RZ 3709 (# 99719) 06/20/2008 (Revised Version: October 2008)
Computer Science 18 pages

Research Report

Algebraic Properties in Alice and Bob Notation (Extended Version, revised October 2008)

Sebastian Mödersheim

IBM Research GmbH
Zurich Research Laboratory
8803 Rüschlikon
Switzerland

E-mail: smo@zurich.ibm.com

Algebraic Properties in Alice and Bob Notation (Extended Version) *

Sebastian Mödersheim
IBM Zurich Research Laboratory, Switzerland
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.

1. 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. [17]. 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 [16] which translates an Alice and Bob specification into the process calculus CSP [13] for model-checking with FDR. Similarly, the CAPSL Integrated Protocol Environment (CIPE) [11] 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 [16, 11] 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 algebraic reasoning is needed for constructing a message.

The parsing problem was first addressed in [14, 9], 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 § 5, we discuss the partial support for algebraic properties in [14, 9].

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 [6]. 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. Last but not least, also generating real implementations as the output of the translation has been considered [7, 19, 22].

Contributions While many approaches to the formal analysis of protocols have considered algebraic properties (e.g. [8, 20, 5]), 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.

*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.

`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` :

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

`Goals` :
  B `authenticates` A `on` Msg
  Msg `secret of` A, B

Figure 1. Example protocol in AnB.

We define the formal protocol specification language *AnB* (for *Alice and Bob*) and thereby generalize previous approaches, giving the first formal semantics for an Alice and Bob style language that is 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, according to our semantics, an honest agent can perform an infinite number of checks on a received message. 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 [4] as a way to describe transition systems. IF is the input language to a variety of tools [2], and we have implemented a prototype of the translator.

Organization The rest of the paper is organized as follows. In § 2 we introduce the syntax and intuition of our AnB language as well as our running example. In § 3, 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 § 4, we go into the details of modeling messages and their construction and deconstruction by honest agents in the presence of algebraic properties. In § 5 we conclude with an outlook on future work.

2. 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 Appendix A. 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 and the behavior of agents 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 $A \to B : M$, meaning that role A sends the message M to role B. The receiver of a message must be the sender of the next one (except for the last message). The core of the semantics/translation is to generate a "program" for each of the roles 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.

3. 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 [4].

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.

3.1. 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.

3.2. 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 Appendix B.

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 concrete agent names. Let M_j denote the initial knowledge of role R_j, then the initial state is:

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

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 gives the intruder the necessary knowledge to execute the protocol in role B under his real name i.

3.3. IF Transition Rules

We consider IF transition rules of the following form (for details on the semantics see [4]):

$$L \mid EQ =\!\![V]\!\!\Rightarrow R$$

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

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

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.

3.4. 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:

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

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 in every transition the local state M of the honest agents (which initially consists only of the initial knowledge) is extended by the incoming message and the freshly created variables. As often done, transitions include both receiving a message and replying [14] and also we identify the intruder knowledge and the insecure communication medium [21]. The mth transition rule for an honest agent in role $\mathcal{R}$ is then 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 [16, 11], 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 of figure 1 is the following IF 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. [14] 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 [6]. 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.

3.5. Goals.

We follow a standard definition of secrecy and authentication goals by means of attack states, found in Appendix C.

After this overview of the semantics and the problems of previous approaches, we now look at the details of interpreting messages and computing how honest agents compose and decompose/check messages.

4. Message Model

We now 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 § 3.4.

4.1. 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:*

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

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

The deduction closure $\mathcal{DY}$ is defined in the spirit of the standard Dolev-Yao intruder model [12], 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 the honest agents is a subset of the behavior of the intruder: he cannot perform operations that honest agents cannot, only he is not bound by the protocol.

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 AVISPA IF is based. We use the following signature* Σ *of predefined operators with their intuitive meanings:* 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 (non built-in) 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 to IF).* $\Sigma_p = \Sigma \setminus \{\mathsf{inv}(\cdot)\}$*, i.e. all functions are public except* $\mathsf{inv}(\cdot)$*.*

The congruence $\approx$ *that we consider with this example signature is defined by the following properties:*

$$\{\!|\{\!|m|\!\}_k|\!\}_k \approx m \tag{2}$$

$$\{\{m\}_k\}_{\mathsf{inv}(k)} \approx m \tag{3}$$

$$\mathsf{inv}(\mathsf{inv}(k)) \approx k \tag{4}$$

$$\pi_i(\langle m_1, m_2 \rangle) \approx m_i \tag{5}$$

$$\langle \pi_1(m), \pi_2(m) \rangle \approx m \tag{6}$$

$$\mathsf{exp}(\mathsf{exp}(B, X), Y) \approx \mathsf{exp}(\mathsf{exp}(B, Y), X) \tag{7}$$

$$A \oplus B \approx B \oplus A \tag{8}$$

$$A \oplus (B \oplus C) \approx (A \oplus B) \oplus C \tag{9}$$

$$A \oplus A \approx e \tag{10}$$

$$A \oplus e \approx A \tag{11}$$

Properties (2) and (3) express that encryption and decryption cancel each other out. In this model encryption and decryption are the same operation (only with inverse keys in case of asymmetric encryption) while in general one may model them as different operations. Property (4) expresses that two $\mathsf{inv}(\cdot)$ *cancel each other out (together with (3 this implies* $\{\{m\}_{\mathsf{inv}(k)}\}_k \approx m$*). Properties (5) and (6) express the relationship between concatenation and the projections. Property (7) is the one needed for Diffie-Hellman based protocols. Finally, properties (8)–(11) characterize the xor operation. There are other properties one could formalize (e.g. the relationships between exponentiation and multiplication). However, for the AnB semantics we are concerned with properties that are essential for the execution of the protocol by honest agents; in this regard, the example theory is sufficient for 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):*

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

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. Consider for instance the message labeled $\mathcal{X}_1$: according to the protocol it is supposed to be an encrypted message, but it is not enforced that a message one receives from the network has indeed the correct format. As the label of the derived message Msg in the example 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.

