Formal modeling of cryptographic protocols: Dolev-Yao model

Vaishnavi Sundararajan

Chennai Mathematical Institute, Chennai, India
vaishnavi@cmi.ac.in

1 The Dolev-Yao Model

Formal verification of cryptographic protocols requires an abstract model of agents' capabilities and communications, and the Dolev-Yao model has been the bulwark of such modelling. This model and its various extensions have been the object of study for the last 30 years. This model views the message space as a term algebra. Term derivation rules specify how agents can obtain new terms from old ones.

Fix countable sets $\mathscr{A}$, $\mathscr{N}$, $\mathscr{K}$ and $\mathscr{V}$, denoting the set of agents, nonces, keys, and variables, respectively. The set of basic terms is $\mathscr{B} = \mathscr{A} \cup \mathscr{N} \cup \mathscr{K}$. For each $A, B \in \mathscr{A}$, assume that $\mathsf{sk}(A)$, $\mathsf{pk}(A)$ and $\mathsf{shk}(A, B)$ are keys. Further, each $k \in \mathscr{K}$ has an inverse defined as follows: $\mathit{inv}(\mathsf{pk}(A)) = \mathsf{sk}(A)$, $\mathit{inv}(\mathsf{sk}(A)) = \mathsf{pk}(A)$ and $\mathit{inv}(k) = k$ for the other keys.

Fix a signature $\mathscr{F}$, which contains all constructors used for building terms. In the original paper by Dolev and Yao, the authors only consider pairing and encryption as the constructors, but since the model has been extended with many other constructors in the intervening years, we present here a version with not only pairing and encryption, but hashing and signatures as well.

We use the notation $\mathscr{T}[\mathscr{F}; \mathscr{S}]$ to denote Dolev-Yao terms over a signature $\mathscr{F}$ and an underlying term set $\mathscr{S}$. In general, when we say $\mathscr{T}$, we will mean $\mathscr{T}[\mathscr{F}; \mathscr{B} \cup \mathscr{V}]$. *Ground terms* are terms without variables. This is the set $\mathscr{T}[\mathscr{F}; \mathscr{B}]$, which will be denoted by $\mathscr{T}_g[\mathscr{F}]$. A *substitution* σ is a partial map from $\mathscr{V}$ to $\mathscr{T}_g[\mathscr{F}]$. σ can be lifted for terms in the standard manner.

There are various ways of presenting the terms under consideration in the Dolev-Yao model. One way is to present a term grammar with a derivation system for deriving new terms. In the grammar shown below, $t \in \mathscr{T}[\mathscr{F}; \mathscr{B} \cup \mathscr{V}]$ with $m \in \mathscr{B} \cup \mathscr{V}$ and $k \in \mathscr{K} \cup \mathscr{V}$. The terms presented here are over the signature $\mathscr{F} = \{\mathsf{pair}, \mathsf{senc}, \mathsf{aenc}, \mathsf{hash}, \mathsf{sign}\}$. (Having fixed this signature, we will now choose to omit $\mathscr{F}$ from our term set notation).

$$t ::= m \mid \mathsf{pair}(t_0, t_1) \mid \mathsf{senc}(t, k) \mid \mathsf{aenc}(t, k) \mid \mathsf{hash}(t) \mid \mathsf{sign}(t, k)$$

The derivation system is shown in Table 1. Note that all derivations must only mention ground terms, and so $X \subseteq \mathscr{T}_g$ and $t, t_0, t_1 \in \mathscr{T}_g$, $k \in \mathscr{T}_g \cap \mathscr{K}$.

Another way to present Dolev-Yao terms is using an equational theory. Note that in the previous presentation, our signature only included 'constructors' (pair, senc, etc) but no 'destructors' (fst, snd, sdec, etc). The derivation system handled the breaking down of big terms into smaller ones, and effectively provided rules for capturing destructors. In this equational theory system, we will expand the signature to include destructors as well, and then provide equations for when a term formed by application of multiple functions from the signature might be equivalent to a smaller term.

