Verifying Security Properties of Cryptoprotocols: A Novel Approach

Mohamed Saleh Mourad Debbabi
Concordia Institute for Information Systems Engineering
Concordia University
Montreal, Quebec, Canada
{m_saleh, debbabi}@ ciise.concordia.ca

Abstract

We model security protocols as a game tree using concepts of game semantics. Using this model we ascribe semantics to protocols written in the standard simple arrow notation. According to the semantics, a protocol is interpreted as a set of strategies over a game tree that represents the type of the protocol. Moreover, in order to specify properties of the model, a logic that deals with games and strategies is developed. A tableau-based proof system is given for the logic, which can serve as a basis for a model checking algorithm. This approach allows us to model a wide range of security protocol types and verify different properties instead of using a variety of methods as is currently the practice. Furthermore, the analyzed protocols are specified using only the simple arrow notation heavily used by protocol designers and by practitioners.

1. Introduction

Cryptoprotocols are a special type of communication protocols, they are used whenever the communicating agents (also called principals) would like to achieve a certain security-related objective. A basic assumption about cryptoprotocols is the presence of a malicious intruder that will use all its capabilities in order to break the protocol security. A protocol is said to be secure if it satisfies certain security properties. Examples of security properties are [1]: Secrecy, authentication, integrity, non-repudiation, fairness, money atomicity, and good atomicity. The problem of assuring that a certain protocol design actually satisfies its intended security property (or properties) is by no means trivial or easy. Formal specification, modeling and verification methods have been extensively used to tackle this problem. Techniques from logic [13], process calculi [4], rewrite systems [14], model checking [15], and theorem proving [24] have been developed in order to verify security properties.

The work most related to this paper is the one using algebraic symbolic modeling and analysis of cryptoprotocols. Several research efforts exist in this direction and we can cite [12] that contains an excellent overview and links to many references. We can break down the basic ideas into modeling and analysis. The modeling part involves the modeling of exchanged messages (the data), the modeling of computation and communication operations (the algorithms) and finally a model of the intruder. The analysis part is concerned with the development of verification techniques which can ensure that the required security properties are satisfied. Messages are modeled as terms of a term algebra having a signature Σ. Computation and communication operations done on messages by honest agents are expressed by modeling agents as processes running in parallel while sending and receiving messages to/from the environment. In this approach, a number of process calculi have been used in the literature, e.g., [4] and [3]. There exist several variations of this framework such as modeling message computations by rewrite rules [14] or using the idea of strand spaces [19]. The model used for the intruder is in most cases that of Dolev Yao [18], which basically means that the intruder has total control over the environment and can block, create, send and receive messages. However, the computational capabilities of the intruder are limited to formal algebraic operations, i.e, the intruder does not perform cryptanalysis. Lately however, and in the direction of increasing the intruder powers, research efforts [2], [5] [10], and [17] investigated the idea of enriching the algebra of messages with an equational theory. This means the intruder would be capable of doing some sort of cryptanalysis. Concerning verification, several techniques are used for specifying security properties and verify them under some assumptions, the two most important of which are the bound on the number of protocol sessions and the bound on message size [12]. Several decidability results for verification have been published under various versions of these two assumptions [17].

In this paper we introduce a new model for crytpoprotocols based on game semantics. A protocol is viewed as a

0-7695-2884-8/07 $25.00 © 2007 IEEE
DOI 10.1109/SEFM.2007.34

game between honest agents and the intruder. Interleaved runs of the protocol form a game tree. Moves of the defined games are protocol messages which are constructed as terms of a term algebra with signature Σ. Moreover, the algebra is equipped with an equational theory E to enrich the analysis by inducing equivalences between terms. Computation steps of the protocol are expressed as abstract computations [25] based on Σ. We find that this simplifies the model as opposed to using a process calculus. Communication steps are modeled implicitly via the interaction in each game between the intruder and an honest agent. In our model, the intruder has total control over the network and can block, create, send and receive messages. We provide rules for the intruder knowledge that make use of the equational theory defined over the message algebra. To verify the model, a logic is defined based on the ideas from [8] and [9] in order to specify security properties over the game tree. Semantics of the logic is defined in terms of game strategies as opposed to simple traces, and a tableau-based proof system is provided. Our contribution can be summarized in the following:

- We can express malicious behavior of the intruder in addition to dishonest behavior of agents who do not follow protocol rules. The latter situation is used in the analysis of fair exchange protocols.
- Our model is built directly from the standard arrow notation commonly used in specifying protocols. Also we ascribe formal semantics to such a notation.
- Reasoning in terms of game strategies allows us to specify a wide range of properties such as secrecy, authentication, fairness, etc. using a unified approach.

This paper contains five sections starting with the introduction. In Section 2, we introduce game semantics and some basic concepts. Section 3 contains the definitions of our protocol games. In Section 4, we introduce our logic and present its semantics. Finally, Section 5 concludes the paper.

2 Basic Concepts

2.1 Game Semantics

The idea of using games in logic specifications dates back to Lorenzen [23] who viewed a logic proposition as a game between two players one trying to assert it (the proponent) and the other trying to attack it (the opponent). Further development was done by Andeas Blass [11], who used it to give semantics to linear logic. Game semantics [7] and [20] is an approach to the semantics of programming languages that makes explicit the interaction with the environment in each computation step. This interaction is modeled by a game in which the players are the environment (the opponent) and the system (the proponent). Any computational step done by the system can be modeled by a possible sequence of moves over the game that describes the system. Here, we make the distinction between the program type (e.g. function signature) and the algorithm (e.g. function definition) this program implements. A program of a certain type is modeled by a certain game, whereas the algorithm specifies the rules according to which the game should be played. So, any particular run of the program (execution of the algorithm) represents a ceratin sequence of moves over the game that describes the program. In this case, the game-semantics specification of the program is the set of all such sequences. In game semantics, each move takes the form of a question Q or an answer A. For instance, the environment can ask for a value (question), and the system supplies this value (answer) directly, *or* asks the environment for more detail (question), and so on. We adopt the convention that the opponent always makes the first move then the game proceeds as alternating moves between player and opponent. Formally a game G is a structure (M_G, λ_G, P_G) where [6]:

$$\begin{array}{ll} M_G & \text{Set of game moves} \\ \lambda_G : M_G \rightarrow \{P, O\} \times \{Q, A\} & \text{Labeling function signature} \\ \lambda_G = \langle \lambda_G^{PO}, \lambda_G^{QA} & \text{Labeling function definition} \\ \lambda_G^{PO} : M_G \rightarrow \{P, O\} & \text{Labeling proponent/opponent moves} \\ \lambda_G^{QA} : M_G \rightarrow \{Q, A\} & \text{Labeling question/answer moves} \\ P_G \subseteq^{nepref} M_G^{alt} & \text{Non-empty, prefix closed set of sequences} \end{array} \quad (1)$$

We write M_G^* for the set of finite sequences over M_G. A sequence $s = s_1.s_2 \ldots s_n$ has length $|s| = n$. Then, M_G^{alt} is a subset of M_G^* containing sequences s such that for even i, $\lambda_G^{PO}(s_i) = P$, and for odd i, $\lambda_G^{PO}(s_i) = O$. The domain P_G (the game tree) is a set of sequences, each of these sequences represents a path in the game tree. The domains P_G^{even} and P_G^{odd} are the sets of even- and odd- length sequences respectively. For any two sequences s and t, s is called a prefix of t if $t = s.u$ where u is any sequence, $len(s)$ is a natural number that indicates the length (number of moves) of s and , by function overloading, $pref(s, i)$ is the sequence t such that t is a prefix of s and $len(t) = i$. For a set of sequences P_G, $\mathsf{Pref}(P_G) = \{\mathsf{pref}(s, i) \mid s \in P_G \wedge i \leq len(s)\}$. A set of sequences is prefix closed when $\mathsf{Pref}(P_G) = P_G$. A deterministic *strategy* σ on a game G is a subset $\sigma \subseteq P_G^{even}$ satisfying: $\epsilon \in \sigma$, $sab \in \sigma \Rightarrow s \in \sigma$, and $sab \in \sigma \wedge sac \in \sigma \Rightarrow b = c$. Here $s, t, u, \ldots$ represent sequences, and $a, b, c, \ldots$ represent single moves. Intuitively, a strategy is a set of even-length paths of the game tree that represent a certain "plan" P follows in response to O's moves.