4.2. Checking Messages

We now turn to the question 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* $\mathcal{I}$ *is a* model *of 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 function* $h \in \Sigma$. *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. 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 for a given finite M.*

The proof is found in Appendix D.

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 § 3.4. 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 in a straightforward way how messages are parsed in an arbitrary algebraic theory without using complicated, specialized procedures to obtain message patterns as in [14, 9, 6].

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 the knowledge*

$$\begin{aligned} M &= \{(a \oplus b \oplus c \oplus d)^{\mathcal{X}_1}, (a \oplus b)^{\mathcal{X}_2}, \\ &\quad (a \oplus c)^{\mathcal{X}_3}, h(b \oplus d)^{\mathcal{X}_4}, h(c \oplus d)^{\mathcal{X}_5}\} \end{aligned}$$

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 following 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:

$$\mathit{verify}(\{\!|m|\!\}_k, k) \approx \mathit{true}$$

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 $(\mathit{verify}(\mathcal{X}_1, \mathcal{X}_2), \mathit{true})$. 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.

4.3. 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 list $m_1, \ldots, m_{k-1}$ of messages. 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 $M = \{m_i^{\mathcal{X}_i} \mid 1 \leq i \leq k+l\}$. 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 un-executable *and its semantics is undefined. The transition rule is then (where $ccs(M)$ is considered as a set of equations):*

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

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:*

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

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 $\mathsf{exp}(\{\pi_2(Z)\}_{pk(B)}, X)$.

5. Conclusions

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

[14] was 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 [9] 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 similarities between this problem of checks and the notion of static equivalence [1]; the results on static equivalence could help to identify more generally classes of algebraic properties 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 [3]. Recently, we have also extended AnB with a notation of secure and pseudonymous channels, as well as support for non-interactive zero-knowledge proofs, and we will report on this soon.

References

[1] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *ACM symposium on principles of programming languages*, pages 104–115, 2001.

[2] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. Hankes Drielsma, P.-C. Héam, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, and L. Vigneron. The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications. In *CAV'05*, 2005.

[3] AVISPA. The AVISPA library of security protocols. `avispa-project.org/library/`.

[4] AVISPA. Deliverable 2.3: The Intermediate Format. Available at `www.avispa-project.org`, 2003.

[5] D. Basin, S. Mödersheim, and L. Viganò. Algebraic intruder deductions. In *LPAR 2005*, volume 3835 of *LNAI*, pages 549–564, 2005.

[6] C. Caleiro, L. Viganò, and D. Basin. On the semantics of Alice&Bob specifications of security protocols. *Theoretical Computer Science*, 367(1):88 – 122, 2006.

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

[8] 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*, LNCS 2914, pages 124–135, 2003.

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

[10] H. Comon-Lundh and V. Cortier. Security properties: two agents are sufficient. In *ESOP'2003*, LNCS 2618, pages 99–113, 2003.

[11] G. Denker, J. Millen, and H. Rueß. The CAPSL Integrated Protocol Environment. Technical Report SRI-CSL-2000-02, SRI International, Menlo Park, CA, 2000.

[12] D. Dolev and A. Yao. On the Security of Public-Key Protocols. *IEEE Transactions on Inform. Theory*, 2(29), 1983.

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

[14] F. Jacquemard, M. Rusinowitch, and L. Vigneron. Compiling and Verifying Security Protocols. In *LPAR 2000*, LNCS 1955, pages 131–160, 2000.

[15] G. Lowe. A hierarchy of authentication specifications. In *CSFW'97*, pages 31–43. IEEE Computer Society Press, 1997.

[16] G. Lowe. Casper: a Compiler for the Analysis of Security Protocols. *Journal of Computer Security*, 6(1):53–84, 1998.

[17] LSV. Security Protocols Open Repository (SPORE). `www.lsv.ens-cachan.fr/spore/`.

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

[19] J. Millen and F. Muller. Cryptographic protocol generation from CAPSL. Technical Report SRI-CSL-01-07, SRI International, 2001.

[20] J. K. Millen and V. Shmatikov. Symbolic protocol analysis with products and Diffie-Hellman exponentiation. In *CSFW'03*, pages 47–61, 2003.

[21] S. Mödersheim. *Models and Methods for the Automated Analysis of Security Protocols*. PhD-thesis, ETH Zürich, 2007.

[22] B. Tobler and A. Hutchison. Generation, Analysis and Verification of Cryptographic Protocol Implementations. In *Information Security South Africa Conference*, pages 297–306, Sandton, South Africa, 2003.

