Towards a Quantitative Analysis of Security Protocols

Pedro Adão [1,5,6]

CLC, Department of Mathematics, IST, Lisbon, Portugal

Paulo Mateus [2,5]

CLC, Department of Mathematics, IST, Lisbon, Portugal

Tiago Reis [3,5,7]

Department of Computer Science, ETH Zurich, Switzerland
and
CLC, Department of Mathematics, IST, Lisbon, Portugal

Luca Viganò [4]

Department of Computer Science, ETH Zurich, Switzerland

Abstract

This paper contributes to further closing the gap between formal analysis and concrete implementations of security protocols by introducing a quantitative extension of the usual Dolev-Yao intruder model. This extended model provides a basis for considering protocol attacks that are possible when the intruder has a reasonable amount of computational power, in particular when he is able, with a certain probability, to guess encryption keys or other particular kind of data such as the body of a hashed message. We also show that these extensions do not augment the computational complexity of the protocol insecurity problem in the case of a finite number of interleaved protocol sessions.

Key words: Security protocols, Dolev-Yao intruder, probabilistic intruder, symbolic protocol analysis, computational protocol analysis.

Submitted for publication in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

1 Introduction

In the last years, a number of tools have been proposed for the formal analysis of security protocols, e.g. [1,8,10,15,22,24,26,28,30]). These tools have helped prove several protocols correct and have uncovered flaws in several other protocols. However, there are many protocols that have not yet been analyzed and which, in fact, are out of the scope of these tools. One of the main reasons for this is that the tools analyze protocols under the assumptions of *perfect cryptography* and that the protocol messages are exchanged over a network that is under the control of a *Dolev-Yao ($\mathcal{DY}$) intruder* [16]. That is, protocols are analyzed by considering the standard protocol-independent, asynchronous model of an active intruder who controls the network but cannot break cryptography (in particular, the intruder can intercept messages and analyze them, but only if he possesses the corresponding keys for decryption, and he can generate messages from his knowledge and send them under any agent name). Hence, the $\mathcal{DY}$ model is too weak to specify several types of security protocols, namely, those including coin tossing and intrinsic cryptographic primitives (such as zero-knowledge proof systems, oblivious transfer, secure computation, and bit commitment).

Moreover, it is well known that there are protocols that can be proved secure for the $\mathcal{DY}$ model, but are insecure when considering specific cryptosystems. For example, a "correct" implementation of the Needham-Schroeder-Lowe protocol can be attacked if El-Gamal is used as the underlying cryptosystem [31]. However, problems with the real implementation of the protocols are not addressed by the current generation of security protocol analysis tools, and even when results that relate symbolic and complexity models [2,3,4,5,6,9,11,12,14,19,20,23,29] are exploited, the equivalence is up to a negligible function and so attacks might be found when we are in the "real world".

The main goal of our work is to further close the gap between formal analysis and concrete implementations of security protocols by introducing a quantitative extension of the classical Dolev-Yao model, which we call $p\mathcal{DY}$. This allows us to consider protocol attacks that are possible when the intruder has a reasonable amount of computational power, in particular when he is able to guess encryption keys or other particular kind of data.

We formalize such an intruder model by extending the $\mathcal{DY}$ model with additional structure and intruder capabilities. As an example, while in the $\mathcal{DY}$ model encryption is considered a "black box" operation that can only

[1] Email: pad@math.ist.utl.pt

[2] Email: pmat@math.ist.utl.pt

[3] Email: tireis@inf.ethz.ch

[4] Email: vigano@inf.ethz.ch

[5] Partially supported by FEDER/FCT project QuantLog POCI/MAT/55796/2004.

[6] Additional support from FCT grant SFRH/BD/8148/2002.

[7] Additional support from FCT grant SFRH/BD/22928/2005.

be undone with the right decryption key, in the $p\mathcal{DY}$ model we introduce new intruder deduction rules, parameterized by the encryption scheme that is being used, which allow the intruder to obtain the decryption key from the encrypted message. Similarly, as another relevant example, we also introduce a deduction rule where we allow the intruder to deduce the body of a hashed message. As one may expect, these operations (should) only occur in the "real world" with small probability, hence the derivations obtained using these rules should also reflect attacks that are almost infeasible.

When using the $\mathcal{DY}$ model, it is customary to formalize interleaved executions of a protocol as an infinite-state transition system, which can then be searched for states that represent protocol attacks (e.g. a state where the intruder gets hold of some data that was intended to be a secret between two honest agents). This transition system defines a computation tree in the standard way, where the root is the initial system state and children represent the ways that a state can evolve in one transition. In the $p\mathcal{DY}$ model, we extend this search tree so that each possible transition is weighed with a probability; for instance, a state where it is possible to guess a key will have a successor that includes the knowledge of that key, but this transition will be weighed with the probability of guessing the key. Since each transition of the tree will be weighed with the probability of that transition, we can compute the probabilities associated with the branches of the tree, and in the end we will be able to tell with which probability each attack is possible.

Another contribution of our work is to show that the computational complexity of protocol insecurity in the case of a finite number of interleaved protocol sessions is not augmented by our extensions. As shown, for instance, in [25], searching for an attack in the classical $\mathcal{DY}$ intruder model is an NP-complete problem when considering finite numbers of protocol sessions, and we prove that if we consider the extended probabilistic intruder model $p\mathcal{DY}$, the problem remains NP-complete.

We proceed as follows. In Section 2, we introduce the intruder deduction problem for the classical $\mathcal{DY}$ intruder model and provide a complexity analysis of the decision version of a model-checking algorithm for finite numbers of protocol sessions. In Section 3, we introduce a quantitative extension of the classical $\mathcal{DY}$ model and provide a complexity analysis of the corresponding extension of the model-checking algorithm. In Section 4, we consider a simple example to illustrate how our probabilistic intruder $p\mathcal{DY}$ can be applied for protocol analysis. In Section 5 we compare with related work and give an outlook on future research directions.

2 Model checking protocols under the Dolev-Yao intruder

Roughly speaking, a Dolev-Yao intruder [16] is an agent that completely controls the network but cannot break cryptography. By controlling the network we mean that the intruder can impersonate other agents, prevent messages from reaching their destination, or reroute them to other agents. The intruder can also generate messages from the knowledge he has acquired and send them to any agent, but he cannot break encryption unless he knows the corresponding key and cannot compute the content of a hashed message unless he knows it already. For this reason, the knowledge of the intruder plays a fundamental role in the $\mathcal{DY}$ model.

In fact, one of the core problems of security protocol analysis is the so-called *intruder deduction problem*: given a state of the protocol execution, can the intruder derive a given message M? Derivation here is relative to the terms the intruder currently knows, i.e. relative to the closure under a set of deduction rules of his initial knowledge augmented with the messages that he has observed during protocol execution.

The intruder deduction problem provides the basis for solving a number of practically relevant protocol analysis problems. One can, for instance, use it to determine whether the intruder is able to construct a message of the form that some honest agent is expecting to receive, or whether he is able to obtain a message that is intended to be a secret, e.g. a key shared by two honest agents.

Since the $\mathcal{DY}$ intruder model abstracts away cryptography, complexity and probability, it reduces attacks to unexpected protocol interleavings that leak information to the attacker. Hence, security is reduced to checking a safety condition; roughly speaking, is it true that the attacker never obtains/produces some private data? Model checking is particularly appealing to deal with such problems. The basic idea for model checking a safety condition is to check whether it is possible to reach a state where the condition fails. The model-checking algorithm consists in generating all possible reachable states and checking for each of these states whether the condition holds or not.

In general, in the case of an arbitrary number of protocol sessions that can be executed in an interleaved way, the problem is undecidable since it can be reduced to the halting problem. Under the restriction to a finite number of sessions, the problem of searching for an attack in the $\mathcal{DY}$ intruder model is an NP-complete problem [25] (and therefore checking if a protocol is secure is co-NP complete).

