Algebraic Properties in Alice and Bob Notation

Sebastian Mödersheim

IBM Zurich Research Laboratory, Switzerland
Email: smo@zurich.ibm.com

***Abstract*—Alice and Bob notation is a popular way to describe security protocols: it is intuitive, succinct, and yet expressive. Several formal protocol specification languages are based on this notation. One of the most severe limitations of these languages is the lack of algebraic reasoning, which is required for instance for the correct interpretation of Diffie-Hellman based protocols. As a consequence, previous approaches either cannot handle such protocols at all or require manual annotation. We generalize previous approaches and give the first formal semantics for a language based on Alice and Bob notation that is defined over an arbitrary algebraic theory. In particular, it defines unambiguously how the protocol is supposed to be executed by honest agents, based on the considered algebraic properties of the operators.**

***Keywords*-Protocol Verifcation, Alice and Bob, Algebraic Reasoning, Semantics**

I. Introduction

In the literature on security protocols, the semi-formal Alice and Bob notation is very popular, since it is a simple, succinct and intuitive way to describe security protocols, see e.g. [1]. Therefore, Alice and Bob notation has been used as the basis of several formal specification languages in the context of formal analysis of security protocols.

One of the first approaches is [2] which translates an Alice and Bob specification into the process calculus CSP [3] for model-checking with FDR. Similarly, the CAPSL Integrated Protocol Environment (CIPE) [4] is based on an Alice and Bob style input language, CAPSL, that is translated into a low-level intermediate format, CIL, based on multi-set rewrite rules. The idea of this intermediate format is that a variety of different analysis tools can be connected to CIPE based on the same input language CIL. Both [2], [4] are limited in recognizing how messages are parsed and generated; they require manual annotation when an agent is not supposed to fully decrypt a received message or when one needs algebraic reasoning to construct a message.

The parsing problem was first addressed in [5], [6], which is the basis of the Casrul system. In particular, Casrul can automatically handle protocol specifications where an agent first receives an encrypted message and in a later step the decryption key for it: the agent can then check that the message received earlier has the required contents. In § V, we discuss the partial support for algebraic properties in [5], [6].

All these approaches define a language with a formal semantics by the translation to another formally defined language, like process calculi, multi-set rewriting or so-called incremental symbolic runs [7]. The translation from Alice and Bob to more low-level formats therefore links theoretical aspects, namely a formal semantics, with practical aspects, namely integrating a variety of verification tools in a meaningful way. The value of such an integration is the complementarity of approaches: writing one specification, we can use a broad spectrum of tools with different strengths to analyze the protocol. Finally, the semantics of Alice and Bob notation is also useful beyond formal verification: the output of the translation can be a real implementation of the protocol with calls to an appropriate cryptographic library [8], [9].

Contributions: While many approaches to the formal analysis of protocols have considered algebraic properties (e.g. [10]–[12]), little has been done to also support algebraic properties in Alice and Bob style specification languages. This rules out (or requires manual annotation of) protocols that crucially depend on algebraic properties of the cryptographic primitives, such as Diffie-Hellman based protocols.

We define the formal protocol specification language *AnB* (for *Alice and Bob*) and thereby generalize previous approaches, giving the first formal semantics for an AnB-style language defined over an arbitrary algebraic theory. In particular, our semantics defines unambiguously how the protocol is executed by honest agents, based on the algebraic properties of the operators.

In general, the semantics of a formal language should be designed in a clear and declarative way, without regard to implementation problems. In a second step, one can then prove that a certain way to implement it is correct. In this spirit, the semantics of AnB is defined without regard to decidability problems of the considered algebraic theories and similar issues. Also,

Protocol : *Diffie-Hellman*

Types :

  Agent A, B;

  Number g, X, Y, Msg;

  Function pk;

Knowledge :

  A : $A, B, g, pk, \mathsf{inv}(pk(A))$;

  B : $B, g, pk, \mathsf{inv}(pk(B))$;

Actions :

$$A \rightarrow B : A, \{\mathsf{exp}(g, X)\}_{\mathsf{inv}(pk(A))}$$
$$B \rightarrow A : B, \{\mathsf{exp}(g, Y)\}_{\mathsf{inv}(pk(B))}$$
$$A \rightarrow B : \{\!|A, Msg|\!\}_{\mathsf{exp}(\mathsf{exp}(g,X),Y)}$$

Goals :

  B authenticates A on Msg

  Msg secret of A, B

Figure 1. Example protocol in AnB.

according to our semantics, an honest agent can perform an infinite number of checks on a received message. In the extended version of this paper [13], we show for one practically relevant algebraic theory that we can always compute an equivalent finite set of checks and that the relevant problems are decidable.

The target language of our translation is the AVISPA Intermediate Format IF [14] as a way to describe transition systems. IF is the input language to a variety of tools [15], and we have implemented a prototype of the translator.

Organization: The rest of the paper is organized as follows. In § II we introduce the syntax and intuition of our AnB language as well as our running example. In § III, we give an overview of the semantics, including its definition in previous approaches and the limitations that arise there. We also use this section to give an overview of our target formalism AVISPA IF. In § IV, we go into the details of modeling messages and their construction and deconstruction by honest agents in the presence of algebraic properties. In § V we conclude with an outlook on future work.

