Formalization of Security Proofs Using PVS in the Dolev-Yao Model*

Rodrigo B. Nogueira[2,3], Anderson C. do Nascimento[2,3], Flávio L.C. de Moura[1,2], and Mauricio Ayala-Rincón[1,2]**

[1] Grupo de Teoria da Computação, Departamentos de Matemática,
[2] Ciência da Computação and
[3] Engenharia Elétrica, Universidade de Brasília, Brasília D.F., Brasil

rodrigo.nogueira@dprf.gov.br,{andclay@ene,flavio@cic,ayala@}unb.br

Abstract. The security analysis of cryptographic protocols is a difficult issue. We can find many examples in the literature of protocols once believed to be secure and later proven to be flawed. The Dolev-Yao model came as a simple and useful framework to study the security of cryptographic protocols. In this study we report on a mechanical verification of the security characterization of a class of protocols in the Dolev-Yao model (two-party cascade protocols) following an algebraic specification approach with the *Prototype Verification System PVS*.

1 Introduction

Motivation. Even assuming perfectly secure cryptographic primitives, the security analysis of cryptographic protocols is a tricky issue. Proofs of security are rather difficult to check and there are many cases reported in the literature of protocols and security proofs which were later proven to be wrong [Mea03]. Automated reasoning and formal methods came up to the scene as a possible way to turn the security analysis of protocols more reliable and less error prone. Arguably, the most popular example of this success is the discovery of a possible attack upon the Needham-Schroeder protocol [NS78]. With the help of formal methods, Gavin Lowe discovered a gap in the Needham-Schroeder protocol after seventeen years of its introduction, a period during which the protocol was assumed correct [Low95]. The protocol was then modified and mechanically proved correct [Low96]. For a survey in the area see [Mea03] and for recent results [ABC+06].

Our Contribution. We show how the proof techniques available in the proof assistant PVS [ORS92, OS97] can be applied to the security analysis of protocols by following an algebraic approach in which instances of applications of steps of the protocols are represented over the monoid freely generated by the cryptographic operators. We illustrate our approach by verifying the security of two-party cascade protocols in the well-known original Dolev-Yao (DY)

* Work supported by the District Federal Research Foundation - FAP-DF 8-004/2007.
** Corresponding author partially supported by the Brazilian Research Council - CNPq.

model [DY83]. The main advantage of the proposed approach is the simplicity of the specification that is close to the original model because it is based on the representation of operators as generators of a monoid and the use of elaborated types and higher-order logic that are available in PVS.

Related Work. There are not so many works applying PVS to the security analysis of cryptographic protocols. In [DS97, ES00] PVS was used to analyze the security of authentication protocols. Our analysis, on the other hand, is restricted to guarantee the property of security of two-party protocols. Even concerning PVS, in [MR00] it is presented a dedicated strategy in order to perform proofs of security protocols, which is based on protocol representation theories on a state-transition model. This contrasts with the simple representation of protocols as sequences of words used in our algebraic approach that we consider more adequate for the treatment of the original DY model. The paper [LHT07] provided a specification and verification in PVS of the intrusion-tolerant protocol enclaves [DCS02]. The problem treated in [DCS02] is of a rather different nature than the ones we address here. [DCS02] deals with a distributed protocol where the protocol goal has to be fulfilled even when a subset of the players are corrupted by a malicious party and can arbitrarily deviate from the protocol specifications (the so-called Byzantine faults). Moreover, [LHT07] is not fully analyzed by using PVS. Its authenticity was treated using the model checker Murphi. Also, [BJ03] reports the use of PVS for formally verifying a system for ordered secure message transmission, but it does not provide the corresponding PVS specification code.

Many work on formalization of security of protocols has been done in other proof assistants [ABC+06], this includes, among others, the remarkable inductive approach by L. Paulson in Isabelle [Pau99] and more recently the `Coq` development `CertiCrypt` which includes probability, complexity and game theoretical techniques in order to verify security [BGZB09]. Recently Benaïssa presented a verification of security of the DY model in `event B` [Ben08]. Although the existence of these works, we believe the current PVS development is of great interest because it improves the scenario of available public libraries for the analysis of security and because it chooses an algebraic straightforward specification of the DY framework, framework that has been widely used to model cryptographic protocols and is known, in some cases, to provide security against all possible adversaries even when we do not consider perfect cryptography (i.e. taking into account complexity theoretical assumptions, asymptotic and probabilities) [Her05]. Thus, we hope to provide security people with a useful set of libraries to simplify the task of obtaining a PVS verification of cryptographic protocols.

