Formalization in PVS of Balancing Properties Necessary for the Security of the Dolev-Yao Cascade Protocol Model

Yuri Santos Rêgo[1] and Mauricio Ayala-Rincón[1,2*]
Departments of [1]Matematics and [2]Computer Science
Universidade de Brasília, 70910-900 Brasília D.F., Brazil
e-mail: `ayala@unb.br`

February 22, 2012

Abstract

Nowadays, formalizing computationally the security of cryptographic protocols is a highly sophisticated task of great relevance. In this work, we present an algebraic approach for modeling the two-party cascade protocol of Dolev-Yao in the specification language of the *Prototype Verification System* PVS. Although cascade protocols could be argued to be a very limited model, it should be stressed here that they are the basis of more sophisticated protocols of great applicability such as those which allow treatment of multiparty, tuples, nonces, stamps, signatures, etc. Thus, obtaining a complete computational formalization of this basic model is in fact a non trivial task of great scientific and practical interest as a first fundamental and necessary step towards the complete formalization of security, authenticity and other relevant properties of more elaborated classes of protocols. In the current algebraic approach, steps of the protocol are modeled in a monoid freely generated by the cryptographic operators. Words in this monoid are specified as finite sequences and the whole protocol is a finite sequence of (sequences) protocol steps. In previous work,

*Corresponding author.

assuming that for *balanced protocols* admissible words produced by a potential intruder should be balanced, a formalization of the characterization of security of this kind of protocols was given in PVS. In this work the previously assumed property is also formalized obtaining in this way a more robust mechanical proof which mathematically guarantees the security of these protocols. Although this property is relatively easy to be specified, when several aspects related with the data structures applied are axiomatized (i.e., assumed without proofs), its proof requires an exhaustive effort, because several algebraic properties should be formalized in the underlying data structure of finite sequences used in the specification. The difficulties presented in this formalization are presented in detail in this work.

1 Introduction

Motivation and proposal

A diversity of cryptographic protocols that are based on the seminal Dolev-Yao (DY) model are currently applied in computational systems. Rules stablished by these protocols are usually guided by a given algorithm implemented in software or hardware and applied in order to preserve information in several manners. Although the programing techniques used in this kind of development are of high quality in general, formal mathematical and logical analysis is necessary in several steps of this algorithmic development in order to guarantee that the implemented protocol is in fact secure and efficient. In a broader context, 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. Perhaps, 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].

In this work, a formal approach to certify security of cryptographic systems is applied in order to proof that the DY two-party cascade protocol model [DY83] is in fact secure, whenever the conditions of security are fulfilled. Elements of the DY model conform the basis of a great number of more elaborated useful protocols and because of this, a great deal of the effort necessary to formalize their security consists in a full formalization of the security of the DY basic model as has been done in the current work. The current presentation focuses on the difficulties inherent to the proof of algebraic properties reflected in the selected data structures used to represent protocols. The complexity of the formalization of these basic properties at the level of granularity of the selected data structures is higher than the one of the proofs of the security properties of the protocols, from the logical point of view.

In previous work [NdMNAR10], the general lines of the formalization were presented, but several facts related with properties of words and finite sequences were assumed. In that work, the characterization of security of the protocol was formalized; that is, security holds whenever two conditions can be guaranteed:

- firstly, an *initial condition* that forces the existence of encryption operators in the first step of the protocol and,
- secondly, a *balancing property* that holds for each step of the protocol.

The latter property, essentially forces occurrences of decryption operators for each user for which at least an encryption operator occurs in any step of the protocol. The security is a consequence of the fact that following this balancing discipline in the construction of protocols the admissible language used by any potential intruder will be also balanced, which makes it impossible the isolation of the decryption operators from the whole language allowed to the intruder. Axiomatization of this fact makes it possible the formalization of the characterization of security of cascade protocols in the deductive language of the proof assistant PVS ([OS97]), as presented in [NdMNAR10], but in order to obtain a *complete* formalization no assumptions are admissible, which makes it necessary the proof of an exhaustive series of basic lemmas related with the balancing property of the admissible language, both

- in the context of this algebraic approach of specification of the protocols (as words in a monoid freely generated by the cryptographic operators modulo the congruence of inversion of the respective encryption-decryption operators) and

- in the level of granularity of the data structure of finite sequences used to represent words of this monoid (that is, protocol steps or words of the admissible language of the intruder) and finite sequences of words (that is, protocols and sequences of words extracted by the intruder).

Related work

In addition to [NdMNAR10], other works apply PVS to check properties of cryptographic protocols. In [DS97, ES00] PVS was used to analyze the security of authentication protocols. 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 the current algebraic approach that can be considered more adequate for the treatment of the original DY model. In [CMR01] the inductive engine of PVS has been used in order to develop a methodology of proof of inductiveness of secrecy and authenticity properties. That methodology is sound but incomplete failing to proof secrecy of secure protocols; in fact, secrecy is known to be a undecidable property. The paper [LHT07] provided a specification and verification in PVS of the intrusion-tolerant protocol enclaves [DCS02]. That work 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]. More 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.

Organization

Section 2 presents the algebraic approach used to model the DY model and analytic sketches of the proofs. Section 3 presents details of the formalization of some of the key lemmas involved in the PVS development. Section 4 concludes and presents future work. As usual, in papers related with formalizations, in the onymous version of this work the whole development in PVS will be made available through a link to the author's institution web page.

2 The Dolev-Yao Model and its Security Characterization

The model is based on a system of public key cryptography in which each user $u \in U$, where U is a finite set of users, owns an encryption operator E_u and a decryption operator D_u. A public secure directory includes all pairs (u, E_u), but $\forall u \in U$, only u has knowledge about D_u. Suppose that the malicious users belong also to the set of users U of the system, and that they can obtain information through passive observation or active interaction in the communication net.