II. AnB Syntax

We introduce the syntax of our formal language AnB by the example in figure 1, a protocol based on the Diffie-Hellman key exchange.The full syntax is found in [13]. Our definition of AnB is independent of the concrete set of operators and algebraic properties considered. For concreteness, we use an example theory which is informally explained in the following and formally defined in example 2.

The specification consists of the following sections:

Types: AnB requires that the types of all identifiers of the protocol specification are declared, except those that are predefined like exp (set in sans-serif, see example 2). We distinguish two kinds of identifiers. The ones that start with an upper-case letter are called *protocol variables* and are instantiated during the protocol execution. The ones that start with a lower-case letter represent global constants and functions. Protocol variables of type Agent are called *roles*. In the example, we have the roles A and B which get instantiated by arbitrary agents when executing the protocol. The numbers g, X, and Y are the group and the random exponents of the Diffie-Hellman key exchange. Finally, pk is a function that we use as a public-key table, yielding the public key for every agent.

Knowledge: The next item of an AnB specification is the initial knowledge attached to each role, consisting of a set of messages. We require that all variables that occur in the initial knowledge section are of type Agent. For long-term keys, we use functions like pk. Here, both A and B know the function pk, meaning that they can look up the public key of every agent. Also they know their own private key $\mathsf{inv}(pk(A))$ and $\mathsf{inv}(pk(B))$, respectively. The function $\mathsf{inv}(P)$ gives the private key belonging to a given public-key P. Unlike other functions, $\mathsf{inv}(\cdot)$ is not a *public* function that can be applied by agents (see example 2).

The initial knowledge is essential for the semantics, as the construction of messages depends on it. Variables that do not occur in the initial knowledge of any role represent values that are freshly created by the agent who first uses them. In the example, X and Msg are created by A and Y is created by B.

Actions: The core of the specification is the list of exchanged messages. They describe the ideal, unattacked run of the protocol. Every action has the form $A \rightarrow B : M$, meaning that role A sends the message M to role B. The receiver of a message must be the sender of the next one. The core of the semantics is to define a "program" for each role of the protocol.

In the example, A and B generate random values X and Y and exchange the Diffie-Hellman *half keys* $\mathsf{exp}(g, X)$ and $\mathsf{exp}(g, Y)$ (we omit the modulus in our notation). Both half keys are signed by the respective agent with their private key as indicated by the $\{m\}_k$ notation that represents asymmetric encryption. The final message is a payload message Msg (modeled as a random number) from A to B that is encrypted symmetrically using the new Diffie-Hellman key $\mathsf{exp}(\mathsf{exp}(g, X), Y)$ of A and B. Note that this description is only meaningful if exp has the property

$\mathsf{exp}(\mathsf{exp}(g, X), Y) \approx \mathsf{exp}(\mathsf{exp}(g, Y), X)$ as otherwise A cannot construct this key.

Goals: Finally, we specify the goals that the protocol is supposed to achieve, in this case secrecy and authentication of a transmitted message Msg.

III. Semantics Overview

We now give an overview of the semantics, including how it was defined in previous approaches to AnB-style languages. Also, we summarize the main features of the target formalism in which we express the semantics: the AVISPA Intermediate Format IF [14].

The AnB description defines the possible behaviors of a system composed of an unbounded number of sessions (protocol runs) of honest agents and an intruder. We specify these possible behaviors by means of an infinite-state transition system, namely an initial state and a transition relation on states. The reachable states of this system represent what can happen in the idealized world we consider. Goals are defined using predicates on the states that specify which states are *attack states*. The verification question is whether an attack state is reachable. Alternatively, one may define goals using predicates on traces of events or of states.

A. IF States

In IF, a state is a finite set of *facts*, separated by dots (“.”). For now, we only need two kinds of facts: $\mathsf{iknows}(m)$ expresses that the intruder knows the message m, and $\mathsf{state}_{\mathcal{R}}(m_1, \ldots, m_n)$ expresses that an agent, playing role $\mathcal{R}$, is in a (local) state of a protocol execution characterized by the list of messages $m_1, \ldots, m_n$. Usually this list contains the messages that the agent initially knows, the messages received so far, and fresh constants it has generated in the session. We use one state fact for every session that an agent participates in. Recall that the role names like A and B that appear in the AnB specification are protocol variables of type agent; since the role parameter of $\mathsf{state}_{\mathcal{R}}(\cdot)$ facts should identify the role of the respective agent rather than being instantiated with the concrete agent name, we need a special constant for each role, which we denote with calligraphic letters. Also, we silently assume that every $\mathsf{state}_{\mathcal{R}}(\cdot)$ fact contains a unique identifier; this allows for several instances of an agent in the same local state of the protocol execution. All reachable states are ground terms, i.e. they do not contain variables: the initial state does not contain variables and no transition rule introduces variables.

B. IF Initial State

Most protocol analysis tools require a bound on the number of sessions that honest agents can perform. In this case, the initial state contains the initial local state for each honest agent and session, as well as the initial intruder knowledge. For the general case without a bound on the number of sessions, we need an initialization theory as discussed in [13].