Organization of the paper. In section 2 the necessary notions on the Dolev-Yao model and PVS are presented. Sections 3 and 4 illustrate the algebraic approach of specification and formal verification of the characterization of security. Section 5 concludes and presents future work. The complete PVS development is available at `ayala.mat.unb.br/publications.html`.

2 Background and Methodology

2.1 The Dolev-Yao protocol

The DY intruder model (as introduced in [DY83]) is based on the so-called DY rules for public-key encryption. The attacker obtains new information from previous knowledge he/she obtained by *passive* observation of the communication network and by *active* participation (sending new messages according to the protocol rules). The model uses the *perfect cryptography assumption*, which states that the only way to obtain information on a *plaintext* (that is the information which the sender wishes to transmit to the receiver) is by knowing the decryption key, and that it is impossible to obtain this key from the ciphertexts.

In a public-key cryptosystem, every user u (including the attacker) over an enumerable set of users U has both an encryption and a decryption key, E_u and D_u, respectively. A secure public directory contains all the encryption functions $E = \{E_u \mid u \in U\}$ which are known by all users, while the decryption function D_u is known by its own user u only, for all user u in U. D denotes the set $\{D_u \mid u \in U\}$. Encryption and decryption functions are inverse operators for all users u and plaintexts M: $E_u(D_u(M)) = D_u(E_u(M)) = M$. It is assumed that knowing $E_u(M)$ and the public directory E does not reveal anything about M.

We consider two-party protocols in which only the two users who wish to communicate are involved in the transmition process without any assistance of a third party. In addition, all users should follow the same communication rules in which the only possible operators are the encryption and decryption functions under the previous restrictions of use.

Let $\Sigma = E \cup D$ and Σ^* the set of finite words over symbols in Σ where λ denotes the empty word, as usual. For simplicity, instead of working over the algebra of these operators we will work over the monoid freely generated by the symbols in Σ modulo the set of congruences

$$E_u D_u = D_u E_u = \lambda, \forall u \in U \tag{1}$$

In this monoid, for all words $\delta \in \Sigma^*$ there exists a unique *canonical form* $\overline{\delta}$ such that $\delta = \overline{\delta}$ and there are neither subwords in $\overline{\delta}$ of the form $E_u D_u$ nor $D_u E_u$ for any $u \in U$. $\overline{\delta}$ is called the *reduced form* of δ and whenever $\delta = \overline{\delta}$, δ is said to be in reduced form.

For all $\delta \in \Sigma^*$, $|\delta|$ denotes its length and for all $0 \leq j < m$, where $|\delta| = m \in \mathbb{N}$, δ_j denotes the $(j+1)^{th}$ symbol in δ and for $0 \leq j \leq k < m$, δ_{j_k} denotes the subword from the $(j+1)^{th}$ symbol until the $(k+1)^{th}$ symbol of δ. For all $u \in U$, E_u^c and D_u^c denote respectively D_u and E_u. For all $\delta \in \Sigma^*$, δ^c denotes its *complement* defined inductively as: $\lambda^c = \lambda$ and $(G\delta')^c = \delta'^c G^c, \forall G \in \Sigma, \delta' \in \Sigma^*$.

Definition 1 (Two-party cascade protocol). *A two-party cascade protocol, which states how pairs of users should communicate in the communication network, is defined by a nonempty finite sequence $\alpha = \alpha_0, \ldots, \alpha_{n-1}$, where $n \geq 1$, of functions from pairs of users to words in Σ^*, that is, $\alpha_i : U \times U \longrightarrow \Sigma^*, \forall 0 \leq$*

$i < n$. *In addition, it should satisfy the following conditions:*

$$\forall 0 \leq i < n, \forall x, y, u, v \in U \begin{cases} 1. \quad \alpha_i(x,y) \neq \lambda \text{ and is in reduced form} \\ 2.1.\ \alpha_i(x,y) \in \{D_x, E_x, E_y\}^*, \text{ whenever } i \text{ is even} \\ 2.2.\ \alpha_i(x,y) \in \{D_y, E_x, E_y\}^*, \text{ whenever } i \text{ is odd} \\ 3. \quad |\alpha_i(x,y)| = |\alpha_i(u,v)| \\ 4. \quad \forall 0 \leq j < |\alpha_i(x,y)| : \begin{cases} \alpha_i(x,y)_j = E_x \text{ iff } \alpha_i(u,v)_j = E_u \\ \alpha_i(x,y)_j = E_y \text{ iff } \alpha_i(u,v)_j = E_v \\ \alpha_i(x,y)_j = D_x \text{ iff } \alpha_i(u,v)_j = D_u \\ \alpha_i(x,y)_j = D_y \text{ iff } \alpha_i(u,v)_j = D_v \end{cases} \end{cases}$$