For any two sets X and Y, let the set $Z = X \uplus Y$ be their disjoint union. If the sequence $s \in Z^*$, then $s \upharpoonright X \in X^*$, which means $s \upharpoonright X$ is the sequence obtained by removing all the elements not in X from s. For any two games G and H, the tensor product $G \otimes H$ defines a game whose set of moves is $M_{G \otimes H} = M_G \uplus M_H$, the labeling function is then defined to be $\lambda_{G \otimes H} = [\lambda_G, \lambda_H]$. The game tree is: $P_{G \otimes H} = \{s \in M^{alt}_{G \otimes H} \mid (s \upharpoonright M_G \in P_G) \wedge (s \upharpoonright M_H \in P_H)\}$. As previously mentioned, all games start by O making a move. Here O can decide to make a move in G or H. The way the tensor product is defined enforces the rule that for any two consecutive moves s_i and s_{i+1} if s_{i+1} is a move of a subgame different than that of s_i, then $\lambda^{PO}_{G \otimes H}(s_i) = P$ and $\lambda^{PO}_{G \otimes H}(s_{i+1}) = O$. This is called the *switching condition* [6]. For any game G, its dual $G^{\perp}$ is obtained by interchanging the roles of the two players (P and O). The set of moves remains the same for both games, just the labeling function is changed, i.e., moves of O become those of P and vice versa. For any two games G and H, the game $G \multimap H$ is defined as $G^{\perp} \otimes H$. In this case, the first move (by O) will always be in H. The switching condition in $G \multimap H$ states that for any two consecutive moves s_i and s_{i+1} if s_{i+1} is a move of a subgame different than that of s_i, then $\lambda^{PO}_{G \otimes H}(s_i) = O$ and $\lambda^{PO}_{G \otimes H}(s_{i+1}) = P$ [6]. An *enabling* relation is defined over the set $M_G \cup \{\star\}$. The enabling relation means that a move cannot be played unless it was enabled (justified) by another move. The first move in the game is justified by $\star$. The enabling relation: $m \rightsquigarrow_G m'$ means that m' cannot be played unless m was played first. It is important to note that this relation is *not* transitive.

2.2 Protocols

A cryptoprotocol specifies a number of steps in which agents are exchanging messages. The structure of a message represents the computations that must be done by the agent sending the message. Moreover, some messages are stored in the agent knowledge and are persistent in all protocol executions, while other are freshly created by the agent in each protocol run such as nonces. Taking these facts into account, the syntax we use to specify protocols consists mainly of two parts: Declaration, and communication as shown in the table below:

$$\begin{array}{rcl} \mathcal{P}rot & ::= & \mathcal{D}ecl \,.\, \mathcal{C}omm \mid \epsilon \\ \mathcal{D}ecl & ::= & \kappa_A \rhd m \,.\, \mathcal{D}ecl \mid \nu_A \rhd m \,.\, \mathcal{D}ecl \mid \epsilon \\ \mathcal{C}omm & ::= & step\ i \rhd A \to B :\ m \,.\, \mathcal{C}omm \mid \epsilon \end{array}$$

Here A, B are communicating agents, m is a certain message, $\kappa_A \rhd m$ means m is part of the initial persistent knowledge of A, and $\nu_A \rhd m$ means m is freshly produced by A for a certain protocol session. As for $step\ i \rhd A \to B : m$, it means that message m is sent by agent A and is intended to be received by agent B. Structures of messages are presented in the next section.

2.3 Messages

Given a signature Σ of operation symbols, we denote by $\mathbb{T}_\Sigma$ and $\mathbb{T}_\Sigma(X)$ the sets of ground terms and terms with variables over Σ. The set $\mathbb{M}$ of messages is a proper subset of $\mathbb{T}_\Sigma$ and is defined by a grammar, all operation symbols (of arity one or higher) in this grammar are called constructors. An example is shown below:

Constants (names) include agent names, keys, and text	
Constructors:	$\{< \_, \_ >,\ \{\_\}_\_\}$
Destructors:	$\{\mathbf{fst}(\_),\ \mathbf{snd}(\_),\ \mathbf{dec}(\_,\_)\}$
Messages	$m ::= a \mid c \mid k \mid\ < m, m >\ \mid \{m\}_k$

In the example above, constructors represent the pairing and encryption operations. Destructors represent projection operations to obtain the first and second elements of a pair and the decryption operation. Valid messages are given by the BNF grammar shown in the example, where a represents agent names, c represents constant (text) messages, and k represents cryptographic keys. All previous messages are called atomic. The term $< m, m >$ represents concatenated messages and $\{m\}_k$ denotes encrypted messages, where k is the encryption key. We equip the algebra of messages with an equational theory E that is chosen according to the specific properties of the protocol and its underlying cryptosystem. In this case, the congruence induced by the equational theory is used to define equivalence classes between messages. We write $m =_E n$ to say that m is congruent to n under the equational theory E. We focus our attention to equational theories for which there exists a convergent (complete) term rewriting system. For these systems, each term has a unique normal form. Therefore, the problem of determining whether $m =_E n$ for any two closed terms m and n is decidable by rewriting both m and n and checking if they have the same normal form. As an example, in the Dolev Yao model, E will contain:

$$\begin{array}{ll} \mathbf{dec}(\{x\}_y, y) = x & \mathbf{fst}(< x, y >) = x \\ \mathbf{snd}(< x, y >) = y & \end{array}$$

2.4 Frames and Computation

During one session of a protocol, an agent A receives a number of messages, we follow [3] and [2], with different interpretation, and organize these messages into a frame $\nu\mathbb{C} \,.\, \theta_f$, where $\mathbb{C}$ is a set of names (constants), ν binds the names in $\mathbb{C}$ to the frame, and θ_f is a partial mapping from a finite set $\{0, \ldots, n\}$ to sets of messages. The number of messages received by A in one protocol session is n since $\theta_f(0)$ is the set containing atomic messages that are initially known to A and $\theta_f(i), 0 < i \leq n$ is the set containing the message expected by A at step i of the protocol. A frame encapsulates three pieces of information: The new names in $\mathbb{C}$ which are created in a protocol session such as nonces, the order in which the messages is received, and the structure of each message. We define a function ϕ (of type Frm) that maps agents names to frames, $\phi(A)$ is the frame of messages received by agent A. For a frame $\phi(A) = \nu\mathbb{C} \,.\, \theta_f$, the function **size**$(\phi(A))$ is defined to be $|dom(\theta_f)|$; the cardinality of the domain of the substitution, moreover for any $0 < i \leq$ **size**$(\phi(A))$, **trunc**$(\phi(A), i), i \in dom(\theta_f)$ is the new frame obtained by restricting the domain of θ_f to the set $\{0, \ldots, i\}$, i.e., it is the frame of A just after the i-th step of the protocol. The function **frame** that parses the protocol and constructs the frame is defined below:

$$
\begin{aligned}
&\mathbf{frame} : \mathcal{P}rot \rightarrow \mathsf{Frm} \rightarrow \mathsf{Frm}\\
&\mathbf{frame}[\![\ \nu A \triangleright m.\mathcal{D}.\mathcal{C}\ ]\!](\phi) = \mathbf{frame}[\![\ \mathcal{D}.\mathcal{C}\ ]\!](\phi \dagger [A \mapsto \nu\mathbb{C} \cup m \,.\, \theta_f])\\
&\mathbf{frame}[\![\ \kappa A \triangleright m.\mathcal{D}.\mathcal{C}\ ]\!](\phi) =\\
&\qquad \mathbf{frame}[\![\ \mathcal{D}.\mathcal{C}\ ]\!](\phi \dagger [A \mapsto \nu\mathbb{C} \,.\, \theta_f[0 \mapsto \theta_f(0) \cup \{m\}]])\\
&\mathbf{frame}[\![\ \epsilon.\mathcal{C}\ ]\!](\phi) = \mathbf{frame}[\![\ \mathcal{C}\ ]\!](\phi)\\
&\mathbf{frame}[\![\ step\ i \triangleright B \rightarrow A :\ m.\mathcal{C}\ ]\!](\phi) =\\
&\qquad \mathbf{frame}[\![\ \mathcal{C}\ ]\!](\phi \dagger [A \mapsto \nu\mathbb{C} \,.\, \theta_f[i \mapsto m]])\\
&\mathbf{frame}[\![\ \epsilon\ ]\!](\phi) = \phi
\end{aligned}
$$

Intuitively, the frame $\phi(A) = \nu\mathbb{C} \,.\, \theta_f$ is constructed as follows:(1) The set of all fresh names for an agent A is added to $\mathbb{C}$, (2) all atomic terms that are initially known to agent A are added to the set $\theta_f(0)$, and (3) for each message m_i received at step i of the protocol run, θ_f is augmented by the mapping $i \mapsto m_i$. To simplify the notation, the message received by an agent A at step k of the protocol is denoted by $r_A(k)$, i.e., $\forall k > 0 \,.\, r_A(k) \in \theta_f(k)$. A term $m \in T_\Sigma$ is deducible by agent A from a frame $\phi(A) = \nu\mathbb{C} \,.\, \theta_f$, written $\phi(A) \vdash m$, if it can be obtained by the application of one or more of the following rules:

$$(\mathbf{rcv})\frac{}{\phi(A) \vdash m} \exists i \in dom(\theta_f) \,.\, m \in \theta_f(i)$$

$$(\mathbf{new})\frac{}{\phi(A) \vdash m} m \in \mathbb{C}$$

$$(\mathbf{apl})\frac{\phi(A) \vdash m_1 \ldots \phi(A) \vdash m_n}{\phi(A) \vdash f(m_1, \ldots m_n)}$$

$$(\mathbf{eqn})\frac{\phi(A) \vdash m \quad m =_E m'}{\phi(A) \vdash m'}$$

We define the set **deduce**$(\phi(A)) = \{m \in \mathbb{M} \mid \phi(A) \vdash m\}$, this definition is necessary to avoid terms that are not messages, e.g., **fst**$(\{m\}_k)$. Moreover, the set **deduce**$(\phi(A))$ can be expressed as $\bigcup_{d \geq 0} \mathbf{deduce}^d(\phi(A))$, where:

$$
\begin{aligned}
\mathbf{deduce}^0(\phi(A)) &= \textstyle\bigcup_{i \in dom(\theta_f)} \theta_f(i) \cup \mathbb{C}\\
\mathbf{deduce}^{d+1}(\phi(A)) &= \mathbf{deduce}^d(\phi(A)) \cup \mathbb{X}^{d+1}_{\mathbf{apl}} \cup \mathbb{X}^{d+1}_{\mathbf{eqn}}\\
\mathbb{X}^{d+1}_{\mathbf{apl}} &= \{m = f(t_1, \ldots, t_n) \mid f \in \Sigma \wedge\\
&\qquad t_1, \ldots, t_n \in \mathbf{deduce}^d(\phi(A))\}\\
\mathbb{X}^{d+1}_{\mathbf{eqn}} &= \{m =_E m' \mid m' \in \mathbf{deduce}^d(\phi(A)\}
\end{aligned}
$$