Recall that the initial knowledge for each role may only contain protocol variables of type agent (i.e. roles)—these need to be instantiated for each session. Let thus $\sigma_1, \ldots, \sigma_n$ be mappings (for n sessions) from the roles $R_1, \ldots, R_m$ to agent names. Let M_j be the initial knowledge of role R_j, then the initial state is:

$$\bigcup_{1 \le i \le n, 1 \le j \le m} \begin{cases} \mathsf{state}_{\mathcal{R}_j}(M_j\sigma_i) & \text{if } R_j\sigma_i \neq \mathsf{i} \\ \mathsf{iknows}(M_j\sigma_i) & \text{if } R_j\sigma_i = \mathsf{i} \end{cases}$$

where i is a special constant: the name of the intruder. Here, we have used σ as a substitution that replaces all protocol variables in the message M_j with agent names. Again, each $\mathcal{R}_j$ is a constant identifying the role R_j (to avoid variables in the IF initial state). The initial state thus consists of the local states of honest agents ($R_j\sigma_i \neq \mathsf{i}$) and the initial knowledge of the dishonest agent ($R_j\sigma_i = \mathsf{i}$).

For the example of figure 1 and the instantiation $\sigma = [A \mapsto a, B \mapsto \mathsf{i}]$, we get the following facts for instance:

$$\mathsf{state}_A(a, \mathsf{i}, g, pk, \mathsf{inv}(pk(a))).\mathsf{iknows}(\mathsf{i}, g, pk, \mathsf{inv}(pk(\mathsf{i})))$$

The second fact allows the intruder to execute the protocol in role B under his real name i.

C. IF Transition Rules

We consider IF transition rules of the following form (for details on the semantics see [14]):

$$L \mid EQ \;\text{=}[V]\Rightarrow\; R$$

with sets L and R of facts, a set EQ of equations of terms, and a list of variables V. The semantics of this rule is defined by the state transitions it allows: we can get from a state S to a state S' with this rule iff there is a substitution σ of all rule variables such that $L\sigma \subseteq S$, $S' = S \setminus L\sigma \cup R\sigma$, $V\sigma$ are fresh constants (that do not appear in S) and all equations of EQ are satisfied under the substitution σ. All equalities between terms/facts are modulo the considered algebra.

As an example, the protocol independent transition rules of the intruder can be expressed as transition rules based on $\mathsf{iknows}(\cdot)$ facts, e.g

$$\mathsf{iknows}(\{|M|\}_K).\mathsf{iknows}(K) \Rightarrow \mathsf{iknows}(M) \quad (1)$$

allows the intruder to obtain the plaintext of a symmetrically encrypted message if he knows the encryption key. Actually, the left-hand side facts should be repeated on the right-hand side as the intruder does not forget the encrypted message and the key. For simplicity of notation, however, $\mathsf{iknows}(\cdot)$ facts are defined to be *persistent*: when present in a state, they are automatically present in all successor states. Thus we do not need to repeat these facts on the right-hand side of transitions. The intruder-behavior is often fixed in verification tools and handled in a special way (e.g. without a transition for every intruder deduction); we only mention these intruder rules for completeness of the semantics.

D. Transitions of Honest Agents

The core of the translation from AnB to transition systems is to generate rules for the behavior of the honest agents. From the point of view of a particular role R, the protocol looks like this:

$$\begin{array}{cccc} \_ & \to & R & : & i_1 \\ R & \to & \_ & : & o_1 \\ & & \vdots & & \\ \_ & \to & R & : & i_l \\ R & \to & \_ & : & o_l \end{array}$$

If R is the sender of the first message of the protocol, then we simply set $i_1 = \mathsf{i}$ as a dummy message; similarly, if R is the receiver of the last message of the protocol, $o_l = \mathsf{i}$.

The idea is that the local state M of the honest agents starts with the initial knowledge of the role and is extended in every transition by the incoming message and the freshly created variables. As often done, transitions include both receiving a message and replying. Also, we identify the intruder knowledge and the communication medium [17]. The mth transition rule for an honest agent in role $\mathcal{R}$ is thus the following:

$$\begin{array}{l} \mathsf{state}_{\mathcal{R}}(\mathit{init}, i_1, f_1, \ldots, i_{m-1}, f_{m-1}).\mathsf{iknows}(i_m) \\ =\![f_m]\!\Rightarrow \\ \mathsf{state}_{\mathcal{R}}(\mathit{init}, i_1, f_1, \ldots, i_m, f_m).\mathsf{iknows}(o_m) \end{array}$$

where f_i denotes the list of fresh variables of o_i (that did not occur in any message of the protocol specification before) and *init* is the initial knowledge of role $\mathcal{R}$. This rule thus allows for a transition for an agent in role $\mathcal{R}$ that has processed the first $m-1$ steps of its protocol execution and receives the mth incoming message i_m on the insecure medium (i.e. the intruder has sent some matching message to this agent). The agent creates fresh constants for the values f_m (as denoted by the arrow), appends them with the incoming message to its current state, and sends the outgoing message o_m to the communication medium (i.e. the intruder).