A AnB Syntax

In EBNF, the syntax of AnB can be described as follows:

$$
\begin{array}{rcl}
\mathit{Protocol} & ::= & \texttt{Protocol} : \mathit{Identifier} \\
 & & \mathit{Types}\ \mathit{Knowledge}\ \mathit{Actions}\ \mathit{Goals} \\
\mathit{Types} & ::= & \texttt{Types} : (\mathit{Type}\ \mathit{Identifier}\ (,\mathit{Identifier})^{*};)^{+} \\
\mathit{Type} & ::= & \texttt{Agent} \mid \texttt{Number} \mid \texttt{Function} \mid \texttt{PublicKey} \mid \texttt{SymmmetricKey} \\
\mathit{Knowledge} & ::= & \texttt{Knowledge} : (\mathit{Agent} : \mathit{Msg};)^{+} \\
\mathit{Actions} & ::= & \texttt{Actions} : (\mathit{Agent}\ \texttt{->}\ \mathit{Agent}\text{: } \mathit{Msg})^{+} \\
\mathit{Goals} & ::= & \mathit{Agent}\ \texttt{authenticates}\ \mathit{Agent}\ \texttt{on}\ \mathit{Msg} \\
 & \mid & \mathit{Msg}\ \texttt{secret of}\ \mathit{Agent}(,\mathit{Agent})^{*} \\
\mathit{Agent} & ::= & \mathit{Identifier} \\
\mathit{Identifier} & ::= & \mathsf{Alpha}\ (\mathsf{Alphanum} \mid {}'-' \mid {}'\_')^{*} \\
\mathit{Msg} & ::= & \mathit{Identifier} \\
 & \mid & \{\ \mathit{Msg}\ \}\ \mathit{Msg} \\
 & \mid & \{|\ \mathit{Msg}\ |\}\ \mathit{Msg} \\
 & \mid & \mathit{Identifier}'('\mathit{Identifier}\ (,\mathit{Identifier})^{*}')' \\
 & \mid & \mathit{Msg}, \mathit{Msg} \\
 & \mid & '('\mathit{Msg}')'
\end{array}
$$

We have used here $'('$ and $')'$ to distinguish parenthesis of the language from the ones of the grammar.

A specification must satisfy the following further conditions:

- All identifiers that appear in a description except for the protocol name and built-in identifiers (see example 2) are uniquely declared.
- The non-terminal symbol $Agent$ can only be an identifier that was declared to be of type agent.
- Identifiers starting with an upper-case letter are called variables, the ones starting with lower-case letter are constants.
- There is exactly one knowledge declaration for each identifier of type agent in the specification.
- All variables in the knowledge specification are of type agent.
- In a sequence of actions, the receiver of a message is the sender of the next message.
- In a message term of the form $\mathit{Identifier}'(' \ldots ')'$, the first identifier must be of type function (i.e. declared or built-in). (Note that an identifier of type function without arguments is also a message.) For built-in functions, the number of arguments must agree with its arity; for other functions, the translator gives a warning when using the function several times with a different number of arguments.

Note that pairing of messages such as $m_1, m_2, \ldots, m_k$ is *not* associative, rather it is a short-hand for $m_1, (m_2, (\ldots, m_k))$. The expression $\{m\}m_1, m_2$ is a short-hand for $(\{m\}m_1), m_2$ (and similarly for symmetric encryption).

B IF Initialization Theory

In § 3.2, we describe the initial state for a given finite number of instances of all roles. The general case of an unbounded number of sessions cannot be defined this way, since the initial state would then contain infinitely many facts (which is not allowed by the AVISPA IF). We therefore use *initialization rules* that allow for opening new sessions at any time. First we introduce a new fact symbol agent for introducing an unbounded number of agents (although there are results that we can restrict ourselves usually to two agents in order to find all attacks [10]):

$$=\!\![A]\!\!\Rightarrow \mathsf{agent}(A)\ .$$

The initial state contains the fact $\mathsf{agent}(\mathsf{i})$.

Let now $A_1, \ldots, A_k$ be the variables of type agent, i.e. the roles, in a protocol specification, and let M_i be their initial knowledge in the specification (recall that it may contain only variables of type agent, i.e. the A_i). Then we have for every $i \in \{1, \ldots, k\}$ the following two rules:

$$\begin{array}{rcl} \mathsf{agent}(A_1).\ \ldots\ .\mathsf{agent}(A_k)\ \&\ A_i \neq \mathsf{i} & \Rightarrow & \mathsf{state}_{\mathcal{A}_i}(M_i) \\ \mathsf{agent}(A_1).\ \ldots\ .\mathsf{agent}(A_k)\ \&\ A_i = \mathsf{i} & \Rightarrow & \mathsf{iknows}(M_i) \end{array}$$

This takes any k-tuple of agent names created by the above rule and instantiates an initial local state of an honest agent or gives the intruder the respective initial knowledge. Recall that we silently assume a session number in every state fact for uniqueness that is not displayed here.

C IF Attack States

We describe here how goals are expressed by defining which states violate the goals. To that end, we need auxiliary fact symbols that express goal relevant information in a declarative way so that the goal checks can be specified in a protocol independent way. The way we describe goals here is close to the definitions in other approaches such as [15, 2, 4].

C.1 Secrecy

Consider the goal M `secret of` $A_1, \ldots, A_k$. For each A_i we check that A_i is able to generate M in the last transition that A_i is involved in. This is a simple check as if M was a plaintext-part of the last out-going message of A_i. If it cannot be generated, the AnB description is refused because of a meaningless goal. Otherwise let d be the label for generating M. The respective transition of A_i is augmented with the following right-hand side facts, where S is an additional freshly created variable of the transition:

$$\mathsf{secret}(d, S).\mathsf{contains}(S, d_{A_1}).\ \ldots\ .\mathsf{contains}(S, d_{A_k})$$

where d_{A_i} are labels for agent A_i's view of the agent names. This declares the secrecy of the term derived as d between the set S of agents.

