AVISPA

www.avispa-project.org

IST-2001-39252

Automated Validation of Internet Security Protocols and Applications

Deliverable D3.2: Assumptions on Environment

Abstract

In protocol analysis, the *environment* refers to the formal definition of the conditions under which a security protocol executes. This environment comprises many elements. Among them we count, for instance, the deductive powers assumed of the intruder and the properties of the communication channels over which messages are sent, including the (perhaps differing) intruder types to which said channels are vulnerable. Modern Internet protocols are designed to be executed in a variety of environments. The ability to specify a rich set of assumptions on the protocol analysis environment is therefore important for the specification and analysis of such protocols. This deliverable presents four techniques that yield a more expressive set of assumptions on the environment.

Deliverable details

Deliverable version: *v1.0*
Date of delivery: *30.07.2004*
Classification: *public*

Person-months required: *20*
Due on: *30.06.2004*
Total pages: *37*

Project details

Start date: *January 1st, 2003*
Duration: *30 months*
Project Coordinator: *Alessandro Armando*
Partners: *Università di Genova, INRIA Lorraine, ETH Zürich, Siemens AG*

Information Society Technologies

Project funded by the European Community under the *Information Society Technologies* Programme (1998-2002)

Contents

1 Introduction

With some exceptions (e.g. [37]), protocol analysis is usually carried out with a relatively uniform set of assumptions on the environment in which the protocol is executed. For instance, the network connecting the various parties involved in a protocol is generally assumed to be homogeneous, with all communication channels sharing identical characteristics. A standard Dolev-Yao intruder [26] is generally assumed to control the network.

This standard set of environmental assumptions, while able to describe a vast range of protocol analysis scenarios, is not always suited to the analysis of modern, industrial-scale protocols like those of the AVISPA library [7]. In this deliverable, we discuss why this is the case and present a set of techniques that aims to enable the modelling of a richer set of assumptions on the protocol analysis environment. We divide these techniques into the following three sub-areas.

Message Interpretation by Honest Principals Many protocols (perhaps most famously Kerberos [29, 36]) involve agents receiving messages containing parts that they cannot actually decrypt, but rather must forward to their intended recipients. In §2, we describe compound typing, our approach to specifying how principals interpret the messages they receive, both those that they can decrypt and those that they cannot.

Protected Communication Channels The idea of a homogeneous communication network in which all links share similar or identical characteristics is often not an accurate model of the environment in which protocols are designed to operate. Networks are often heterogeneous, and different communication channels may have different characteristics. For instance, some media are wired and some wireless. In the case of the former, we generally adopt the worst-case assumption that the intruder might gain control of a router and be able to intercept messages, preventing them from arriving at their destinations. In the latter case, communication takes place over wireless links, and it is less likely that the intruder can perform such intercepts. We thus need a way to specify a richer set of assumptions on channel types, in particular the amount of control exerted by the intruder over a given type of channel. Assumptions on communication channels are discussed in §3.

Extended Abilities of the Intruder In §4 and §5, we discuss augmenting standard intruder models with extended abilities. We achieve this by specifying additional deduction rules available to the intruder as well as, in the latter subsection, extending the message algebra to include a new type of term. We extend the intruder's abilities using the following two techniques.

- *Oracle rules:* Oracle rules, described in Deliverable 2.2 [3], can be used to extend the deductive powers of the intruder. In §4, we focus on the application of oracle rules to the intruder's handling of properties of cryptographic operators. In particular, we show how they can be used to model so-called low-exponent attacks on certain implementations of the RSA encryption scheme.

- *Guessing*: Many attacks exploit the fact that users often choose poor passwords. Such weak passwords are vulnerable to guessing attacks. In situations where such vulnerabilities may arise, it is thus reasonable to assume that the intruder can attempt to mount guessing-based attacks. §5 describes a novel approach to model a guessing intruder.

2 Compound Typing

Typing is an important modelling technique used in the analysis of security protocols. In a *typed model* of a protocol, messages are built out of a typed language thereby preventing that agents accept ill-formed messages. A typed model therefore corresponds to reasonable implementations of the protocol in which ill-formed messages are rejected by their receivers. If no typing discipline is assumed, then the more liberal *untyped model* is considered. In an untyped model, one agent can receive and misinterpret an ill-formed message for a legal one thereby leading to *type-flaw attacks*. Whether type-flaw attacks should be considered serious threats on protocols or not is a matter of ongoing debate.[1] In any case, both typed and untyped models are currently supported by the AVISPA Tool.

An important feature of the typed model (as opposed to the untyped one) is that it leads to a smaller search space. This is particularly important in model-checking tools such as the SATMC back-end, which are based on a preliminary encoding and unfolding of the transition relation. In this context, the adoption of the typed model is crucial and plays a pivotal role in the scalability of the approach to industrial-scale security protocols.

An important observation in this context is that many industrial-scale security protocols (e.g. Kerberos [29, 36]) specify communications in which a principal receives a message containing parts that he cannot decrypt, but that must be simply forwarded to other recipients. In such situations, the intermediate receiver cannot check the "well-formedness" of the message to be forwarded and ill-formed messages are later rejected by the final recipients. In terms of the search tree, this corresponds to the exploration of a huge number of useless branches. While it is not possible (in principle) to prevent this exploration in an untyped model, by using a typed model it is possible to restrict the patterns of those parts of a message that a receiver cannot decrypt. In order to achieve this, we use the notion of *compound typing*. Compound types have been already introduced in the IF language (see Section 4.1 of Deliverable 2.3 [4]); here we lift compound types to the HLPSL and to the HLPSL2IF Translator.

In the next subsection, §2.1, we introduce the Kerberos protocol as a motivating example. The notion of compound typing is introduced in §2.2 and some experimental results conducted with a new version of SATMC [5] supporting compound types are given in §2.3. The experimental results show that compound types allow SATMC *(i)* to gain up to three orders of magnitude in the time spent for analysing protocols (e.g., the AAAMobileIP protocol [15]), and *(ii)* to analyse protocols (e.g., the Kerberos protocol), that were previously

[1]Tagging mechanisms have been proposed in order to enforce the typed discipline [28] thereby ruling out certain kinds of type-flaw attacks.

out of reach.