Although many details are different, this formulation of the translation reflects the semantics of the first formal AnB-style approaches [2], [4], and it is sufficient for a large class of protocols. However, there are some aspects that are not appropriately handled by this semantics which we illustrate by an example.

Example 1. *According to this schema, the first transition of B in the example figure 1 is the following rule:*

$$\begin{array}{l} \mathsf{state}_{\mathcal{B}}(B, g, pk, \mathsf{inv}(pk(B))). \\ \mathsf{iknows}(A, \{\mathsf{exp}(g, X)\}_{\mathsf{inv}(pk(A))}) \\ =\![Y]\!\Rightarrow \\ \mathsf{state}_{\mathcal{B}}(B, g, pk, \mathsf{inv}(pk(B)), \mathsf{exp}(g, X), Y). \\ \mathsf{iknows}(B, \{\mathsf{exp}(g, Y)\}_{\mathsf{inv}(pk(B))}) \end{array}$$

In this rule, B accepts as an incoming message only instances of the pattern $A, \{\mathsf{exp}(g, X)\}_{\mathsf{inv}(pk(A))}$. At least for the exponentiation this is quite unrealistic: not knowing the value X, it should be impossible to check that the received value is really of this "form". This neither makes sense cryptographically nor in the idealized world of black-box cryptography, and the agent should rather accept anything here. The rule should thus use a fresh variable GX instead of $\mathsf{exp}(g, X)$ that can be matched against any term.

Thus, the most important problem of the described translation based on pattern matching is the assumption that honest agents only accept messages that are instances of the messages in the protocol description, even if they actually cannot always check that. Another frequent example is a protocol where A sends to B a message $m = \{\!|A, B, N|\!\}_{k(A,T)}$ encrypted for a third party T (that B should forward to T in a later step). Not knowing the shared key $k(A, T)$ of A and T, B cannot check that the received message has the appropriate form (i.e. is an instance of m)—B should accept any message here.

The first approaches used the so-called Lowe-operator to annotate manually how such a message is received. [5] was the first approach to automatically compute the correct pattern. In a nutshell, the approach checks "how deep" the receiver can analyze the message based on its current knowledge; what cannot be analyzed (due to an unknown key) is replaced by a fresh variable, a "blinding" in the terminology of [7]. Running this receive check on the entire knowledge and received message in every step of the agent allows for the correct handling even of the following situation. An agent later receives a key that allows for the decryption and check

of previous messages, e.g. in the above example, B receives $k(A,T)$.

The problems with this blinding/pattern matching approach begin when we consider algebraic properties, such as for the Diffie-Hellman example in figure 1. In fact, the meaning of the protocol crucially depends on an algebraic property of exponentiation.

Finally, the definition of goals by means of attack states is standard, see [13]. After this overview of the semantics and the problems of previous approaches, we look at the details of interpreting messages and computing how honest agents compose and check messages.

IV. Message Model

We formally define the basis of the AnB semantics, the model of messages as terms in the presence of algebraic properties. First, we define how an agent can derive new messages from existing ones. To that end we use a Dolev-Yao style model with labels that reflect the actual operations performed by the agents. Second, we define what checks an agent can perform on received messages. Finally, we integrate this model into the generation of IF transition rules for the honest agents, replacing the naïve rule generation sketched in § III-D.

A. Message Derivation

Definition 1. *A* message model $(\Sigma, \approx, \Sigma_p)$ *consists of a signature* Σ *(i.e. a countable set of symbols with arities), a congruence relation* $\approx$ *between terms over* Σ*, and a subset* $\Sigma_p \subseteq \Sigma$ *called the* public *symbols. We require that* Σ_p *contains the name of the intruder* i*. Note that* Σ *contains all identifiers of AnB, including protocol variables.*

Let $\mathcal{V} = \{\mathcal{X}_0, \mathcal{X}_1, \ldots\}$ *be a set of variable symbols disjoint from* Σ*. A* ground term *is a term without variables. A* labeled *message* t^l *consists of a ground term* $t \in \mathcal{T}_\Sigma$ *that is labeled by a term* $l \in \mathcal{T}_\Sigma(\mathcal{V})$ *where* l *may have variables.*

For a set of labeled messages M*, the* deduction closure $\mathcal{DY}(M)$ *is the least set closed under the following rules:*

$$\frac{}{m^l \in \mathcal{DY}(M)}\ m^l \in M \qquad \frac{t^l \in \mathcal{DY}(M) \quad s \approx t}{s^m \in \mathcal{DY}(M)}\ l \approx m$$

$$\frac{t_1^{l_1} \in \mathcal{DY}(M) \quad \ldots \quad t_n^{l_n} \in \mathcal{DY}(M)}{f(t_1,\ldots,t_n)^{f(l_1,\ldots,l_n)} \in \mathcal{DY}(M)}\ f \in \Sigma_p$$

The deduction closure $\mathcal{DY}$ is defined in the spirit of the standard Dolev-Yao intruder model [18], hence the name. Additionally, we label terms with the way they have been deduced, which we need for our definition of the rule generation. The first rule expresses that every initially given term of M can be deduced. The second rule expresses that deduction is closed under algebraic properties. The third rule expresses that deduction is closed under application of public operations, i.e. knowing terms $t_1, \ldots, t_n$ one can apply an n-ary public operation f to them; the label is added accordingly. Since $\mathcal{DY}$ is defined as the least set closed under the rules, it reflects the black-box cryptography model: the intruder can only perform standard encryption and decryption operations with known keys, and no rule allows him to break the cryptography.