Encryption and decryption operators are algebraically inverse operators for each user: $\forall u \in U$, $E_u D_u = D_u E_u = \lambda$, where λ denotes the empty word. Users interchange arbitrary information that is codified and decoded through *encryption* and *decryption* operators. Thus, this can be modeled as the monoid freely generated by the cryptographic operators. In this structure only strings built with the generators should be considered.

Let $\Sigma = E \cup D = \{E_u \mid u \in U\} \cup \{D_u \mid u \in U\}$, and Σ^* the set of all finite strings over the alphabet of symbols in Σ. $\forall \gamma \in \Sigma^*$, $|\gamma|$ denotes the length of the string γ, and $\forall i$ such that $0 \le i < |\gamma|$, γ_i denotes the $(i+1)^{th}$ symbol (operator) of the string γ. Also, $\forall i, j$ such that $0 \le i \le j < |\gamma|$, $\gamma_{i,j}$ denotes the substring of γ from the $(i+1)^{th}$ until the $(j+1)^{th}$ symbol.

An arbitrary cryptographic operator will be denoted as O, and when necessary as O_u, in order to refer to its user $u \in U$. The opposite cryptographic operator of O_u is denoted as O_u^c. Thus, $E_u^c = D_u$ and $D_u^c = E_u$.

Whenever $\sigma \in \Sigma^*$ has some substring of the form $E_u D_u$ or $D_u E_u$, that is either $\sigma = \sigma' E_u D_u \sigma''$ or $\sigma = \sigma' D_u E_u \sigma''$, for some $u \in U$, $\sigma', \sigma'' \in \Sigma^*$, it is said that σ can be normalized with respect to u, and one denotes as $\overline{\sigma}^u$ the normalization of σ with respect to u, and as $\overline{\sigma}$ the normalization of σ with respect to all users. Thus, normalization means the repeatedly elimination from each word of all pairs of contiguous cancelable operators, according to the congruence of the monoid: $\forall u \in U$, $E_u D_u = D_u E_u = \lambda$. From the algebraic point of view, in this quotient monoid it is only necessary to work with normal or canonical forms, but in the context of the specification, that is the same of cryptography, discrimination between different representations of words or sequences of operators in the same equivalence class is essential.

Definition 1 (Two-party Cascade Protocol)**.** *A two-party cascade protocol determines how users in a communication net should communicate and consists of a finite and non empty sequence of functions from pairs of different users into sequences of operators, $\alpha = \alpha_{n-1}\alpha_{n-2}\cdots\alpha_2\alpha_1\alpha_0$, where $n \geq 1$, and $\alpha_i : U \times U \to \Sigma^*$, $\forall i$ such that $0 \leq i < n$. Additionally, $\forall i$ such that $0 \leq i < n$, $\forall x, y, u, v \in U$, the following constraints hold:*

i. $\alpha_i(x,y) \neq \lambda$ *and is normalized;*

ii. $\alpha_i(x,y) \in \{E_x, D_x, E_y\}^*$ *if i is even;*

iii. $\alpha_i(x,y) \in \{E_y, D_y, E_x\}^*$ *if i is odd;*

iv. $|\alpha_i(x,y)| = |\alpha_i(u,v)|$;

v. $\forall\ 0 \leq j < |\alpha_i(x,y)|$:

v.1) $(\alpha_i(x,y))_j = E_x \iff (\alpha_i(u,v))_j = E_u$;

v.2) $(\alpha_i(x,y))_j = E_y \iff (\alpha_i(u,v))_j = E_v$;

v.3) $(\alpha_i(x,y))_j = D_x \iff (\alpha_i(u,v))_j = D_u$;

v.4) $(\alpha_i(x,y))_j = D_y \iff (\alpha_i(u,v))_j = D_v$.

Given $x, y \in U$ and M, the communication is done in the following manner:

x sends a message to y following the first step of the protocol, $\alpha_0(x,y)M$;

y answers to x with $\overline{\alpha_1(x,y)\alpha_0(x,y)}M$;

x answers to y with $\overline{\alpha_2(x,y)\overline{\alpha_1(x,y)\alpha_0(x,y)}}M$,

and so on.

Following the rules of a given protocol α, two users in communication x and $y \in U$, the constraints **ii)** and **iii)** basically restrict the use of the decryption operator D_x and D_y, respectively, to the user sending the message in each step of the protocol (even steps for x and odd for y). Constraints **iv)** and **v)** guarantee that the protocol has exactly the same behavior for each pair of users.

2.1 Security Characterization of Cascade Protocols

The following items characterize the admissible language of a possible saboteur in a communication net in which the users follow a well-defined cascade protocol α.

Let x, y and $z \in U$, where z is a possible saboteur. z can force applications of any step α_i of the protocol α, for $i > 0$, twofold: either supplanting x in order to obtain answers from y (odd steps of the protocol) or intercepting an eventual communication started between x and y and supplanting y in order to obtain answers from x (even steps of the protocol). This is described in detail in the two items presented below.

1. z can obtain $\alpha_i(x,y)$, for all $1 \leq i < |\alpha|$ odd, starting a communication with y supplanting x. In the i^{th} step of the communication, z sends to y any message M, obtaining as answer $\alpha_i(x,y)M$ (since y is following the protocol). This allows z to apply $\alpha_i(x,y)$ to any selected message M, for i odd;

2. z can obtain $\alpha_i(x,y)$, for all $2 \leq i < |\alpha|$ even, observing passively the net and waiting until the moment in that x establishes communication with y. Then, in the $i-1^{th}$ step of the communication, z intercepts the answer from y to x and replaces it sending to x any selected message M. Thus, x answers to z $\alpha_i(x,y)M$, allowing z to apply $\alpha_i(x,y)$ to any selected M, for $|\alpha| > i \geq 2$ even.

In addition to the two previous tricks, a potential saboteur z can use the language of words of the monoid generated by all the encryption operators and its own decryption operator.