2.1 The Kerberos Protocol

Kerberos is a trusted third-party authentication service based on the key distribution model developed by Needham and Schroeder [35]. In Alice&Bob-style notation, the Kerberos protocol is as follows:

```
(1) C -> A : C, G, Lifetime_1,N_1
(2) A -> C : C, Ticket_1, {G, K_CG, Tstart_1, Texpire_1, N_1}K_CA
(3) C -> G : S, Lifetime_2, N_2, Ticket_1, {C, T_1}K_CG
(4) G -> C : C, Ticket_2, {S, K_CS, Tstart_2, Texpire_2, N_2}K_CG
(5) C -> S : Ticket_2, {C, T_2}K_CS
(6) S -> C : {T_2}K_CS
```

where the terms `Ticket_1` and `Ticket_2` are defined as:

```
Ticket_1 := {C, G, K_CG, Tstart_1, Texpire_1}K_AG
Ticket_2 := {C, S, K_CS, Tstart_2, Texpire_2}K_GS
```

In such a protocol, a client `C` obtains credentials from the trusted authentication server `A` (steps `(1)` and `(2)`) and from the ticket granting server `G` (steps `(3)` and `(4)`) in order to request access to a service provided by the server `S` (steps `(5)` and `(6)`). A detailed explanation of the above protocol goes beyond the scope of this deliverable and the interested reader should consult [29, 36]. Let us concentrate our attention on the term `Ticket_1` (the same ideas apply also to `Ticket_2`). One can immediately see that since `C` does not know the key `K_AG` shared between `A` and `G`, then `C` cannot decrypt `Ticket_1` sent by `A` at protocol step `(2)`. The client `C` in the protocol step `(3)` just forwards such a term to the intended receiver that will be able to decrypt it and to check its content. This can be easily expressed in HLPSL as follows:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Definition of the Trusted Authentication Server Role
%%%
role Kerberos_A (A, C, S, G: agent,
                 Snd, Rcv:   channel (dy),
                 K_CA, K_AG: symmetric_key) played_by A def=

  exists St:                  nat,
         K_CG:                symmetric_key (fresh),
         Tstart_1, Texpire_1: text (fresh),
         Lifetime_1, N_1:     text,

  ... % Other declarations
  transition

   1. St=0 /\ Rcv(C.G.Lifetime_1'.N_1')
      =|>
      St'=1 /\
      % NOTE: Part_2 is {C.G.K_CG'.Tstart_1'.Texpire_1'}K_AG
      Snd(C.{C.G.K_CG'.Tstart_1'.Texpire_1'}K_AG.
            {G.K_CG'.Tstart_1'.Texpire_1'.N_1'}K_CA)
end role

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Definition of the Client Role
%%%
role Kerberos_C (C, A, G, S: agent,
                 Snd, Rcv:   channel (dy),
                 K_CA:       symmetric_key) played_by C def=

  exists St:                        nat,
         K_CG:                      symmetric_key,
         N_1, N_2, Lifetime_2, T_1: text (fresh),
         Tstart_1, Texpire_1:       text,
         % NOTE: Ticket_1 is declared of type message
         Ticket_1:                  message

  ... % Other declarations

  transition

   ... % transition 1

   2. St=1  /\
      % NOTE: Ticket_1 can be every message
      Rcv(C.Ticket_1'.{G.K_CG'.Tstart_1'.Texpire_1'.N_1}K_CA)
      =|>
      St'=2 /\ Snd(S.Lifetime_2'.N_2'.Ticket_1'.{C.T_1'}K_CG')

   ... % transitions 3 and 4

end role
```

In the above HLPSL specification, any agent playing the role `Kerberos_A` sends in its first transition a message that is composed of the identity of `C` and of two other sub-messages:

the second is encrypted with the symmetric key K_CA, while the first (i.e. Ticket_1) is encrypted with the symmetric key K_AG and contains the identities of C and G, a fresh symmetric key K_CG and two fresh numbers Tstart_1 and Texpire_1.[2] When an agent playing the role of Kerberos_C receives such a message (see transition 2 in this role), it cannot decrypt and check Ticket_1, since it does not know the key K_AG. Moreover, due to the fact that Ticket_1 is declared to belong to the most generic type message it turns out that even under the typed model the intruder can send whatever he wants to the agents playing role Kerberos_C in place of Ticket_1. This could cause a blow-up of the search space that may dramatically affect the performance of the back-ends. Such an explosion can be avoided by extending the HLPSL language for supporting the specification of compound types.

2.2 The Notion of Compound Typing

Compound types allow the protocol designer to declare HLPSL variables of sorts restricted and specialised in a particular way. For instance, with reference to the Kerberos protocol presented in §2.1, we could be more specific about the type of the HLPSL variable Ticket_1 (similarly for Ticket_2) instead of declaring it to be of type message. In fact, even if the first principal who receives Ticket_1 cannot decrypt it, Ticket_1 is forwarded and later on it will be decrypted and checked for well-formedness. In more detail, we would like to be able to declare that the HLPSL variable Ticket_1 can be only instantiated with terms of the form {A_1.A_2.K_1.Num_1.Num_2}K_2 where A_1 and A_2 are of type agent, K_1 and K_2 are of type symmetric_key, and Num_1 and Num_2 are of type text. To this end, we have extended the syntax of the HLPSL by changing the grammar rule Subtype_of (see the appendix of Deliverable 2.1 [2]) and by adding the grammar rule Compound_type as follows:

```
Subtype_of ::=
     Simple_type
   | Compound_type                  % New declaration

% New grammar rule for expressing compound types
Compound_type ::=
     Subtype_of "list"              % List
   | Subtype_of "set"               % Set
   | "{" Subtype_of "}" Subtype_of  % Encryption
   | Subtype_of "." Subtype_of      % Concatenation, right-associative
```

[2]Since the current version of the AVISPA Tool does not support timestamps, we model Tstart_1 and Texpire_1 as numbers. This amounts to applying an abstraction on the concrete system by assuming that timestamps never expire. It is immediate to see that if a security property holds for the abstract system, it will also hold for the concrete one that adopts timestamps. On the other hand, an attack on the abstract system does not always correspond to an attack on the concrete one and thus spurious attacks can be reported by the tool. We note also that timestamps might be considered guessable in the sense of §5. It will be interesting future work to analyse protocols involving timestamps guessable by the intruder.