In the two equations above, **deduce**$^{d+1}(\phi(A))$ is obtained by application of the rules (**apl**) and (**eqn**) to the messages in **deduce**$^d(\phi(A))$. Given a frame $\phi(A) = \nu\mathbb{C}.\theta_f$ that belongs to a certain agent A, and a message m_j that this agent is supposed to send at a certain communication step j, we would like to find the abstract computation procedure that should be followed to get m_j, we call this procedure **out**$(\phi(A), j)$. In other words, **out**$(\phi(A), j)$ is the algorithm followed by agent A, having a frame $\phi(A)$ in order to compute the message m_j that he is *supposed* to send at step j of the protocol, i.e., according to protocol specification. We define below an algorithm **Proc** such that **out**$(\phi(A), j) =$ **Proc**$(\phi(A), m, d)$, where m is the message m_j such that $m \in \mathbf{deduce}^d(\phi(A))$:

```
$\mathbf{Proc}(\phi(A), m, d)$
case $m \in \mathbb{X}^d_{\mathbf{apl}} \Rightarrow$
    $f(\mathbf{Proc}(\phi(A), t_1, d-1), \ldots, \mathbf{Proc}(\phi(A), t_n, d-1))$
    $m = f(t_1, \ldots, t_n)$
case $m \in \mathbb{X}^d_{\mathbf{eqn}} \Rightarrow$
    $\mathbf{Proc}(\phi(A), m', d-1)$
    $m' =_E m$
case $m \in \mathbf{deduce}^{d-1}(\phi(A)) \Rightarrow$
    $\mathbf{Proc}(\phi(A), m, d-1)$
$\mathbf{Proc}(\phi(A), m, 0)$
case $m = n, n \in \theta_f(0) \Rightarrow n$
case $m \in \theta_f(k), k > 0 \Rightarrow r_A(k)$
case $m = c, c \in \mathbb{C} \Rightarrow c$
```

We say that the algorithm $out(\phi(A), j)$ is feasible if $\neg\exists i \geq j \,.\, \theta_f(i) \preccurlyeq out(\phi(A), j)$, where $\preccurlyeq$ is the subterm relation defined over $\mathbb{T}_\Sigma(X)$, i.e., $x \preccurlyeq y$ if $x = y$ or $y = f(t_1, \ldots t_n) \wedge \exists i \,.\, x \preccurlyeq t_i$. In other words, the algorithm is feasible if it constructs the message that is supposed to be sent by A at step j of the protocol using only the initial knowledge of A and the messages received by A up to step j. This condition of feasibility is one simple and preliminary check on the correctness of protocol specification. As an example, consider the Shamir Rivest Adelman three pass protocol [21]: $A \rightarrow B : \{m\}_{K_A}$, $B \rightarrow A : \{\{m\}_{K_A}\}_{K_B}$, $A \rightarrow B : \{m\}_{K_B}$. This protocol uses the commutative property of RSA encryption. It is assumed that A and B share no information about their keys. The signature Σ contains the constants m, K_A, and K_B, and the operations $\{\_\}_\_$, $dec(\_, \_)$ and $(\_)^{-1}$ representing encryption, decryption and key inverses, respectively. We add the equations that: $\{\{x\}_y\}_z = \{\{x\}_z\}_y$ and $dec(\{x\}_y, y^{-1}) = x$. In this case, $\phi(A) = \nu\{m\} \,.\, \{0 \mapsto \{K_A\}, 2 \mapsto \{\{\{m\}_{K_A}\}_{K_B}\}\}$, we can deduce the following:

$$
\begin{aligned}
&\mathbf{deduce}^0(\phi(A)) = \{m, K_A, \{\{m\}_{K_A}\}_{K_B}\}\\
&\mathbf{deduce}^1(\phi(A)) = \mathbf{deduce}^0(\phi(A)) \cup\\
&\qquad \{K_A^{-1}, dec(\{\{m\}_{K_A}\}_{K_B}, K_A)\} \cup \{\{\{m\}_{K_B}\}_{K_A}\}\\
&\mathbf{deduce}^2(\phi(A)) = \mathbf{deduce}^1(\phi(A)) \cup\\
&\qquad \{dec(\{\{m\}_{K_B}\}_{K_A}, K_A^{-1})\} \cup \{\ldots\}\\
&\mathbf{deduce}^3(\phi(A)) = \mathbf{deduce}^2(\phi(A)) \cup \{\ldots\} \cup \{\{m\}_{K_B}\}
\end{aligned}
$$

Now we want to know how agent A is able to construct the message he is sending in step $j = 3$ as a function from the knowledge he gathered thus far. In other words we want to get **out**$(\phi(A), 3)$ since the message sent by A at step 3 is $\{m\}_{K_B}$, and $\{m\}_{K_B} \in \mathbf{deduce}^3(\phi(A))$, i.e., $d = 3$, then **out**$(\phi(A), 3) =$ **proc**$(\phi(A), \{m\}_{K_B}, 3)$:

$$
\begin{aligned}
&\mathbf{proc}(\phi(A), \{m\}_{K_B}, 3)\\
&\quad = \mathbf{proc}(\phi(A), dec(\{\{m\}_{K_B}\}_{K_A}, K_A^{-1}, 2)\\
&\quad = dec(\mathbf{proc}(\phi(A), \{\{m\}_{K_B}\}_{K_A}, 1), \mathbf{proc}(\phi(A), K_A^{-1}, 1))\\
&\quad = dec(\mathbf{proc}(\phi(A), \{\{m\}_{K_A}\}_{K_B}, 0), (\mathbf{proc}(\phi(A), K_A, 0))^{-1})\\
&\quad = dec(r_A(2), K_A^{-1})
\end{aligned}
$$

The last line means that in order to obtain the message that should be sent at step 3 from the frame $\phi(A)$, agent A must decrypt the message he received at step 2 by the inverse of his key K_A.

2.4.1 Agents Responses

The intruder's behavior is considered nondeterministic in the sense that at any point in time we cannot tell exactly which message he's going to send. Honest agents on the other hand have a deterministic behavior; the message an agent A sends at any step i is determined by $\mathbf{out}(\phi(A), i)$, i.e., the message is dependent on the frame of messages seen by A and the step number. The frame $\phi(A)$ is constructed based on the protocol specification, we call it a specification frame as the underlying assumption is that the protocol is executed exactly as specified. This is not the actual case however, since we assume the presence of an intruder and/or dishonest agents. The difference between the real frame $\rho(A) = \nu\mathbb{C} \centerdot \theta_f$ and the specification frame $\phi(A)$ of an agent A lies in the substitution θ_f. Both frames agree at $\theta_f(0)$ since it contains the initial knowledge. At any other $i > 0$, $\theta_f(i)$ of the specification frame is the message specified to be received by the agent at step i. For an actual frame however the message $\theta_f(i)$ is the one that was actually received from the network during the current session of the protocol execution. This message could have been manipulated by the intruder. Therefore, the message sent by an agent A at step i of an actual protocol execution will be $\mathbf{out}(\rho(A), i)$, this message describes the behavior of an honest agent. A dishonest agent D, however, can send any message in $\mathbf{deduce}(\rho(D))$, we note here that we used the actual frame $\rho(D)$.

2.4.2 Intruder Frames

An intruder frame is always an actual frame, where $\rho(I) = \nu\mathbb{C} \centerdot \theta_f$. In this case, $\mathbb{C}$ is a set containing fresh values created by the intruder. The substitution θ_f in this case maps the set $\{0, 1, \ldots, N\}$ to messages in $\mathbb{M}$, where $\theta_f(0)$ contains a concatenation of all messages initially known to I such as public keys and $\theta_f(i) = m$ means that m is the i^{th} received by the intruder. Deduction rules for the intruder are similar to those defined above for agents, the difference is that $\phi(A)$ should be replaced by $\rho(I)$.

3 Games for Security Protocols

Similar to the discussion above about game semantics and programs, in order to build the game that represents the protocol, we have to assign a certain type to the protocol. This type will determine the game to be played. The specific syntax of the protocol (protocol steps specifying messages), on the other hand, will determine how this game should be played.