3. Since z is a user of the communication net, he can use the language generated by the admissible alphabet $\Sigma_0(z) := E \cup D_z$.

Definition 2 (Admissible Language)**.** *Given a well-defined cascade protocol α and denoting the set of words, related to the first and second items above, as $\Sigma_1 := \{\alpha_i(x, y) \mid x, y \in U,\ x \neq y,\ 0 < i < |\alpha|\}$, one defines the* **admissible language** *of a possible saboteur z as*

$$\mathcal{AL}(z) := (\Sigma_0(z) \cup \Sigma_1)^*$$

Definition 3 (Insecure/Secure Protocol)**.** *Consider a well-defined two-party cascade protocol given as $\alpha = \alpha_{n-1} \cdots \alpha_1 \alpha_0,\ \ n \geq 1$, and let $x, y, z \in U$ be different users. The protocol is said to be insecure if $\exists \gamma \in \mathcal{AL}(z)$ such that, for some $0 < j < n$*

$$\overline{\gamma(\alpha_{j-1}(x, y) \cdots \alpha_0(x, y))} = \lambda$$

Otherwise the protocol is said to be secure.

2.2 Characterization of the Security of Cascade Protocols

An initial condition and a balancing property characterize security of two-party cascade protocols.

Definition 4 (Security Initial Condition - **IC**)**.** *A cascade protocol satisfies the initial condition (IC) if $\forall x, y \in U,\ \exists i,\ 0 \leq i < |\alpha_0(x, y)|$ such that $(\alpha_0(x, y))_i = E_u$, where $u \in \{x, y\}$; i.e., if the initial step of the protocol includes at least an encryption operator.*

Definition 5 (Balanced Word - **BP**)**.** *A word $\sigma \in \Sigma^*$ owns the balancing property (BP) with respect to a user $u \in U$ if, whenever there is some i, $0 \leq i < |\sigma|$ such that $\sigma_i = D_u$, there is j, $0 \leq j < |\sigma|$, such that $\sigma_j = E_u$.*

Definition 6 (Balanced Protocol)**.** *A cascade protocol α is said to be balanced if, $\forall x, y \in U,\ \forall\ |\alpha| > i \geq 0,\ \alpha_i(x, y)$ satisfies BP with respect to x if i is even and with respect to y if i is odd. In other words, for any step of the protocol, if it has a decryption operator, then it has an encryption operator, both for the same user.*

The following lemma, perhaps the most difficult or at least the most elaborated part of the analytic theory of the DY model, deals with the balancing property relative to words in the admissible language of an intruder z, $\mathcal{AL}(z)$. This lemma simplifies the proof of the Theorem 1 of characterization of security, presented at the end of this section, because it encapsulates the subjacent technicalities involved in its formalization.

Lemma 1 (BP for Normalized Words of the Admissible Language of Balanced Protocols)**.** *Given a balanced cascade protocol α and $z \in U$. Then, $\forall \eta \in \mathcal{AL}(z)$, $\overline{\eta}$ satisfies BP with respect to* ***all*** *users $a \in U,\ a \neq z$.*

Lemma 1, was only axiomatized (i.e., assumed without proof) in [NdMNAR10] in order to present a formal proof of the Theorem 1 of characterization of security, and its current formalization depends on Lemmas 9, 10 and 11 (according to the original numeration in [DY83]), that will be presented in the sequel. Lemma 9 is necessary in order to prove Lemma 10 and both the latter lemma and Lemma 11 are necessary in order to conclude the proof of Lemma 1. In the formalization of these Lemmas two additional definitions related with the balancing property relative to a specific user are necessary.

Definition 7 (Word User-Balanced)**.** *Let $a \in U$ and $\pi \in \Sigma^*$. π is said to be a-balanced if the following implication is true: $\pi = D_x \delta D_y$, for some $x, y \in U, x \neq a \neq y$ and $\overline{\delta}^a \cap D \subseteq \{D_a\}$ imply that $\overline{\delta}^a$ is balanced with respect to a.*

Definition 8 (Linkage Property - **LP**)**.** *Let $z \in U$ and $\eta \in \Sigma^*$. η is said to satisfy the linkage property (w.r.t. z) if for any π subword of the word $D_z \eta D_z$, π is a-balanced, $\forall a \in U, a \neq z$.*

The analysis of whether a word is balanced is reduced to the verification of the balancing property for all its subwords in which only decryption operators for a unique user happens, that is done through the verification of the linkage property.

Lemma 9 (Linkage Property)**.** *Let μ and $\eta \in \Sigma^*$ words that satisfy LP w.r.t. z. $\eta\mu$ satisfies LP w.r.t. z.*

Lemma 10 (Linkage Property for the Admissible Language)**.** *Consider a balanced cascade protocol α, $z \in U$ and let $\eta \in \mathcal{AL}(z)$, then η satisfies LP w.r.t. z.*

Lemma 11 (Normal Forms Preserve Linkage Property). *Let $\eta \in \Sigma^*$ such that η satisfies LP. Then $\overline{\eta}$ satisfies LP too.*

Sketches of analytical proofs of the previous three lemmas are available in [DY83], but their formalizations require an exhaustive series of proofs of mundane properties of the data structures being used in order to represent protocols. The proof of Lemma 10 is by induction on the inductive construction of the admissible language $((\Sigma_0(z) \cup \Sigma_1)^*)$ and depends on proving that words in $\Sigma_0(z) \cup \Sigma_1$ satisfy LP (basis of the induction) and application of Lemma 9 in the inductive step. The proof of Lemma 11 is also done by induction, in this case on the number of recursive steps applied in the normalization of the word η. In the induction basis, it is proved that after eliminating from η the first, from left to right, occurrence of contiguous opposite operators, either $D_u E_u$ or $E_u D_u$, for some $u \in U$, the resulting word satisfies LP. In the inductive step this argumentation is applied once again.