```
   | Subtype_of "->" Subtype_of     % Function, left-associative
```

The semantics of the above new declarations can be easily expressed in terms of the semantics of the `Simple_type` declaration already specified in [2]. In fact, any term that can be generated from the grammar rule `Compound_type` is simply a "macro" for specifying the definitions of a new type and its associated domain. For instance, concerning the Kerberos protocol, the HLPSL declaration

```
Ticket_1: {agent.agent.symmetric_key.text.text}symmetric_key
```

specifies the type of the variable `Ticket_1`. The domain of this type comprises all the terms of the form {`A_1.A_2.K_1.Num_1.Num_2`}`K_2` such that `A_1` and `A_2` are of type `agent`, `K_1` and `K_2` are of type `symmetric_key`, and `Num_1` and `Num_2` are of type `text`.

While the IF language presented in Deliverable 2.2 [4] already allows for the specification of compound types, the HLPSL2IF translator has required some modifications in order to be able to translate HLPSL compound typing declarations into the corresponding IF compound typing declarations. For instance, the HLPSL compound typing declaration described above for the Kerberos protocol is simply compiled into the following IF typing declaration:

```
Ticket_1: scrypt(symmetric_key,pair(agent,
                                    pair(agent,
                                         pair(symmetric_key,pair(text,
                                                                 text)))))
```

Besides this, small changes are required in the pattern of the dummy values occurring in the state-facts describing the initial state of honest agents.[3] Let us clarify this with an example. Given the HLPSL specification of the Kerberos protocol partially shown in §2.1, the state of any honest agent playing the role `Kerberos_C` will, of course, store its current value of `Ticket_1`. If `Ticket_1` is declared of type `message`, then the initial value of such a variable will be `dummy_msg`. On the other hand, if `Ticket_1` is declared to be of the compound type mentioned above, then the initial dummy value for `Ticket_1` will be:

```
scrypt(dummy_sk,pair(dummy_agent,pair(dummy_agent,
                                      pair(dummy_sk,pair(dummy_nonce,
                                                         dummy_nonce)))))
```

The rest of the IF specification (rules, goals, etc.) is not, syntactically, affected by the introduction of compound types. However, under the typed model, the search space associated with the IF specification with compound types will be much smaller than that associated with the IF specification without compound types. For instance, let us consider the IF rule corresponding to the second HLPSL transition of the role `Kerberos_C`:

[3]Dummy values are needed to give an initial value to those fields of a state-fact whose values will be fixed only after some protocol step executions.