The goal is that the intruder knows a secret between a set of people that he is not a member of, as expressed by the following error-rule:

$$\mathsf{secret}(M, S).\mathsf{iknows}(M)\ \&\ \mathsf{not}(\mathsf{contains}(S, \mathsf{i})) \Rightarrow \mathsf{error}$$

This can be considered as a standard transition rule with the resulting symbol error, and we define attack states to be those that contain the symbol error. Note that here we also use the IF feature of negative predicates, i.e. the transition has the additional condition that no matching fact is contained in the state [4].

C.2 Weak Authentication

For the goal B `weakly authenticates` A `on` M (called *non-injective agreement* in [15]), we generate facts that in the transition rules of A and B as follows.

In the last transition of B, B must be able to generate M (otherwise the specification is meaningless); let d be the label for generating M. On the right-hand side of this transition we add the following fact:

$$\mathsf{wrequest}(d_B, d_A, \mathcal{M}, d)$$

where $\mathcal{M}$ is a special constant identifying the protocol's description of the message to be transmitted authentically.[1] Roughly speaking, with this fact, B declares that he accepts the message derived as d to come from A and being meant as M in the protocol.

[1]This is to prevent that in the case of several goals, a potential confusion between some values is not detected. The labels d_B and d_A are B's view on the names of A and B.

We go back in the protocol description from B's last transition to the last transition of A before that and check that A can generate M at that point, say with label d (again, the protocol description is meaningless if M cannot be generated):

$$\mathsf{witness}(d_A, d_B, \mathcal{M}, d)$$

This statement reflects the counter-part of the $\mathsf{wrequest}(\cdot)$ statement of B, namely that A believes to share with B the message derived as d to be the message M in the protocol description.

The attack rule is:

$$\mathsf{wrequest}(B, A, M, D).\mathsf{not}(\mathsf{witness}(A, B, M, D))\ \&\ A \not\approx \mathsf{i} \Rightarrow \mathsf{error}$$

This expresses that it is an attack if B has accepted something in a different way then ever meant by A. We use here the feature of IF to specify inequalities in transitions, which have the expected meaning [4].

C.3 Authentication

The stronger variant B `authenticates` A on M (called *injective agreement* in [15]) is very similar. It additionally takes replay into account, i.e. B should accept a message M more than once. This is based on the assumption that M contains a constant that is supposed to be generated freshly in the protocol. Instead of the $\mathsf{wrequest}(\cdot)$ we now have a $\mathsf{request}(\cdot)$ fact with an additional session identifier. Recall that we have required in § 3.1 that every state-fact has a unique identifier (to distinguish several instances of an agent in the same local state). Thus in B's last transition we have the additional fact:

$$\mathsf{request}(d_B, d_A, \mathcal{M}, d, S)$$

where S is the respective identifier of agent B.

The $\mathsf{witness}(\cdot)$ fact in A's last transition before B's last transition is the same as in the weak authentication case.

In this case, there are two attack rules. The first rule covers the weak authentication attacks as before:

$$\mathsf{request}(B, A, M, D).\mathsf{not}(\mathsf{witness}(A, B, M, D))\ \&\ A \not\approx \mathsf{i} \Rightarrow \mathsf{error}$$

The second rule is for detecting replay:

$$\mathsf{request}(B, A, M, D, S).\mathsf{request}(B, A, M, D, S')\ \&\ S \not\approx S'\ \&\ A \not\approx \mathsf{i} \Rightarrow \mathsf{error}$$

This rule can fire when two different request terms ($S \not\approx S'$) are present in a state.

D Derivability and Checks

In § 4 we have defined labeled intruder deduction and consistency checks on which the role generation is based. The underlying problems of algebraic reasoning are in general undecidable, e.g whether two terms are equal under a given congruence $\approx$, see for instance [21]. In this section, we give the proof of Theorem 1, namely that the relevant problems are decidable/computable for our example theory described in example 2.

Definition 4. *We split the example theory of example 2 into two sets F and C of equations, where F contains the property (7) of exponentiation and the properties (8) and (9) that $\oplus$ is commutative and associative; C contains all other properties.*

In the following, F and C generally refer to our example theory, as well as $\approx$ denotes equivalence in the algebraic theory induced by $F \cup C$, and we use $\approx_F$ for the restricted equivalence in F alone.

We denote with $\mathcal{DY}_F$ a restriction of the labeled Dolev-Yao style deduction relation: instead of the $\approx$ relation we use the restricted congruence $\approx_F$. Similarly, we denote with $ccs_F(M)$ the restriction of $ccs(M)$ to theory $\approx$, i.e. the set of checks that can be obtained for $\mathcal{DY}_F(M)$ and $\approx_F$ instead of $\mathcal{DY}(M)$ and $\approx$.

Note that F is a *finite theory*, i.e. the equivalence class $[t]_F$ of any term t under F is finite, and C is a *cancellation theory*, i.e. each equation has the form $l \approx r$ where r is a constant or a subterm of l. We regard these equations as rewrite rules $l \to r$, and consider the relation $\to_{C/F}$ on F-equivalence classes (i.e. $[s]_F \to_{C/F} [t]_F$ iff exist $s' \in [s]_F$ and $t' \in [t]_F$ with $s' \to_C t'$). For our theory, this relation is convergent, i.e. for every $[t]_F$ we reach a unique normal form, denoted $[t]_F \downarrow$, after finitely many steps. For simplicity, we will silently just write a term rather than its equivalence class. The word problem $s \approx t$ is decidable in our theory by normalizing (equivalence classes of) terms modulo $\to_{C/F}$. For a more detailed exposition, see e.g. [21].