In PVS, the formalization of Lemma 11 depends on the specification of the notion of normalization that is given basically through two specified functions presented below. The function `first_cancelable` takes as argument a reducible sequence and detects the first contiguous occurrence of opposite operators. The second function, `normalizeseq` uses the first function in order to detect recursively the first occurrence of contiguous opposite operators and eliminate them from the sequence.

```
first_cancelable(seq : reducibleseq) : RECURSIVE nat =
 IF   areopcomplements?(seq(0),seq(1)) THEN 0
 ELSE 1 + first_cancelable(^(seq,(1,seq‘length-1)))
 ENDIF
 MEASURE seq‘length-1

normalizeseq(seq : seqOps) : RECURSIVE seqOps =
 IF normalseq?(seq) THEN seq
 ELSE LET (firstCancPos : nat) = first_cancelable(seq) IN
  IF firstCancPos=0 THEN normalizeseq(seq^(2,seq‘length-1))
  ELSE normalizeseq(seq^(0,firstCancPos-1) o
           seq^(firstCancPos+2,seq‘length-1))
  ENDIF
 ENDIF
 MEASURE seq‘length
```

Several decisions taken during the specification are relevant in order to obtain a full formalization of the main lemmas. Observing the function `fist_cancelable`, one notices that the input sequence `seq` is indexed from 0 to its length minus one (`seq`length - 1`). The function `areopcomplements?` checks whether two operators either are opposite or not. The type of the parameter of the function `first_cancelable` is the type of reducible sequences that is a subtype of the type of sequences of operators (`SeqOps`) as used for the parameter of the function `normalizeseq`. The operators "ˆ" and "o" denote respectively, subsequences and concatenation or append of sequences. Thus, `(seq^(0,firstCancPos-1) o seq^(firstCancPos+2,seq`length-1)` denotes the sequence obtained by eliminating the operators at positions `firstCancPos` and `firstCancPos + 1` of the sequence `seq`, that, in other words, is the sequence obtained by eliminating the first contiguous occurrence of opposite operators in `seq`, since `firstCancPos` was set as the position of the first cancelable contiguous occurrence.

The next theorem, whose proof depends on Lemma 1, characterizes security of two-party cascade protocols.

Theorem 1 (Characterization of Security of Cascade Protocols ([DY83])). *A two-party cascade protocol is secure if and only if*

- *it satisfies the initial condition and*
- *is balanced.*

The proof of Theorem 1 is divided in the proof of necessity and the proof of sufficiency:

> The former, that is to prove that a secure protocol satisfies the initial condition and should be balanced, is obtained by contrapositive argumentation: if a protocol does not satisfy the initial condition or is not balanced it is proved to be insecure.
>
> The latter is proved by contradiction. Let $x, y \in U$, and suppose that x starts communication with y. Suppose, by reduction to the absurd, that the protocol satisfies IC and is balanced, but it is insecure; thus, there exists $\gamma \in \mathcal{AL}(z)$, such that $\overline{\gamma\alpha_0(x,y)} = \lambda$. A separation in two cases is then possible: in the first case, E_y appears in $\alpha_0(x,y)$, and then it is possible to show, that $\overline{\gamma}$ is not balanced, because it should contain an operator D_y (since D_y does not occur in $\alpha_0(x,y)$), which contradicts Lemma 1; in the second case, E_y does not appear in $\alpha_0(x,y)$, which implies that D_x also does not occur in the first step of the protocol and consequently Lemma 1 is contradicted again.

3 Formalization in PVS

In this section, several deductive techniques applied in the formalization of the balancing properties of the DY model are presented. The formalization is available in the PVS files. Here only a brief description is possible focusing on the most relevant aspects.

The Prototype Verification System PVS is a higher order proof assistant with an elaborated type system in which subtyping and dependent types are allowed. In a higher order logic language, as the one of the specification language of PVS, one can quantify relational variables. This makes straightforward the specification of properties of two-party cascade protocols that are sequences of relational (functional) objects according to Definition 1. For instance, sufficiency lemma related to the proof of Theorem 1 is specified as the lemma below, in which a well-defined protocol `prot` is universally quantified. Also dependent types are used in order to quantify triplets of users `x, y` and `z` such that are mutually different.

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

The basic data structures used in the formalization are `finite_sequences` and `sets` that are available in the *prelude* theory of PVS[OS97]. The whole hierarchy of the formalization is presented in Fig. 1. The main theory, named `CascadeProtocolsSecurity`, contains the specification of Theorem 1 and imports subtheories for the formalization of sufficiency and necessity, respectively, `SecurityNecessity` and `SecuritySufficiency`. The focus in this paper, is on the formalization of lemmas related with balancing properties (Lemmas 9, 10 and 11), which are formalized inside the subtheories `UserBalancingProperties` and `UserMonoidCryptOps`.

The subtheory `finite_sequences_extras` imports the prelude theory for finite sequences and includes additional necessary lemmas and properties about this data structure as well as about finite sets that are not available in the PVS *prelude* library. The subtheory `MonoidCryptOps` includes