```
step step_1 (C,..,Dummy_Ticket_1,Ticket_1,..,CID) :=
    state_Kerberos_C(C,..,1,Dummy_Ticket_1,..,CID).
    iknows(pair(C,pair(Ticket_1,..)))
=[..]=>
    state_Kerberos_C(C,..,2,Ticket_1,..,CID).
    iknows(pair(..,pair(..,pair(..,pair(Ticket_1,..)))))
```

It is immediate to see that the number of possible instantiations of such a rule heavily depends, aside from the domains of the other parameters, on the domain of `Ticket_1`. Hence, if this domain is more specific and restricted, then the number of possible rule instances and, therefore, the size of the search space will be accordingly smaller as well.

An alternative approach to deal with the same problem has been proposed in [31]. It is based on the Lowe operator and allows for the restriction of pattern terms. Use of the Lowe operator is expressive enough to specify our compound types, but we believe that our approach is simpler and more intuitive. In fact, starting from a HLPSL specification, we can generate an IF specification that can be analysed with respect to both the typed and the untyped models simply by considering or neglecting the type information. This seems not to be the case for Casper [31] and CAPSL [16], both using the Lowe's operator, in which such a pattern restriction explicitly affects the syntax of the transition rules making them unsuited for analysing security protocols with respect to the untyped model.

2.3 Experimental results

We have updated the SATMC back-end in order to support compound types and we have experimented with the AAAMobileIP and Kerberos security protocols. An explanation of how SATMC works goes beyond the scope of this deliverable and the interested reader should consult [5] and the forthcoming Deliverable 4.5 [8].

Table §1 reports the results of our experiments obtained by running SATMC against IF specifications with the compound types disabled (**No Compound Types**) and enabled (**Compound Types**). Experiments have been carried out under Linux on a Pentium IV 2.4 GHz with a resource limit of 1 hour CPU time and 1 GB memory. For each protocol and for each IF specification with and without the compound types, we give the number of steps at which the search halted (**K**), the number of propositional variables (**Atoms**) and clauses (**Clauses**) in the SAT formula, and the time (**Time**) spent by SATMC in doing the analyses.

It is immediate to see that for both the security protocols analysed, the application of compound types significantly improves the performance of SATMC. In the case of the Kerberos protocol, if compound types are disabled then SATMC is unable to analyse the protocol because memory is exhausted after only 5 steps. By declaring the appropriate compound types for `Ticket_1` and `Ticket_2` (see §2.1 and §2.2), SATMC is able to analyse the Kerberos protocol reaching the fixed-point after the exploration of 15 steps in less than 20 seconds. For what concerns the AAAMobileIP protocol, we gain three orders of magnitude in the time spent and one order of magnitude in the size of the propositional

Table 1: Experimental data using SATMC with compound types disabled and enabled.

	No Compound Types				Compound Types			
Protocol	**K**	**Atoms**	**Clauses**	**Time**	**K**	**Atoms**	**Clauses**	**Time**
AAAMobileIP	10	7,849	30,061	262.95	9*	630	1,743	0.14
KerberosV	5	-	-	-	15*	30,094	186,379	19.55

Legenda:

-	:	memory out
*	:	fixed-point reached after the number of steps indicated

formula generated and the fixed-point is reached after 9 steps (without compound types, SATMC does not reach the fixed-point after 10 steps).

These preliminary experimental results confirm the effectiveness compound typing and show that it allows back-ends like SATMC to scale up to protocols of industrial complexity where un-decryptable messages characterised by a complex structure are frequently exchanged between participants.

The correctness of compound typing is obvious as it excludes computations and therefore any attack that exists in the model with compound typing also exists in the model without it. As far as completeness is concerned, preliminary work in this direction indicates that compound typing is indeed attack-preserving (that is, if there is an attack on the protocol in the model without compound typing, then there is also an attack in the model with compound typing) for protocols satisfying the following two conditions: *(i)* they work under the free-algebra assumption, and *(ii)* every time a principal receives a message of a compound type that he cannot decrypt, then such a message is simply forwarded in cleartext to a third party and it has no further relevance to that receiver. One can immediately see that the Kerberos protocol belongs to this class. In addition, the Kerberos protocol is free of loops[4] and therefore, as soon as the fixed-point is reached, SATMC can conclude the analysis by stating that in the considered scenario the Kerberos protocol is secure (see the forthcoming Deliverable 4.5 [8]). On the other hand, since the AAAMobileIP protocol does not satisfy condition *(ii)*, we cannot conclude that it is secure, as attacks might have been excluded by the compound typing restriction. However, we conjecture that the above results can be extended to protocols like AAAMobileIP and we are currently working in this direction.

[4]A protocol is free of loops if and only if a single execution of the protocol can not involve an unbounded number of steps of any role.

3 Channel Assumptions

It is important that the AVISPA tool supports different models of an intruder's capabilities. One of the critical capabilities of an intruder is the way in which he can influence communication between agents, and the way in which this information influences him. This section discusses the motivation for having different types of channels, presents the new HLPSL syntax for specifying these channels, explains the semantics behind the channel types, and finally provides a sample specification which makes use of some of the channel types.

3.1 Motivation

Formal analysis of cryptographic protocols is usually based on the Dolev-Yao intruder model [26]. This model describes an intruder who has complete control over the network, and who is only restricted by cryptography, which is assumed to be perfect. A Dolev-Yao intruder can read all traffic, block or redirect any message, and compose or decompose messages within the restrictions of perfect cryptography.

There are many situations where the Dolev-Yao intruder model is inappropriate (see, for instance, [24]). A number of protocols assume a weaker intruder model and are not designed to provide security against such a strong intruder. Other protocols assume some level of security is provided by mechanisms beyond the scope of the protocol. For example an application level protocol might assume that TLS is being used to provide a private channel. The assumptions made about the communication channels differ in subtle ways and often it is not explicitly stated with respect to which intruder model a protocol is secure. In order to analyse protocols in terms of their assumptions about their environment, different types of channels with different properties are required. The properties of these channels define the intruder model for each channel and allow flexible specifications where more than one type of intruder can be considered.

If model checking is to be integrated into the design phase of protocol standardisation bodies, it will be useful for designers to be able to experiment with different environmental assumptions. This will allow them to investigate assumptions which they can relax, and also to identify assumptions which should be explicitly stated. An example of this is the so-called ASW protocol [1]. When this protocol was designed, the authors required that the communication channels between the trusted third party and each participant were confidential. Subsequent analysis [38] found that this assumption could be relaxed without compromising the security requirements of the protocol.

Supporting different types of communication channels is important for the AVISPA tool. It increases the variety of protocols which can be modelled. This makes the AVISPA tool truly state-of-the-art in terms of scope, because most other security protocol analysis tools are limited to the Dolev-Yao intruder model. Indeed, many tools have the Dolev-Yao intruder built-in to the analysis technique, making the support of different intruder models impractical.

Experimenting with different assumptions during the design phase will be useful to protocol developers and will improve the quality of protocol specifications by encouraging explicit statements about the assumptions a protocol makes. This support will facilitate the migration of the AVISPA tool into industrial standardisation organisations such as the IETF.

3.2 Additions to HLPSL

Channels are declared as described in Deliverable 2.1 [2] with the intruder model attribute in parentheses. For example:

```
SND : channel (confidential)
```

In order to support new types of channels, channel parameters must be introduced. These are necessary for channel models which do not allow the intruder to decide the destination of every message. To write to a channel, the syntax is

```
SND(A,B;Msg)
```

where `SND` is a channel passed to the role as an argument, `A` and `B` are arbitrary values and `Msg` is any message. The semi-colon is used to separate the parameters of the message from the message itself. The receive action which is supposed to correspond to this send will have matching parameters, for example:

```
RCV(A,B;Msg)
```

The parameters of send and receive actions give us a binding between sender and receiver which may or may not be honoured by the network, depending on the type of the channel.

The types of channels proposed for HLPSL are as follows:

- Dolev-Yao channels `(dy)`: All messages are sent to the intruder, who may redirect or modify them arbitrarily [26].
- Confidential channels `(confidential)`: Whenever you send something on a confidential channel `SND(a,b;msg)` you may be sure that the message will only be read by `b`.
- Authentic channels `(authentic)`: Whenever you receive something on an authentic channel `RCV(a,b;msg)` you may be sure that the message was sent by `a` and the tuple `(a,b;msg)` was not modified. In particular, the message was sent on *this* channel.
- Location-limited channels `(location_limited)`: Whenever you send something on a location-limited channel `SND(a,b;msg)` you may be sure that the message will only be readable by `b` and the tuple `(a,b;msg)` will not be modified. See [9].

3.3 Semantics

The precise definition of the formal semantics of the different channel types, as well as their incorporation into the AVISPA tool, is work in progress. In this subsection, however, we give an intuition into the intended semantics.

Each channel has a state, also referred to as its knowledge. When a message is sent to a channel, its knowledge is updated accordingly. Depending on the type of the channel, the knowledge of the intruder may be updated during either send or receive actions, or not at all.

The parameters used for sending and receiving on a channel link the sender and the intended recipient together. The only way a message `msg` on a channel with certain parameters `param` can be received is if the channel knows the tuple `(param;msg)`, or if the intruder knows `(param;msg)` and the channel model allows the intruder to send on it.

If the intruder does not know the parameters of a channel, he may not read or write on this channel. This is independent of the intruder model of this channel. For example, if the intruder does not know the parameters of a Dolev-Yao channel, he may not send or receive on it. This corresponds to knowledge of the agents participating in a protocol session. For instance, the intruder cannot send a message to A with a from address of B unless he *knows* A and B.

Alternatively, even if the intruder knows the parameters of a confidential channel, he may not receive on it. He may only send or receive on a channel if he knows the channel's parameters *and* the channel is of a type that allows this. Other agents may also not send or receive on a channel unless they know its parameters. The pi-calculus is based on similar behaviour [32, 33]. This semantics allows agents (or the intruder) to dynamically learn other agents' names for future correspondence.

Channels which *guarantee* the delivery of messages have not been defined. They require fairness constraints, which are currently not supported by the AVISPA tool. [39] shows how certain fairness constraints can be modelled as safety properties, thus mitigating the need for fairness constraint support. However this approach does not apply to all types of fairness constraints and the value of introducing support for fairness constraints is currently under discussion.

3.4 A Sample Specification

The Purpose-Built Keys framework [13] describes a protocol for verifying that the source of successive messages does not change. The protocol does not rely on any public key infrastructure, but rather attempts to "improve overall security by narrowing the window of vulnerability". The initial exchange of the protocol is vulnerable to a man-in-middle attack, however if this exchange is completed successfully, further messages received by the initiator can be verified as coming from the original source. It is useful to model this protocol using two types of channels: a confidential channel for the initial exchange and a Dolev-Yao channel for all other messages.

The Purpose-Built Key framework in Alice&Bob notation is:

```
A -> B: IP_A, PK_A, hash(PK_A)
A -> B: {Msg}inv(PK_A), hash(PK_A)
B -> A: Nonce
A -> B: {Nonce}inv(PK_A)
```

Where PK_A is a public key generated by A, inv(PK_A) is the private key corresponding to PK_A, hash is a one-way function, IP_A is A's IP address, Msg is arbitrary text, and Nonce is a random number generated by B.

An HLPSL specification of the Purpose-Built Keys protocol is provided below. A confidential channel is used to protect the initial message.

```
role Alice (
    A,B         : agent,
    SND,RCV     : channel (dy),
    SND_S       : channel (confidential),
    Hash        : function,
    IP_A        : text
) played_by A def=

  local
    State       : nat,
    PK_A        : public_key (fresh),
    Msg         : text (fresh),
    Nonce       : text

  init
    State = 0

  transition

 1. State  = 0 /\ RCV(i,A;start) =|>
    State' = 2 /\ SND_S(A,B;IP_A.PK_A'.Hash(PK_A'))
 2. State  = 2 /\ RCV(i,A;start) =|>
    State' = 4 /\ SND(A,B;{Msg'}inv(PK_A).Hash(PK_A))
               /\ witness(A,B,msg_id,Msg')
 3. State  = 4 /\ RCV(B,A;Nonce') =|>
    State' = 6 /\ SND(A,B;{Nonce'}inv(PK_A))

end role

role Bob (
    B,A         : agent,
    SND,RCV     : channel (dy),
    RCV_S      : channel (confidential),
    Hash       : function

) played_by B def=

  local
    State      : nat,
    Nonce      : text (fresh),
    Msg        : text,
    PK_A       : public_key,
    IP_A       : text

  init
    State = 1

  transition

 1. State  = 1 /\ RCV_S(A,B;IP_A'.PK_A'.Hash(PK_A')) =|>
    State' = 3
 2. State  = 3 /\ RCV(A,B;{Msg'}inv(PK_A).Hash(PK_A)) =|>
    State' = 5 /\ SND(B,A;Nonce')
 3. State  = 5 /\ RCV(A,B;{Nonce}inv(PK_A)) =|>
    State' = 7
           /\ request(B,A,msg_id,Msg)
end role

role Session (
    A,B         : agent,
    SND,RCV     : channel (dy),
    SND_S,RCV_S : channel (confidential),
    Hash        : function,
    IP_A        : text
) def=

  composition
    Alice(A,B,SND,RCV,SND_S,Hash,IP_A)
 /\ Bob(B,A,SND,RCV,RCV_S,Hash)

end role

role Environment() def=

  const
```

```
    a,b        : agent,
    snd,rcv    : channel (dy),
    snd_s      : channel (confidential),
    rcv_s      : channel (confidential),
    hash       : function,
    msg_id     : protocol_id,
    ip_a       : text

  knowledge(i) = {a,b,i,hash,ip_a,snd,rcv}

  composition
    Session(a,b,snd,rcv,snd_s,rcv_s,hash,ip_a)
 /\ Session(i,b,snd,rcv,snd_s,rcv_s,hash,ip_a)