$\mathcal{DY}$ is central to the definition of AnB: the honest agents must be able to form every message of the protocol according to the $\mathcal{DY}$ model. This also ensures that the behavior of honest agents is subsumed by the abilities of the intruder, i.e. given the initial knowledge of a role, the intruder can execute the protocol in that role as a dishonest participant.

Example 2. *For concreteness, we now describe the message model that we use as a running example and on which the current implementation of the translator for AnB to IF is based.* Σ *contains the following operators:* i *is the name of the intruder,* $\langle m_1, m_2 \rangle$ *is the concatenation of* m_1 *and* m_2*,* $\{\!|m|\!\}_k$ *and* $\{m\}_k$ *are the symmetric and asymmetric encryption of* m *with* k*,* $\mathsf{inv}(k)$ *is the private key belonging to public key* k*,* $\mathsf{exp}(b,x)$ *is the modular exponentiation of* b *to* x*,* $m_1 \oplus m_2$ *is the bitwise exclusive or (xor) of* m_1 *and* m_2*,* e *is the neutral element of* $\oplus$*,* $f(m)$ *is the application of a function symbol* f *to* m*. Additionally, there are the operations* π_1 *and* π_2 *for projection, i.e. the destructor for concatenation, which may not occur in the AnB specification itself (but only in the translation). All these symbols are public but* $\mathsf{inv}(\cdot)$*.*

The congruence $\approx$ *that we consider with this example signature is defined by the following properties. Encryption and decryption cancel each other out, i.e.* $\{\!|\{\!|m|\!\}_k|\!\}_k \approx m$ *and* $\{\{m\}_k\}_{\mathsf{inv}(k)} \approx m$*. In this model, encryption and decryption are the same operation while in general one may model them as different operations. The property* $\mathsf{inv}(\mathsf{inv}(k)) \approx k$ *expresses that two* $\mathsf{inv}(\cdot)$ *cancel each other out. (Note that our theory thus implies* $\{\{m\}_{\mathsf{inv}(k)}\}_k \approx m$*). The properties* $\pi_i(\langle m_1, m_2 \rangle) \approx m_i$ *and* $\langle \pi_1(m), \pi_2(m) \rangle \approx m$ *express the relationship between concatenation and projections. For Diffie-Hellman based protocols we need the property* $\mathsf{exp}(\mathsf{exp}(B,X),Y) \approx \mathsf{exp}(\mathsf{exp}(B,Y),X)$*. Finally, the properties* $A \oplus B \approx B \oplus A$*,* $A \oplus (B \oplus C) \approx (A \oplus B) \oplus C$*,* $A \oplus A \approx e$*, and* $A \oplus e \approx A$ *characterize the xor operation. There are other properties one could formalize, however, these are sufficient for understand-*

ing the execution of a large class of protocols.

Consider now the following given knowledge, where each term is labeled with a variable:

$$M = \{\, \{\!|Msg|\!\}^{\mathcal{X}_1}_{\mathsf{exp}(\mathsf{exp}(g,X),Y)}\,,\; X^{\mathcal{X}_2}\,,\; \mathsf{exp}(g,Y)^{\mathcal{X}_3}\,\}$$

For instance, one can derive Msg from M as follows (omitting "$\in \mathcal{DY}(M)$" in the statements for brevity):

$$\dfrac{\dfrac{\{\!|Msg|\!\}^{\mathcal{X}_1}_{\mathsf{exp}(\mathsf{exp}(g,X),Y)}}{\mathsf{exp}(\mathsf{exp}(g,X),Y)^{\mathsf{exp}(\mathcal{X}_3,\mathcal{X}_2)}} \quad \dfrac{\dfrac{\overline{\mathsf{exp}(g,Y)^{\mathcal{X}_3}} \quad \overline{X^{\mathcal{X}_2}}}{\mathsf{exp}(\mathsf{exp}(g,Y),X)^{\mathsf{exp}(\mathcal{X}_3,\mathcal{X}_2)}}}{Msg^{\{\!|\mathcal{X}_1|\!\}_{\mathsf{exp}(\mathcal{X}_3,\mathcal{X}_2)}}}$$

The root label $\{\!|\mathcal{X}_1|\!\}_{\mathsf{exp}(\mathcal{X}_3,\mathcal{X}_2)}$ exactly describes how Msg is obtained from the components of the given knowledge.