A protocol α is applied in the communication between x and $y \in U$ as follows:

1. x sends y the message $\alpha_0(x,y)M$;
2. y answers x the reduced form of $\alpha_1(x,y)$ applied to $\alpha_0(x,y)M$, that is, $\overline{\alpha_1(x,y)\alpha_0(x,y)}M$;
3. x sends y the reduced form of $\alpha_2(x,y)$ applied to $\overline{\alpha_1(x,y)\alpha_0(x,y)}M$, that is, $\overline{\alpha_2(x,y)\overline{\alpha_1(x,y)\alpha_0(x,y)}}M$; ...

The conditions in definition of cascade protocols are explained as follows:

- 2.1 and 2.2 guarantee that when users x and y exchange messages both apply only the allowed deciphering operators: x may apply D_x and y D_y.
- 3 and 4 establish that all pairs of users exchange messages following the same policy; in fact, these necessary conditions guarantee that the images $\alpha_i(x,y)$ and $\alpha_i(u,v)$ are identical except for renamings of user's variables.

Example 1. Consider the protocol $\alpha := \alpha_0, \alpha_1$, where for all $x, y \in U$: $\alpha_0(x,y) \mapsto E_y$ and $\alpha_1(x,y) \mapsto E_x D_y$. Following this protocol, when a would like to send to b the plaintext M, in the first step, a sends b the message $E_b M$ and then, in the second step, b answers a $\overline{E_a D_b E_b}M = E_a M$. Notice that b can find the plaintext M by decoding $E_b M$ and a is able to verify that b actually received the message, because after the second step he/she can decode $E_a M$.

Given a two-party cascade protocol α, a potential saboteur $z \in U$ may use the language of words over $\Gamma_z^\alpha := \Sigma_0(z) \cup \Sigma_1$, where $\Sigma_0(z) := \{D_z\} \cup E$ and $\Sigma_1 := \{\alpha_i(u,v) \mid u \neq v \in U, 0 < i < |\alpha|\}$.
$\Sigma_0(z)$ is the language allowed to z. The motivation for including Σ_1 is that for any z and $u \neq v$

- z may obtain $\alpha_i(u,v)$, for $1 \leq i < |\alpha|$ odd, starting a communication with v, claiming itself to be u. In the i^{th} step of the communication, z sends v any message M obtaining as answer $\alpha_i(u,v)M$, which allows z to apply $\alpha_i(u,v)$ to any chosen M, for i odd.

– z may obtain $\alpha_i(u,v)$, for $2 \leq i < |\alpha|$ even, attending the communication network until u establishes a communication with v. In the i^{th} step of the communication, z supplants v (intercepting the answer v gives to u) and sends u any message M. Then u answers to z $\alpha_i(u,v)M$. This allows z to apply $\alpha_i(u,v)$ to any chosen M, for i even, greater than 0.

Example 2. Consider again the protocol α given in the example 1. A saboteur $z \in U$ can break this protocol easily. In fact, consider z would like to know the plaintext M that a user a sent to b. z will proceed in the following way: firstly, z intercepts $\alpha_0(a,b)M = E_bM$; afterwards, z starts communication with b and sends to b E_bM; finally, following the protocol, b answers to z $\overline{\alpha_1(z,b)E_b}M = E_zM$. Notice that after intercepting the message E_bM and sending it to b, the saboteur z is applying the admissible language of words over Γ_z^α: firstly, $\alpha_1(z,b) = E_zD_b \in \Sigma_1$ and, secondly, to decode E_zM, z uses D_z that belongs to $\Sigma_0(z)$.

Given a protocol α, for $0 \leq i \leq l < |\alpha|$, α_{i_l} denotes the reverse subsequence of functions $\alpha_l, \ldots, \alpha_i$ and, by abuse of notation, $\forall x, y \in U$, $\alpha_{i\_l}(x,y)$ will denote the word $\alpha_l(x,y) \cdots \alpha_i(x,y)$.