end role

Environment()
```

The PBK example shows how we can easily specify protocols which are based on more than one intruder model. In the example, we used confidential channels for the initial exchange, then Dolev-Yao channels after that. The ability to model different intruder capabilities within a single specification is useful in a variety of situations. [24] describes a number of such scenarios.

4 Oracle Rules

4.1 Introduction

In previous works, we have already described how the deduction power of the intruder could be extended using additional deduction rules as long as these rules follow some patterns. We have given the generic name of *Oracle rules* to these additional deduction rules. They permit us to extend the intruder's deduction power in various directions. It is possible, for example, to consider algebraic properties of some operator such as the *Exclusive-or* or the exponential (see [17, 18] for more details) or even properties of block-encryption [17]. These extensions are already described in Deliverable 2.2 [3].

Another direction is to extend intruder's deduction power in order to *verify* protocols. This direction was first described in [20], and its complexity is given in [19]. The rational behind this use is to *accelerate* the execution of an unbounded number of sessions. This permits us to over-approximate the deductions the intruder can make by interacting with other principals than those currently under attack. This acceleration is among the few that permit to handle goals such as strong authentication while having a finite system. These rules are more precisely described in Deliverable 5.1 [6].

Table 2: Intruder Rules

	Decomposition rules	Composition rules
Pair	$\mathrm{L_{p1}}(\langle a,b\rangle)$: $\langle a,b\rangle \to a$ $\mathrm{L_{p2}}(\langle a,b\rangle)$: $\langle a,b\rangle \to b$	$\mathrm{L_{dy,c}}(\langle a,b\rangle)$: $a,b \to \langle a,b\rangle$
Asymmetric	$\mathrm{L_{ad}}(\{a\}^p_K)$: $\{a\}^p_K, K^{-1} \to a$	$\mathrm{L_{dy,c}}(\{a\}^p_K)$: $a, K \to \{a\}^p_K$
Symmetric	$\mathrm{L_{sd}}(\{a\}^s_b)$: $\{a\}^s_b, b \to a$	$\mathrm{L_{dy,c}}(\{a\}^s_b)$: $a, b \to \{a\}^s_b$
Guess	$\mathrm{L_{O,d}}(a)$: $E \to a$ with a subterm of E	$\mathrm{L_{O,c}}(a)$: $E \to a$

In this deliverable we will focus on the use of these Oracle rules to the handling of low-level cryptographic properties. More precisely, after recalling the definition of Oracle rules in the next subsection, we present how to apply them to the detection of the so-called *Low-exponent attacks* which occur in poorly implemented but well-spread implementations of the RSA encryption scheme [22].

4.2 Definition of the Intruder's Rules

In this subsection the intruder's deduction power is expressed by rewrite rules on ground terms. In contrast with the symbolic representation considered e.g. for the lazy intruder, this permits a finer complexity analysis of problems.

The rewrite rules in Table 2 model the deduction power of the intruder. The operations taken into account are concatenation ($\langle a,b\rangle$), symmetric encryption ($\{a\}^s_b$) and asymmetric encryption ($\{a\}^p_b$). We have also added in this table *guess rules* which are Oracle rules if they satisfy the conditions of Definition 1.

We gather the rewrite rules in the following sets:

- $\mathrm{L_{dy,d}}(t) := \mathrm{L_{p1}}(t) \cup \mathrm{L_{p2}}(t) \cup \mathrm{L_{ad}}(t) \cup \mathrm{L_{sd}}(t)$ for every message t. In case, for instance, $\mathrm{L_{p1}}(t)$ is not defined, i.e., the head symbol of t is not a pair, then $\mathrm{L_{p1}}(t) = \emptyset$; analogously for the other rule sets,
- $\mathrm{L_{dy,d}} := \bigcup_a \mathrm{L_{dy,d}}(a)$, $\mathrm{L_{dy,c}} := \bigcup_a \mathrm{L_{dy,c}}(a)$,
- $\mathrm{L_{O,c}} := \bigcup_a \mathrm{L_{O,c}}(a)$, $\mathrm{L_{O,d}} := \bigcup_a \mathrm{L_{O,d}}(a)$,
- $\mathrm{L_O}(t) := \mathrm{L_{O,c}}(t) \cup \mathrm{L_{O,d}}(t)$, $\mathrm{L_O} := \mathrm{L_{O,c}} \cup \mathrm{L_{O,d}}$,
- $\mathrm{L_d}(t)$ is the set of all decomposition t-rules in Table 2, i.e., all t-rule in the left column of the table,
- $\mathrm{L_d} := \bigcup_a \mathrm{L_d}(a)$,
- $\mathrm{L_c}(t)$ is the set of all composition t-rules in Table 2.

- $\mathrm{L_c} := \bigcup_a \mathrm{L_c}(a)$.
- $\mathrm{L} := \mathrm{L_d} \cup \mathrm{L_c}$.

Given a system of rewrite rules L, we say this system is *local* if, in order to know whether a term t can be deduced from a set of terms E, it suffices to consider the subterms of t or E. If a rewriting rules system L is local, reachability of t from E can be decided easily provided it can be decided whether a rule can apply on a finite set of terms.

4.3 Oracle rules

Oracle rules are by definition extensions of the set $\mathrm{L_{dy}}$ of intruder's deduction rules for the Dolev-Yao intruder. In the next definition, I stands for the name of their intruder. It has the particularity of being member of all sets of terms considered.

Definition 1. *Let* $\mathrm{L_O} = \mathrm{L_{O,c}} \cup \mathrm{L_{O,d}}$ *be a (finite or infinite) set of Guess rules, where* $\mathrm{L_{O,c}}$ *and* $\mathrm{L_{O,d}}$ *denote disjoint sets of composition and decomposition Guess rules, respectively. Then,* $\mathrm{L_O}$ *is a* set of oracle rules *(w.r.t.* $\mathrm{L_{dy,c}} \cup \mathrm{L_{dy,d}}$ *as defined above) iff:*

1. *The system L of rewriting rules is* local.
2. *If $F \rightarrow_{\mathrm{L_{O,c}}(t)} F, t$ and $F, t \rightarrow_{\mathrm{L_{dy,d}}(t)} F, t, a$, then there exists a derivation D from F with goal a such that* $\mathrm{L_{dy,d}}(t) \notin D$.
3. *For every rule $F \rightarrow s \in \mathrm{L_{O,c}}(s)$, every proper subterm of s is a subterm of F.*
4. *For every non atomic message u, there exists a message $\epsilon(u)$ with $|\epsilon(u)| < |u|$ such that: for every finite set E of messages and every $F \subseteq E$, and for any message t, if $F \rightarrow u$ is in $\mathrm{L_c}$ and $E \rightarrow_{\mathrm{L_O}(t)} E, t$, then $t[u \leftarrow \epsilon(u)] \in \overline{E[u \leftarrow \epsilon(u)]}^{\mathrm{L}}$ and $\epsilon(u) \in \overline{E}^{\mathrm{L}}$.*

The first condition permits us to ensure that ground reachability is decidable. The second condition is rather technical. Informally, it means that the result of composition rules is the application of a constructor f on subterms of the left-hand side. The third and fourth conditions are also technical, and are only used to prove maximal bounds on the size of minimal attacks.

4.4 Application: Low Exponent Attacks

First, let us give a brief survey of what low-encryption attacks on RSA are. In RSA, messages are interpreted as numbers *modulo* a positive and large integer N. RSA encryption and decryption are simple exponentiations modulo N. The public key d and the private key e are numbers to which the message to be encrypted is exponentiated. Given a message (a number modulo N) m, the encryption of m by d is m^d modulo N. The message m^d is decrypted by computing $(m^d)^e$ modulo N. The numbers d and e are chosen such that the latter is equal to m modulo N. The relation between d and e can be expressed as follows.