We use the expanded signature $\mathscr{F} = \{\mathsf{pair}, \mathsf{fst}, \mathsf{snd}, \mathsf{sdec}, \mathsf{adec}, \mathsf{hash}, \mathsf{sign}, \mathsf{ver}, \mathsf{msg}, \mathsf{true}, \mathsf{false}\}$. The destructors fst and snd give the first and second projection of a pair object, respectively. ver verifies the content of a signed message against a particular term, while msg returns the

$\dfrac{}{X \cup \{t\} \vdash t}\ ax$	$\dfrac{X \vdash t}{X \vdash \mathsf{hash}(t)}\ hash$
$\dfrac{X \vdash t_0 \quad X \vdash t_1}{X \vdash \mathsf{pair}(t_0, t_1)}\ pair$	$\dfrac{X \vdash \mathsf{pair}(t_0, t_1)}{X \vdash t_i}\ split_i$
$\dfrac{X \vdash t \quad X \vdash k}{X \vdash \mathsf{enc}(t, k)}\ enc$	$\dfrac{X \vdash \mathsf{enc}(t, k) \quad X \vdash inv(k)}{X \vdash t}\ dec$
$\dfrac{X \vdash t \quad X \vdash k}{X \vdash \mathsf{sign}(t, k)}\ sign$	$\dfrac{X \vdash \mathsf{sign}(t, k)}{X \vdash t}\ unsign$

Table 1 The Dolev-Yao derivation system (enc could be senc or aenc depending on the kind of key k used, and similarly for dec)

underlying message alone from the signed object. true and false are 0-ary functions standing for the boolean values true and false, which we require to ensure that the result of applying a ver to a sign, for example, is also a term in our syntax.

We present the syntax of terms $\mathscr{T}$ (over this expanded signature $\mathscr{F}$), where $m \in \mathscr{B} \cup \mathscr{V}$, $t, t_0, t_1 \in \mathscr{T}$, and $k \in \mathscr{K} \cup \mathscr{V}$.

$$\begin{aligned} t = \mathsf{true} \mid \mathsf{false} \mid m \mid \mathsf{pair}(t_0, t_1) \mid \mathsf{fst}(t) \mid \mathsf{snd}(t) \\ \mid \mathsf{senc}(t, k) \mid \mathsf{sdec}(t, k) \mid \mathsf{aenc}(t, k) \mid \mathsf{adec}(t, k) \\ \mid \mathsf{hash}(t) \mid \mathsf{sign}(t, k) \mid \mathsf{ver}(t_0, t_1, k) \mid \mathsf{msg}(t) \end{aligned}$$

The equational theory E over this signature is given below, where $t, t', r \in \mathscr{T}, k \in \mathscr{K}, a \in \mathscr{A}$. The function ver takes three inputs, and returns true only if the first input to it (the signed message) is the sign of the second input with the inverse of the third.

$$\begin{aligned} \mathsf{fst}(\mathsf{pair}(t, t')) &= t \\ \mathsf{snd}(\mathsf{pair}(t, t')) &= t' \\ \mathsf{sdec}(\mathsf{senc}(t, k), k) &= t \\ \mathsf{adec}(\mathsf{aenc}(t, \mathsf{pk}(a)), \mathsf{sk}(a)) &= t \\ \mathsf{msg}(\mathsf{sign}(t, k)) &= t \\ \mathsf{ver}(\mathsf{sign}(t, \mathsf{sk}(a)), t, \mathsf{pk}(a)) &= \mathsf{true} \qquad (\mathsf{false}\ \text{otherwise}) \end{aligned}$$

For the sake of readability, we will hereafter use $\{t\}_k$ to represent $\mathsf{enc}(t, k)$ (with the type of k used – symmetric key versus public key – disambiguating between senc and aenc) and (t_0, t_1) for $\mathsf{pair}(t_0, t_1)$.

1.1 Intruder Capabilities

The intruder has access to all public information, and can block and replay any terms sent by honest agents. The intruder is essentially the network itself – any terms sent by honest agents are received by the intruder and then forwarded onto the intended recipient (or not), and any terms received by honest agents are assumed to have come from the intruder. He

can also send terms under masquerade, in the name of any agent (all agents' names are public). The intruder can send out any terms he can derive from the set of terms he has access to, using the derivation rules given in Table 1. Note that in particular, this means that while an agent A might send out a term t for the intended recipient B, the intruder might send a term t' to B, pretending to be A, thereby effecting a 'forgery' in A's name. A key component in checking for what terms the intruder can get access to is solving the *derivability problem*. (We will see later how this becomes important when we try to prove various security properties about the protocols under examination, in Section 1.4.)

Derivability Problem

Given a set of terms X and a term t, can one check whether $X \vdash t$ according to the derivation rules given in Table 1?