The labeled deduction is now employed to reflect two *views* on a term: t^l represents a message that, according to the protocol, should have the structure t, while l reflects how an honest agent sees or constructs the message. Thus, the actual messages transmitted in the protocol will be instances l. Consider for instance the message labeled $\mathcal{X}_1$: according to the protocol, it is an encrypted message, but a priori one cannot check that a received message has such a format. As the label of the derived message Msg suggests, an honest agent will take as *Msg* the result of decrypting message $\mathcal{X}_1$ using as decryption key whatever results from exponentiating $\mathcal{X}_3$ with $\mathcal{X}_2$. If $\mathcal{X}_1$ is not a properly encrypted message, the result of this decryption will be garbage—which can not be detected by the agent in general.

B. Checking Messages

We now define how agents can check the messages they receive. We describe this based on the deduction closure: we look for two derivations $t_1^{l_1}$ and $t_2^{l_2}$ such that $t_1 \approx t_2$ (the terms are equal according to the protocol), but $l_1 \not\approx l_2$ (they have been derived in different ways).

Definition 2. *A* consistency check *is a pair (l_1, l_2) of terms with variables. An* interpretation $\mathcal{I}$ *is a mapping from $\mathcal{V}$ to $\mathcal{T}_\Sigma$. An interpretation $\mathcal{I}$* models *a set of consistency checks C iff $\mathcal{I}(l_1) \approx \mathcal{I}(l_2)$ for all $(l_1, l_2) \in C$.*

For a set of labeled messages M we define the complete set of consistency checks of M *as*

$$ccs(M) = \{(l_1, l_2) \mid t_1^{l_1}, t_2^{l_2} \in \mathcal{DY}(M) \wedge t_1 \approx t_2 \wedge l_1 \not\approx l_2\}\,.$$

A subset $C \subseteq ccs(M)$ of the complete set of consistency checks is called sufficient *iff all interpretations $\mathcal{I}$ that are are models of C are also a model of $ccs(M)$. Note that, vice-versa, all models of $ccs(M)$ are also models of C, since $C \subseteq ccs(M)$.*

Example 3. *Let $M = \{\, h(N)^{\mathcal{X}_1}\,,\, N^{\mathcal{X}_2}\,\}$ for a public h. We can derive $h(N)$ in two ways, namely $\mathcal{X}_1$ and $h(\mathcal{X}_2)$ obtaining the check $(\mathcal{X}_1, h(\mathcal{X}_2))$. While $ccs(M)$ is infinite (e.g. one can also check $(h(\mathcal{X}_1), h(h(\mathcal{X}_2)))$), the set $\{(\mathcal{X}_1, h(\mathcal{X}_2))\}$ is already sufficient.*

In practice we cannot use an infinite set of checks in transition rules of honest agents. Therefore we need to restrict the set of checks to a finite one in the real translator, if possible a sufficient one. (It is not clear if such a finite sufficient set exists in general.) The restriction to an insufficient subset of $ccs(M)$ bears only the risk of false attacks (that are caused by honest agents accepting messages in our model that they do not accept in reality). However, a precise solution is desirable. In the extended version [13], we show that this is always possible for the example theory:

Theorem 1. *For the algebraic theory of example 2, it is decidable whether a d exists such that $t^d \in \mathcal{DY}(M)$ for a given t and a given finite M, and if it exists, such a d is computable. Moreover, for this theory, a finite sufficient $C \subseteq ccs(M)$ is computable, given a finite M.*

Some remarks about our definition are in order. Our model of consistency checks on messages differs from all previous approaches to AnB notation such as the first sketch of a semantics in § III-D. Previous approaches have used pattern matching to describe the set of messages acceptable to the recipient, where variables in the pattern represent arbitrary subterms. We have entirely replaced this pattern matching concept with the checks of definition 2.

The main advantages of our approach are declarativity and generality: $ccs(\cdot)$ allows us to define straightforwardly how messages are parsed in an arbitrary algebraic theory without using complicated, specialized procedures to obtain message patterns as in [5]–[7].

We illustrate this point by an example that cannot be handled correctly by any previous approach to formalizing Alice and Bob notation:

Example 4. *Consider knowledge $M = \{(a \oplus b \oplus c \oplus d)^{\mathcal{X}_1}, (a \oplus b)^{\mathcal{X}_2}, (a \oplus c)^{\mathcal{X}_3}, h(b \oplus d)^{\mathcal{X}_4}, h(c \oplus d)^{\mathcal{X}_5}\}$*

According to our definitions, one can compute for instance $(c \oplus d)^{\mathcal{X}_1 \oplus \mathcal{X}_2}$ and $(b \oplus d)^{\mathcal{X}_1 \oplus \mathcal{X}_3}$ from the knowledge M and therefore obtain the checks $(h(\mathcal{X}_1 \oplus \mathcal{X}_2), \mathcal{X}_5)$ and $(h(\mathcal{X}_1 \oplus \mathcal{X}_3), \mathcal{X}_4)$. These are exactly the checks an agent can realistically perform on M. In contrast, previous approaches are based on identifying the largest subterms of a message that cannot be "parsed"

by the receiver, a notion which is ambiguous here.

Another subtle issue is whether agents can generally recognize correct encryptions when the decryption key is known. For instance, consider a protocol that contains the step $A \to B : \{\!|N|\!\}_K$ where K is a shared key of A and B, and N is a fresh random number. The question is then whether B can distinguish a properly encrypted message $\{\!|\cdot|\!\}_K$ from anything else. (This is not an issue if the plaintext contains something that can be checked by the receiver, like a tag or a known message part.) Many cryptosystems indeed have this property of recognizability while plain xor does not, for instance.