The Euler function φ permits us to compute a number $\varphi(N)$ such that a public/private key pair (d, e) verifies:

$$d \cdot e \equiv 1 \mod \varphi(N)$$

The security of RSA encryption is based on the facts that the only known way to compute $\varphi(N)$ from N is to know the factors of N, and that to compute these factors from N is difficult. Thus without knowing $\varphi(N)$, which is supposed to remain a secret to the intruder, there is no easy way to compute the private key e knowing N and d.

This situation has led to software that carefully chooses a random number N for the modulus in such a way that $\varphi(N)$ is known[5], and then computes two integers d and e such that $d \cdot e \equiv 1$ modulo $\varphi(N)$. Since the actual value of d is of no importance, it is often chosen in software as a low integer, such as 3 or 5. This choice permits to speed up encryption, and does not, from what is described above, introduce any security flaw.

However, Franklin and Reiter [22] showed that if several messages are sent encrypted by the same public key d, with d a small integer, and if there is a known polynomial relationship between two messages m_1 and m_2, it is possible to recover both m_1 and m_2 from m_1^d and m_2^d. The value used for d in PGP, 17 is *not* out of reach of an intruder with a large computing power. In this case, an extrapolation [27] shows that 512-bits encryption can be broken in 130 years on a single Pentium III-850MHz processor.

In our setting, it is hard to express the existence of a polynomial relationship between two messages. While the probability is low for two arbitrary messages it gets higher when concatenating messages. Consider three messages a, b and c of length l_a, l_b and l_c. Moreover, let us assume the total length $l = l_a + l_b + l_c$ is smaller than the size of an encryption block (512 bits, for example). Let m be the concatenation of the three messages a, b and c. Noting $N(x)$ the interpretation of a message x as an integer, we have:

$$\begin{cases} m = a, b, c \\ N(m) = 2^{l_b + l_c} a + 2^{l_c} b + c \end{cases}$$

A special case of the above mentioned polynomial relationship is when there exists a known constant c such that $m_2 = m_1 + c$. This case may occur in two different situations.

First, it may happen that a same message m is sent twice with two different timestamps. This can occur, *e.g.*, when an encrypted message is sent to a mailing list. The messages sent in this case are: $\{m, t_1\}_d^p$ and $\{m, t_2\}_d^p$. Assuming the intruder can know the small interval t between the construction of the two messages, he can deduce the message m. This situation is modelled by rules:

$$\{m, t_1\}_d^p, \{m, t_2\}_d^p \to m \qquad t_1 \neq t_2$$

One easily checks that such rules are Oracle decomposition rules, and thus can be expressed in our setting.

Second, a more general situation is when a same message m is sent twice with the same key, but with different endings n_1 and n_2. If these endings are known to the intruder, it

[5]Usually, N is chosen as the product of two primes p and q. In this case, $\varphi(N) = (p - 1) \cdot (q - 1)$

can easily computes the difference between messages $m_1 = m, n_1$ and $m_2 = m, n_2$. The low exponents attack can then be expressed as:

$$\{m, n_1\}_d^p, \{m, n_2\}_d^p, n_1, n_2 \rightarrow m \qquad n_1 \neq n_2$$

Finally as shown by the above equation, the case where the message m is in the middle of two other messages, and is sent several times encrypted with these two messages different, can again be modelled by Oracle rules.

4.5 Use of Oracle rules

The above example seem to generalise to a wide range of low-level cryptographic properties. We believe a joint work with cryptographers on actual implementation of the protocols will permit us to express possible additional deduction rules of the intruder. Though this remains future work, we also believe that it will permit a reliable verification of cryptographic protocols taking into account the concrete cryptographic primitives used for implementing them.

5 The Guessing Intruder

5.1 Introduction

The Achilles' heel of a great many security measures lies in the simple fact the users tend to choose poor passwords [34]: that is, ones that are easy to remember and accordingly easy to guess (English words, for instance). This can give rise to *guessing attacks* in which an intruder is able to guess a password and then somehow verify that his guess is correct. Indeed, protocols like those of the EKE family [11, 12] have already been devised in an attempt to provide resilience to such guessing or dictionary attacks.

Guessing attacks come in two variants: *online* and *offline*. Online guessing involves the intruder actively sending messages whose contents are based on yet unverified guesses. Systems often provide simple resilience mechanisms to online guessing: automatic bank machines, for instance, may confiscate the customer's bank card if he or she enters the wrong PIN three times in a row. Formal analysis considering on-line guessing has been performed for instance in [14]. In this deliverable, we focus on offline guessing. We intend to examine online guessing in future work. Here, the intruder is able to verify his guess offline by comparing it to values in a dictionary which, with high probability, contains the correct value for the guess.

Several works consider off-line guessing, starting with Lowe's paper [30] which inspired [21] and [23]; the most advanced approach to date is probably [25]. However, about all these works we felt somewhat unsatisfied. First, [30] and [23] both deal with only one guess at a time; this may be interesting for falsification (quickly detecting some guessing attacks), but it does not provide a reasonable basis for verifying that a given protocol is invulnerable to guessing. Second, all these approaches present sets of rules that describe how the intruder can derive new messages. It is not at all clear that these rules correctly and completely describe guessing in a certain sense; in particular, none of the cited approaches presents a formal meta-model with respect to which the correctness and completeness of the approach could be proven; rather they *define* guessing by simply giving a set of intruder inference rules. Third, the inference rules in all these approaches are very tricky: it must somehow be ensured that the intruder really has two *different* ways of deriving a message in order to verify his guesses. In other words, one must prevent trivially false derivations of the intruder; this involves either restrictions as in [30] or very complicated conditions like proofs normalised in a certain way [25]. To further complicate matters, adding algebraic properties like the properties of exponentiation adds yet other ways to derive guesses and we see no way to adapt the existing approaches in order not to allow false derivations in these cases, making them unusable for protocols like EKE2 or SPEKE which use Diffie-Hellman exponentiation in order to be invulnerable for guessing.