3.1 Definition of Protocol Games

When defining games for security protocols, each move m in the game represents a message sent from a principal to the other. As mentioned earlier the set of messages is $\mathbb{M} \subset T_\Sigma$. The generation of a single message by an honest principal is represented as a strategy over the game Msg which is defined as follows:

$$\begin{array}{rcl} M_{\mathsf{Msg}} & = & \{q\} \cup \mathbb{M} \\ \lambda(q) & = & OQ \\ \lambda(m) & = & PA \ \forall m \in \mathbb{M} \\ P_{\mathsf{Msg}} & = & \{q.m \mid m \in \mathbb{M}\} \end{array} \qquad \begin{array}{rcl} \star & \rightsquigarrow & q \\ q & \rightsquigarrow & m \ \forall m \in \mathbb{M} \end{array}$$

Here, we assumed that any $m \in \mathbb{M}$ can be played at any time. This is not an accurate assumption, as any principal in the protocol can play only those messages m that they are able to construct, i.e., $\rho(A) \vdash m$. A single communication step, on the other hand, is represented by a strategy over the game $\mathsf{Csg} = \mathsf{Msg} \multimap \mathsf{Msg}$. The formal definition of the game is given hereafter:

$$M_{\mathsf{Csg}} = \{q^1\} \cup \{q^2\} \cup \{m^i \mid m \in \mathbb{M}, i \in \{1, 2\}\}$$
$$\lambda_{\mathsf{Csg}}(q^i) = \begin{cases} PQ & i = 1 \\ OQ & i = 2 \end{cases}$$
$$\lambda_{\mathsf{Csg}}(m^i) = \begin{cases} OA & \forall m \in \mathbb{M} \wedge i = 1 \\ PA & \forall m \in \mathbb{M} \wedge i = 2 \end{cases}$$

The following enabling relation is defined over M_{Csg}:

$$\begin{array}{rcll} \star & \rightsquigarrow_{\mathsf{Csg}} & q^2 & \\ q^2 & \rightsquigarrow_{\mathsf{Csg}} & q^1 & \\ q^1 & \rightsquigarrow_{\mathsf{Csg}} & m^1 & \forall m \in \mathbb{M} \\ m^1 & \rightsquigarrow_{\mathsf{Csg}} & n^2 & \forall m, n \in \mathbb{M} \end{array} \tag{2}$$

We notice here that we used a superscript to differentiate between moves in each game, since the set of moves is the *disjoint* union of the sets of the individual games. We used the superscript 1 to denote moves of the game to the left of $\multimap$, and the superscript 2 for the other game. This is equivalent to denoting the games as $\mathsf{Msg}^1 \multimap \mathsf{Msg}^2$. The definition of the enabling relation in (2) makes sure that n^2 cannot be played unless m^1 is played first (any sequence in the game tree will be the prefix of a sequence in the form $q^2.q^1.m^1.n^2$). This results from the fact that we assumed that in any communication step (Csg game) an agent (the proponent) only sends a message in response to a message that they received from the intruder (the opponent). We clarify these ideas by taking as example the first three steps of the Woo and Lam authentication protocol [26]: Step1. $A \to B : A$, Step2. $B \to A : N_b$, Step3. $A \to B : \{N_b\}_{K_{as}}$. Examining Step 1, we notice that A initiates the protocol. Since, in game semantics, O (the channel) always plays first, we assume A gets a start message "$start$" from the channel and replies with his identity A. The "$start$" message serves as an action to begin the execution of the protocol. In Step 2, B receives the message A from the channel and replies with a nonce N_b, and other steps follow. We rewrite the first three steps as:

$$\begin{array}{lccl} \text{Step1.} & I \to A & : & start \\ & A \to I & : & A \\ \text{Step3.} & I \to A & : & N_b \\ & A \to I & : & \{N_b\}_{K_{as}} \end{array} \qquad \begin{array}{lccl} \text{Step2.} & I \to B & : & A \\ & B \to I & : & N_b \end{array}$$

The protocol description above makes clear the role of the intruder I. Each communication step has the form: $I \rightarrow X : m_i$ followed by $X \rightarrow I : m_j$ where X is an honest agent. To respect the notation, a protocol will always end by an agent X sending a *terminate* message to the intruder. The *terminate* message marks the end of the execution of one protocol session. Expressed this way, each communication step can be captured as a strategy σ over the game Csg. The execution of a number of steps in succession can be represented by a strategy over the tensor product of a number of Csg games. As an example, Step1 and Step2 are represented as a strategy over the game $(\mathsf{Msg}^{11} \multimap \mathsf{Msg}^{12}) \otimes (\mathsf{Msg}^{21} \multimap \mathsf{Msg}^{22})$. This strategy is the sequence $q^{12}.q^{11}.start^{11}.A^{12}.q^{22}.q^{21}.A^{21}.N_b^{22}$. In the case of multiple sessions of a protocol, the game Spr is defined as:

$$\mathsf{Spr}_{[\mathsf{N}]} \overset{def}{=} \mathsf{Csg}^{1,1} \ldots \otimes \mathsf{Csg}^{1,N} \otimes \mathsf{Csg}^{2,1} \ldots \otimes \mathsf{Csg}^{2,N} \ldots \quad (3)$$

Here $\mathsf{Csg}^{i,j}$ is the game representing communication step i in session j, where N is the maximum number of sessions. The enabling relation defined for $\mathsf{Spr}_{[\mathsf{N}]}$ contains eight conditions for moves. The first condition states that the play begins in Session 1 in any step where an agent expects to receive his first message in the protocol. Notice that the first move of this step has to be a question by the opponent (i.e., $q^{1,t2}$). Once the game has started in a step in a certain session, *and* the intruder has received a message in this step ($m^{j-1,i2}$), he can start the play in any step in the next session ($q^{j,i'2}$) provided this is also the first step in the interaction with a certain agent. This is stated by the second condition. The third condition is special for the start message, i.e., the start message is always enabled in communication step 1 in any session. The fourth condition states the condition for the termination of one session of the protocol (i.e., the reception of $m^{j,N1}$). The fifth, sixth and seventh rules put a condition on the sequence of moves in any communication step in a certain session. They simply state that in any communication step we cannot have a sequence $q^{j,i2}.m^{j,i2}$, this sequence means that an agent sends a message to the intruder without first getting a message form the intruder. This is to emphasize the rule that we established before that each communication step is an exchange between the intruder and an agent, where the intruder has to supply a message in order to get a message in return. The eighth rule imposes order on the messages of the intruder in the same session. Basically the intruder when playing with a certain agent, has to supply messages in the order that this agent expects. It worth noting that games defined this way give rise to a category where objects are games and a morphism between any two games G and H is a strategy over the game $G \multimap H$, details can be found in [6]. Moreover, the game defines above allow us to ascribe formal semantics to protocol specifications written in the standard arrow notation. A protocol is interpreted as a set of strategies over the game $\mathsf{Spr}_{[\mathsf{N}]}$, where $[\mathsf{N}]$ is the maximum number of sessions. The semantics is given in the appendix.

3.1.1 Quantification over Strategies

The game Spr represents an interleaving of actions from different copies if the Csg games. Each Csg game is played between an agent and the intruder, therefore a strategy over Spr may involve several agents. The identity of the agent playing against the intruder in a certain copy of Csg is given by $Id(copy)$, where $copy$ represents the superscript identifying a specific copy of Csg, e.g., the pair i, j in 3. The set of all strategies over Spr is denoted $\mathcal{S}_{\mathsf{Spr}}$. For any strategy $\sigma \in \mathcal{S}_{\mathsf{Spr}}$ and a set A of honest agents identities, we define σ^A as:

$$\sigma^A = \{s \upharpoonright M \mid s \in \sigma\}$$
$$M = \{m \in \mathsf{Csg}^{copy} \mid Id(copy) \in A\}$$

Intuitively, σ_A is obtained from σ by eliminating all moves from any copy of the Csg game whose player P is not a member of A. The set of all strategies involving a set A of players is denoted $\mathcal{S}^A_{\mathsf{Spr}}$ and is defined as: $\mathcal{S}^A_{\mathsf{Spr}} = \{\sigma^A \mid \sigma \in \mathcal{S}_{\mathsf{Spr}}\}$.

4 Logic for Security Properties

In this section, we present a new logic for the specification of security properties. It can be used to specify security properties that should be satisfied by the game tree that represents protocol interactions. A tableau-based proof system is also developed for the verification of such properties. The logic is based on ideas from [8] and [9]. The main differences are that in [9], there is no explicit mention of the intruder, only adversarial behavior between players is considered. The logic was not specifically designed to be used for security properties, although its ideas were used later in [22] to verify non-repudiation protocols. Interaction between players is not explicitly modeled, i.e., no messages exchanged between players. This complicates the specification of properties based on traces of messages. Moreover, in [8], the model considered is a single trace, verification of a protocol amounts to verification of a property over all traces which limits the analysis. Also, the logic considers only traces with atomic actions, it cannot specify a certain structure for the exchanged messages. In our model, the logic is based on the idea of interaction, where the game tree is built from possible interactions between players and intruder. Logic formulas specify properties over the game tree. Moves of the game are actual messages that are exchanged in a protocol run. We can therefore specify properties on the structure of messages, and on traces of messages. We can quantify existentially, universally or by players strategies, i.e., all traces in which certain players are interacting. We give examples for a number of security properties specified in this logic.

