))]\!]_{\mathcal{P}} \equiv m'$. $\square$

Intuitively, this theorem says that is T_{f_Y} is functional, the receiver can decode messages generated to it by the strategy of the pair $\{S, R\}$. It gives sufficient conditions for making possible an encoding of a message through decisions in a protocol It is easy to see why this condition is only sufficient. Consider the HMSC of Figure 7. A message $m \in \{0,1\}^*$ of arbitrary length can be encoded by scenario $Data^n.Close$, where n is the number which binary representation is m. We have only defined the property for an encoding of 0 and 1 from encoding nodes, which is sufficient to characterize information transmission. However, more accurate partitions of transition sets can be used to transfer more than 1 bit at each encoding node. If a decision node y has n_y outgoing transitions with pairwise disjoint observable languages, then a decision at node y can encode up to $log_2(n_y)$ bits of information.

Determining if a transducer is functional is quadratic [8]. However, as we are studying abstract and incomplete models, the size of the transducers is usually small. Note also that transducer functionality can be replaced by a simpler requirement. One can just ask the language on the receiving instance to be a code [7]. This property detects less covert channels, but can be computed very efficiently.

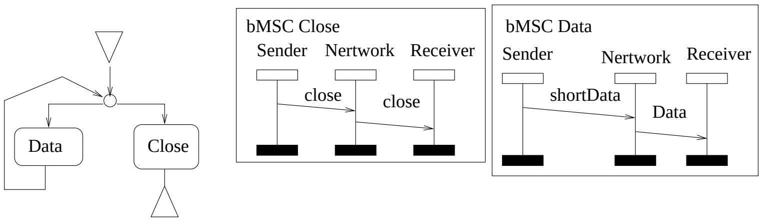

Fig. 7. A different encoding

7 Conclusion and future work

This paper shows how to characterize the presence of covert channels from a scenario representation H of a system. The first step consists in identifying a "live" subset of H from which a message encoding is possible in a bounded number of decisions. Then, searching a maximal strategy in the live subset of H and the construction of the transducer associated to this strategy allows us to determine if a non ambiguous message transmission is possible. The fact that the covert channel game is defined as a Muller game, and that strategies are maximal and non deterministic allows a wide range of behaviors that were possible in the initial specification. This increases the capacity of our channels, and makes online detection of obfuscated uses of a system harder (the more normally a system behaves in presence of corrupted users, the harder it is to detect a covert channel use).

Ambiguous transducers do not mean that covert information passing is impossible, but rather that the channel contains noise. A study to compute noisy covert channels capacity using information theory is currently undergone. The main difficulty to compute capacities and rates is that in the asynchronous systems depicted by our scenarios, all encodings are not performed in constant time. Note that to build an efficient strategy, one does not need to remember the order between visited nodes, but only the visited nodes since the last visit of an encoding node. This lets us suppose that a more efficient strategy may be found.

So far, we have considered centralized strategies, where R can take decision to help S transmitting data. This provides necessary conditions for an attack and synthesizes global scenarios exhibiting the channel. However, in a distributed framework, such a strategy might not be implementable without introducing additional ambiguity, due to the fact that instances $\{S, R\}$ only have a partial view of the system. Finally, we only considered a pair $\{S, R\}$ of attackers. An extension of this work is to study when a team of processes can create a covert channel. This includes several senders and receivers, and processes being able to act successively as senders or as receivers. However, if considering several senders seems very easy with our approach, having several receivers raises some more complicated issues, and potentially undecidable problems. When several receivers are considered for the encoded message, projections on a set of processes is not necessarily a regular language, and finding encoding nodes (which relies on language intersection emptiness) may become an undecidable problem.

References

[1] G.R. Andrews and R.P. Reitmans. An axiomatic approach to information flows in programs. *ACM transactions on Programming languages and Systems*, 2(1):56–76, January 1980.

[2] J.L. Autebert and L. Boasson. *Transductions Rationnelles.* Etudes et Recherches en Informatique. Masson, 1988.

[3] D.E Bell and J.J La Padula. Secure computer systems: a mathematical model. MITRE technical report 2547, MITRE, may 1973. Vol II.

[4] D.E Bell and J.J La Padula. Secure computer systems: mathematical foundations. Mitre technical report 2547, MITRE, march 1973. Vol I.

[5] H. Ben-Abdallah and S. Leue. Syntactic detection of process divergence and non-local choice in message sequence charts. In *Proc. of TACAS'97*, volume 1217 of *LNCS*, pages 259 – 274. Springer-Verlag, April 1997.

[6] J. Berstel. *Transductions and Context-Free-Languages.* B.G. Teubner, Stuttgart, 1979.

[7] J. Berstel and D. Perrin. *Theory of codes.* Academic Press, 2002.