Definition 2 (Security). *A protocol is said to be* insecure *whenever for three different users $x, y, z \in U$, there exists $\gamma \in (\Gamma_z^\alpha)^*$ such that for some $0 \leq i < |\alpha|$, $\overline{\gamma\alpha_{0\_i}(x,y)} = \lambda$. Otherwise the protocol is said to be* secure.

Notice that since conditions 3 and 4 in the definition of two-party cascade protocols guarantee that the behavior of the protocol is the same for each user, the definition of security is independent of the triplet of users.

Before presenting the DY characterization of secure protocols, two additional notions are necessary.

1. (*Initial condition*) A protocol α satisfies the **initial condition** whenever the word $\alpha_0(x,y)$ has symbols in E;
2. (*Balancing property*) it satisfies the **balancing property** whenever for all $0 \leq i < |\alpha|$, $u \in \{x,y\}$, D_u occurs in $\alpha_i(x,y)$ implies E_u occurs as well.

The main property to be verified characterizes security.

Characterization of Security. A two-party cascade protocol is *secure* if, and only if it satisfies the *initial condition* and the *balancing property*.

2.2 PVS

The PVS prover used in the verification of the security characterization of two-party cascade protocols is briefly described. PVS consists of a specification language with support tools and a proof assistant, that provides an integrated environment for the development and analysis of formal specifications.

The *specification language* of PVS is built on higher-order logic, which supports modularity by means of parameterized *theories*, with a rich type-system,

including the notions of subtypes and dependent types. It provides a large set of built-in constructs for expressing a variety of notions. The PVS specifications are organized as a collection of *theories*, from which the most relevants are collectively referred as the *prelude*. Each *theory* is composed essentially of *declarations*, which are used to introduce names for types, constants, variables, axioms and formulas, and `IMPORTINGs`, that allow to import another theories. Parameterized theories are very convenient since the use of parameters allows more generic specifications, as we can see with the `finite_sequences_extras` *theory* below which imports the PVS *theory* for manipulation of finite sequences over which the theory of monoids over $D \cup E$ is built.

```
finite_sequences_extras[T: TYPE]: THEORY   ...
IMPORTING finite_sequences[T]  ...
```

`T` is treated as a fixed uninterpreted type. Consequently, when the PVS *theory* `finite_sequences_extras` is invoked by another *theory*, `T` must be instantiated. For example, the *theory* of finite sequences of cryptographic operators `op` is just `finite_sequences_extras[op]`. The verification uses mainly the PVS theories `set` and `finite_sequences`.

An important step in PVS specifications is type-checking the *theory*, which checks for semantic errors, such as undeclared names and ambiguous types. Type-checking may build new files or internal structures such as `TCCs` (type-correctness conditions). These `TCCs` represent *proof obligations* that must be discharged before the *theory* can be considered type-checked, and its proofs may be postponed indefinitely. Although, the *theory* is considered *complete* when all `TCCs` and formulas upon which the proof is dependent have been completed.

The PVS *Prover* provides a variety of commands to construct proofs of the different theorems. It is used interactively in a sequent-style proof representation to display the current proof goal for the proof in progress. The prover maintains a proof tree for the current theorem being proved. The user aims at constructing a complete proof tree, in the sense that all the leaves are recognized as `true`. Each node of the tree is a proof goal that results from the application of a prover command (*rule* or *strategy*) to its parent node. Each proof goal is a sequent consisting of two sequences of formulas called the antecedents and the consequents (logically connected by conjunctions/disjunctions and numbered with negative/-positive integers, respectively).

3 Methodology of specification

The PVS development follows an algebraic approach in which instances of two-party cascade protocols are words specified as finite sequences of operators over the alphabet Σ. The complete development runs in PVS 4.1 and consists of 215 lemmas specified in 1211 lines (55K) and 22876 lines (1.4M) of proofs. PVS build 105 TCCs (type-correctness conditions) whose proofs are included in the latter number. The formalization of the characterization of security *per se* consists of 110 of these lemmas, from which the remaining are TCCs.

The hierarchy of *theories* of the PVS development is presented in Fig. 1.

The specification is built over the prelude *theories* for `finite_sequences` and `sets`. The former was enlarged with some necessary extra properties in the *theory* `finite_sequences_extras` over which the monoid freely generated by the operators in Σ is specified in the *theory* `MonoidCryptOps`.