The approach that we consider in this paper is general and technology-independent, and can thus be effectively incorporated in different techniques and tools for security protocol analysis. For concreteness, we consider the constraint-based *on-the-fly-model-checker OFMC* [1,7,30]. [8] We will now summarize the key definitions and results that underlie OFMC (as well as a num-

[8] OFMC has been developed in the context of the AVISPA project. The AVISPA Tool is a fully automated protocol analysis environment that comprises of OFMC and three other tools (called CL-AtSe, SATMC, and TA4SP) and that has been successfully applied for the push-button analysis of a large number of industrial-scale security protocols.

ber of other approaches and tools), which in Section 3 we will then extend to model the probabilistic intruder that we consider here. More concretely, we now give a model-checking algorithm inspired to the approach of OFMC for finite numbers of sessions and provide a detailed complexity analysis of this algorithm. To that end, we begin by considering the context-free grammar given in [7] for formalizing protocol descriptions:

Definition 2.1 Let $\mathcal{C}$ and $\mathcal{V}$ be disjoint countable sets of *constants* (denoted by lowercase letters) and *variables* (denoted by uppercase letters). The syntax of our protocol specification language is defined by the following context-free grammar:

$$
\begin{aligned}
\mathit{ProtocolDescr} &::= (\mathit{State}, \mathit{Rule}^*, \mathit{AttackRule}^*)\\
\mathit{Rule} &::= \mathit{LHS}{\Rightarrow}\mathit{RHS}\\
\mathit{AttackRule} &::= \mathit{LHS}\\
\mathit{LHS} &::= \mathit{State}\ \mathit{NegFact}\ \mathit{Condition}\\
\mathit{RHS} &::= \mathit{State}\\
\mathit{State} &::= \mathit{PosFact}\ (\ .\ \mathit{PosFact})^*\\
\mathit{NegFact} &::= (\ .\ \mathsf{not}(\mathit{PosFact})\ )^*\\
\mathit{PosFact} &::= \mathsf{state}(\mathit{Msg}) \mid \mathsf{msg}(\mathit{Msg}) \mid \mathsf{i\_knows}(\mathit{Msg}) \mid \mathsf{secret}(\mathit{Msg}, \mathit{Msg})\\
\mathit{Condition} &::= (\ \wedge\ \mathit{Msg}\ \neq\ \mathit{Msg}\ )^*\\
\mathit{Msg} &::= \mathit{AtomicMsg} \mid \mathit{ComposedMsg}\\
\mathit{ComposedMsg} &::= \langle \mathit{Msg}{,}\mathit{Msg} \rangle \mid \{\mathit{Msg}\}_{\mathit{Msg}} \mid \{|\mathit{Msg}|\}_{\mathit{Msg}} \mid \mathit{Msg}(\mathit{Msg}) \mid \mathit{Msg}^{-1}\\
\mathit{AtomicMsg} &::= \mathcal{C} \mid \mathcal{V} \mid \mathbb{N} \mid \mathsf{fresh}(\mathcal{C}{,}\mathbb{N})
\end{aligned}
$$

We write $\mathcal{L}(n)$ for the context-free language associated with the nonterminal n. We write $\mathit{vars}(t)$ to denote the set of variables occurring in a (message, fact, or state) *term* t, and when $\mathit{vars}(t) = \emptyset$, we say that t is *ground*, and write $\mathit{ground}(t)$. We straightforwardly extend the functions *vars* and *ground* to the more complex terms.

In this paper, we give a quick intuitive top-down explanation of the used grammar. We refer the reader to [7] for a detailed explanation.

A protocol is represented by a term in $\mathcal{L}(\mathit{ProtocolDescr})$, that is, a triple (I, R, AR), where I is the initial state, R is the set of transition rules, and AR is the set of rules for identifying an attack state.

A state denotes the internal status of the network and is represented by a set of positive facts. There are four types of positive facts: (i) $\mathsf{state}(\mathit{Msg})$, which represents the local state of an honest agent; (ii) $\mathsf{msg}(\mathit{Msg})$, which represents a message in transit through the network (i.e. one sent but not yet received); (iii) $\mathsf{i\_knows}(\mathit{Msg})$, which represents a message known by the intruder; and (iv) $\mathsf{secret}(\mathit{Msg}, \mathit{Msg})$, which represents a secret message, where $\mathsf{secret}(m, a)$ means that m is a secret and that agent a is allowed to know it.

Rule describe state transitions. The left-hand side *lhs* of a rule $r = \mathit{lhs} \Rightarrow \mathit{rhs}$ consists of a set of positive facts P, a set of negative facts N, and a condition Cond, where $\mathit{vars}(P) \supseteq \mathit{vars}(N) \cup \mathit{vars}(\mathit{Cond})$. Negative facts are negations of positive facts, and usually only negations of state and secret facts are considered. Conditions are just conjunctions of inequalities of messages. Intuitively, the rules model the communication steps of honest agents executing a protocol. More specifically, rules model the transitions of the honest agents executing a protocol, i.e. transitions in which honest agents send a message in reply to another message they have received. The states where such a transition are enabled are described by the *lhs* of the associated rule. The state after the execution of the step is described by the *rhs* of the rule, which consists only of positive facts. Attack rules model the conditions for which an attack on the protocol is possible, and consist only of a *lhs* that characterizes the states for which the attack condition holds.

Message terms are used to describe all relevant information, like the intruder knowledge, keys, states of the network, etc. Messages can be either atomic or composed. An *atomic message* is a constant, a variable, a natural number, or a *fresh constant*. The fresh constants are used to model the creation of random data, e.g. nonces, during a protocol session. Following [7], we model each fresh data item by a unique term $\mathsf{fresh}(c,n)$, where c is an identifier and the number n denotes the particular protocol session that c is intended for. Messages can be composed using *pairing* $\langle m_1,m_2\rangle$, or the *cryptographic operators* $\{m_2\}_{m_1}$ and $\{|m_2|\}_{m_1}$ (for *asymmetric* and *symmetric encryption* of m_2 with m_1), $f(m)$ (for application of the function f to the message m, representing a hash-function or key-table), or m^{-1} (for the *asymmetric inverse* of m).

Before we present the model-checking algorithm for protocols described in the language above, we consider a simplification presented in [7]. First, we assume that AR is constituted just by one attack rule. Moreover, we only consider protocols (I, R, AR) where the rules $r = \mathit{lhs} \Rightarrow \mathit{rhs} \in R$ are of the form

$$\mathsf{msg}(m_1).\mathsf{state}(m_2).P_1.N_1 \wedge \mathrm{Cond} \Rightarrow \mathsf{state}(m_3).\mathsf{msg}(m_4).P_2\,, \qquad (1)$$

where N_1 is a set of negative facts that do not contain i_knows or msg facts, P_1 and P_2 are sets of positive facts that do not contain state or msg facts. Moreover, we require that if $\mathsf{i\_knows}(m) \in P_1$ then $\mathsf{i\_knows}(m) \in P_2$; this ensures that the intruder knowledge is *monotonic*, i.e. that the intruder never forgets messages during transitions.

The state facts appearing in both the *lhs* and the *rhs* of the rule mean that the rule describes a transition of an honest agent. Also, in both sides we have a msg fact representing the incoming message that the agent expects to receive in order to make the transition (in the *lhs*) and the agent's answer message (in the *rhs*). The rule corresponding to the initial (respectively, final) protocol step contains no incoming (respectively, outgoing) message. However, the rule form (1) is not a restriction, as one may always insert a dummy message that can be generated by the intruder. In fact, as argued in [7], rules of the form (1) are adequate to describe a very large class of industrial-scale security protocols (including protocols in the Clark/Jacob protocol library [13] as well as Internet protocols such as Kerberos, SET, IPSec, IKE, TLS, and H.530).

The basic idea underlying model checking for a protocol description (I, R, AR) is to build a state transition system modeling interleaved executions of the protocol by starting from the initial state I, which contains the initial knowledge of the honest protocol agents and of the intruder, and compute each successor state by applying the transition rules in R. For each of these successor states, we check whether an attack-rule in AR can be applied. If this is the case, then we have found an attack, otherwise we compute the next successor states and iterate the process until an attack is found or no more states can be generated. The latter situation only occurs in the case of a finite number of sessions.