4.1 Syntax

Before presenting the syntax of formulas, we present the concept of a sequence pattern r. A sequence pattern has the following syntax:

$$r ::= \epsilon \mid a^{j,in}.r \mid x_r.r \qquad a ::= m \mid \lceil m \rceil \tag{4}$$

Here a represents a move in the game and has the form m or $\lceil m \rceil$, where $i, j \in \mathbb{N}$, $n \in \{1, 2\}$, and $m \in T_\Sigma(X)$, the free term algebra over Σ. Intuitively, a represents a move played in a copy of the Msg game , where the message of this move is either m or a term containing m. The term $\lceil m \rceil$ is defined as m or $f(t_1, t_2, \ldots t_n)$ such that $\exists t_i \centerdot t_i = \lceil m \rceil$. Here $f \in \Sigma$ is any function symbol in the signature of the algebra. The variable x_r represents a sequence of moves of zero or any finite length, the subscript r is added to avoid confusion with variables x of the message algebra. The sets of variables and moves in r are written $var(r)$ and $mov(r)$ respectively. Moreover, for any pattern r, the symbol $r|_i$ represents the variable or move at position i of r, where $i \in \{1, \ldots n\}$.

We define the substitution $\theta_r : var(r) \to M^*_{\mathsf{Msg}}$ that maps variables x_r in a sequence pattern to sequences of moves of the game Msg and the substitution $\theta_m : \mathbb{X} \to \mathbb{T}_\Sigma(X)$ that maps variables inside messages into terms of $T_\Sigma(X)$.

We define the predicate **satisfy**$(\sigma, r, \theta_m, \theta_r)$, which is true when a strategy σ in the game tree *satisfies* a pattern r. Intuitively, a strategy σ satisfies a pattern r if there is a sequence $s \in \sigma$ such that s matches r. Formally:

$$\textbf{satisfy}(\sigma, r, \theta_m, \theta_r) = \exists s \in \sigma \centerdot \textbf{match}(s, r, \theta_m, \theta_r)$$

The predicate **match**$(s, r, \theta_m, \theta_r)$, where $s = s_1 s_2 \ldots s_n$ is a sequence of moves in the game tree, is defined as follows:

$$\begin{array}{l}
\textbf{match}(\epsilon, \epsilon, \theta_m, \theta_r) = true \\
\textbf{match}(s, \epsilon, \theta_m, \theta_r) = false \quad \text{if } s \neq \epsilon \\
\textbf{match}(s, a.r, \theta_m, \theta_r) = (s_1 =_E a\theta_m) \wedge \textbf{match}(s_2 \ldots s_n, r, \theta_m, \theta_r) \\
\textbf{match}(s, x_r.r, \theta_m, \theta_r) = \exists j \leq n \centerdot x_r\theta_r = s_1 \ldots s_j \wedge \\
\qquad\qquad \textbf{match}(s_{j+1} \ldots s_n, r, \theta_m, \theta_r)
\end{array}$$

A substitution $\theta : \mathcal{R} \to M^*_{\mathsf{Msg}}$ from patterns to sequences of moves, where $\theta = \theta_m \cup \theta_r$, is defined as follows:

$$\begin{array}{lcl}
\theta(\epsilon) & = & \epsilon \\
\theta(a.r) & = & \theta_m(a).\theta(r) \\
\theta(x_r.r) & = & \theta_r(x_r).\theta(r)
\end{array}$$

We follow the usual notation for substitutions and write $r\theta$ for $\theta(r)$, where $r\theta \in \{q \cup T_\Sigma\}^*$. From the definitions of the predicate **match** and the substitution θ above, we notice that the condition for a match between a pattern and a sequence is the existence of one or more substitutions θ, we can therefore write the predicates above as **satisfy**(σ, r, θ) and **match**(s, r, θ).

The syntax of a formula ϕ of our logic is expressed by the following grammar:

$$\varphi ::= Z \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid [r_1 \looparrowright r_2]\varphi \mid \nu Z.\varphi \mid \langle\langle A \rangle\rangle \varphi \tag{5}$$

We require the following two syntactic conditions:

- In $[r_1 \looparrowright r_2]$, $var(r_1) = var(r_2)$ and $\forall i \centerdot (r_1|_i \in var(i) \Leftrightarrow r_1|_i = r_2|_i) \wedge (r_1|_i \in mov(i) \Leftrightarrow r_1|_i = r_2|_i \vee r_2|_i = \circledast)$.
- In $\nu Z.\varphi$, any free Z in φ appears under the scope of an even number of negations.

The first condition above means that r_2 is obtained from r_1 by replacing some of the moves of r_1 by the dummy symbol $\circledast$, where $\theta(\circledast) = \circledast$. This condition is necessary to ensure that $r_1\theta$ and $r_2\theta$ have the same length. Hence, if $r_1\theta \in \sigma$, we can replace $r_1\theta$ by $r_2\theta$ and still get a strategy σ', which is written $\sigma' = \sigma[r_2\theta / r_1\theta]$. The second condition is necessary for the semantic interpretation function as will be explained in the semantics section. Intuitively, the formula $[r_1 \looparrowright r_2]\varphi$ is true if there is a sequence in a strategy in the game tree that matches r_1 and when modified to match r_2 will satisfy φ. The quantifier $\langle\langle A \rangle\rangle$ chooses the subtree of the game tree where the agents in the set A interact together through their strategies. The formula to the right of $\langle\langle A \rangle\rangle$ operates on this subtree. The rest of the formulas have their usual meaning in modal μ-calculus [16].

4.2 Semantics

A formula in the logic is interpreted over a game tree. Given a certain game tree $\mathcal{G}$, a substitution θ, and a an environment e that maps formulae variables to strategies in $\mathcal{G}$, the semantic function $\llbracket\, \varphi \,\rrbracket^{\mathcal{G}}_e$ maps a formula φ to a set of strategies $S \subseteq \mathcal{S}_{\mathcal{G}}$, where $\mathcal{S}_{\mathcal{G}}$ is the set of all strategies in $\mathcal{G}$.

$$\begin{array}{rcl}
\llbracket\, Z \,\rrbracket^{\mathcal{G}}_e & = & e(Z) \\
\llbracket\, \neg\varphi \,\rrbracket^{\mathcal{G}}_e & = & \mathcal{S}_{\mathcal{G}} \setminus \llbracket\, \varphi \,\rrbracket^{\mathcal{G}}_e \\
\llbracket\, \varphi_1 \wedge \varphi_2 \,\rrbracket^{\mathcal{G}}_e & = & \llbracket\, \varphi_1 \,\rrbracket^{\mathcal{G}}_e \cap \llbracket\, \varphi_2 \,\rrbracket^{\mathcal{G}}_e \\
\llbracket\, [r_1 \looparrowright r_2]\varphi \,\rrbracket^{\mathcal{G}}_e & = & \{\sigma \in \mathcal{S}_{\mathcal{G}} \mid \forall\theta \centerdot \textbf{satisfy}(\sigma, r_1, \theta) \Rightarrow \\
 & & \sigma' \in \llbracket\, \varphi \,\rrbracket^{\mathcal{G}'}_e\} \\
\multicolumn{3}{l}{\text{where } \sigma' = \sigma[r_2\theta/r_1\theta], \mathcal{G}' = \mathcal{G}[r_2\theta/r_1\theta]} \\
\llbracket\, \nu Z.\varphi \,\rrbracket^{\mathcal{G}}_e & = & \bigcup\{S \subseteq \mathcal{S}_{\mathcal{G}} \mid S \subseteq \llbracket\, \varphi \,\rrbracket^{\mathcal{G}}_{e[Z \mapsto S]}\} \\
\llbracket\, \langle\langle A \rangle\rangle\varphi \,\rrbracket^{\mathcal{G}}_e & = & \mathcal{S}^A_{\mathcal{G}} \cap \llbracket\, \varphi \,\rrbracket^{\mathcal{G}}_e
\end{array} \tag{6}$$

From the semantic equations above it can be seen that the meaning of the recursive formula $\nu Z.\varphi$ is taken to be the greatest fixpoint of a function $f : 2^{\mathcal{S}_{\mathcal{G}}} \to 2^{\mathcal{S}_{\mathcal{G}}}$, where $f(S) = \llbracket\, \varphi \,\rrbracket^{\mathcal{G},\theta}_{e[Z \mapsto S]}$. The function f is defined over the lattice $(2^{\mathcal{S}_{\mathcal{G}}}, \subseteq, \cup, \cap)$, the syntactic condition on φ (X appears under the scope of an even number of negations) ensures that $f(S)$ is monotone [16] and hence has a greatest fixpoint.

We use the following shorthand notations:

$$\begin{array}{rcl}
\neg(\neg\varphi_1 \wedge \neg\varphi_2) & \equiv & \varphi_1 \vee \varphi_2 \\
\neg\varphi_1 \vee \varphi_2 & \equiv & \varphi_1 \Rightarrow \varphi_2 \\
\neg[r_1 \looparrowright r_2]\neg\varphi & \equiv & \langle r_1 \looparrowright r_2\rangle\varphi \\
\neg\nu Z.\neg\varphi[\neg Z/Z] & \equiv & \mu Z.\varphi
\end{array}$$

Moreover, we use the following two notation $\texttt{tt} \equiv \nu Z.Z$ and $\texttt{ff} \equiv \mu Z.Z$, now we prove some important results regarding the logic.