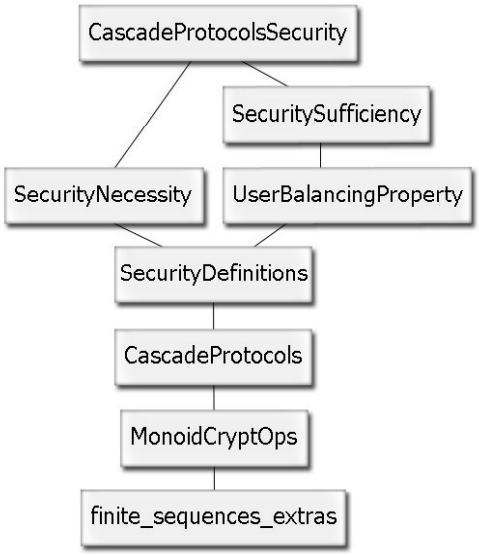

Fig. 1. Hierarchy of the `CascadeProtocolsSecurity` *theory*

The *theory* `MonoidCryptOps`. Words in Σ^* are specified by the type `seqOps : TYPE = finite_sequence[op]`, where `op` is given as the type of pairs of crypt type operators (either encrypt or decrypt) and user over the type `U` of sets of users: `op : TYPE = [# crTyp : cryptType, user : U #]`.

The *theory* `MonoidCryptOps` specifies the recursive function `normalizeseq` that build the canonical forms in Σ^* detecting and eliminating pairs of adjacent opposite operators from left to right. This function uses elements from the *theory* `finite_sequence` such as `o`, operator for concatenation of words, and `seq(l,m)`, that extracts the subsequence from positions `l` to `m` of the sequence `seq`. The boolean `normalseq?` detects whether a sequence of operators has adjacent opposite operators and the function `firstCancPos` that returns the first position, from left to right, of adjunct opposite operators in a reducible sequence `seq`.

```
normalizeseq(seq : seqOps) : RECURSIVE seqOps =
  IF normalseq?(seq) THEN seq
  ELSE LET firstCancPos  = first_cancelable(seq) IN
            IF firstCancPos=0 THEN normalizeseq(seq(2,|seq|-1))
            ELSE normalizeseq(seq(0,firstCancPos - 1) o
                              seq(firstCancPos + 2,|seq|-1))
```

This specific construction of canonical forms implies the necessity of proving lemmas such as idempotence of the normalization: `normalizeseq(seq) = normalizeseq(normalizeseq(seq))` (in standard notation, $\forall \delta \in \Sigma^*, \overline{\delta} = \overline{\overline{\delta}}$) and `normalizeseq(d1 o normalizeseq(d2) o d3) = normalizeseq(d1 o d2 o d3)`, given in Section 2 as the formula $\forall \delta_1, \delta_2, \delta_3 \in \Sigma^*, \overline{\delta_1 \overline{\delta_2} \delta_3} = \overline{\delta_1 \delta_2 \delta_3}$.

Other necessary notions of the theory of monoids modulo the congruences (1), such as *complement* (δ^c, for $\delta \in \Sigma^*$) are specified in the *theory* `MonoidCryptOps`.

The *theory* `cascadeProtocols` specifies the type `welldefined_protocol` of well-defined two-party cascade protocols as finite sequences of functions from pairs of users to sequences of operators that satisfy all conditions of Definition 1. `extract_eN(prot, i, x, y)` builds the application of the $i+1$ first steps of the protocol `prot` for the pair of users `x` and `y`, that was denoted in Section 2 as $\alpha_{0\_i}(x, y)$.

The *theory* `SecurityDefinitions` defines the function `gamma_welldef?` that specifies the language of a potential saboteur for a given protocol, that is written as $\gamma \in (\Gamma_z^\alpha)^*$ in Section 2: `gamma_welldef?(prot,gamma,z)`. Having the admissible language of the saboteur, it is possible to specify the notion of insecurity as the boolean function `insecure_protocol?`:

```
insecure_protocol?(prot, x, y, z | x /= y) : bool =
EXISTS (gamma | gamma_welldef?(prot,gamma,z),
        i : nat | i < |prot|) :
 normalizeseq(extract_gamma(gamma) o extract_eN(prot,i,x,y)) =
                                                    empty_seq
```

In this function `empty_seq` represents the empty sequence λ (cf. Def. 2).