In Definition 2.3 below, we give (the decision version of) a model-checking algorithm for finite numbers of protocol sessions under the classical Dolev-Yao intruder. This algorithm is (implicitly) given in [7] but its complexity is not investigated there. To provide a complexity analysis of the algorithm, we need to formalize the ways for generating successor states and for finding attacks. We start by describing how to generate successor states. Since rules model protocol steps, the successor states of a state S are obtained by analyzing the rules that are applicable to this S, that is, the rules for which S "satisfies" the pre-condition associated to the *lhs*.

Let substitutions σ be mappings from $\mathcal{V}$ to $\mathcal{L}(Msg)$. As we will formally define below, a rule is applicable to a state if (i) the positive facts are contained in the state for some substitution σ of the rule's variables, (ii) the negative facts under σ are not contained, and (iii) the condition *Cond* is satisfied under σ. Recall that the right-hand side *rhs* of a rule $lhs \Rightarrow rhs$ is just a set of positive facts, where $vars(lhs) \supseteq vars(rhs)$. The successors of a state S are then the states generated by replacing in S the facts that match the positive facts of the *lhs* of some applicable rule with the *rhs* of that rule.

Observe that the intruder has the power to replace the messages involved in the applicable rule with any messages he knows or can generate from his knowledge. For this reason, it is essential to describe the set of messages known by the intruder.

Definition 2.2 For a set M of messages, we define the Dolev-Yao intruder knowledge $\mathcal{DY}(M)$ as the smallest set closed under the generation ($G_{\_}$) and analysis ($A_{\_}$) rules given in Fig. 1.

The generation rules express that the intruder can compose messages from known messages using pairing, asymmetric and symmetric encryption, and

$$\frac{m \in M}{m \in \mathcal{DY}(M)}\, G_{\text{axiom}} \qquad \frac{m_1 \in \mathcal{DY}(M) \quad m_2 \in \mathcal{DY}(M)}{\langle m_1, m_2 \rangle \in \mathcal{DY}(M)}\, G_{\text{pair}}$$

$$\frac{m_1 \in \mathcal{DY}(M) \quad m_2 \in \mathcal{DY}(M)}{\{m_2\}_{m_1} \in \mathcal{DY}(M)}\, G_{\text{crypt}} \qquad \frac{m_1 \in \mathcal{DY}(M) \quad m_2 \in \mathcal{DY}(M)}{\{|m_2|\}_{m_1} \in \mathcal{DY}(M)}\, G_{\text{scrypt}}$$

$$\frac{m_1 \in \mathcal{DY}(M) \quad m_2 \in \mathcal{DY}(M)}{m_1(m_2) \in \mathcal{DY}(M)}\, G_{\text{apply}} \qquad \frac{\langle m_1, m_2 \rangle \in \mathcal{DY}(M)}{m_i \in \mathcal{DY}(M)}\, A_{\text{pair}}$$

$$\frac{\{m_2\}_{m_1} \in \mathcal{DY}(M) \quad m_1^{-1} \in \mathcal{DY}(M)}{m_2 \in \mathcal{DY}(M)}\, A_{\text{crypt}} \qquad \frac{\{m_2\}_{m_1^{-1}} \in \mathcal{DY}(M) \quad m_1 \in \mathcal{DY}(M)}{m_2 \in \mathcal{DY}(M)}\, A_{\text{crypt}}^{-1}$$

$$\frac{\{|m_2|\}_{m_1} \in \mathcal{DY}(M) \quad m_1^{-1} \in \mathcal{DY}(M)}{m_2 \in \mathcal{DY}(M)}\, A_{\text{scrypt}}$$

Fig. 1. The generation and analysis rules of the classical $\mathcal{DY}$ intruder.

function application. The analysis rules describe how the intruder can decompose messages. Note that no rules are given that allow the intruder to analyze function applications, for example to recover m from $f(m)$. Moreover, note that this formalization correctly handles non-atomic keys, for instance $m \in \mathcal{DY}(\{\{|m|\}_{f(k_1,k_2)}, k_1, k_2, f\})$.

We are now able to give an analytical presentation of the set of successor states, but before let us fix some notation. Let $\overline{P_1}$ be obtained from P_1 by removing all $\mathsf{i\_knows}$ facts

$$\overline{P_1} = P_1 \setminus \{f \mid \exists m.\ f = \mathsf{i\_knows}(m)\}\,.$$

We define the applicability of a rule r of the form (1) by the function *Applicable* that maps a state S and the left-hand side *lhs* of r to the ground substitution σ, under which the rule can be applied to the state:

$$\begin{aligned}
\mathit{Applicable}_{\mathit{lhs}}(S) = \{\sigma \mid \\
&\mathrm{ground}(\sigma) \wedge \mathrm{dom}(\sigma) = \mathrm{vars}(m_1) \cup \mathrm{vars}(m_2) \cup \mathrm{vars}(P_1) \\
&\wedge \{m_1\sigma\} \cup \{m\sigma | \mathsf{i\_knows}(m) \in P_1\} \subseteq \mathcal{DY}(\{m | \mathsf{i\_knows}(m) \in S\}) \\
&\wedge \mathsf{state}(m_2\sigma) \in S \wedge \overline{P_1}\sigma \subseteq S \\
&\wedge (\forall f.\, \mathsf{not}(f) \in N_1 \implies f\sigma \notin S) \wedge \sigma \vdash \mathit{Cond}\}.
\end{aligned}$$

Moreover, we define the successor function

$$\mathit{Succ}_R(S) = \bigcup_{r \in R} \mathit{Step}_r(S)$$

that, given a set R of rules of the form (1) and a state S, yields the corresponding set of successor states by means of the *Step* function

$$\begin{aligned} \mathit{Step}_{\mathit{lhs} \Rightarrow \mathit{rhs}}(S) = \{S' \mid \exists \sigma. \\ & \sigma \in \mathit{Applicable}_{\mathit{lhs}}(S) \\ & \wedge S' = (S \backslash (\mathsf{state}(m_2\sigma) \cup \overline{P_1}\sigma)) \cup \mathsf{i\_knows}(m_4\sigma) \cup P_2\sigma\}. \end{aligned}$$

As described before, the transition system that models a protocol (I, R, AR) is built by applying the successor function to the initial state I of the protocol and then to its successors. In this way, we obtain the set of reachable states, and it is important to note that this set is a *ground model.* No reachable state contains variables because of the way we define the transition.

Formally, the set of reachable states of the protocol (I, R, AR) is the set

$$\mathit{Reach}(I, R) = \bigcup_{n \in \mathbb{N}} \mathit{Succ}_R^n(I)\,.$$

Finally, we describe how to characterize insecure states. For this purpose, we introduce an attack predicate $\mathit{isAttack}_{AR}(S)$ that is true when an attack determined by the attack-rule AR is found at state S. This attack predicate $\mathit{isAttack}_{AR}(S)$ is true iff the rule AR can be applied to the state S, i.e. $\mathit{Applicable}_{AR}(S) \neq \emptyset$.

For simplicity, we consider here only a decision version of the model-checking algorithm. This means the algorithm just determines whether there exists an attack or not; while the attack itself is not returned. We initialize S as the knowledge available initially to the intruder and the honest protocol agents, that is, given a protocol $P = (I, R, AR)$ state S equals I.

Definition 2.3 Decision version of the model-checking algorithm for a finite number of protocol sessions:

```
ModelCheck(P)

Input: Protocol P = $(S, R, AR)$
Output: 1 if the protocol is secure, 0 otherwise

1. If ($\mathit{isAttack}_{AR}(S)$) then Return 0;
2. Compute the set $\mathit{Applicable}_{\mathit{lhs}}(S)$ for each $r \in R$ with $r = \mathit{lhs} \Rightarrow \mathit{rhs}$;
3. Compute $\mathit{Succ}_R(S)$;
4. For each $S' \in \mathit{Succ}_R(S)$;
5.      B=ModelCheck($S', R, AR$);
6.      If(B==0) then Return 0;
7. Return 1.
```

Before proceeding with the complexity analysis, we introduce some notation and make some observations. First, we define inductively the depth of a message $m \in \mathcal{L}(Msg)$ as follows:

- depth(m)=1 if m is atomic;
- depth($c(m)$)=1+depth(m) if c is a unary constructor;
- depth($c(m_1, m_2)$)=1+depth(m_1)+depth(m_2) if c is a binary constructor.