1.2 Actions

Agents can send and receive terms. A send action of a term t by $A \in \mathscr{A} \cup \mathscr{V}$ to an intended recipient $B \in \mathscr{A} \cup \mathscr{V}$ looks as follows: $A!B : (\vec{m})\ t$, where $\vec{m} \subseteq \mathscr{T}$ is the set of fresh variables and nonces used for generating the term t (the $\vec{m}$ may be omitted in sends where there are no fresh values involved). Similarly, a receive action by A from a purported sender B looks as follows: $A?B : t$.

▶ **Definition 1.** An *A-action* is a send by A or a receive by A. An *A-role* is a finite sequence of A-actions, and a *role* is an A-role for some $A \in \mathscr{A} \cup \mathscr{V}$. A *protocol* is a finite set of roles.

For a role $\eta = a_0 \cdots a_{n-1}$ and $i \in [0..n]$, we denote the partial completion of η till $i-1$ by (η, i). (This denotes that a_i is the next action in this role that has to happen.) We say that the variable x *originates* at i if x occurs in a_i and does not occur in a_j for any $j < i$. A variable x is *active* at (η, i) if one of the following conditions holds:

- η is an x-role
- x originates at some j and either $j < i$ or a_j is a send.

Even though the roles of a protocol mention variables, its executions consist only of ground terms and assertions exchanged in various instances of the roles. An instance of a role is formally specified by a substitution σ, which, as mentioned earlier, is a partial map from $\mathscr{V}$ to $\mathscr{T}_g$, and is lifted for terms and actions in the standard manner. A substitution σ is said to be suitable for an action a if $\sigma(a)$ is an action, i.e. a typing discipline is followed. A substitution is suitable for a (partially completed) role (η, i) if it is defined on all variables active at (η, i), and suitable for all actions in η.

The notion of what messages an agent can construct at any given point of time is formalized by a "knowledge state", which represents all the terms and assertions that each agent knows. A "control state" is a record of progress made by an agent in the various sessions he/she participates in.

Each agent A has a set of terms she has access to – initially this set just includes her secret key, all other agents' public keys, all constants of the protocol etc. An agent's knowledge set grows as he/she participates in a protocol. The global knowledge state is the tuple of the term sets of all agents (including the intruder). Any action, when enabled at a knowledge state, takes the system from that knowledge state to another, with modifications to the term sets of some agents.

In order to check whether a send action $A!B : (\vec{m})\ t$ is enabled at a state, we check whether the term t is derivable by A, using all the terms A has access to at that point and

$\vec{m}$. As mentioned in the previous section, any send is assumed to be captured by the intruder. So this action leads to t being added to the intruder's set of terms. The receive action $A?C : t$ is the complement of the send action, in that enabling is equivalent to checking whether the intruder can derive the received term bound to x, and the update is to add the term to the recipient's term set.

A knowledge state ks is a tuple $(X_A)_{A \in \mathscr{A}}$, where X_A is the set of ground terms belonging to an agent A. A control state S is a finite set of triples of the form (η, i, σ), where η is a role of the protocol, i is an index in the range $[0, |\eta|]$ that indicates the current action of the role η as $\eta[i]$ (or that the session has ended, when $i = |\eta|$), and σ is a substitution suitable for η. A protocol state is a pair (ks, S) where ks is a knowledge state and S is a control state. One can think of the transition system induced by the protocol states – the transitions are given according to Definition 2. The protocol itself is characterized by this transition system.

▶ **Definition 2.** Let (ks, S) and (ks', S') be two states of a protocol Pr, and let b be a ground action. We say that $(ks, S) \xrightarrow{a} (ks', S')$ iff $\exists (\eta, i, \sigma) \in S$ and $\sigma' \supseteq \sigma$ such that:

- σ' is defined for all variables in $\eta[i]$
- $a = \sigma'(\eta[i])$
- $S' = S \setminus \{(\eta, i, \sigma)\} \cup \{(\eta, i+1, \sigma')\}$
- $ks \xrightarrow{a} ks'$ as given in Table 2.

In Table 2, for any agent A, we use X_A and X'_A to denote agent A's set of terms in ks and ks' respectively (I is the intruder). Term sets that do not undergo any change are omitted. Note also that we add σ' (not σ), in order to update the substitution associated with the session on executing the action. This update reflects the new bindings for input variables (in case of a receive).