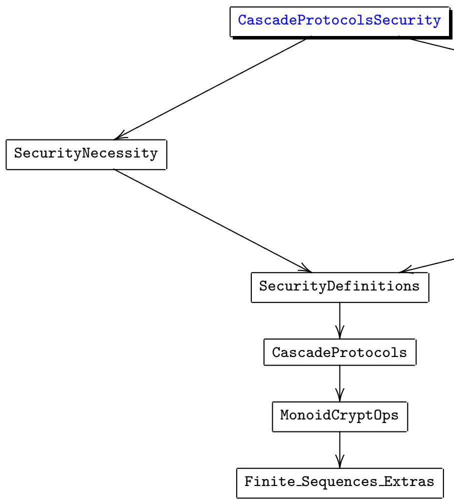

Figure 1: Hierarchy of theories and subtheories — formalization of security of cascade protocols

general specifications about the cryptographic operators and the theory of monoids freely generated by the language of cryptographic operators modulo the congruence given by elimination of opposite operators. In this theory, the notion of normal form is given. The subtheory `CascadeProtocols` formalizes the basic notions about two-party cascade protocols. The subtheory `SecurityDefinitions` formalizes the notions of security of cascade protocols. The subtheory `UserMonoidCryptOps` includes specifications and formalizations about properties of sequences of cryptographic operators relative to specific users. In this subtheory, notions similar to the ones given in `MonoidCryptOps` are specified; for instance, the notion of normal form relative to an specific user is given. These relativizations are necessary in order to deal with notions such as user balanced and linkage property (e.g., Defs. 7 and 8) among others, that are necessary to formalize the central balancing property (Lemma 1) necessary in the proof of sufficiency of the main security characterization theorem.

As previously mentioned, the crucial part of the theory is included in the subtheories `SecurityNecessity` and `SecuritySufficiency` formalizing necessity and sufficiency of Theorem 1. Here, the focus is on the balancing properties necessary for the sufficiency that are formalized in the subtheories `UserBalancingProperty` and `UserMonoidCryptOps` containing, the former, the formalizations of Lemmas 1, 9, 10 and 11, and the latter, specifications of properties of the theory of monoids relative to specific users.

3.1 Verification of Balancing Lemma 1

Assuming Lemmas 9, 10 and 11, Lemma 1 was formalized following the analytic proof in [DY83] essentially, but it was detected the necessity of an additional technical property in order to guarantee the integrity of the formalization: for any word of the form $D_z \eta D_z$, where $\eta \in \Sigma^*$, if for some $a \in U$, D_a occurs in η, then there exists a subsequence of $D_z \eta D_z$ of the form $D_x \delta D_y$ containing the occurrence of D_a and such that $x \neq a \neq y$. Analytically, this property is very simple but technically its formalization is non trivial. This kind of mundane properties are recurrent in the formalization and represent a great deal of the whole formalization effort.

Lemma 1 was formalized inside the subtheory `UserBalancingProperty`, and its specification is given as below.

```
userBalancing : LEMMA
  FORALL (prot : welldefined_protocol,
              z : U,
          gamma : gammaT | gamma_welldef?(prot,gamma, z),
              w : U | w /= z) :
 balanced_cascade_protocol?(prot) =>
   balancedseq_wrt?(normalizeseq(extract_gamma(gamma)), w)
```

This PVS lemma specifies the following: let `prot` be a well-defined protocol, $z, w \in U$ be different users and $\gamma \in \mathcal{AL}(z)$ a word in the admissible language of the protocol `prot` for the intruder z. Thus, if `prot` is a balanced protocol, then the normalization of γ, $\overline{\gamma}$, is balanced with respect to the user w. In other words, for all users different from a possible saboteur z, the normalization of any admissible word is balanced with respect to the other users.

In the specification above, `gammaT` represents the type of finite sequences of allowed strings (finite sequences of operators) for the model of protocols under consideration; `gama_welldef?` is a tertiary relation that expresses the fact that the finte sequence of words `gamma` is a sequence of words either in $\Sigma_0(z)$ or Σ_1 (according to the protocol `prot`), that is, the concatenation of words in `gamma` belongs to the admissible language of the intruder z. `balanced_cascade_protocol?` and `balancedseq_wrt?` are *boolean* unary and binary relations, respectively, for balanced protocols and words balanced with respect to a user. The function `extract_gamma` builds the word of concatenation of words in the finite sequence `gamma`, that is a word in $\mathcal{AL}(z)$, the admissible language for the intruder z according to the protocol `prot`. `normalizeseq`, as previously mentioned, recursively builds the normalization of the input word according to the congruence of the monoid, eliminating all contiguous opposite operators.

The proof of Lemma 1 is done applying Lemmas 10 and 11 as follows:

> Let $\eta \in \mathcal{AL}(z)$ be a word in the admissible language. Suppose, by contradiction, that for some $a \in U$, $\overline{\eta}$ does not satisfy the balancing property with respect to a. Thus, D_a occurs in $\overline{\eta}$, but E_a does not, which implies that $\overline{\eta}$ does not satisfy the linkage property. This contradicts Lemmas 10 and 11, since in first place, η satisfies the linkage property because it is an admissible word built from a balanced protocol and, in second place, normalizations of words that satisfy the linkage property preserve this property.

The language of proof of PVS follows the Gentzen sequent style. PVS uses an interactive proof language in which inference rules of the sequent calculus are applied by means of proof commands. The proof is started by a sequent containing as antecedents the premisses of the conjecture or objective to be proved and as succedents the conclusion of the conjecture to be proved. Proof commands should be applied until the proof is concluded or until one detects errors in the conjecture. Proofs are stored in a file of proof commands.

The command `prove` starts the proof of some selected objective. This is illustrated below for the Lemma 1. Items above the symbol |------- represent the antecedents or premisses of the sequent, and items below this symbol, the succedents or conclusions.

```
|-------
[1] FORALL (prot : welldefined_protocol, z : U,
           gamma : gammaT | gamma_welldef?(prot,gamma, z),
               w : U | w /= z) :
  balanced_cascade_protocol?(prot) =>
     balancedseq_wrt?(normalizeseq(extract_gamma(gamma)), w)
```

The succedent starts exactly as Lemma 1. By an application of the proof command of Skolemization one obtains the following sequent.

```
{-1} balanced_cascade_protocol?(prot)
|-------
{1}balancedseq_wrt?(normalizeseq(extract_gamma(gamma)),w)
```