Lemma 4.1 $\llbracket\, \varphi[\psi/Z] \,\rrbracket^{\mathcal{G}}_e = \llbracket\, \varphi \,\rrbracket^{\mathcal{G}}_{e[Z\mapsto\llbracket\, \psi \,\rrbracket^{\mathcal{G}}_e]}$

The proof is done by structural induction over φ.
proof
Base case: $\varphi = Z$
$\llbracket\, \psi/Z \,\rrbracket^{\mathcal{G}}_e = \llbracket\, \psi \,\rrbracket^{\mathcal{G}}_e$
But: $\llbracket\, Z \,\rrbracket^{\mathcal{G}}_e = e(Z)$, so $\llbracket\, \psi/Z \,\rrbracket^{\mathcal{G}}_e = \llbracket\, Z \,\rrbracket^{\mathcal{G}}_{e[Z\mapsto\llbracket\, \psi \,\rrbracket^{\mathcal{G}}_e]}$
We demonstrate the case $\varphi = [r_1 \looparrowright r_2]\varphi'$ and the other cases can be easily proved:
$\llbracket\, \varphi[\psi/Z] \,\rrbracket^{\mathcal{G}}_e = \{\sigma \in \mathcal{S}_{\mathcal{G}} \mid \forall\theta \,.\, \textbf{satisfy}(\sigma, r_1, \theta) \Rightarrow \sigma' \in \llbracket\, \varphi'[\psi/Z] \,\rrbracket^{\mathcal{G}'}_e\}$
By induction hypothesis:
$\llbracket\, \varphi[\psi/Z] \,\rrbracket^{\mathcal{G}}_e = \{\sigma \in \mathcal{S}_{\mathcal{G}} \mid \forall\theta \,.\, \textbf{satisfy}(\sigma, r_1, \theta) \Rightarrow \sigma' \in \llbracket\, \varphi' \,\rrbracket^{\mathcal{G}'}_{e[Z\mapsto\llbracket\, \psi \,\rrbracket^{\mathcal{G}}_e]}\}$
$\llbracket\, \varphi[\psi/Z] \,\rrbracket^{\mathcal{G}}_e = \llbracket\, \varphi \,\rrbracket^{\mathcal{G}}_{e[Z\mapsto\llbracket\, \psi \,\rrbracket^{\mathcal{G}}_e]}\}$ □

As a result, we have $\llbracket\, \nu Z.\varphi \,\rrbracket^{\mathcal{G}}_e = \llbracket\, \varphi[\nu Z.\varphi/Z] \,\rrbracket^{\mathcal{G}}_e$. This follows from the fact that $\llbracket\, \nu Z.\varphi \,\rrbracket^{\mathcal{G}}_e = \llbracket\, \varphi \,\rrbracket^{\mathcal{G}}_{e[Z\mapsto T]}$, where $T = \bigcup\{S \subseteq \mathcal{S}_{\mathcal{G}} \mid S \subseteq \llbracket\, \varphi \,\rrbracket^{\mathcal{G}}_{e[Z\mapsto S]}\} = \llbracket\, \nu Z.\varphi \,\rrbracket^{\mathcal{G}}_e$.

We can now prove that the semantics of the expression $\mu Z.\varphi$ defined earlier as $\neg\nu Z.\neg\varphi[\neg Z/Z]$ is the least fixpoint of the function $f(S) = \llbracket\, \varphi \,\rrbracket^{\mathcal{G}}_{e[Z\mapsto S]}$.

$$\begin{array}{l}
\llbracket\, \neg\nu Z.\neg\varphi[\neg Z/Z] \,\rrbracket^{\mathcal{G}}_e \\
= \mathcal{S}_{\mathcal{G}} \setminus \bigcup\{S \subseteq \mathcal{S}_{\mathcal{G}} \mid S \subseteq \llbracket\, \neg\varphi[\neg Z/Z] \,\rrbracket^{\mathcal{G}}_{e[Z\mapsto S]}\} \\
= \mathcal{S}_{\mathcal{G}} \setminus \bigcup\{S \subseteq \mathcal{S}_{\mathcal{G}} \mid S \subseteq \mathcal{S}_{\mathcal{G}} \setminus \llbracket\, \varphi \,\rrbracket^{\mathcal{G}}_{e[Z\mapsto\llbracket\, \neg Z \,\rrbracket^{\mathcal{G}}_{e[Z\mapsto S]}]}\} \\
= \mathcal{S}_{\mathcal{G}} \setminus \bigcup\{S \subseteq \mathcal{S}_{\mathcal{G}} \mid S \subseteq \mathcal{S}_{\mathcal{G}} \setminus \llbracket\, \varphi \,\rrbracket^{\mathcal{G}}_{e[Z\mapsto \mathcal{S}_{\mathcal{G}}\setminus S]}\}
\end{array}$$

For any set of strategies $S \subseteq \mathcal{S}_{\mathcal{G}}$, let $S^c = \mathcal{S}_{\mathcal{G}} \setminus S$. By De Morgan laws, for any two sets A and B:
$(A \cap B)^c = A^c \cup B^c$, $(A \cup B)^c = A^c \cap B^c$, $A \subseteq B \Rightarrow B^c \subseteq A^c$.

$$\begin{array}{l}
\llbracket\, \neg\nu Z.\neg\varphi[\neg Z/Z] \,\rrbracket^{\mathcal{G}}_e \\
= (\bigcup\{\mathcal{S}_{\mathcal{G}} \setminus S^c \subseteq \mathcal{S}_{\mathcal{G}} \mid S \subseteq (\llbracket\, \varphi \,\rrbracket^{\mathcal{G}}_{e[Z\mapsto S^c]})^c\})^c \\
= (\bigcup\{\mathcal{S}_{\mathcal{G}} \setminus S^c \subseteq \mathcal{S}_{\mathcal{G}} \mid \llbracket\, \varphi \,\rrbracket^{\mathcal{G}}_{e[Z\mapsto S^c]} \subseteq S^c\})^c \\
= \bigcap(\{\mathcal{S}_{\mathcal{G}} \setminus S^c \subseteq \mathcal{S}_{\mathcal{G}} \mid \llbracket\, \varphi \,\rrbracket^{\mathcal{G}}_{e[Z\mapsto S^c]} \subseteq S^c\})^c \\
= \bigcap\{S^c \subseteq \mathcal{S}_{\mathcal{G}} \mid \llbracket\, \varphi \,\rrbracket^{\mathcal{G}}_{e[Z\mapsto S^c]} \subseteq S^c\}
\end{array}$$

Moreover, we investigate the semantics of the the expression $\langle r_1 \looparrowright r_2\rangle\varphi$ as defined above:

$$\begin{array}{l}
\llbracket\, \langle r_1 \looparrowright r_2\rangle\varphi \,\rrbracket^{\mathcal{G}}_e = \llbracket\, \neg[r_1 \looparrowright r_2]\neg\varphi \,\rrbracket^{\mathcal{G}}_e \\
= \{\sigma \in \mathcal{S}_{\mathcal{G}} \mid \neg\forall\theta \,.\, \textbf{satisfy}(\sigma, r_1, \theta) \Rightarrow \sigma' \in \mathcal{S}_{\mathcal{G}} \setminus \llbracket\, \varphi \,\rrbracket^{\mathcal{G}'}_e\} \\
= \{\sigma \in \mathcal{S}_{\mathcal{G}} \mid \neg\forall\theta \,.\, \textbf{satisfy}(\sigma, r_1, \theta) \Rightarrow \neg\sigma' \in \llbracket\, \varphi \,\rrbracket^{\mathcal{G}'}_e)\} \\
= \{\sigma \in \mathcal{S}_{\mathcal{G}} \mid \neg\forall\theta \,.\, \neg(\textbf{satisfy}(\sigma, r_1, \theta) \wedge \sigma' \in \llbracket\, \varphi \,\rrbracket^{\mathcal{G}'}_e)\} \\
= \{\sigma \in \mathcal{S}_{\mathcal{G}} \mid \exists\theta \,.\, (\textbf{satisfy}(\sigma, r_1, \theta) \wedge \sigma' \in \llbracket\, \varphi \,\rrbracket^{\mathcal{G}'}_e)\}
\end{array}$$

In the derivation above we used the sequent $\psi \Rightarrow \neg\varphi \vdash \neg(\psi \wedge \varphi)$, which can be easily proved by propositional calculus. We also used the fact that for any set of strategies S, $S \cap (\mathcal{S}_{\mathcal{G}} \setminus S) = \emptyset$. It is worth noting here that the semantics of $\langle r_1 \looparrowright r_2\rangle\varphi$ is consistent with the definition of the modality $\langle\ \rangle$ from modal μ-calculus.