D.1 Labeled $\mathcal{DY}_F$ Deduction

Within finite theories, intruder deduction is no problem, as the finite equivalence class of every term is easily computed:

Lemma 1. *Given a term t and a finite set of labeled terms M, then it is decidable whether there is a derivation d such that $t^d \in \mathcal{DY}_F(M)$. Moreover, the set of such derivations is finite and computable.*

Proof. For every $t' \in [t]_F$, we first take the set of labels d such that ${t'}^d \in M$. Moreover, if $t' = f(t_1, \dots, t_n)$ we recursively compute the set $D_i = \{d_i \mid t_i^{d_i} \in \mathcal{DY}_F(M)\}$ for every $i \in \{1, \dots, n\}$ and take additionally the labels $\{f(d_1, \dots, d_n) \mid d_1 \in D_1, \dots, d_n \in D_n\}$. This computation cannot run into a non-terminating recursion as otherwise there were an infinite chain $t_1 \approx_F s_1 \sqsupset t_2 \approx_F s_2 \sqsupset t_3 \dots$ of terms; we could thus obtain the infinite F-equivalence class

$$t_1 \approx s_1 \approx s_1[t_1 \mapsto s_2] \approx s_1[t_1 \mapsto (s_2[t_2 \mapsto s_3])] \approx \dots ,$$

contradicting the finiteness of F. □

While for a particular term t, the set of labels d with $t^d \in \mathcal{DY}_F(M)$ is finite for a finite M. However, the set $\mathcal{DY}_F(M)$ is infinite and thus the set of terms that is potentially relevant for a finding checks. We show that for the finite theory F we can find a finite set of checks.

Definition 5. *In the following, we use the notion of a* derivation tree *for labeled messages, i.e. a proof $t^d \in \mathcal{DY}(M)$. where the root is t^d, intermediate nodes are labeled terms within the deduction and leaves are labeled terms of M. For simplicity, we do also collapse nodes that are equivalent modulo F, i.e. transforming*

$$\cfrac{\cfrac{\dots}{t^d}}{s^l} \quad \textit{into} \quad \cfrac{\dots}{s^l}$$

if $s \approx_F t$ and $l \approx_F d$. Also, by slight abuse of notation, we do not distinguish syntactically different forms of F-equivalent terms, e.g. if $s \approx_F t$, we may simply call both t, using t as a representative for its F-equivalence class.

Further, when we later consider also equivalences in $\approx$ in general (i.e. based on the example theory $F \cup C$), we consider a special kind of node, called analysis node*, denoted as follows:*

$$\cfrac{\cfrac{\dots}{t^\star}}{s^\star} \downarrow$$

where t is a redex of $\rightarrow_{C/F}$, and s is the normal form, and that all terms that are not the child node of an analysis node like $t^\star$ here, are in normal form. Also we assume that all labels throughout the tree are normalized (even those of an analysis node). The notation $t^\star$ is used to denote a labeled term where the derivation is irrelevant.

It is without restriction to transform an arbitrary derivation tree into one where all nodes except children of analysis nodes are normalized: all nodes besides $\approx$ and leaf nodes are composition nodes for which is does not matter whether they are executed on normalized or unnormalized terms. So we can transform every derivation tree into one that has only analysis nodes for equational reasoning (hiding the equalities in $\approx_F$).

Lemma 2. *For given finite M, we can compute a finite set of checks C sufficient for $ccs_F(M)$.*

Proof. Let $d(t)$ denote the depth of the term t, $\mathsf{maxd}(M)$ denote the depth of the largest term $t' \in [t]_F$ for any $t \in M$. For a set of terms M, let $|M|$ denote its cardinality.

Let $M' = \{t^d | t^d \in \mathcal{DY}_F(M) \wedge d(t) \leq 2 \cdot |M| \cdot \mathsf{maxd}(M) + 1\}$. M' is finite and computable, since there are only finitely many terms t with a bounded depth contained such that $t^\star \in \mathcal{DY}_F(M)$, and for a given term t there are finitely many labels such that $t^d \in \mathcal{DY}_F(M)$, according to lemma 1.

We take as equations the set $C = \{(d_1, d_2) \mid t^{d_1}, t^{d_2} \in M', d_1 \not\approx d_2\}$. We show that this set is sufficient for $ccs_F(M)$ as follows.

Consider now the derivation trees for two derivations $t^{d_1}, t^{d_2} \in \mathcal{DY}_F(M)$ with $d_1 \not\approx d_2$. If any of these labeled terms is already contained in M, i.e. if one derivation tree is a single leaf, then the check $d_1 \approx d_2$ is already included in C. So let us consider that both trees are not leaves. By the tree form of definition 5, and since we consider only equalities in F, the trees can only contain composition nodes. This composition step must be with the same operator, as equivalence in F does not allow the root-symbol of t to change (i.e. all equations of F have the same root-symbol on both sides). We do a case split according to this root-symbol:

- All operators that do not occur in the equations F (such as $\{\!|\cdot|\!\}_{\cdot}$): the corresponding subterms must be equivalent in F to give the same resulting term t. Thus, we can reduce the problem to the equivalence checks of these proper subterms of t.
- $\oplus$: Since the derivation contains only normalized terms, the worst case is an $\oplus$-composition of all terms of M but one and comparing it to the remaining one. The depth of this composed term is still smaller than $|M| \cdot \mathsf{maxd}(M)$.
- exp: Summarizing a set of exp steps, we have the two derivations (omitting the derivation labels):

$$\frac{\overline{a} \quad \frac{\cdots}{x_1} \quad \ldots \quad \frac{\cdots}{x_n}}{\mathsf{exp}(\ldots(\mathsf{exp}(a, x_1), \ldots), x_n)}\ \mathsf{exp}^n \text{ and } \frac{\overline{a} \quad \frac{\cdots}{y_1} \quad \ldots \quad \frac{\cdots}{y_m}}{\mathsf{exp}(\ldots(\mathsf{exp}(b, y_1)\ldots), y_m)}\ \mathsf{exp}^n\ .$$

  Here, we do not assume anything about the derivation of the exponents, only we have eventually a leaf at the basis of exponentiation. This summarizes all compositions of exponentiations that can occur.

  In order for both root terms of the derivation trees to be equal, we need the property that $a = \mathsf{exp}(\ldots(\mathsf{exp}(a_0, a_1), \ldots), a_k)$ and $b = \mathsf{exp}(\ldots(\mathsf{exp}(b_0, b_1), \ldots), b_l)$ and that the *multi*-sets $\{a_1, \ldots, a_k, x_1, \ldots, x_n\}$ and $\{b_1, \ldots, b_l, y_1, \ldots, y_m\}$ are equal modulo F.

  Now suppose that there are i and j such that $x_i \approx y_j$. Then we have an exponent that is added in the construction of both derivation trees. The idea is that we can thus reduce the problem to smaller terms to be checked, namely the two derivations of t without the exponent $x_i \approx y_j$—leaving out x_i and y_j in both trees we get the same resulting term at the root—and the check of the derivations of x_i and y_j.

  Using this reduction to simpler comparisons inductively, we have two compositions with exp as above where the x_i contain exactly those exponents of the b_i that are missing in the a_i and vice-versa the y_i contain exactly those exponents of the a_i that are missing in the b_i. So we compose a term t with $d(t) \leq d(a) + d(b)$, thus we have still a term of depth smaller than $2 \cdot \mathsf{maxd}(M)$.

Thus, the checks C derived using M' are sufficient for $ccs_F(M)$. $\square$

D.2 Labeled $\mathcal{DY}$-Deduction

When we now bring in the cancellation rules C of our example theory. In general, the set of derivations for every term is now infinite (even when considering only normalized derivations). The idea is to use the notion of an *analyzed* knowledge from [5, 21]:

Definition 6. *M is called* analyzed *(with respect to $\rightarrow_{C/F}$) iff for every term $t^d \in \mathcal{DY}(M)$, the normal-form of t can be generated by composition alone, i.e. there is a derivation d' such that $(t\downarrow)^{d'} \in \mathcal{DY}_F(M)$.*

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 for a given finite M.*

Proof. We first define an analysis closure *ANA*$(\cdot)$ on unlabeled terms, similar to $\mathcal{DY}(\cdot)$, but which is finite (as we need this for the rest of the construction).

We slightly abuse notation using $\mathcal{DY}_F$ for unlabeled terms also, and introducing in single rules two consequences $a, b \in$

ANA(M) rather writing separate rules with consequences $a \in \mathit{ANA}(M)$ and $b \in \mathit{ANA}(M)$:

$$\frac{}{m \in \mathit{ANA}(M)}\; m \in M$$

$$\frac{\{\!|m|\!\}_k \in \mathit{ANA}(M)}{m, k \in \mathit{ANA}(M)}\; k \in \mathcal{DY}_F(M)$$

$$\frac{\{m\}_k \in \mathit{ANA}(M)}{m, \mathsf{inv}(k) \in \mathit{ANA}(M)}\; \mathsf{inv}(k) \in \mathcal{DY}_F(M)$$

$$\frac{\{m\}_{\mathsf{inv}(k)} \in \mathit{ANA}(M)}{m, k \in \mathit{ANA}(M)}\; k \in \mathcal{DY}_F(M)$$

$$\frac{\langle m_1, m_2\rangle \in \mathit{ANA}(M)}{m_1, m_2 \in \mathit{ANA}(M)}$$

$$\frac{t \in \mathit{ANA}(M)}{m_1, m_2 \in \mathit{ANA}(M)}\; t \approx_F (m_1 \oplus m_2), m_1 \in \mathcal{DY}_F(M)$$

$$\frac{t \in \mathit{ANA}(M)}{(m_1 \oplus m_3), (m_2 \oplus m_3) \in \mathit{ANA}(M)}\; t \approx_F (m_1 \oplus m_2), (m_1 \oplus m_3) \in \mathcal{DY}_F(M)$$

For a finite set of messages M, *ANA*(M) is finite and computable, since there are finitely many $t \in \mathcal{DY}_F(M)$, and all rules but the last two add only subterms of M to the set. The last two rules are bounded by the powerset of all xored subterms of M and their F-equivalence classes.

We show below that *ANA*(M) gives us an analyzed set, albeit ignoring the derivation labels, and thus allows us to decide $t^\star \in \mathcal{DY}(M)$.

We first define a set M' of labeled terms and a set equations EQ as follows. Let M_0 be the set M with all labels stripped off and let $M_1 = \mathit{ANA}(M_0)$ be the least closure under the *ANA*$(\cdot)$ rules. Let M' be the set M_1 with each term normalized and labeled with a fresh variable $\mathcal{X}_i$ that did not occur in M. Let EQ be the set of equations that links previous labels of M with those of M':

$$EQ = \{(d, d') \mid t^d \in M, t^{d'} \in M'\} \; .$$

Let EQ_0 be a finite set of equations that is sufficient for $ccs_F(M')$ (which can be computed according to lemma 2).