As antecedent or premise one has that `prot` is a balanced protocol (that is well-defined), and one should prove that the normalization of `extract_gamma(gamma)`, that is `normalizeseq(extract_gamma(gamma))`, named `reducedGamma` below, is balanced with respect to w. Lemmas 10 and 11 can be invoked applying the PVS proof command `lemma`. The application of this command includes as new premisses of the sequent the selected lemma and it can be instantiated according to the objective being proved.

```
{-1} linkage_property?(extract_gamma(gamma), z) =>
                              linkage_property?(reducedGamma, z)
[-2] balanced_cascade_protocol?(prot) =>
                       linkage_property?(extract_gamma(gamma), z)
[-3] balanced_cascade_protocol?(prot)
|-------
{1} balancedseq_wrt?(reducedGamma, w)
```

At this point, according to the analytic proof previously explained, the conclusion appears trivial, but it involves technicalities as those mentioned. In fact, the development of the proof is exhaustive and several properties related with finite sequences and the theory of monoids are necessary. More than ten additional technical lemmas were necessary in order to conclude the proof of this lemma. The proof of this lemma uses more than 160 proof commands.

3.2 Verification of LP for the Admissible Language (Lemma 10)

Lemma 10 about linkage property for the admissible language is specified as

```
  balanced_prot_imp_linkage_in_sigmas : LEMMA
    FORALL(prot:  welldefined_protocol, z : U,
           eta : gammaT | gamma_welldef?(prot,eta, z)) :
      balanced_cascade_protocol?(prot) =>
                   linkage_property?(extract_gamma(eta), z)
```

This states that given a balanced cascade protocol `prot` and a user $z \in U$, all words of the admissible language $\mathcal{AL}(z)$, built in the specification as `extract_gamma(eta)`, satisfy the linkage property.

The proof is started by applying the command `prove` obtaining the initial sequent below.

```
  |-------
{1} FORALL (prot: welldefined_protocol, z: U,
             eta: gammaT | gamma_welldef?(prot, eta, z)):
      balanced_cascade_protocol?(prot) =>
         linkage_property?(extract_gamma(eta), z)
```

The proof is by induction in the length of the sequence `eta`. This method is selected by applying the proof command `(measure-induct+ "eta`length" ("eta"))` to which PVS returns the following sequent having as first premise the inductive hypothesis, that is for any sequence with length less than the length of the initial sequence, called now `x!1`, it satisfies the linkage property.

```
{-1} FORALL (y: {eta: gammaT | gamma_welldef?(prot, eta, z)}):
       y‘length < x!1‘length =>
         balanced_cascade_protocol?(prot) =>
          linkage_property?(extract_gamma(y), z)
{-2} balanced_cascade_protocol?(prot)
  |-------
{1} linkage_property?(extract_gamma(x!1), z)
```

Analytically, it is enough to apply induction and Lemma 9, but some specificities of the data structure should be considered. The case in which the length of `x!1` is zero, is proved easily. Now, if `x!1` has length equal to one some considerations are necessary. Supposing that `x!1`length = 1`, one applies an auxiliary lemma called `admissible_language_sat_link_property`, that states that all words of the language $\Sigma_0(z)$ or Σ_1 satisfy the linkage property. Invoking this auxiliary lemma gives the sequent below.

```
{-1}  FORALL (prot: welldefined_protocol, z: U, delta: seqOps):
        (balanced_cascade_protocol?(prot) AND
          ( member(delta, sigma2_3(prot)) OR
    wellDefInSigma1?(delta, z)) )
         => linkage_property?(delta, z)
[-2]  x!1‘length = 1
[-3]  FORALL (y: {eta: gammaT | gamma_welldef?(prot, eta, z)}):
        y‘length < x!1‘length =>
         balanced_cascade_protocol?(prot) =>
          linkage_property?(extract_gamma(y), z)
[-4]  balanced_cascade_protocol?(prot)
  |-------
[1]   x!1‘length = 0
[2]   linkage_property?(extract_gamma(x!1), z)
```

Lemma `admissible_language_sat_link_property` appearing as premise {`-1`} guarantees in this case that if `x!1` is a unique word, that is a sequence of operators, it satisfies the linkage property.

Supposing now that the length of `x!1` is greater than one, one has the sequent below. Observe that the succedent formulas {`1`} and {`2`} correspond to the antecedent `x!1` is greater than one.

```
[-1]  FORALL (y: {eta: gammaT | gamma_welldef?(prot, eta, z)}):
        y`length < x!1`length =>
         balanced_cascade_protocol?(prot) =>
          linkage_property?(extract_gamma(y), z)
[-2]  balanced_cascade_protocol?(prot)
  |-------
{1}   x!1`length = 1
[2]   x!1`length = 0
[3]   linkage_property?(extract_gamma(x!1), z)
```

In order to apply the induction hypothesis, one instantiates it, that is the antecedent `[-1]`, with the finite sequence `x!1` without its first word, that is the sequence `x!1^(1, x!1`length - 1)`, by applying the command `(inst -1 ''x!1^(1, x!1`length - 1)'')`. Since the sequence `x!1^(1, x!1`length - 1)` has length less than the length of `x!1`, this sequence can be used in the induction hypothesis. This together with the fact that `prot` is a balanced protocol gives rise to the simplification of the induction hypothesis (after this instantiation) as the premise `{-1}` in the sequent below, that states that `extract_gamma(x!1^(1, x!1`length - 1))` satisfies the linkage property.

```
{-1}  linkage_property?(extract_gamma(x!1^(1, x!1`length - 1)), z)
[-2]  balanced_cascade_protocol?(prot)
  |-------
[1]   x!1`length = 1
[2]   x!1`length = 0
[3]   linkage_property?(extract_gamma(x!1), z)
```