Additionally, the properties that characterize security, i.e., *initial condition* and *balancing property*, are specified, The latter is given as the boolean function

```
balanced_cascade_protocol?(prot) : bool =
  FORALL(x, y | x /= y, i | 0 < i < |prot|) :
     IF even?(i) THEN balanced_wrt?(prot(i),x,y,x)
     ELSE balanced_wrt?(prot(i),y,x,y)
```

In this function, `balanced_wrt?` is a boolean function that checks for quadruplets α_i, x, y, z, where z is either equal to x or to y, and whenever the operator D_z occurs in $\alpha_i(x, y)$, E_z occurs as well.

The former is straightforwardly specified as the boolean function

```
alpha0ContainsE?(prot, x, y) : bool =
  EXISTS(i | 0<= i < |prot(0)(x,y)|):
     member(prot(0)(x,y)(i),{encrypt(x), encrypt(y)})
```

4 Methodology of verification

Induction is exhaustively applied to prove auxiliary lemmas in the *theories* discussed in the previous section, namely, `MonoidCriptOps`, `CascadeProtocols` and `SecurityDefinitions`. For instance the fact that $\forall \delta_1, \delta_2, \delta_3 \in \Sigma^*, \overline{\delta_1 \overline{\delta_2} \delta_3} = \overline{\delta_1 \delta_2 \delta_3}$ is proved by induction on the length of δ_2.

The formalization of the *characterization of security* of two-party cascade protocols is given by proving *necessity* in the *theory* `SecurityNecessity` and sufficiency in the *theory* `SecuritySufficiency` (see Fig. 1).

The *theory* `SecurityNecessity` formalizes the fact that any *secure* protocol satisfies the *initial condition* and the *balancing property*.

```
secProt_imp_alpha0ContainsE : LEMMA
 FORALL (prot, x, y, z | x /= y) :
    secure_protocol?(prot, x, y, z)
     =>
        alpha0ContainsE?(prot, x, y)

secure_impl_balanced : COROLLARY
    FORALL (prot, x, y, z | x /= y AND z /= x AND z /= y) :
       secure_protocol?(prot, x, y, z)
        =>
           balanced_cascade_protocol?(prot)
```

The former is proved by contraposition showing that any well-defined protocol α for which the *initial condition* does not hold, satisfies $\alpha_0(x, y) = D_x^k$, for some $k > 0$. The complement of D_x^k, $(D_x^k)^c = E_x^k \in \Sigma_0(z)^* \subset (\Gamma_z^\alpha)^*$; that is, it belongs to the admissible language of the saboteur.

The latter is also formalized by contraposition, proving that for an unbalanced protocol α there exists $\gamma \in (\Gamma_z^\alpha)^*$ such that $\overline{\gamma \alpha_0(x, y)} = \lambda$. Essentially, the saboteur z uses an unbalanced step of the protocol for cancelling encrypt operators E_x and E_y (lemma `extractable_decUser` formalized in this *theory*). This formalization is done by a series of auxiliary lemmas which express this cancellation. Suppose α_i is an unbalanced step. Then, for any user w, case i is even, $D_w \in \alpha_i(w, z)$, but $E_w \notin \alpha_i(w, z)$ and case i is odd, $D_w \in \alpha_i(z, w)$, but $E_w \notin \alpha_i(z, w)$. In both cases, $\alpha_i(w, z)$ and $\alpha_i(z, w)$ belong to the language $\{D_w, E_z\}^*$ and consequently the saboteur can use words $\tau_1(w)$ and $\tau_2(w)$ in the admissible language (specifically, in $(\Sigma_0(z))^*$), such that either $\overline{\tau_1(w)\alpha_i(z, w)\tau_2(w)} = D_w$ or $\tau_1(w)\alpha_i(w, z)\tau_2(w) = D_w$. The proof is formalized considering all possible cases by an elaborated induction on the length of the unbalanced step. To

conclude, since $\alpha_0(x,y) \in \{E_x, D_x, E_y\}^*$, for some word γ either in $(\{D_z\} \cup \{\tau_1(x)\alpha_i(x,z)\tau_2(x)\} \cup \{\tau_1(y)\alpha_i(y,z)\tau_2(y)\})^*$ or in $(\{D_z\} \cup \{\tau_1(x)\alpha_i(z,x)\tau_2(x)\} \cup \{\tau_1(y)\alpha_i(z,y)\tau_2(y)\})^*$, which in both cases belong to $(\Gamma_z^\alpha)^*$, the admissible language of z, one has $\overline{\gamma\alpha_0(x,y)} = \lambda$.