We now derive a superset $EQ' \supseteq EQ \cup EQ_0$ of equations as follows, where $\in_F$ represents containment modulo $\approx_F$:

$$\begin{array}{rcl} EQ' & = & EQ \cup EQ_0 \\ & & \{(\{\!|d_1|\!\}_{d_2}, d_3) \mid \{\!|m|\!\}_k^{d_1}, m^{d_3} \in_F M' \wedge k^{d_2} \in \mathcal{DY}_F(M)\} \\ & & \{(\{d_1\}_{d_2}, d_3) \mid \{m\}_k^{d_1}, m^{d_3} \in_F M' \wedge \mathsf{inv}(k)^{d_2} \in \mathcal{DY}_F(M)\} \\ & & \{(\{d_1\}_{d_2}, d_3) \mid \{m\}_{\mathsf{inv}(k)}^{d_1}, m^{d_3} \in_F M' \wedge k^{d_2} \in \mathcal{DY}_F(M)\} \\ & & \{(d_1, \langle d_2, d_3\rangle \mid \langle m_1, m_2\rangle^{d_1}, m_1^{d_2}, m_2^{d_3} \in_F M', \} \\ & & \{(d_1 \oplus d_2, d_3) \mid (a \oplus b)^{d_1}, b^{d_3} \in_F M', a^{d_2} \in \mathcal{DY}_F(M)\} \\ & & \{(d_1 \oplus d_2, d_3) \mid (a \oplus b)^{d_1}, (a \oplus c)^{d_3} \in_F M', (b \oplus c)^{d_2} \in \mathcal{DY}_F(M)\} \end{array}$$

EQ' is finite and computable according to the properties $\mathcal{DY}_F(M)$ and M'.

We now show that for every labeled term $t^d \in \mathcal{DY}(M)$ where t and all terms of M are normalized, we can find a derivation $t^{d'} \in \mathcal{DY}_F(M')$ such that in every model $\mathcal{I}$ of EQ', $\mathcal{I}(d) \approx \mathcal{I}(d')$. We will also simply say EQ' implies $d \approx d'$ in this case.

Let thus $t^d \in \mathcal{DY}(M)$ be any derivation, and consider a derivation tree according to definition 5. If the tree contains no analysis nodes, then we already have $t^d \in \mathcal{DY}_F(M)$. Otherwise, consider an analysis node in this derivation tree such that no subtrees below the analysis node contain further analysis nodes, namely

$$\cfrac{T_1 \quad \dots \quad T_n}{\cfrac{s_0^{d_0}}{t_0^{d_0}} \downarrow}$$

where the subtrees T_i contain only normalized leaves and composition nodes, and s_0 is a redex, and t_0 is a normal form.

We now show that we can replace the analysis subtree with root $t_0^{d_0}$ with a leaf node such that the whole derivation is still covered by M' and EQ', i.e. the leaf is indeed part of M' and the replacement of the labels is implied by EQ'.

Depending on the root symbol of s_0, we do a case split:

- Case $s_0 = \{|m|\}_k$. This can only be a redex if $m = \{|m'|\}_k$, i.e. we have

$$\cfrac{\cfrac{\cfrac{T_1}{\{|m'|\}_k^{d_1}} \quad \cfrac{T_2}{k^{d_2}}}{\{|\{|m'|\}_k|\}_k^{\{|d_1|\}_{d_2}}}}{m'^{\{|d_1|\}_{d_2}}} \downarrow$$

  Recall that the subtrees T_1 and T_2 contain no analysis nodes by assumption.

  We have two further subcases:

  - The case $\{|m'|\}_k^{d_1} \in M'$, i.e. T_1 is a leaf. Since tree T_2 contains no analysis node, $k^{d_2} \in \mathcal{DY}_F(M)$. Therefore by ANA, $m^{d_m} \in M'$ for some derivation d_m and by the definition of EQ', $(\{|d_1|\}_{d_2}, d_m) \in EQ'$. Since $d_0 = \{|d_1|\}_{d_2}$, we can safely replace the subtree $t_0^{d_0}$ with the leaf $(m')^{d_m}$.
  - Otherwise, when $\{|m'|\}_k^{d_1} \notin M'$, the term was composed from k^{d_3} and m'^{d_4} for some derivations d_3 and d_4. Thus we have another (possibly different) derivation of k using only composition, thus $(d_2, d_3) \in ccs_F(M')$. Since $EQ_0 \subseteq EQ'$ is sufficient for $ccs_F(M')$, the derivation $d_0 = \{|\{|d_4|\}_{d_3}|\}_{d_2}$ reduces to d_4 under any interpretation that satisfies EQ'. Thus it is safe to replace the subtree $t_0^{d_0}$ with subtree m'^{d_4}.

- The cases $s_0 = \{m\}_k$ and $s_0 = \langle m_1, m_2\rangle$ are treated similarly. The case $s_0 = \pi_i(m)$ is trivially simplified (as it cannot be a leaf for M' contains only normalized terms).