Selecting the previous sequence for the instantiation of the induction hypothesis is adequate for the application of Lemma 9, because this lemma guarantees that the concatenation of words that satisfy the linkage property also satisfies this property. Lemma 9 is specified in PVS with the name `linkage_property_composition` and its invocation at this point of the proof gives the sequent below in which the first premise corresponds to this lemma.

```
{-1}  FORALL (mu, eta: seqOps, z: U):
        linkage_property?(mu, z) AND linkage_property?(eta, z) =>
         linkage_property?(mu o eta, z)
[-2]  linkage_property?( extract_gamma(x!1^(1, x!1`length - 1)), z )
[-3]  balanced_cascade_protocol?(prot)
  |-------
[1]   x!1`length = 1
[2]   x!1`length = 0
[3]   linkage_property?(extract_gamma(x!1), z)
```

The first word of the finite sequence `x!1`, that is ``x!1`seq(0)``, belongs either to $\Sigma_0(z)$ or Σ_1 and consequently, it satisfies the linkage property, as previously mentioned. By induction hypothesis, the rest of the sequence, that is ``x!1^(1, x!1`length - 1)``, satisfies this property as well. Lemma 9 is instantiated with `mu` as ``x!1`seq(0)``, the first element of `x!1`, and `eta` as the rest of the sequence. Since ``x!1`seq(0) o extract_gamma(x!1 ^ (1, x!1`length - 1)) = extract_gamma(x!1)``, one concludes that the linkage property also holds for `x!1`. Proving this equality also requires several additional technicalities (almost ninety PVS proof commands are applied) not presented here, but available in the PVS formalization. One obtains as last sequent the one presented below.

```
{-1}  linkage_property?(x!1‘seq(0), z) =>
       linkage_property?(extract_gamma(x!1), z)
[-2]  linkage_property?( extract_gamma(x!1^(1, x!1‘length - 1)), z )
[-3]  balanced_cascade_protocol?(prot)
  |-------
[1]   x!1‘length = 1
[2]   x!1‘length = 0
[3]   linkage_property?(extract_gamma(x!1), z)
```

At this point, it is enough to guarantee that ``x!1`seq(0)`` satisfies the linkage property. This is done in the same way as in the case in that ``x!1`length = 1`` by application of the auxiliary lemma `admissible_language_sat_link_property`.

3.3 Formalization of technical properties

A great amount of the effort done in this formalization is related with the construction of proofs of specific properties of the sequences representing the quotient monoid of cryptographic operators. Here we present the formalization of a specific lemma applied in the proof of Lemma 9, in order to guarantee that the word user-balancing property of Definition 7 holds for subsequences of the concatenation of sequences satisfying the linkage property of definition 8. Then, it is necessary to characterize the normalization relative to a user z of concatenation of sequences δ and σ, $\overline{\delta\sigma}^{z}$. Two cases are to be considered: either the last operators of δ and the first of σ are opposite for the user z, or not. In the first case, let $\delta = \delta' O_z^k$ and $\sigma = (O_z^c)^j \sigma'$ normal with respect to z, where $k, j > 0$. In the second case, $\delta = \delta' O_u$ or $\sigma = O_u \sigma'$ for $u \neq z$. Two specific lemmas arise:

Lemma 2 (Relative normalization with separation)**.** *Let $\delta, \sigma \in \Sigma^*$, $z, a \in U$, such that $z \neq a$. Then*

$$\overline{\delta O_a \sigma}^z = \overline{\delta}^z O_a \overline{\sigma}^z$$

Lemma 3 (Relative normalization)**.** *Let $\delta, \sigma \in \Sigma^*$ be normal sequences with respect to $z \in U$ and let $j, k \geq 1$ such that j and k are maximal with $\delta = \delta' O_z^j$ and $\sigma = (O_z^c)^k \sigma'$. Then, $j \geq k$ implies*

$$\overline{\delta\sigma}^z = \delta' O_z^{j-k} \sigma'$$

Contrariwise,

$$\overline{\delta\sigma}^z = \delta' (O_z^c)^{k-j} \sigma$$

Here, we explain the formalization of the former lemma, that has been specified in PVS as the lemma `user_normalize_break` included below.

```
user_normalize_break : LEMMA FORALL (seq : seqOps, z, a : U, i : nat) :
  (a /= z & i < seq‘length - 1 & user(seq(i)) = a) =>
     normalizeseqZ(seq, z) =
      IF i = 0 THEN
        seq^(0,0) o normalizeseqZ(seq^(1, seq‘length - 1), z)
      ELSE
        normalizeseqZ(seq^(0,i-1), z) o seq^(i,i) o
                      normalizeseqZ(seq^(i+1,seq‘length - 1), z)
      ENDIF
```

The formalization of this lemma basically is based on the application of a number of auxiliary lemmas proved by induction, from which two key lemmas discriminate the case in which the first part of the sequence is normal with respect to z and the opposite case. The latter case, is specified as the lemma below.

`user_normalize_break` included below.

```
  user_normalize_separation2 : LEMMA FORALL (seq : seqOps,
                    a: U, z:U | a /= z,  i : below[seq‘length]) :
  (reduzibleseqZ?(seq,z) AND user(seq(i)) = a ) =>
    LET k = first_cancelableZ(seq,z) IN
     k < i =>
      normalizeseqZ(seq,z) =
       normalizeseqZ(seq^(0, i-1),z) o seq^(i,i) o
                    normalizeseqZ(seq^(i + 1 , length(seq) - 1), z)
```

The proof of this lemma consists of more than two hundred proof steps and is done basically by application of two additional auxiliary technical lemmas: the first one states that when δ is normal with respect to z, $\overline{\delta O_a \sigma}^z =$

$\delta O_a \overline{\sigma}^z$ and the second one that, in general $\overline{\overline{\alpha}^z \beta}^z = \overline{\alpha\beta}^z$. From these lemmas one has that $\overline{\delta O_a \sigma}^z = \overline{\overline{\delta}^z O_a \sigma}^z = \overline{\delta}^z O_a \overline{\sigma}^z$. The former lemma is specified as `user_normalize_separation1`.

```
user_normalize_separation1 : LEMMA FORALL (seq : seqOps,
                a: U, z:U | a /= z, i : nat | i < seq‘length) :
(reducibleseqZ?(seq,z) AND user(seq(i)) = a ) =>
 LET k = first_cancelableZ(seq,z) IN
 k > i =>
  normalizeseqZ(seq,z) =
     seq^(0, i) o normalizeseqZ(seq^(i + 1 , seq‘length - 1), z)
```

The formalization of this lemma is done by induction on the length of the sequence `seq` and consists of more than five hundred lines of proof commands in which thirty one invocation to other auxiliary technical lemmas are done.