[8] O. Carton, CH. Choffrut, and Ch. Prieur. Two quadratic time algorithms for functions defined by finite automata. Technical report, LIAFA, 2000.

[9] Common Criteria. Common criteria for information technology security evaluation part 3: Security assurance requirements. Technical Report CCIMB-99-033, CCIMB, 1999.

[10] R. Focardi, R. Gorrieri, and F. Martinelli. Non interference for the analysis of cryptographic protocols. In *Int. Colloquium on Automata, Languages and Programming (ICALP'00)*, number 1853 in LNCS, pages 744–755. Springer Verlag, July 2000.

[11] B. Genest, A. Muscholl, H. Seidl, and M. Zeitoun. Infinite-state High level MSCs: realizability and model-checking. In *Proc. Of ICALP'02*, number 2382 in LNCS, pages 657–668. Springer Verlag, 2002.

[12] J.A. Goguen and J. Meseguer. Security policies and security models. In IEEE Computer Society Press, editor, *Proc of IEEE Symposium on Security and Privacy*, pages 11–20, April 1982.

[13] E. Grädel, W. Thomas, and T. Wilke, editors. *Automata Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*. Number 2500 in LNCS. Springer Verlag, 2002.

[14] D. Harel and W. Damm. Lscs: breathing life into message sequence charts. Technical Report CS98-09, Weizmann Institute, April 1998.

[15] L. Hélouët and C. Jard. Conditions for synthesis of communicating automata from hmscs. In *5th International Workshop on Formal Methods for Industrial Cr itical Systems (FMICS)*, Berlin, april 2000. GMD FOKUS. http://www.gmd.de/publications/report/0091.

[16] L. Hélouët, M. Zeitoun, and C. Jard. Covert channels detection in protocols using scenarios. In *Proc. of SPV'03 Security Protocols Verification*, pages 21–25, sep. 2003.

[17] ITU-TS. *ITU-TS Recommendation Z.120: Message Sequence Chart (MSC)*. ITU-TS, Geneva, September 1993.

[18] R.A. Kemmerer. Shared ressources matrix methodology: an approach to indentifying storage and timing channels. *ACM transactions on Computer systems*, 1(3):256–277, 1983.

[19] B. Lampson. A note on the confinement problem. *Communication of the ACM*, 16(10):613–615, October 1973.

[20] G. Lowe. Quantifying information flow. In *Proceedings of the 7th European Symposium on Research in Computer Security(ESORICS)*, pages 18–31, 2002.

[21] J. Millen. 20 years of covert channel modeling and analysis. In *Proc of IEEE Symposium on Security and Privacy*, page 113, 1999.

[22] I. Moskowitz and M. Kang. Covert channels - here to stay ? In *Proceedings of COMPASS'94*, pages 235–243. IEE Press, 1994.

[23] NSA/NCSC. *A guide to Understanding Covert Channel Analysis of Trusted Systems*. Number NCSC-TG-030 [Light Pink Book] in Rainbow Series. NSA/NCSC, Nov. 1993.

[24] Object Management Group. Unified modeling language specification version 2.0: Superstructure. Technical Report pct/03-08-02, OMG, 2003.

[25] V. Pratt. Modeling concurrency with partial orders. *International Journal of Parallel Programming*, 15(1):33–71, 1986.

[26] M. Reniers. *Message Sequence Charts: Syntax and Semantics.* PhD thesis, Eindhoven University of Technology, 1998.

[27] M. Reniers and S. Mauw. High-level message sequence charts. In A. Cavalli and A. Sarma, editors, *SDL97: Time for Testing - SDL, MSC and Trends*, Proceedings of the Eighth SDL Forum, pages 291–306, Evry, France, September 1997.

[28] C.H Rowland. Covert channels in the tcp/ip protocol suite. Technical Report Tech. Rep. 5, First Monday, Peer Reviewed Journal on the Internet, July 1997.

[29] E. Rudolph, P. Graubmann, and J. Grabowski. Tutorial on Message Sequence Charts. *Computer Networks and ISDN Systems*, 28(12):1629–1641, 1996.

[30] P. Ryan, J. McLean, and J. Millen. Non-interference, who needs it ? In *Proceedings of the 14th IEEE Computer Security Foundations Workshop*, 2001.

[31] A. Sabelfeld and A.C. Myers. Language-based information-flow security. *IEEE Journal on selected areas in communications*, 21(1), Jan. 2003.

[32] M.P. Schützenberger. Sur les relations rationnelles. In *Automata Theory and Formal Languages*, number 33 in LNCS, pages 209–213, 1975.

[33] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal of Computing*, 1(2), 1992.

[34] D. Volpano and G. Smith. Eliminating covert flows with minimum typings. In *Proc. 10th IEEE Computer Security Foundations Workshop*, pages 156–168, June 1997.