Note that in Step 2 of the algorithm above, a set of substitution needs to be computed accordingly with $Applicable_{lhs}(S)$. These substitutions allow the intruder to replace a message with one he knows. Moreover, by looking at Definition 2.2, it is easy to realize that the set of messages that the intruder knows is infinite thanks to the generation rules. However, by fixing the number of sessions to s and assuming that each session consists of at most j steps, it is easy to see that only a finite number of messages can lead to an attack.

The intuitive argument behind this fact is that the interaction with the agents can only increase/decrease the depth of the messages involved in the computation by a finite amount. Since the overall number of steps for the session is finite, there is a limit k such that if there is an attack then there is an attack with messages exchanged with depth at most k. This value depends on all the parameters of the protocol: initial state, rules and attacks rules; as well as on the number s of sessions of the protocol executed in parallel. Moreover, without loss of generality, we assume that k is greater than the depth of all message terms occurring in the protocol. Still in the context of messages terms, the following result gives an upper-bound to the number of terms up to some depth that are freely generated from an algebra signature.

Lemma 2.4 The number of terms with depth less than or equal to k for a free algebra with b binary constructors, u unary constructors, and a atomic symbols is $O((a+b+u)^{2^k})$.

We now address the question of the number of parallel sessions of a protocol. Assume that there are s sessions of a protocol with j steps running in parallel. Since we model each step of (a session of) the protocol with a rule $r \in R$, we have j rules for each session. Moreover, nothing prevents the protocol from being completely parallel, that is, nothing prevents all the rules in R from being applicable to all reachable states but the final ones. So, we know that for s sessions of the protocol with j steps we may have at most $j \times s = n$ rules that can be applicable at each state.

We are now able to start the complexity analysis. From Lemma 2.4, we conclude that the number of message terms with depth less than or equal to k generated from a atomic symbols is $O((a+4)^{2^k})$. Since we assume that k upper-bounds the depth of the message terms of the protocol, the intruder can only learn at most k atomic symbols per message exchange. If we assume that the intruder initially knows i atomic messages, we conclude that the total amount of relevant terms the intruder may learn is $O((nk+i+4)^{2^k}) =$ *num_iknow*. By assuming that the *lhs* and *rhs* of the rules in R have at most v variables, the number of substitutions to check if they are *applicable* is $O(\mathit{num\_iknow}^v) = O((nk+i+4)^{v2^k}) = \mathit{num\_sub}$. Since for each rule we have at most *num_sub* new successor states, and given that there are at most n rules, the number of successors for a given state is $O(n \times \mathit{num\_sub}) = O(n(nk+i+4)^{v2^k}) = \mathit{num\_suc}$.

Observe that the depth of the recursion tree of the model-checking algorithm is $j \times s = n$, and so the recursion tree will generate at most $O(\mathit{num\_suc}^n) = O(n^n(nk+i+4)^{nv2^k}) = \mathit{num\_tot\_st}$ total states. In the worst case, the *isAttack* predicate has to be checked over all these states, and so the overall time complexity for Step 1 is obtained by multiplying *num_tot_st* with the time complexity of checking the *isAttack* predicate.

Recall that the *isAttack(S)* predicate reduces to checking whether

$$\mathit{Applicable}_{AR}(S) \neq \emptyset\,.$$

Hence, the time complexity is upper-bounded by the time complexity of computing $\mathit{Applicable}_{AR}$. For any *lhs*, to compute $\mathit{Applicable}_{lhs}$ we need to range over all *num_sub* ground substitutions of v variables by *num_iknow* terms known by the intruder and, moreover, verify for each of these substitutions if certain conditions hold. The latter verification takes polynomial time on the size of the *lhs*, which for simplicity we consider to be at most g time for all rules $r \in R$. So, assuming that all message terms the intruder knows are stored in memory, with access $O(1)$, in order to generate all *num_sub* possible substitutions it takes $O(\mathit{num\_sub} \times v \times \log(\mathit{num\_iknow}))$ (this corresponds to generating *num_sub* increments on a number with v digits written in basis *num_iknow*). Thus, the time complexity of computing $\mathit{Applicable}_{lhs}(S)$ is

$$\begin{aligned} O(g \times \mathit{num\_sub} \times v \times \log(\mathit{num\_iknow})) &= O(gvn(nk+i+4)^{v2^k}\log((nk+i+4)^{v2^k})) \\ &= O(gv^2 2^k n(nk+i+4)^{v2^k}\log(nk+i+4)) \\ &= \mathit{time\_app}. \end{aligned}$$

Hence, we have that the time complexity of checking the *isAttack* predicate over all states is

$$O(\mathit{time\_app} \times \mathit{num\_tot\_st}) = O(gv^2 2^k n^{n+1}(nk+i+4)^{(n+1)v2^k}\log(nk+i+4))\,,$$

which takes care of Step 1 of the algorithm.

To compute the complexity of Step 2, we just have to multiply the total number of states *num_tot_st* by the time complexity of computing $\mathit{Applicable}_{lhs}(S)$ *time_app* and the total number or rules n, obtaining thus

$$O(gv^2 2^k n^{n+\ 2}(nk+i+4)^{(n+1)v2^k}\log(nk+i+4))\,.$$

The time complexity of Step 3 consists of generating the successor states, which is bounded by the total number of states. Since we assume that the intruder knowledge in each state is stored in memory we need to update it. In the worst case, the number of message terms known by the intruder in each state is $O(\mathit{num\_iknow})$. The time required to update the message terms up to a depth of k is bounded by $O(\mathit{num\_iknow}^2)$ multiplied by the number of binary rules existing in the $\mathcal{DY}$ model plus $O(\mathit{num\_iknow})$ multiplied by the number of unary rules. The idea is to consider all pairs of known messages and apply all possible binary rules, and similarly for unitary rules. In detail, for each binary rule, we have to consider each pair of messages the intruder knows, apply the rule (if possible) and verify if we obtain an unknown message. If this is the case, we have to add this message to the knowledge of the intruder and henceforward consider pairs with this new message. A similar approach to update the knowledge of the intruder is done for unary rules. Since there are seven binary rules and one proper unary rule (G_{axiom} does not need to be considered) and by assuming that all $\mathcal{DY}$ intruder rules take at most d time to be applied, we conclude that the total time to update the intruder knowledge is

$$O(d(7\mathit{num\_iknow}^2 + \mathit{num\_iknow})) = O(d(nk+i+4)^{2^{k+1}}) = \mathit{upd\_know}\,.$$

Finally, since we need to update the intruder knowledge for every non initial state, the overall complexity of Step 3 is given by

$$O(\mathit{upd\_know} \times \mathit{num\_tot\_st}) = O(dn^n(nk+i+4)^{(2+nv)2^k})\,.$$

The remaining steps of the algorithm correspond to traversing the recursion tree, so we conclude that:

Proposition 2.5 *The overall complexity of the model-checking algorithm* `ModelCheck(P)` *for a finite number of protocol sessions under the classical Dolev-Yao intruder is*

$$O(gv^2 2^k n^{n+\ 2}(nk+i+4)^{(n+1)v2^k}\log(nk+i+4) + dn^n(nk+i+4)^{(2+nv)2^k})\,.$$

The above analysis gives an upper-bound on the complexity of the model-checking algorithm and suggests that the naive model checker is exponential in the number of protocol steps, protocol sessions and exchanged messages per step, double exponential in the maximum depth of the messages the intruder needs to know, and polynomial in the remaining factors. Nevertheless, it hints that the presented algorithm is rather time expensive. Indeed, this is mostly due to the excessive number of successor states of the transition system caused by the huge amount of messages the intruder gets to derive at each protocol step (this leads to the double exponential factor in the complexity analysis over the maximum depth of messages the intruder needs to know). A way to avoid this problem is to consider a symbolic "lazy" intruder (see [7] and the various references there), an intruder for which the substitutions at Step 2 are postponed and substantially restricted. A much better performance is achieved for the lazy intruder, however the complexity analysis is out of the scope of this paper, and is far from being straightforward. The purpose of the above computation is to set a basis for comparing the complexity of model-checking under the classical Dolev-Yao intruder with the complexity under the probabilistic extension of the intruder that we introduce in the next section.