4.3 Tableau-based Proof System

Before we present the rules of the tableau, we define the immediate subformula relation [16] $\prec_I$ as:

$$\begin{array}{ll}
\varphi \prec_I \neg\varphi & \varphi \prec_I [r_1 \looparrowright r_2]\varphi \\
\varphi_i \prec_I \varphi_1 \wedge \varphi_2 \;\; i \in \{1,2\} & \varphi \prec_I \nu Z.\varphi
\end{array}$$

We define $\prec$ to be the transitive closure of $\prec_I$ and $\preceq$ to be its transitive and reflexive closure. A tableau based proof system starts from the formula to be proved as the root of a proof tree and proceeds in a top down fashion. In every rule of the tableau, the conclusion is above the premises. Each conclusion of a certain rule represents a node in the proof tree, whereas the premises represent the children to this node. In our case, the proof system proves sequents of the form $H, b \vdash \sigma \in \varphi$, which means that under a set H of hypotheses and the symbol b, then the strategy σ satisfies the property φ. The set H contains elements of the form $\sigma : \nu Z.\varphi$ and is needed for recursive formulas. Roughly, the use of H is to say that in order to prove that a strategy σ satisfies a recursive formula φ_{rec}, we must prove the following: Under the hypothesis that σ satisfies φ_{rec}, then σ also satisfies the unfolding of φ_{rec}. We also define the set $H \Rsh \nu Z.\varphi = \{\sigma \in L \mid \sigma : \nu Z.\varphi \in H\}$. The use of H, and b will be apparent after we state the rules of the proof system:

$$\begin{array}{lcl}
R_{\neg} & \dfrac{H, b \vdash \sigma \in \neg\varphi}{H, \neg b \vdash \sigma \in \varphi} & \\
R_{\wedge} & \dfrac{H, b, \vdash \sigma \in \varphi_1 \wedge \varphi_2}{H, b_1 \vdash \sigma \in \varphi_1 \quad H, b_2 \vdash \sigma \in \varphi_2} & b_1 \times b_2 = b \\
R_{\nu} & \dfrac{H, b \vdash \sigma \in \nu Z.\varphi}{H' \cup \{\sigma : \nu Z.\varphi\}, b \vdash \sigma \in \varphi[\nu Z.\varphi/Z]} & \sigma : \nu Z.\varphi \notin H \\
R_{[]} & \dfrac{H, b \vdash \sigma \in [r_1 \looparrowright r_2]\varphi}{\xi_1 \quad \xi_2 \quad \ldots \xi_n} & Condition \\
R_{\langle\langle\rangle\rangle} & \dfrac{H, b \vdash \sigma \in \langle\langle A\rangle\rangle\varphi}{H, b \vdash \sigma^A \in \varphi} &
\end{array}$$

Where, $H' = H \setminus \{\sigma' : \Gamma \mid \nu Z.\varphi \prec \Gamma\}$
$\xi_i = H, b_i \vdash r_2\theta_i \in \varphi$

$$condition = \begin{cases} & \forall\theta_i \,.\, \textbf{match}\,(\sigma, r_1, \theta_i) \\ \wedge & b_1 \times b_2 \ldots \times b_n = b \\ \wedge & n > 0 \end{cases}$$

The first rule concerns negation of formulas where $b \in \{\epsilon, \neg\}$ serves as a "memory" to remember negations, in this case $\epsilon\varphi = \varphi$. We define $\epsilon\epsilon = \epsilon$, $\epsilon\neg = \neg\epsilon = \neg$, and $\neg\neg = \epsilon$. Moreover, we define $\epsilon \times \epsilon = \epsilon$, $\epsilon \times \neg = \neg \times \epsilon = \neg$, and $\neg \times \neg = \neg$. The second rule says that in order to prove the conjunction, we have to prove both conjuncts. The third rule concerns proving a recursive formulas, where the construction of the set H, via H', ensures that the validity of the sequent $H, b \vdash \sigma \in \nu Z.\varphi$ is determined only by subformulas of φ [16]. The fourth rule takes care of formulas matching sequences to patterns. Finally, the fifth rule deals with formulas dealing with subtrees of the game tree. Starting from the formula to be proved at the root of the proof tree, the tree grows downwards until we hit a node where the tree cannot be extended anymore, i.e., a leaf node. A formula is proved if it has a successful tableau, where a successful tableau is one whose all leaves are successful. A successful leaf meets one of the following conditions:(1) $H, \epsilon \vdash \sigma \in Z$ and $\sigma \in [\![Z]\!]_e^{\mathcal{G}}$, (2) $H, \neg \vdash \sigma \in Z$ and $\sigma \notin [\![Z]\!]_e^{\mathcal{G}}$, (3) $H, \epsilon \vdash \sigma \in \nu Z.\varphi$ and $\sigma : \nu Z.\varphi \in H$, or (4) $H, \epsilon \vdash \sigma \in [r_1 \looparrowright r_2]\varphi$ and $\{\sigma \in \mathcal{S}_{\mathcal{G}} \mid \exists\theta\,.\ \mathbf{match}\,(\sigma, r_1, \theta)\} = \emptyset$.

4.4 Properties of Tableau System

We would like to prove three main properties, namely the finiteness of the tableau for finite models, the soundness, and the completeness. Soundness and completeness are proved with respect to a relativized semantics that takes into account the set H of hypotheses. The new semantics is the same as the one provided above for all formulas except for recursive formulas where is it defined as:

$$[\![\nu Z.\varphi]\!]_e^{\mathcal{G},H} = (\nu [\![\varphi]\!]_{e[Z \mapsto S \cup S']}^{\mathcal{G},H}) \cup S'$$
$$\text{where, } S' = H \upharpoonright \nu Z.\varphi$$

In the equation, the greatest fixpoint operator is applied to a function $f(S) = [\![\varphi]\!]_{e[Z \mapsto S]}^{L,\theta}$ whose argument is $S \cup S'$. Since the function is monotone over a complete lattice, as mentioned earlier, then the existence of a greatest fixpoint is guaranteed. We now list some results regarding the proof system. The detailed proofs are not provided due to space limitation.

Theorem 4.1 Finiteness. *For any sequent $H, b \vdash \sigma \in \varphi$ there exists a finite number of finite tableaux.*

The idea of the proof is that for any formula at the root of the proof tree we begin applying the rules $R_\neg$, $R_\wedge$, $R_{[]}$, $R_{\langle\langle\rangle\rangle}$, and R_ν. The application of the first four rules results in shorter formulas, while the application of the R_ν rule results in larger hypothesis sets H. The proof shows that shortening a formula and increasing the size of H cannot continue infinitely. Hence no path in the tree will have infinite length. Branching happens in the proof tree whenever we have an expression of the form $\varphi_1 \wedge \varphi_2$ or $[r_1 \looparrowright r_2]\varphi$. Finite branching is guaranteed in the first case by the finite length of any expression and in the second case by the finiteness of the model.

Theorem 4.2 Soundness. *For any sequent $H, b \vdash \sigma \in \varphi$ with a successful tableau, $\sigma \in [\![\varphi]\!]_e^{\mathcal{G},H}$*

The idea behind the proof is to show that all the successful leaves described above are valid and that the application of the rules of the tableau reserves semantic validity.

Theorem 4.3 Completeness. *If for a strategy $\sigma \in \mathcal{S}_{\mathcal{G}}$, $\sigma \in [\![\varphi]\!]_e^{\mathcal{G},H}$, then the sequent $H, b \vdash \sigma \in \varphi$ has a successful tableau.*

The proof relies on showing that we cannot have two successful tableaux for the sequents $H, b \vdash \sigma \in \varphi$ and $H, b \vdash \sigma \in \neg\varphi$.

5 Conclusion

We presented a game semantics model for security protocols that is based on an algebraic description of messages. The message algebra is equipped with an equational theory that enriches the analysis by expressing algebraic properties over the algebra of messages. Moreover the definition of the games makes explicit the role of the intruder in the protocol communications. We also developed a logic that can be used to express security properties. The model and logic can be used to analyze a wide variety of protocols specified in the standard arrow notation.

References

[1] M. Abadi. Security protocols and their properties. In F. Bauer and R. Steinbrueggen, editors, *Foundations of Secure Computation, 20th Int. Summer School, Marktoberdorf, Germany*, pages 39–60. IOS Press, 2000.

[2] M. Abadi and V. Cortier. Deciding knowledge in security protocols under (many more) equational theories. In *Proceedings of the 18th IEEE Computer Security Foundations Workshop*, 2005.

[3] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *POPL*, pages 104–115, 2001.

[4] M. Abadi and A. B. Gordon. A calculus for cryptographic protocols: The SPI calculus. In *Proceedings of the 4th ACM Conference on Computer and Communications Security*, 1997.

[5] M. Abadi and P. Rogaway. Reconciling two views of cryptography (The computational soundeness of formal encryption). *Journal of cryptology*, 15(2):103–127, 2002.

[6] S. Abramsky. Semantics of interaction: An introduction to game semantics. In *Proceedings of the 1996 CLiCS Summer School, Isaac Newton Institute, P. Dybjer and A. Pitts, eds. (Cambridge University Press)*, 1997.

[7] S. Abramsky, P. Malacaria, and R. Jagadeesan. Full abstraction for PCF. In *Theoretical Aspects of Computer Software*, pages 1–15, 1994.

[8] K. Adi, M. Debbabi, and M. Mejri. A new logic for electronic commerce protocols. *Theor. Comput. Sci*, 291(3):223–283, 2003.