Our semantics allows for modeling both recognizable and unrecognizable encryption schemes: whether one can check for correct decryption is determined by the considered algebraic properties of the operators.

Example 5. *In our example theory, all operators are unrecognizable. (Example 6 illustrates the behavior of an agent who cannot check anything about the content of a supposedly encrypted message.) It is straightforward to define an alternative model where for instance symmetric encryption is recognizable, e.g. by adding the following property to the algebraic theory:* $verify(\{\!|m|\!\}_k, k) \approx true$ *for two new public symbols* $verify$ *and* $true$*. The receiver of an encrypted message with the knowledge* $M = \{\{\!|m|\!\}_k^{\mathcal{X}_1}, k^{\mathcal{X}_2}\}$ *for instance, can then perform the check* $(verify(\mathcal{X}_1, \mathcal{X}_2), true)$*. A way to achieve recognizability without changing the theory is to explicitly include publicly known tags into the encryptions, but this requires to include such details in the AnB specification of a protocol.*

C. Translation

The generation of IF transition rules for the honest agents is based on the functions $ccs(M)$ and $\mathcal{DY}(M)$:

Definition 3. *We give the transition rule for an agent in role* $\mathcal{R}$ *in a local state characterized by the message list* $m_1, \ldots, m_{k-1}$*. This list represents the initial knowledge of* $\mathcal{R}$ *in the case of the first transition, and the resulting state of the previous transition otherwise. Let* m_k *be the currently incoming message (from the AnB specification), and let* $m_{k+1}, \ldots, m_{k+l}$ *be the freshly generated messages of the transition. Let* $\mathcal{X}_1, \ldots, \mathcal{X}_{k+l}$ *be variables from* $\mathcal{V}$*, and* $M = \{m_i^{\mathcal{X}_i} \mid 1 \le i \le k+l\}$*. Let finally* t *be the outgoing message of the transition.*

Compute a label d *such that* $t^d \in \mathcal{DY}(M)$ *(we discuss below that the meaning of the rule does not depend on the choice of* d *if several such labels exist). If there is no such label, then the agent cannot compose the outgoing message* t*; the protocol is hence called* unexecutable *and its semantics is undefined. The transition rule is then (where* $ccs(M)$ *is considered as a set of equations):*

$$\begin{aligned}&\mathsf{state}_{\mathcal{R}}(\mathcal{X}_1, \ldots, \mathcal{X}_{k-1}).\mathsf{iknows}(\mathcal{X}_k) \;\mid\; ccs(M)\\ &=\![\mathcal{X}_{k+1}, \ldots, \mathcal{X}_{k+l}]\!\Rightarrow\\ &\mathsf{state}_{\mathcal{R}}(\mathcal{X}_1, \ldots, \mathcal{X}_{k+l}).\mathsf{iknows}(d)\end{aligned}$$

When several different derivations for t exist, the choice of d is unspecified. The semantics is unambiguous, however, because in all models of $ccs(M)$, all derivations of t must be equal according to the definition of $ccs(M)$. Thus, the meaning of the rule does not depend on this choice.

We use the variables $\mathcal{X}_1, \ldots, \mathcal{X}_{k+l}$ to refer to the messages that the agent currently knows, the incoming message and the fresh variables. The conditions of $ccs(M)$ determine which interpretation of these variables are consistent with the checks that the agent performs, and thereby, which incoming messages are acceptable. The updated state of the agent contains the incoming message and the freshly generated values. The outgoing message is simply the label d that we have computed for the outgoing message t, a term based on the variables $\mathcal{X}_i$.

Example 6. *Consider the last transition of role* A *in figure 1. For readability, we use the name of the protocol variables rather than the meta variables* $\mathcal{X}_i$ *and leave the constants* g *and* pk *as such. Similarly, we use the variable name* P *for the private key of* A *and* Z *for the incoming message. The rule then reads as follows:*

$$\begin{aligned}&\mathsf{state}_{\mathcal{A}}(A, B, g, pk, P, X).\mathsf{iknows}(Z)\\ &\mid \pi_1(Z) \approx B\\ &=\![Msg]\!\Rightarrow\\ &\mathsf{state}_{\mathcal{A}}(A, B, g, pk, P, X, Z, Msg).\\ &\mathsf{iknows}(\{\!|A, Msg|\!\}_{\mathsf{exp}(\{\pi_2(Z)\}_{pk(B)}, X)})\end{aligned}$$

Here, the equation $\pi_1(Z) \approx B$ *ensures that the first component of the received message is the name of agent* B *that* A *initially wanted to talk to. Note that* A *cannot check anything else about the received message* Z*, since in our example theory, correct encryptions are not recognizable; an alternative model may be based on a function like* $verify$ *in example 5.*

To obtain the message part that is supposed to be $\mathsf{exp}(g, Y)$ *according to the protocol,* A *constructs the term* $\{\pi_2(Z)\}_{pk(B)}$*, i.e. she applies the asymmetric encryption/decryption operation to the second projection of the received message* Z*.* $\pi_2(Z)$ *may thus be an arbitrary term and* A *continues with whatever results from the decryption operation to generate the Diffie-Hellman key for the outgoing message as* $\mathsf{exp}(\{\pi_2(Z)\}_{pk(B)}, X)$*.*

V. Conclusions

We have defined AnB, a formal protocol description language based on Alice and Bob notation. In particular, we have given the first semantics for Alice and Bob notation that works in the context of an arbitrary algebraic theory.