3 Extending the Intruder Model

The classical Dolev-Yao intruder model assumes that cryptography is perfect and that the intruder can only decrypt an encrypted message if he knows the corresponding decryption key. Since one of the ultimate goals of our work is to augment the intruder model in such a way to allow the intruder to perform attacks on cryptography (e.g. representing that he can break an encryption with a certain probability), we can no longer ignore the underlying cryptographic schemes used in the protocol specification.

Our approach allows the intruder to "retrieve" certain data from the messages he knows, but only with some probability of success. The intruder can, for example, guess keys to decrypt encrypted messages or arguments of hash functions from their hashed values. This new ability may possibly result in a significant increase of the intruder knowledge and thus open new ways for attacks. It is therefore important to accurately measure the probability of correctly retrieving data, either by simple guessing or resorting to cryptanalysis techniques.

In general, with this new intruder model we expect to detect flaws in protocols caused by badly chosen cryptographic systems in important positions of the protocol flow, such as a repetitive use of the same encryption key that allows the intruder to gather enough information to be able to compute the key with a high probability of success, or use of cryptographic functions with a low security parameter.

To quantify the efforts of the intruder, we follow an approach based on the dimensions of the key space of a given cryptographic system and co-domain of cryptographic functions, taking into account the fact that additional knowledge may reduce the key space significantly.[9] For formalizing protocol descriptions, we consider the context-free grammar given in Definition 2.1 and extend it to express the probability of a transition when guessing is used by the intruder. To keep track of the transition probabilities, we extend the language with a new positive fact $\mathsf{plabel}(\mathbb{R})$, and we also introduce the function

$$p : \mathcal{L}(\mathit{Msg}) \times \mathcal{P}(\mathcal{L}(\mathit{PosFact})) \to \mathbb{R}$$

[9] Note that other approaches could be used, like the amount of computational power needed for a brute force attack, or the likelihood of a successful attack given a time bound for cryptanalysis. Of course, these approaches are highly dependent on the assumptions on the computational power of the intruder, nonetheless they can be tuned to mirror the capabilities of a relatively powerful intruder. Note also that the approach we propose here is different from, and is in fact complementary to, the symbolic, deductive approach to off-line guessing proposed in [17].

$$\frac{m \in M}{m \in p\mathcal{DY}(M)}\,G_{\text{axiom}} \qquad \frac{m_1 \in p\mathcal{DY}(M) \quad m_2 \in p\mathcal{DY}(M)}{\langle m_1, m_2\rangle \in p\mathcal{DY}(M)}\,G_{\text{pair}}$$

$$\frac{m_1 \in p\mathcal{DY}(M) \quad m_2 \in p\mathcal{DY}(M)}{\{m_2\}_{m_1} \in p\mathcal{DY}(M)}\,G_{\text{crypt}} \qquad \frac{m_1 \in p\mathcal{DY}(M) \quad m_2 \in p\mathcal{DY}(M)}{\{\!|m_2|\!\}_{m_1} \in p\mathcal{DY}(M)}\,G_{\text{scrypt}}$$

$$\frac{m_1 \in p\mathcal{DY}(M) \quad m_2 \in p\mathcal{DY}(M)}{m_1(m_2) \in p\mathcal{DY}(M)}\,G_{\text{apply}} \qquad \frac{\langle m_1, m_2\rangle \in p\mathcal{DY}(M)}{m_i \in p\mathcal{DY}(M)}\,A_{\text{pair}}$$

$$\frac{\{m_2\}_{m_1} \in p\mathcal{DY}(M) \quad m_1^{-1} \in p\mathcal{DY}(M)}{m_2 \in p\mathcal{DY}(M)}\,A_{\text{crypt}} \qquad \frac{\{m_2\}_{m_1^{-1}} \in p\mathcal{DY}(M) \quad m_1 \in p\mathcal{DY}(M)}{m_2 \in p\mathcal{DY}(M)}\,A^{-1}_{\text{crypt}}$$

$$\frac{\{\!|m_2|\!\}_{m_1} \in p\mathcal{DY}(M) \quad m_1 \in p\mathcal{DY}(M)}{m_2 \in p\mathcal{DY}(M)}\,A_{\text{scrypt}}$$

$$\frac{\{\!|m_2|\!\}_{m_1} \in p\mathcal{DY}(M) \quad m_1 \notin p\mathcal{DY}(M)}{m_1 \in p\mathcal{DY}(M)}\,\text{guessSK} \qquad \frac{\{m_2\}_{m_1} \in p\mathcal{DY}(M) \quad m_1^{-1} \notin p\mathcal{DY}(M)}{m_1^{-1} \in p\mathcal{DY}(M)}\,\text{guessPK}$$

$$\frac{H(m_1, \ldots, m_i, \ldots, m_k) \in p\mathcal{DY}(M) \quad m_i \notin p\mathcal{DY}(M)}{m_i \in p\mathcal{DY}(M)}\,\text{guessHash}$$

Fig. 2. The deduction rules of the $p\mathcal{DY}$ intruder.

such that $p(m, \mathit{IK})$ returns the probability of the intruder knowing $m \in \mathcal{L}(\mathit{Msg})$ when his knowledge is IK, a set of positive facts of the form $\mathsf{i\_knows}(\mathit{Msg})$. We will explain later how this probability is computed.

Definition 3.1 For a set M of messages, we define $p\mathcal{DY}(M)$ as the smallest set closed under the rules given in Fig. 2: generation rules ($G_{\_}$), analysis rules ($A_{\_}$), two rules *guessSK* and *guessPK* for the intruder guessing symmetric and inverse (private) keys, and a rule *guessHash* that states that whenever the intruder possesses the hash value of a message he can retrieve that message or part of it (note that we assume that hash functions are known to all participants and thus to the intruder).

Note that, in the presence of other cryptographic operators, we could of course add other, similar, guessing rules. The ones we consider here help us illustrate the principles of our approach.

To incorporate the changes in the intruder model, every state transition must reflect the fact that the intruder may have used a guessing rule. Thus, we need to distinguish the deterministic transitions from the probabilistic ones. To carry out this distinction, we label each transition with the value we obtain from the function p, where p returns 1 if no guessing was needed. When a state S is found that triggers the attack-rule AR, we determine the attack probability by traversing backwards the attack trace (from S to I) and multiplying all the probability labels. Note that by doing so, we only "introduce" new attacks with respect to the analysis in the classical $\mathcal{DY}$ model;

the attacks that were already detectable in the $\mathcal{DY}$ model will also be detected in the $p\mathcal{DY}$ model and will be labeled with probability 1.

We define the extended applicability of a rule r by the extended function *pApplicable* that maps a state S and the left-hand side *lhs* of r to the pair $\langle\sigma,\xi\rangle$, where σ is a ground substitution and $\xi \in \mathbb{R}$, under which the rule can be applied to the state:

$$\begin{aligned}
&\mathit{pApplicable}_{\mathit{lhs}}(S) = \{\langle\sigma,\xi\rangle \mid \\
&\qquad \mathrm{ground}(\sigma) \wedge \mathrm{dom}(\sigma) = \mathrm{vars}(m_1) \cup \mathrm{vars}(m_2) \cup \mathrm{vars}(P_1) \\
&\qquad \wedge \{m_1\sigma\} \cup \{m\sigma \mid \mathsf{i\_knows}(m) \in P_1\} \subseteq p\mathcal{DY}(\{m \mid \mathsf{i\_knows}(m) \in S\}) \\
&\qquad \wedge \xi = \Pi_{t\in\text{co-dom}(\sigma)} p(t, \{m \mid \mathsf{i\_knows}(m) \in S\}) \\
&\qquad \wedge \mathsf{state}(m_2\sigma) \in S \wedge \overline{P_1}\sigma \subseteq S \\
&\qquad \wedge (\forall f.\ \mathsf{not}(f) \in N_1 \Longrightarrow f\sigma \notin S) \wedge \sigma \vdash \mathit{Cond}\}.
\end{aligned}$$

To compute $p(t, \{m \mid \mathsf{i\_knows}(m) \in S\})$, we first assume an algorithm *prule* to compute the probability $\mathit{prule}(\rho)$ for each instance ρ of a guess rule of the $p\mathcal{DY}$ intruder model. This algorithm depends on the cryptographic assumptions, and it is assumed to be given for the model-checking algorithm. Given a derivation δ of m using the $p\mathcal{DY}$ rules, the probability $p_\delta(m)$ of knowing m through δ is given by multiplying all the probabilities $\mathit{prule}(\rho)$ associated to the instances of the rules ρ occurring in δ. We then have that

$$p(t, \mathit{IK}) = \max\{p_\delta(m) \mid \delta \text{ is a derivation of } m \text{ using knowledge } \mathit{IK}\}\,.$$

We will discuss in the extended model-checking algorithm a (naive) way to compute $p(t, \mathit{IK})$.