The *theory* `SecuritySufficiency` formalizes the fact that any protocol that satisfies the *initial condition* and the *balancing property* is secure.

```
alpha0_and_bal_secure : LEMMA
    FORALL (prot, x, y, z | x /= y AND z /= x AND z /= y) :
      alpha0ContainsE?(prot, x, y) AND
      balanced_cascade_protocol?(prot)
           =>
        secure_protocol?(prot, x, y, z)
```

The formalization is by contradiction supposing that for a protocol α both the *initial condition* and the *balancing property* hold, but the protocol is insecure. The existence of $\gamma' \in (\Gamma_z^\alpha)^*$ such that $\exists i, \overline{\gamma'\alpha_{0\_i}(x,y)} = \lambda$ implies the existence of $\gamma \in (\Gamma_z^\alpha)^*$ (namely, either γ' itself, when $i = 0$ or $\gamma'\alpha_{1\_i}(x,y)$, otherwise) such that $\overline{\gamma\alpha_0(x,y)} = \lambda$. As in the analytic proof in [DY83], in the formalization two cases on the operators in $\alpha_0(x,y)$ are considered: either E_y occurs in $\alpha_0(x,y)$ or not. In the former case, $\overline{\gamma}$ will be unbalanced, because it should contain some D_y, but no E_y, in order to be able to cancel the occurrences of E_y in $\alpha_0(x,y)$. Unbalancedness of $\overline{\gamma}$ contradicts the key balancing property on all words γ over the admissible language $(\Gamma_z^\alpha)^*$, that is here formalized in the *theory* `UserBalancingProperty` under specific verifiable assumptions: $\overline{\gamma}$ should be balanced for all users different from z. In the latter case, since $\alpha_0(x,y)$ is in reduced form it should be of the form E_x^k, for some $k > 0$. Again, $\overline{\gamma}$ will be unbalanced, because it should contain some D_x, but no E_x, in order to be able to cancel E_x^k, which gives the contradicition. Consequently, the existence of such γ' is impossible, that implies that the protocol is secure.

5 Conclusions and Future Work

We formalized the characterization of security of two-party cascade protocols in the original DY model following an algebraic methodology in which the steps of a protocol are specified as functions from pairs of users in U into sequences of operators representing words over the monoid freely generated by the symbols in $\{E_u, D_u \mid u \in U\}$ modulo the congruences $E_u D_u = D_u E_u = \lambda, \forall u \in U$. Although the security characterization is a higher-order theorem, most of the efforts in the verification were focused on proving simple properties over finite sequences of these operators and once the necessary properties were proved the formalization of the characterization was supported by proving techniques available in PVS such as propositional techniques (proofs by contradiction, contraposition, etc), first-order logical techniques (skolemization, instantiation, etc) and induction over the structures involved in the specification.

Although the DY model is not used as it was originally published, it is captured by most of nowadays protocol models and consequently straightforward extensions of our higher-order algebraic verification approach over this foundational model can be given in order to analyze in an adequate way extensions of the DY model. Possible extensions include protocol models in which concatenation can be used in several ways, and protocols that allow performing checks on received messages, such as checking that an agent's identity within the message matches the recipient's identity or the claimed sender's identity, among others.

The distinguishing features of the current formalization approach include:

1. the simplicity of monoids in order to specify straightforwardly the algebraic relations between encryption and decryption operators that in contrast with functional approaches avoids inclusion of axioms such as $\forall x \neq y \in U, E_x \neq E_y \& D_x \neq D_y$ (cf. [Ben08] in which the difference among encryption and decryption keys is assured by the injectivity of the functions designing encryption and decryption keys to users) and $\forall x, y \in U, E_x \neq D_y$ (cf. [Ben08] in which and axiom specifying disjunctivity of the set of encryption and decryption operators is necessary).
2. The use of dependent types available in the PVS specification language in order to characterize security and insecurity in terms of universal quantified different users of the system, in contrast to other approaches in which the attacker is axiomatically fixed (cf. [Ben08]).
3. The use of the expressive power of the higher-order language of PVS in order to formalize closely to the original DY analytical approach security characterization properties. This flexibility allows universal quantification of objects such as protocols, that are sequences of functions.