- In the case $s_0 = x \oplus y$, the derivation tree of t_0 has the form

$$\cfrac{\cfrac{\cfrac{\dots}{x^{d_1}} \quad \cfrac{\dots}{y^{d_2}}}{(x \oplus y)^{d_1 \oplus d_2}} \oplus}{z^{d_1 \oplus d_2}} \downarrow \quad .$$

  If $x \approx y$, we have $s_0 = e$ and the label $d_1 \oplus d_2$ also reduces to e, since $x^{d_1}, y^{d_2} \in \mathcal{DY}_F(M')$ and therefore $ccs_F(M')$ implies $d_1 \approx d_2$.

  Otherwise, if both x^{d_1} and y^{d_2} are leaf nodes, then x^{d_1} and y^{d_2} are already contained in M'. The fact that $x \oplus y$ is a redex while x and y itself are not implies that one of the following is the case:

  - $x \approx_F y$ — we have handled this case before.
  - $x \approx_F x_1 \oplus x_2$ and $y \approx_F y_1 \oplus y_2$ and $x_1 \approx_F y_1$ (note that due to $\approx_F$ this case includes all commutations and associations of the $\oplus$-ed subterms). Note that by the last rule of the $ANA(\cdot)$ closure, $z = (x \oplus y) \downarrow$ is also included in the closure, and thus in M' for some derivation d_0' and $d_0' \approx d_0 = (d_1 \oplus d_2) \downarrow$ is implied by EQ'. Thus we can safely replace the analysis node with a leaf.
  - $x \approx_F x_1 \oplus x_2$ and $x_1 \approx y$. Then the second last rule of the $ANA(\cdot)$ closure implies that similarly $(x \oplus y) \downarrow$ is contained in M' for a label that is equivalent to d_0 due to EQ', so we can again safely replace the analysis node with a leaf.
  - The same case as the previous one with the roles of x and y swapped; this case is handled analogously.

Finally, we need to consider the case that at least one of the two terms x and y is not a leaf, but composed, say $x = x_1 \oplus x_2$, resulting from the composition of two terms x_1 and x_2. We thus have the following situation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\cdots}{x_1^{d_1}} \quad \dfrac{\cdots}{x_2^{d_2}}}{(x_1 \oplus x_2)^{d_1 \oplus d_2}} \oplus \quad \dfrac{\cdots}{y^{d_3}}}{((x_1 \oplus x_2) \oplus y)^{d_1 \oplus d_2 \oplus d_3}} \oplus}{z^{d_1 \oplus d_2 \oplus d_3}} \downarrow$$

In this case, the term $(x \oplus y) \downarrow$ is not necessarily generated by the *ANA*$(\cdot)$ closure. The idea is now that in this case $x_1 \oplus y$ or $x_2 \oplus y$ is also a redex. (Otherwise x must be a redex which is excluded by the derivation tree form.) Say $x_1 \oplus y$ is a redex. It is therefore possible to transform the derivation tree into the following one (without changing the resulting root node):

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\cdots}{x_1^{d_1}} \quad \dfrac{\cdots}{y^{d_3}}}{(x_1 \oplus y)^{d_1 \oplus d_3}} \oplus}{(x_1 \oplus y) \downarrow^{d_1 \oplus d_3}} \downarrow \quad \dfrac{\cdots}{x_2{}^{d_2}}}{((x_1 \oplus y) \downarrow \oplus x_2)^{d_1 \oplus d_3 \oplus d_2}} \oplus}{z^{d_1 \oplus d_2 \oplus d_3}} \downarrow$$

(In case that $(x_1 \oplus y) \downarrow \oplus x_2$ is not a redex, the lower analysis node to z is not present.)

Observe that we have now a similar situation as the one we started with, namely an analysis node for $x_1 \oplus y$ (similar to the considered $x \oplus y$), and also an analysis node for $(x_1 \oplus y \downarrow \oplus x_2$. Thus, we have not replaced the subtree for $x \oplus y$ with a leaf as previously, but with a tree that even contains more analysis nodes. However, we now show that we do not get into an infinite loop if we now apply the same transformations to the new nodes. To see that define a measure on an analysis node for $x \oplus y$, namely $size(x) + size(y)$ where $size(t)$ is simply the cardinality of the set of positions of the term t. This is measure is positive for every term. Now our transformation for $x \oplus y$ nodes produces in the worst case two new such analysis nodes, but both with strictly smaller measure, namely in one case $x_1 \oplus y$ where x_1 is a proper subterm of x, so its size is strictly smaller; and in the other case it is $(x_1 \oplus y) \downarrow \oplus x_2$ where $x_1 \oplus y$ is a redex and therefore its normalform is strictly smaller in size. By this argument, the production of new analysis nodes eventually stops.

So we can finally guarantee a leaf node for $(x \oplus y) \downarrow$ with a derivation that is safe according to EQ'.

After applying these transformations to the derivation tree of the main term $t^d \in \mathcal{DY}(M)$ until no analysis nodes are contained in the derivation tree, we thus have a derivation $t^{d'} \in \mathcal{DY}_F(M')$ where EQ' implies $d \approx d'$. In particular, this means that M' is analyzed.

We can therefore decide whether for some derivation d, $t^d \in \mathcal{DY}(M)$ for given t and finite M, since by lemma 1, the deduction problem for $\mathcal{DY}_F$ is decidable. Also, if it exists, such a label d can be computed.

Further, the set of equations EQ' is sufficient for $ccs(M)$ since for any derivations $t^{d_1}, t^{d_2} \in \mathcal{DY}(M)$, there are is a derivation $(d_1', d_2') \in ccs_F(M')$ where $d_1 \approx d_1'$ and $d_2 \approx d_2'$ are implied by EQ' and EQ' is sufficient for $ccs_F(M)$ by construction. □