We define the successor function

$$\mathit{pSucc}_R(S) = \bigcup_{r\in R} \mathit{pStep}_r(S)$$

that, given a set R of rules of the form (1) and a state S, yields the corresponding set of successor states by means of the step function

$$\begin{aligned}
&\mathit{pStep}_{\mathit{lhs}\Rightarrow\mathit{rhs}}(S) = \{S' \mid \exists\sigma.\,\exists\xi. \\
&\qquad \langle\sigma,\xi\rangle \in \mathit{pApplicable}_{\mathit{lhs}}(S) \\
&\qquad \wedge S' = (S \setminus (\mathsf{state}(m_2\sigma) \cup \overline{P_1}\sigma)) \cup \mathsf{i\_knows}(m_4\sigma) \cup P_2\sigma \cup \mathsf{plabel}(\xi)\}.
\end{aligned}$$

In this new setting, the set of reachable states of a protocol (I, R, AR) is the set

$$\mathit{pReach}(I, R) = \bigcup_{n\in\mathbb{N}} \mathit{pSucc}_R^n(I)\,.$$

Given an attack-rule AR and a state S, the attack predicate $\mathit{ispAttack}_{AR}(S)$ is true if and only if the rule AR can be applied to the state S, that is

$pApplicable_{AR}(S) \neq \emptyset$. Moreover, given an attack-rule AR, if $ispAttack_{AR}(S)$ is true, then the probability $pAttack_{AR}(S)$ of such an attack is given by

$$pAttack_{AR}(S) = \prod_{x \in \{t \mid \mathsf{plabel}(t) \in S\}} x \,.$$

We are now able to state the probabilistic version of the intruder deduction problem for a finite number of sessions in the extended Dolev-Yao model and discuss the quantitative model-checking algorithm.

Definition 3.2 Given a protocol (I, R, AR) in the $p\mathcal{DY}$ model, the *probabilistic intruder deduction problem* consists in determining whether the intruder can learn a certain message, with probability greater than or equal to p, as defined by AR, when s sessions of the protocol are run.

It is easy to see that this problem is NP-complete. First notice that the non-quantitative version of intruder deduction problem can be reduced to this one by taking $p = 1$, and so the problem is NP-hard. Moreover, given a trace of the attack (which includes derivations of the intruder knowledge) we can check, in polynomial-time, whether this attack is carried out with probability greater than p. So, there is a polynomial witness verification algorithm for the case of a possible attack. Moreover, if there is no attack, any witness (trace) provided will not result in verifying an attack. So the problem is in NP, and therefore NP-complete.

Theorem 3.3 The probabilistic intruder deduction problem for a finite number of sessions in the $p\mathcal{DY}$ intruder model is NP-complete.

Concerning the model-checking algorithm, the extension we propose does not change much the pseudo-code given in Definition 2.3.

Definition 3.4 Decision version of the quantitative model-checking algorithm:

```
qModelCheck(P)

Input: Protocol $P = (S, R, AR)$ and probability $p$
Output: 1 if the protocol is secure, 0 otherwise

1. If ($ispAttack_{AR}(S) \wedge pAttack_{AR}(S) \geq p$) then Return 0;
2. Compute the set $pApplicable_{lhs}(S)$ for each $r \in R$ with $r = lhs \Rightarrow rhs$;
3. Compute $pSucc_R(S)$;
4. For each $S' \in pSucc_R(S)$;
5.      B=ModelCheck($S', R, AR$);
6.      If(B==0) then Return 0;
7. Return 1.
```

We proceed by analyzing the complexity of the quantitative model-checking algorithm. Note that the number of relevant terms needed to be known by both the classical and the probabilistic intruder is the same, that is, terms with depth at most k. Hence, in the worst case, the total number of states of the recursion tree is the same (asymptotically) for both the classical and quantitative model checker, that is,

$$O(n^n(nk+i+4)^{nv2^k}) = \mathit{num\_tot\_st}\,,$$

where $n = j \times s$ is the product of the number of steps j of the protocol and the number of sessions s, i is the number of initial atomic messages known by the intruder, and v is the maximum number of variables existing in a *lhs* of the rules in R and AR.

There are clearly some differences between the quantitative and the classical model-checking algorithms. Namely, we have to change the way the function $\mathit{pApplicable}_{lhs}(S)$ is computed and the way the successor states are generated (including how the knowledge of the intruder increases). Moreover, we have to compute $\mathit{pAttack}_{AR}(S)$.

We start by noticing that in the extended model, $\mathit{pApplicable}_{lhs}(S)$ returns pairs $\langle \sigma, \xi \rangle$ and that ξ is uniquely determined from σ and the intruder knowledge. Like for the classical $\mathcal{DY}$ model and for the sake of simplicity, we assume that we have in memory all terms known by the intruder with their probabilities, and that we can access these values in $O(1)$ time. So, the time to compute $\mathit{pApplicable}_{lhs}(S)$ in the $p\mathcal{DY}$ model is the same as in the $\mathcal{DY}$ model multiplied by the time to compute ξ. Since ξ is a product of v elements that can be obtained in $O(1)$, this multiplication can be achieved in v time (each multiplication takes $O(1)$ since it is a floating point multiplication). Hence, $\mathit{pApplicable}_{lhs}(S)$ is computed in

$$O(v \times \mathit{time\_app}) = O(gv^3 2^k n(nk+i+4)^{v2^k} \log(nk+i+4))\,.$$

Now we show how $\mathit{pAttack}_{AR}(S)$ can be computed in $O(1)$. Indeed, we will not check for the plabel facts in state S, but rather have a register that stores the probability of reaching S. This register is initialized to 1, and when a successor of S is computed via a pair $\langle \sigma, \xi \rangle$, this register is multiplied by ξ (which takes $O(1)$ time). Thus, this register gives the probability of reaching a state S and when this state has an attack, the register takes the value $\mathit{pAttack}_{AR}(S)$.

Hence, the time complexity of checking the *ispAttack* predicate and computing $\mathit{pAttack}_{AR}(S)$ over all states is

$$O(v \times \mathit{time\_app} \times \mathit{num\_tot\_st}) = O(gv^3 2^k n^{n+1}(nk+i+4)^{(n+1)v2^k} \log(nk+i+4))\,,$$

which takes care of Step 1.

To compute the complexity of Step 2 we just have to multiply the total number of states *num_tot_st* by the time complexity of computing the function

$\mathit{pApplicable}_{lhs}(S)$ and the total number or rules n, obtaining thus

$$O(gv^3 2^k n^{n+\ 2}(nk+i+4)^{(n+1)v2^k}\log(nk+i+4)).$$

In Step 3 we update the intruder knowledge for all the successor states. As stated above, we assume that the intruder knowledge is stored in memory with access time $O(1)$. In the worst case, the intruder knowledge has $O(\mathit{num\_iknow})$ messages, thus the time required for updating is bounded by $O(\mathit{num\_iknow}^2)$ multiplied by the number of binary rules existing in the $p\mathcal{DY}$ model plus $O(\mathit{num\_iknow})$ multiplied by the number of unary rules. We consider the guessing rules to be unary rules and we ignore the non-belonging proviso. When a guessing rule ρ is applied to m to derive m', in order to calculate the probability of getting m', we take from memory the probability p of knowing m, and three situations may happen:

- if m' was not known before, then we introduce this term with probability $p \times \mathit{prule}(\rho)$;
- if m' was known before and its probability was smaller than $p \times \mathit{prule}(\rho)$, then we update this probability;
- if m' was known before and its probability was greater than or equal to $p \times \mathit{prule}(\rho)$, then nothing is done.