This explanation can continue in this way, enumerating a long series of necessary auxiliary lemmas, which are related with the algebraic properties of the monoid freely generated by the cryptographic operators, the quotient monoid and the quotient monoid relative to a specific user as well as to its representation as the data structure of sequences of operators. Summarizing, what is relevant to clarify at this point is that most of the necessary formalization work is related with the mechanical proofs of these auxiliary technical lemmas.

In its current state, the whole PVS development consists of the nine subtheories depicted in Fig. 1 in which the specification part consists of more than 1700 lines of code (or 80 KB) and the proof part consists of more than 38000 lines of proof commands (or 2.4 MB). Auxiliary lemmas related with the data structure of sequences, the monoid and the quotient monoid relative to a specific user were specified respectively in the PVS theories `finite_sequences_extras`, `MonoidCryptOps` and `UserMonoidCryptOps` (see Fig. 1). These three theories alone consist of more than 900 lines of specification code (43 KB) and almost 26000 lines of proof commands (or 1.3 MB).

4 Conclusion and Future Work

The general sketch of the formalization of the theorem of characterization of security of two-party cascade protocols was concluded based on axiomatizations about balancing properties of the admissible language of potential malicious users. A great variety of properties about the monoid freely generated by the language of encryption and decryption operators were necessary as well as properties about the data structure of finite sequences. The latter was used twofold: firstly, in order to represent words of the monoid, that are finite sequences of cryptographic operators and secondly, to represent protocols, that are finite sequences of protocol steps, that are basically words in this monoid.

Several properties related with normalization of words in this monoid according to the convergence given by the elimination of opposite cryptographic operators were necessary, and in particular, these properties were relativized to specific users. The latter was necessary in order to establish properties such as the linkage property and the property of being user balanced.

A great deal of the effort invested in the formalization of the characterization of security of the DY model was concentrated on the proof of basic technical properties over the structure of monoids and its representation as sequences. The formalization of these auxiliary lemmas is worth because it is fundamental in order to formalize the security of the DY model. But more important, it is valuable because this represents an important and robust kernel that can be applied in order to formalize logical properties of other cryptographic protocols and models (e.g., multiparty models, models with authentication mechanisms, models with blind signatures, etc.). In fact, this was illustrated in this paper, when the logical sketch of the proofs of the Theorem 1 of characterization of security and of the Lemma 1 of balancing property of normalizations of words of the admissible language were explained.

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 Reasoning, 2006. 1

[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 Verlag, 2008. 1

[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. 1

[BJ03] M. Backes and C. Jacobi. Cryptographically Sound and Machine-Assisted Verification of Security Protocols. In *20th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 2607 of *Lecture Notes in Computer Science*, pages 675–686. Springer Verlag, 2003. 1

[CMR01] V. Cortier, J. Millen, and Harald Ruess. Proving secrecy is easy enough. In *14th IEEE Computer Security Foundations Workshop (CSFW'01)*, pages 97–110. IEEE Comp. Soc. Press, 2001. 1

[DCS02] B. Dutertre, V. Crettaz, and V. Stavridou. Intrusion-tolerant enclaves. In *Proc. of the IEEE International Symposium on Security and Privacy*, pages 216–224, 2002. 1

[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 Verlag, 1997. 1

[DY83] D. Dolev and A. C. Yao. On the Security of Public Key Protocols. *IEEE. T. on Information Theory*, 29(2):198–208, 1983. 1, 2.2, 2.2, 1, 3.1

[ES00] N. Evans and S. Schneider. Analysing Time Dependent Security Properties in CSP Using PVS. In 6th *European Symposium on Research in Computer Security ESORICS*, volume 1895 of *Lecture Notes in Computer Science*, pages 222–237. Springer Verlag, 2000. 1

[LHT07] M. Layouni, J. Hoofman, and S. Tahar. Formal Specification and Verification of the Intrusion-Tolerant Enclaves Protocol. *International Journal of Network Security*, 5(3):288–298, 2007. 1

[Low95] G. Lowe. An Attack on the Needham-Schroeder Public-Key Authentication Protocol. *Information Processing Letters*, 56(3):131–133, 1995. 1

[Low96] G. Lowe. Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR. *Software - Concepts and Tools*, 17(3):93–102, 1996. 1

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

[MR00] J. K. Millen and H. Rueß. Protocol-independent secrecy. In *IEEE Symposium on Security and Privacy*, pages 110–209, 2000. 1

[NdMNAR10] R.B. Nogueira, F.L.C. de Moura, A. Nascimento, and M. Ayala-Rincón. Formalization Of Security Proofs Using PVS in the Dolev-Yao Model. In *Computability in Europe CiE 2010 (Booklet)*, 2010. 1, 2.2

[NS78] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Comm. of the ACM*, 21:993–999, 1978. 1

[OS97] Sam Owre and Natarajan Shankar. The formal semantics of PVS. Technical report, SRI-CSL-97-2, Computer Science Laboratory, SRI International, Menlo Park, CA, August 1997. Available at http://pvs.csl.sri.com/. 1, 3

[Pau99] L. C. Paulson. Proving Security Protocols Correct. In *14th Annual IEEE Symposium on Logic in Computer Science LICS*, pages 370–383, 1999. 1