[5] is the first Alice and Bob style language that considered algebraic properties, namely to model explicit decryption. Due to the combination of explicit and implicit decryption, the approach however does not work properly for several protocols. The corrected version [6] thus uses only implicit decryption, but adds partial support for xor.

Our semantics defines an infinite number of checks that honest agents can perform on incoming messages. We have shown that we can produce equivalent rules with a finite number of checks for a practically relevant algebraic theory. There seem to be similarities between this problem of checks and the notion of static equivalence [19]. Existing work here may thus help to identify further algebraic theories that allow for a finite set of checks. Such an investigation is left for future work.

Our prototype implementation can successfully handle a growing testsuite of protocols based on the AVISPA library, including IKEv2, H.530, Kerberos, and TLS [20]. Recently, we have also extended AnB with a notation of secure and pseudonymous channels [21].

Acknowledgment

This work is partially supported by the FP7-ICT-2007-1 Project no. 216471, AVANTSSAR: Automated Validation of Trust and Security of Service-oriented Architectures. The author thanks Luca Viganò, David Basin, and Benedikt Schmidt for helpful comments.

References

[1] LSV, "Security Protocols Open Repository (SPORE)," www.lsv.ens-cachan.fr/spore/.

[2] G. Lowe, "Casper: a Compiler for the Analysis of Security Protocols," *Journal of Computer Security*, vol. 6, no. 1, pp. 53–84, 1998.

[3] C. A. R. Hoare, "CSP — Communicating Sequential Processes," 1985.

[4] G. Denker, J. Millen, and H. Rueß, "The CAPSL Integrated Protocol Environment," SRI International, Menlo Park, CA, Tech. Rep. SRI-CSL-2000-02, 2000.

[5] F. Jacquemard, M. Rusinowitch, and L. Vigneron, "Compiling and Verifying Security Protocols," in *LPAR 2000*, ser. LNCS 1955, 2000, pp. 131–160.

[6] Y. Chevalier and L. Vigneron, "Towards Efficient Automated Verification of Security Protocols," in *VERIFY'01*, 2001.

[7] C. Caleiro, L. Viganò, and D. Basin, "On the semantics of Alice&Bob specifications of security protocols," *TCS*, vol. 367, no. 1, pp. 88 – 122, 2006.

[8] U. Carlsen, "Generating formal cryptographic protocol specifications," *IEEE Symposium on Research in Security and Privacy*, pp. 137–146, 1994.

[9] J. Millen and F. Muller, "Cryptographic protocol generation from CAPSL," SRI International, Tech. Rep. SRI-CSL-01-07, 2001.

[10] Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani, "Deciding the Security of Protocols with Diffie-Hellman Exponentiation and Products in Exponents," in *FST TCS'03*, ser. LNCS 2914, 2003, pp. 124–135.

[11] J. K. Millen and V. Shmatikov, "Symbolic protocol analysis with products and Diffie-Hellman exponentiation," in *CSFW'03*, 2003, pp. 47–61.

[12] D. Basin, S. Mödersheim, and L. Viganò, "Algebraic intruder deductions," in *LPAR 2005*, ser. LNAI 3835, 2005, pp. 549–564.

[13] S. Mödersheim, "Algebraic Properties in Alice and Bob Notation (extended version)," IBM Zurich Research Lab, Tech. Rep. RZ3709, 2008, domino.research.ibm.com/library/cyberdig.nsf.

[14] AVISPA, "Deliverable 2.3: The Intermediate Format," 2003, available at www.avispa-project.org.

[15] A. Armando and et al., "The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications," in *CAV'05*, ser. LNCS 3576, 2005.

[16] S. Mauw, M. Reniers, and T. Willemse, "Message sequence charts in the software engineering process," in *Handbook of Software Eng. and Knowledge Eng.*, 2001, pp. 437–463.

[17] S. Mödersheim, "Models and Methods for the Automated Analysis of Security Protocols," PhD-Thesis, ETH Zürich, 2007.

[18] D. Dolev and A. Yao, "On the Security of Public-Key Protocols," *IEEE Transactions on Inform. Theory*, vol. 2, no. 29, 1983.

[19] M. Abadi and C. Fournet, "Mobile values, new names, and secure communication," in *ACM POPL*, 2001, pp. 104–115.

[20] AVISPA, "The AVISPA library of security protocols," www.avispa-project.org/library/.

[21] S. Mödersheim and L. Viganò, "Secure Pseudonymous Channels," IBM Zurich Research Lab, Tech. Rep. RZ3724, 2008, domino.research.ibm.com/library/cyberdig.nsf.