Observe that if a probability of a term is updated, we need to iterate the updating process until no probability is increased. So, if ϵ is the smallest increase possible in a probability term (note that ϵ is the desired precision we want for the probabilities to be computed), in the worst case each rule only increases the probability of each term by ϵ, and so we have to apply each rule $1/\epsilon$ times. Hence, the total amount of time to update the intruder knowledge at one state for the $p\mathcal{DY}$ model is

$$O(1/\epsilon \times d \times \mathit{num\_iknow}^2) = O(1/\epsilon \times d(nk+i+4)^{2^{k+1}}) = O(1/\epsilon \times \mathit{upd\_know}),$$

where d is an upper bound for the time it takes to apply a rule. Finally, since we need to update the intruder knowledge for every non-initial state, the overall complexity of Step 3 is given by

$$O(1/\epsilon \times \mathit{upd\_know} \times \mathit{num\_tot\_st}) = O(1/\epsilon \times dn^n(nk+i+4)^{(2+nv)2^k}).$$

The remaining steps of the algorithm correspond to traversing the recursion tree, so we conclude that:

Proposition 3.5 *The overall complexity of the model-checking algorithm* `qModelCheck(P)` *for a finite number of protocol sessions under the $p\mathcal{DY}$ intruder is*

$$O(gv^3 2^k n^{n+\ 2}(nk+i+4)^{(n+1)v2^k}\log(nk+i+4) + 1/\epsilon \times dn^n(nk+i+4)^{(2+nv)2^k}).$$

1. $A \rightarrow S : A$
2. $S \rightarrow A : N_S$
3. $A \rightarrow S : H(pw, N_A, N_S, A), N_A$
4. $S \rightarrow A : H(pw, N_A)$

Fig. 3. The MS–CHAPv2 Protocol.

Hence, the time for quantitative model checking takes essentially $1/\epsilon$ more time than classical model checking. In other words, the time increase from the $\mathcal{DY}$ and the $p\mathcal{DY}$ model is inversely proportional to the precision ϵ of representing probabilities.

4 Example

To illustrate how the intruder behaves under our extension of the Dolev-Yao model, we consider the MS-CHAPv2 authentication protocol, the Microsoft Challenge/ Response Authentication Protocol, version 2 [32], presented in Fig. 4. MS-CHAPv2 is the authentication mechanism for the Point-to-Point Tunneling Protocol (PPTP [18]), which itself is used to secure PPP connections over TCP/IP.

The objective of the MS-CHAPv2 Protocol is to achieve mutual authentication between a client A and a server S. This authentication is carried out under the assumption that there is an initial password pw between A and S. The protocol goes as follows. In Step 1, the client A sends a message to the server S saying that he wants to start a new session. In Step 2, S replies to A's request by sending him a fresh session-id N_S. A replies to this in Step 3 by hashing the tuple (pw, N_A, N_S, A) where pw is the shared password and N_A is a nonce freshly generated by A, which A also sends in cleartext appended to the hash. When receiving this message, S is sure that he is talking to A as only A knows the value pw. Mutual authentication is achieved if S replies back with the hash of (pw, N_A) as only S knows pw and N_A.

It is already known that this protocol is vulnerable to off-line guessing attacks (see [27] and also [17]). Our extended intruder model allows us to quantify this vulnerability. In this protocol, both agents share a password pw and authentication depends on the agreement of the hash values exchanged in the last two steps of the protocol.

If the intruder is able to guess the password from the hash value he receives in Step 3, then he can impersonate the server. Below is a trace of an attack where the intruder I assumes the role of S and agent A is led into believing he is talking to the server and authenticates himself successfully.

$$\begin{array}{ll}
1.\ A \to I(S) : A & IK_1 = \{H, A, S\} \\
2.\ I(S) \to A : N_S & IK_2 = \{H, A, S, N_S\} \\
3.\ A \to I(S) : H(pw, N_A, N_S, A), N_A & IK_3 = \{H, A, S, N_S, N_A, \\
 & \qquad H(pw, N_A, N_S, A)\} \\
4.\ I(S) \to A : H(pw, N_A) & IK_4 = \{H, A, S, N_S, N_A, \\
 & \qquad H(pw, N_A, N_S, A), H(pw, N_A)\}
\end{array}$$

In the fourth step of the trace, the intruder is able to construct and send the message $H(pw, N_A)$, since $H(pw, N_A) \in p\mathcal{DY}(IK_3)$ as shown by the following derivation:

$$\dfrac{\dfrac{}{H}\,G_{\text{axiom}} \qquad \dfrac{\dfrac{}{N_A}\,G_{\text{axiom}} \qquad \dfrac{\dfrac{}{H(pw, N_A, N_S, A)}\,G_{\text{axiom}}}{pw}\,{}_{\text{guessHash}}, p(pw, IK_3)}{\langle pw, N_A \rangle}\,G_{\text{pair}}}{H(pw, N_A)}\,G_{\text{apply}}$$

This constitutes an attack on the protocol. However, this is only possible if rule guessHash is applied. In the above derivation, the application of guessHash is tagged with a value that represents the probability that the intruder can guess pw having knowledge IK_3. The probability of obtaining $H(pw, N_A)$ is then computed from its derivation from $p\mathcal{DY}(IK_3)$, and the probability of success of the attack is obtained by examining all the probabilities involved in the derivation tree of the attack. In our case, as steps 1, 2 and 3 of the trace have no probabilities involved, the success of this attack is given simply by $p(pw, IK_3)$, which is the only probability in the derivation.

This is a simple example but illustrates how one can use our approach. By introducing the new rules that consider probabilistic behaviors, one can explore other types of attacks that cannot be formalized with the standard Dolev-Yao intruder model. If one can parameterize the model-checking algorithm in such a way that the rules reflect the real probability of breaking the cryptographic primitives, then one may perform a quantitative analysis of the insecurity of a protocol.

5 Related Work and Future Work

In the last few years, much attention has been devoted to the analysis of security protocols, for which several automated tools have been proposed, e.g. [1,8,10,15,22,24,26,28,30]. The major drawback of these tools is that they perform abstractions of cryptographic primitives as functions over an algebra of terms, in the spirit of [16]. Such abstractions may not reflect the cryptographic reality and hence one may be ignoring attacks while doing this (consider as an example the attack given in [31] on a protocol proved to be correct using automated tools [21]).

Recent work has tried to bridge the gap between these abstractions and computational complexity, e.g. [2,3,4,5,6,9,11,12,14,19,20,23,29]. These results show that it is possible to faithfully abstract encryption schemes and signature schemes as functions over a term algebra. More specifically, it is shown that an intruder that interacts with the "real system" does not obtain more information than when interacting with an "ideal system", where decryption is only possible when in possession of the decryption key. This is formalized as saying that two messages that are symbolically indistinguishable, are also computationally indistinguishable.

Let us focus on two of the most recent works. Sprenger *et al.* [29] proposed recently an implementation for the model given in [6] using the theorem prover Isabelle/HOL. By encoding the symbolic model of [6] in Isabelle/HOL, they thus provided the first cryptographically-sound theorem prover for security protocols. While this is a very promising approach, there is still room for further improvements. For instance, it can only reason about protocol functionalities that can be expressed in the reactive simulatability framework, leaving aside important cryptographic tasks such as zero-knowledge, bit-commitment and oblivious transfer.

Blanchet [9] concurrently proposed another mechanized prover for secrecy properties of security protocols. This prover, contrarily to the previous ones, does not rely on the symbolic Dolev-Yao intruder model, but on the computational cryptographic model. The proofs are done by performing a sequence of cryptographic games and transformations, which correspond to the cryptographic reductions that are usually done by hand.

In our work, we propose a third approach, which builds on [7] and combines some of the characteristics of these two other approaches. We continue to use a Dolev-Yao-style model checker but also address the possibility that some cryptographic primitives can be attacked with a certain probability. We formalize such an intruder model by extending the $\mathcal{DY}$ model with additional structure and intruder capabilities, in order to be able to consider more kinds of attacks. The extended $p\mathcal{DY}$ model contains intruder deduction rules that depend on the cryptographic primitives being used. These rules are tagged with a probability of success that, if an attack is found, will give us the probability of success of the attack (just compute the probability of applying all the guessing rules in that branch of the derivation). As concrete examples, we considered here rules related to encryption and hashing, where an intruder may obtain the decryption key out of an encrypted message or could retrieve part of the body of a hashed message. Rules for other cryptographic operators could be introduced similarly.