Action a	Enabling conditions	Updates
$A!B : (\vec{m})\ t$	$X_A \cup \vec{m} \vdash t$	$X'_A = X_A \cup \vec{m}$ $X'_I = X_I \cup \{t\}$
$A?B : t$	$X_I \vdash t$	$X'_A = X_A \cup \{t\}$

Table 2 Enabling and update conditions for $(ks, S) \xrightarrow{a} (ks', S')$.

▶ **Definition 3.** A *run* of the protocol is a run of this transition system.

In the next section, we present an example of how a simple protocol can be modelled in this framework.

1.3 Example Protocol

Consider a (very) simple protocol using which agent A wants to check if agent B is online. A sends B a freshly generated random value, which B should send back to A. So once A receives this nonce back, A is convinced that B is online and has responded to her communication.

$A \to B : n$

$B \to A : n$

With each communication being split into the corresponding send and receive actions, we see the protocol looks as follows.

$A!B : (n)\ n$

$B?A : x$

$B!A : x$

$A?B : y$

The protocol can be split into two roles, the A-role and the B-role. The A-role is composed of the two A-actions where A sends out a nonce, and gets a response (and checks whether this response is the same as the nonce she sent out in the previous action). The B-role is the complement, where B receives a value, and just sends the same value back.

1.4 Security Properties and Verification

Having set up all this machinery, if we use it to model protocols, what properties of protocols are we interested in examining, and how do we test for them in this model? In this section, we will inspect a few security properties of interest and how they are specified in this model.

Security properties are of two types – *trace* properties and *equivalence* properties. Trace properties are those which can be defined for each run of the protocol. A protocol is said to satisfy such a property if every valid run of the protocol satisfies the property. Equivalence properties are those which, if satisfied, mean that the adversary cannot distinguish between two processes. Equivalence properties, therefore, require the inspection of multiple runs of both processes under consideration, unlike trace properties.

1.4.1 Secrecy

For a term t to be deemed 'secret' in a protocol, it must be the case that the intruder never learns it, no matter what the other agents communicate. Formally, a protocol satisfies the secrecy of t if and only if in all runs of the protocol, it is never the case (at any state of the run) that $X_I \vdash t$.

Therefore, given a protocol, checking whether it satisfies the secrecy of a given term reduces to checking the derivability of the term from the intruder's term set in every possible execution of the protocol. Secrecy is thus a trace property. Note that if the derivability problem for the system is efficiently decidable, and if the number of possible runs of a protocol is bounded, we have an obvious brute-force algorithm for verifying the secrecy property.

1.4.2 Correspondence

A correspondence property is of the following form: if an event has occurred during the course of a protocol execution, then a corresponding event must have happened before it in the past. Correspondence properties are trace properties, since it suffices to examine each trace to see whether they are preserved or violated. The actual formalization of the property would depend on the events under question.

1.4.3 Authentication

Authentication is the property that if an agent A executes the protocol with another agent she thinks is B, then B also thinks that he has executed the protocol with A, and vice versa.

This is generally expressed as a correspondence property (and is thus also a trace property), for example, the following: if A terminates the protocol with B, then B started a session of the protocol with A. Several variants of this property can be stated, depending on the protocol.

1.4.4 Unlinkability

The unlinkability property requires that the relationship between any two designated terms should not be known to the adversary. Since multiple executions taken together might reveal some link between the terms under consideration, this property needs the inspection of multiple runs, and is hence an equivalence property. Anonymity, which is a key requirement in many protocols, is a specific instance of the unlinkability property, where the relationship between an agent and some term used by him/her should not be revealed to the adversary.

1.4.5 Strong secrecy

One can define a stronger secrecy notion, strong secrecy, which states that the adversary cannot detect a change in the value of the secret. In other words, the adversary has no information at all on the value of the secret. Unlike the earlier secrecy property, this is an equivalence property – two runs where the possibly secret value has different values ought to be indistinguishable from the attacker's perspective for this property to hold.

▶ **Definition 4** (Verification Problem)**.** For a given property φ specified in this model, the *φ-verification problem* is to check whether the given protocol satisfies φ.

The φ-verification problem requires the examination of some or all runs of the protocol, either individually or as a set (depending on whether φ is a trace property or an equivalence property). In particular, the set of all runs of the protocol will also include runs where the intruder exercises all faculties at her disposal to obtain and/or interfere in other ways with messages from honest agents – a property is said to be satisfied for a protocol if it is satisfied even for these runs, i.e., in spite of the intruder's best efforts to violate it.