As future work, the logical correctness of more general protocols will be formalized. We are particularly interested in secure two-party protocols and in oblivious transfer [Kil88]. Oblivious transfer is a powerful secure two-party primitive known to imply any other secure two-party functionality. We also aim at extending our current model as to contain information theoretically secure cryptographic protocols based on physical assumptions (such as the existence of noisy channels) [NW08].

References

[ABC+06] A. Armando, D. Basin, J. Cuellar, M. Rusinowitch, and L. Vigano, editors. *Special Issue on Automated Reasoning for Security Protocol Analysis*, volume 36. J. of Automated Reasomning, 2006.

[Ben08] N. Benaïssa. Modelling Attacker's Knowledge for Cascade Cryptographic Protocols. In *ABZ '08: Proc. of the 1st Int. Conf. on Abstract State Machines, B and Z*, volume 5238 of *Lecture Notes in Computer Science*, pages 251–264. Springer, 2008.

[BGZB09] G. Barthe, B. Grégoire, and S. Zanella Béguelin. Formal certification of code-based cryptographic proofs. In *36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages POPL*, pages 90–101, 2009.

[BJ03] M. Backes and C. Jacobi. Cryptographically Sound and Machine-Assisted Verification of Security Protocols. In *20^{th} Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 2607 of *Lecture Notes in Computer Science*, pages 675–686. Springer, 2003.

[DCS02] B. Dutertre, V. Crettaz, and V. Stavridou. Intrusion-tolerant enclaves. In *Proc. of the IEEE Int. Symposium on Security and Privacy*, pages 216–224, 2002.

[DS97] B. Dutertre and S. Schneider. Using a PVS Embedding of CSP to Verify Authentication Protocols. In *Theorem Proving in Higher Order Logics, TPHOL's 97*, volume 1275 of *Lecture Notes in Computer Science*, pages 121–136. Springer, 1997.

[DY83] D. Dolev and A. C. Yao. On the Security of Public Key Protocols. *IEEE. Transactions on Information Theory*, 29(2):198–208, 1983.

[ES00] N. Evans and S. Schneider. Analysing Time Dependent Security Properties in CSP Using PVS. In *6^{th} European Symposium on Research in Computer Security ESORICS*, volume 1895 of *Lecture Notes in Computer Science*, pages 222–237. Springer, 2000.

[Her05] J. Herzog. A computational interpretation of Dolev-Yao adversaries. *Theoretical Computer Science*, 340:57–81, 2005.

[Kil88] J. Kilian. Founding Cryptography on Oblivious Transfer. In *20^{th} Annual ACM Symposium on Theory of Computing STOC*, pages 20–31. ACM Press, 1988.

[LHT07] M. Layouni, J. Hoofman, and S. Tahar. Formal Specification and Verification of the Intrusion-Tolerant Enclaves Protocol. *Int. Journal of Network Security*, 5(3):288–298, 2007.

[Low95] G. Lowe. An Attack on the Needham-Schroeder Public-Key Authentication Protocol. *Information Processing Letters*, 56(3):131–133, 1995.

[Low96] G. Lowe. Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR. *Software - Concepts and Tools*, 17(3):93–102, 1996.

[Mea03] C. Meadows. Methods for Cryptographic Protocol Analysis: Emerging Issues and Trends. *IEEE J. on Selected Areas in Communications*, 21(1):44–54, 2003.

[MR00] J. K. Millen and H. Rueß. Protocol-independent secrecy. In *IEEE Symposium on Security and Privacy*, pages 110–209, 2000.

[NS78] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Comm. of the ACM*, 21:993–999, 1978.

[NW08] A. Nascimento and A. Winter. On the Oblivious Transfer Capacity of Noisy Resources. *IEEE Trans. on Information Theory*, 54(6):2572–81, 2008.

[ORS92] S. Owre, J. M. Rushby, and N. Shankar. PVS: A Prototype Verification System. In *11^{th} Int. Conf. on Automated Deduction CADE*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752. Springer, 1992.

[OS97] S. Owre and N. Shankar. The formal semantics of PVS. Technical report, SRI-CSL-97-2, Computer Science Laboratory, SRI Int., Menlo Park, CA, August 1997. Available at http://pvs.csl.sri.com/.

[Pau99] L. C. Paulson. Proving Security Protocols Correct. In *14^{th} Annual IEEE Symposium on Logic in Computer Science LICS*, pages 370–383, 1999.