We also showed that by performing these extensions we do not increase the complexity of searching for an attack when compared to the standard $\mathcal{DY}$ model; the problem remains NP-complete. Hence, searching for an attack with this extended model is as "bad" as with the classical intruder model.

The extension we propose requires a clear commitment to the cryptographic primitives being used in the specified protocol, for instance, for encryption schemes, the size of the keys and the key space. This information is important because it is required in the calculation of the transition probabilities and the attack probability. As future work, we aim to enhance OFMC by extending the underlying model-checking algorithm with a calculation of probabilities. This will allow us to employ the tool to find out if the attack probability is below a given threshold ε, and reduce the search space by abandoning the state exploration when the product of the probability labels of a certain state is less than the given ε.

In order to reduce the complexity of our algorithms, we also intend to apply to our extended model the lazy intruder technique and other search-space-reduction techniques implemented in OFMC. This will allow us to exploit our quantitative approach for the analysis of industrial-scale protocols. The use of abstractions and over-approximations for unbounded, quantitative protocol verification will also be an interesting avenue of future research.

References

[1] *The AVISPA Project — Automated Validation of Internet Security Protocols and Applications*.
URL http://www.avispa-project.org

[2] Abadi, M. and P. Rogaway, *Reconciling two views of cryptography (the computational soundness of formal encryption)*, Journal of Cryptology **15** (2002), pp. 103–127.

[3] Adão, P., G. Bana and A. Scedrov, *Computational and information-theoretic soundness and completeness of formal encryption*, in: *Proceedings of the 18th IEEE Computer Security Foundations Workshop (CSFW)* (2005), pp. 170–184.

[4] Backes, M. and B. Pfitzmann, *A cryptographically sound security proof of the Needham-Schröeder-Lowe public-key protocol*, IEEE Journal on Selected Areas in Communications **22** (2004), pp. 2075–2086, preliminary version presented at FSTTCS'03.

[5] Backes, M. and B. Pfitzmann, *Relating symbolic and cryptographic secrecy*, IEEE Transactions on Dependable and Secure Computing **2** (2005), pp. 109–123, preliminary version presented at S&P'05.

[6] Backes, M., B. Pfitzmann and M. Waidner, *A composable cryptographic library with nested operations*, in: S. Jajodia, V. Atluri and T. Jaeger, editors, *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS)* (2003), pp. 220–230, full version available at IACR ePrint Archive, Report 2003/015, January 2003.

[7] Basin, D., S. Mödersheim and L. Viganò, *OFMC: A symbolic model checker for security protocols*, International Journal of Information Security **4** (2005), pp. 181–208.

[8] Blanchet, B., *An Efficient Cryptographic Protocol Verifier Based on Prolog Rules*, in: *Proceedings of the 14th IEEE Computer Security Foundations Workshop (CSFW)* (2001), pp. 82–96.

[9] Blanchet, B., *A computationally sound mechanized prover for security protocols*, in: *Proceedings of the 2006 IEEE Symposium on Security and Privacy (S&P)* (2006), to Appear. Full version available at IACR ePrint Archive, Report 2005/401, November 2005.

[10] Bodei, C., M. Buchholtz, P. Degano, F. Nielson and H. Riis Nielson, *Static validation of security protocols*, Journal of Computer Security **13** (2005), pp. 347–390.

[11] Canetti, R., *Universally composable security: A new paradigm for cryptographic protocols*, in: *42nd IEEE Symposium on Foundations of Computer Science (FOCS)* (2001), pp. 136–145, full version available at IACR ePrint Archive, Report 2000/067.

[12] Canetti, R. and J. Herzog, *Universally composable symbolic analysis of mutual authentication and key-exchange protocols*, in: *Proceedings of the 3nd Theory of Cryptography Conference (TCC)*, Lecture Notes in Computer Science (2006), pp. 380–403, full version available at IACR ePrint Archive, Report 2004/334.

[13] Clark, J. and J. Jacob, *A survey of authentication protocol literature: Version 1.0, 17. nov. 1997*.
URL www.cs.york.ac.uk/~jac/papers/drareview.ps.gz

[14] Cortier, V. and B. Warinschi, *Computationally sound, automated proofs for security protocols*, in: M. Sagiv, editor, *Proceedings of the 14th European Symposium on Programming (ESOP)*, Lecture Notes in Computer Science **3444** (2005), pp. 157–171.

[15] Cremers, C., *Scyther: Automatic verification of security protocols*.
URL http://www.win.tue.nl/~ccremers/scyther/

[16] Dolev, D. and A. C. Yao, *On the security of public-key protocols*, IEEE Transactions on Information Theory **29** (1983), pp. 198–208, preliminary version presented at FOCS'81.

[17] Drielsma, P. H., S. Mödersheim and L. Viganò, *A Formalization of Off-Line Guessing for Security Protocol Analysis*, in: *Proceedings of LPAR'04*, LNAI 3452 (2005), pp. 363–379.

[18] Hamzeh, K., G. Pall, W. Verthein, J. Taarud, W. Little and G. Zorn, *RFC 2637: Point-to-point tunneling protocol* (1999), status: Informational.

[19] Herzog, J., "Computational Soundness for Standard Assumptions of Formal Cryptography," Ph.D. thesis, Massachussets Institute of Technology (2004),

available at `http://theory.lcs.mit.edu/∼jherzog/papers/herzog-phd.pdf`.

[20] Laud, P., *Symmetric encryption in automatic analyses for confidentiality against active adversaries*, in: *Proceedings of the 2004 IEEE Symposium on Security and Privacy (S&P)* (2004), pp. 71–85.

[21] Lowe, G., *Breaking and fixing the Needham-Shroeder public-key protocol using FDR*, in: T. Margaria and B. Steffen, editors, *Proceedings of the 2nd International Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, Lecture Notes in Computer Science **1055** (1996), pp. 147–166.

[22] Meadows, C., *The NRL protocol analyzer: An overview*, Journal of Logic Programming **26** (1996), pp. 113–131.

[23] Micciancio, D. and B. Warinschi, *Soundness of formal encryption in the presence of active adversaries*, in: M. Naor, editor, *Proceedings of the 1st Theory of Cryptography Conference (TCC)*, Lecture Notes in Computer Science **2951** (2004), pp. 133–151.

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

[25] Rusinowitch, M. and M. Turuani, *Protocol insecurity with finite number of sessions is np-complete*, in: *Proceedings of the 14th IEEE Workshop on Computer Security Foundations (CSFW)* (2001), p. 174.

[26] Ryan, P., S. Schneider, M. Goldsmith, G. Lowe and B. Roscoe, *Modelling and analysis of security protocols* (2000).

[27] Schneier, B., Mudge and D. Wagner, *Cryptanalysis of Microsoft's PPTP authentication extensions (MS-CHAPv2)*, in: R. Baumgart, editor, *CQRE*, Lecture Notes in Computer Science **1740** (1999), pp. 192–203.

[28] Song, D., S. Berezin and A. Perrig, *Athena: a novel approach to efficient automatic security protocol analysis*, Journal of Computer Security **9** (2001), pp. 47–74.

[29] Sprenger, C., M. Backes, D. Basin, B. Pfitzmann and M. Waidner, *Cryptographically Sound Theorem Proving*, in: *Proceedings of 19th Computer Science Foundation Workshop (CSFW)*, 2006, to appear.

[30] Viganò, L., *Automated security protocol analysis with the avispa tool*, Electronic Notes in Theoretical Computer Science (Proceedings of the 21st Annual Conference on Mathematical Foundations of Programming Semantics, MFPS XXI) **155** (2006), pp. 61–86.

[31] Warinschi, B., *A Computational Analysis of the Needham-Schroeder(-Lowe) Protocol*, in: *Proceedings of 16th Computer Science Foundation Workshop (CSFW)* (2003), pp. 248–262.

[32] Zorn, G., *RFC 2759: Microsoft PPP CHAP extensions, version 2* (2000), status: Informational.