[9] R. Alur, T. Henzinger, and O. Kupferman. Alternating-time temporal logic. *JACM: Journal of the ACM*, 49, 2002.

[10] B. Blanchet, M. Abadi, and C. Fournet. Automated verification of selected equivalences for security protocols. In *LICS*, pages 331–340. IEEE Computer Society, 2005.

[11] A. Blass. A game semantics for linear logic. *Annals of Pure and Applied Logic*, 56:183–220, 1992.

[12] M. Boreale and M. Buscemi. A method for symbolic analysis of security protocols. *TCS: Theoretical Computer Science*, 338, 2005.

[13] M. Burrows, M. Abadi, and R. Needham. A logic of authntication. Technical report, Digital Systems Research Center.

[14] I. Cervesato, N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. A meta-notation for protocol analysis. In *PCSFW: Proceedings of The 12th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1999.

[15] E. Clarke, S. Jha, and W. Marrero. Using state space exploration and a natural deduction style message derivation engine to verify security protocols. In *International Conference on Programming Concepts and Methods*, pages 87–106, 1998.

[16] R. Cleaveland. Tableau-based model checking in the propositional mu-calculus. *Acta Informatica*, 27(8):725–748, 1990.

[17] S. Delaune. *Vérification des protocoles cryptographiques et propriétés algébriques*. Thèse de doctorat, Laboratoire Spécification et Vérification, ENS Cachan, France, June 2006.

[18] D. Dolev and A. Yao. On the security of public key protocols. *IEEE transactions on information theory*, 29(2):198–208, 1983.

[19] J. Fabrega, J. Herzog, and J. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, (7):191–230, 1999.

[20] J. M. E. Hyland and C.-H. L. Ong. On full abstraction for PCF: I, II, III. *Info. and Comp.*, 163:285–408, 2000.

[21] J.Clark and J. Jacob. A survey of authentication protocol literature.

[22] S. Kremer and J. Raskin. A game approach to the verification of exchange protocols - application to non-repudiation protocols. In *Proceedings of the Workshop on Issues in the Theory of Security (WITS '00)*, 2000.

[23] K. Lorenz. Basic objectives of dialogue logic in historical perspective. *Synthese (Elsevier)*, 127(1–2), April/May 2001.

[24] L. C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, (6):85–128, 1998.

[25] J. Tucker and J. Zucker. Computable functions and semi-computable sets on many-sorted algebras. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 5, pages 317–523. Oxford University Press, 2000.

[26] T. Woo and S. Lam. A lesson on authentication protocol design. *Operating Systems Review*, pages 24–37, 1994.

A Semantics

A.1 Protocol Types

A type τ of a certain protocol can be defined by the following BNF grammar: $\tau ::= \mathsf{Msg} \multimap \mathsf{Msg} \mid \tau \otimes \tau$. To assign a type to a protocol we use the typing rules below, where $\alpha, \beta, \ldots$ represent single communication steps.

$$\frac{\cdot}{\alpha : \mathsf{Msg} \multimap \mathsf{Msg}} \qquad \frac{\alpha : \mathsf{Msg} \multimap \mathsf{Msg} \quad \mathcal{C}omm : \tau}{\alpha.\mathcal{C}omm : (\mathsf{Msg} \multimap \mathsf{Msg}) \otimes \tau}$$

$$\frac{\mathcal{D}ecl : \tau \quad \mathcal{C}omm : \tau'}{\mathcal{D}ecl.\mathcal{C}omm : \tau'}$$

We define the function $\overline{\tau}$ over the algebra of types where, $\overline{\mathsf{Msg} \multimap \mathsf{Msg}} = 1$ and $\overline{\tau' \otimes \tau'} = \overline{\tau'} + \overline{\tau'}$.

A.2 Protocol Semantics

A protocol is *well-formed* if, in the protocol specification, no message is sent by an agent A unless it can be deduced from $\phi(A)$ (the specification frame). Hereafter we define first the predicate $\mathsf{Wf\_Prot}()$ that checks a protocol for well-formedness, then we define the semantic function of a well-formed protocol.

$$\begin{aligned}
&\mathsf{Wf\_Prot} : \mathcal{C}omm \to \mathsf{Frm} \to \mathsf{Bool} \\
&\mathsf{Wf\_Prot}(\mathcal{C})(\phi) \qquad \phi \text{ is constructed by "}\textbf{frame}\text{" (Section 2.4)} \\
&\quad = \mathsf{Wf\_Prot}(A \to B : m.\mathcal{C})(\phi) \\
&\quad = \phi(A) \vdash m \wedge \mathsf{Wf\_Prot}(\mathcal{C})(\phi)
\end{aligned}$$

In security semantics, we investigate possible manoeuvres that can be performed by the intruder in order to break the protocol's security. For any protocol $\mathcal{P}$ that has type τ, the semantic function assigns to $\mathcal{P}$ a set of strategies over the game $\mathsf{Spr}_{[\overline{\tau}]}$. First, we define the following:

$\mathsf{sess} : \mathsf{Session}$	$\mathsf{Session} : \_ \to \mathsf{N}$ sess is the session number.
$P_G : \mathsf{GameTree}$	Game tree of the game G
$P_G^{sub} : \mathsf{GameTree}$	$P_G^{sub} \subseteq P_G$ a subset of the game tree (subtree)

The semantic function $\mathfrak{P}$ accepts a protocol, a frame, a game tree and a protocol type. It returns a subset of the game tree of the game Spr. This subset represent all possible sequences the game could proceed with:

```
$\mathfrak{P} : \mathcal{P}rot \to \mathsf{Frm} \to \mathsf{GameTree} \to \prod \tau \in \mathcal{T} \,.\, P^{sub}_{\mathsf{Spr}_{[\overline{\tau}]}}$
$\mathfrak{P}[\![\nu A \triangleright m.\mathcal{D}.\mathcal{C}]\!](\rho)(P_{\mathsf{Spr}_{[\overline{\tau}]}})(\tau) \qquad \forall P \,.\, \rho(P) = \nu\emptyset \,.\, [\,]$
    $= \mathfrak{P}[\![\mathcal{D}.\mathcal{C}]\!](\rho[A \mapsto \nu\mathbb{C} \cup m \,.\, \theta_f])(P_{\mathsf{Spr}_{[\overline{\tau}]}})(\tau)$
$\mathfrak{P}[\![\kappa A \triangleright m.\mathcal{D}.\mathcal{C}]\!](\rho)(P_{\mathsf{Spr}_{[\overline{\tau}]}})(\tau)$
    $= \mathfrak{P}[\![\mathcal{D}.\mathcal{C}]\!](\rho[A \mapsto \nu\mathbb{C} \,.\, \theta_f[0 \mapsto < \theta_f(0), m >]])(P_{\mathsf{Spr}_{[\overline{\tau}]}})(\tau)$
$\mathfrak{P}[\![\epsilon.\mathcal{C}]\!](\rho)(P_{\mathsf{Spr}_{[\overline{\tau}]}})(\tau)$
    $= \mathfrak{P}[\![\mathcal{C}]\!](\rho[A \mapsto \nu\mathbb{C} \,.\, \theta_f[0 \mapsto < \theta_f(0), m >]])(P_{\mathsf{Spr}_{[\overline{\tau}]}})(\tau)$
$\mathfrak{P}[\![step\ i \triangleright A \to B : m_i.\mathcal{C}]\!](\rho)(P_{\mathsf{Spr}_{[\overline{\tau}]}})(\tau) =$
    if $i = 1$ then
      $\mathfrak{P}[\![\mathcal{C}]\!](\rho \dagger [I \mapsto \nu\mathbb{C} \,.\, \theta_f[1 \mapsto m_i])$
      $(P_{\mathsf{Spr}_{[\overline{\tau}]}}[q^{11}.start/P_{\mathsf{Spr}_{[\overline{\tau}]}} \upharpoonright \mathsf{Msg}^{\mathsf{sess},11},$
      $q^{12}.m_i^{12}/P_{\mathsf{Spr}_{[\overline{\tau}]}} \upharpoonright \mathsf{Msg}^{\mathsf{sess},12}])(\tau)$
    else
      $\forall m \,.\, \rho(I) \vdash m$ let $n = \mathbf{out}(\rho(A), 3))$
        in
        $\mathfrak{P}[\![\mathcal{C}]\!](\rho \dagger [A \mapsto \nu\mathbb{C} \,.\, \theta_f[i \mapsto m], I \mapsto \nu\mathbb{C} \,.\, \theta_f[1 \mapsto m_i]])$
        $(P_{\mathsf{Spr}_{[\overline{\tau}]}}[q^{i1}.m^{i1}/P_\tau \upharpoonright \mathsf{Msg}^{\mathsf{sess},i1},$
        $q^{i2}.n^{i2}/P_{\mathsf{Spr}_{[\overline{\tau}]}} \upharpoonright \mathsf{Msg}^{\mathsf{sess},i2}])(\tau)$
        end
      end
    end
  $\mathfrak{P}[\![\epsilon]\!](\rho)(P_{\mathsf{Spr}_{[\overline{\tau}]}})(\tau) =$
    $P_{\mathsf{Spr}_{[\overline{\tau}]}}$
```