Our impression is, however, that there is a common intuition about guessing, which appears far simpler than the cited approaches. Our approach is thus to first describe this intuition as precisely as possible in §5.2. We then try to formalise this intuition by a (meta-) model based on dictionaries in §5.3. This gives us a basis to describe Dolev-Yao-style derivation rules for the intruder in §5.4 and to show the correctness and completeness of these rules with respect to formal model (which may be considered as a kind of "semantics for guessing").

In §5.5, we give example derivations using our model of guessing. Finally, in §5.6, we show how our rule system (which allows infinitely many derivations) can be deployed in practice by integrating it into the constraint-based method of the lazy intruder. Implementation of this in the AVISPA tool is still work in progress, but we are confident that it will prove very useful in practice. In this deliverable, however, we place the emphasis on the theoretical foundations underlying our model of guessing.

5.2 Intuition

As is the case in the standard Dolev-Yao model, we assume that the intruder cannot break cryptography; he can, however, attempt to guess poorly chosen passwords. The intuition behind this is that such passwords appear (at least with a high probability) in a dictionary (a set of messages like passwords) that the intruder possesses. We call the messages in the dictionary the *guessable* messages.

Here, our first clash of intuitions arises: if we assume that the intruder "knows" everything that is "guessable" (i.e. if there is no distinction between guessable and known), then this is like assuming that the intruder will always guess correctly — which is clearly not our intuition. The dictionary of guessable messages is thus a different kind of knowledge than the "standard" intruder knowledge. In particular, for an encrypted message $\{|m|\}_k$, if k is guessable, then k occurs in the intruder's dictionary (so he knows k in a certain sense), but he does not know *which* entry in the dictionary is the right one — unless he finds a way to verify the correct guess.

We resolve this clash of intuitions by refining our notion of knowledge and precisely defining what it means to know the elements of a dictionary. The intruder possesses a dictionary and can be certain that every guessable value is contained within it. Furthermore, he *knows* all the entries in the dictionary by virtue of being able to look them up. However, this is a somewhat different way of knowing these data items than the classical way in which he knows a term like an agent name or a cryptographic key. In particular, his possession of the dictionary and knowledge of its entries does not imply that he knows *which* entry corresponds to a given guessable value. He must guess which one it is, and only if he has some way of verifying his guess will he be able to tell whether he has guessed correctly or not.

We take an illustrative example. Assume that the intruder is attempting to guess Alice's password and somehow knows that it is an English word. We further assume, for the sake of example, that the intruder possesses an exhaustive English language dictionary that contains all English words without exception. Given such a dictionary, he can be sure that Alice's password is contained within it. However, his possession of the dictionary and knowledge of its entries does not imply that he knows *which* of them Alice has chosen as her password. He must verify this guess in some way.

Verifying a guess is the difficult point (at least in all attempts to formalise this intuition), since we must precisely define what the intruder can and cannot do to verify a guess.

There are essentially two ways in which the intruder may verify a guess. Firstly, he may construct messages using the dictionary of guesses and then compare these messages with a message he has observed on the network, this observed message having been built using the correct value. For instance, if the intruder observes the exchange of the message $h(pw)$, the cryptographic hash of a guessable password pw, then he may attempt to hash each of the values in his dictionary of guessable values using the function h. Exactly one of the hashed dictionary entries produces the observed value $h(pw)$, and the intruder can thus verify that the respective entry in the dictionary is the correct value of pw.

The second way is to use the (hypothetical) guess to decrypt an encrypted message that contains a known subterm. For example, assume the intruder observes the exchanges of the message $\{|N|\}_{pw}$, the encryption of a nonce N (whose value he knows) with a guessable symmetric key pw. In this case, the intruder can attempt to decrypt the observed message with each of the guessable terms in his dictionary. Precisely one of these will yield the value he expects, N, and he can thus verify that this value is the correct key pw.

Intuitively, it seems clear what the intruder can and cannot do, but formalising this intuition requires ingenuity. Consider, for example, the message $\{|g_2|\}_{g_1}$ where both g_1 and g_2 are guessable. From this message, the intruder can learn both guesses, and this is supported by intuition: if the intruder attempts to decrypt this message with an incorrect key, it is unlikely that the result of the decryption will appear, by coincidence, in the dictionary.

Note that common intuition neglects minor probabilities to the disadvantage of the intruder as in the previous example: even if there is a small probability that the intruder is not able to perform the attack, the mere possibility that the attack *could* work with a certain probability is enough.

There is a further clash in the intuition with the normal Dolev-Yao model: we would like to assume that for an arbitrary message m, the intruder cannot learn whether m is an encrypted term or not by attempting to decrypt it with a guessed key. In particular, he should not get the answer "wrong key," indicating that m is indeed an encryption.

The normal DY analysis rules such as the rule for symmetric decryption,

$$\frac{\{|m|\}_k \in \mathcal{DY}(IK) \quad k \in \mathcal{DY}(IK)}{m \in \mathcal{DY}(IK)},$$

however suggest the existence of such a check, since, of course, there is no rule describing that the intruder learns some nonsense term when decrypting a message with an incorrect key, i.e. of the form

$$\frac{\{|m|\}_k \in \mathcal{DY}(IK) \quad k' \in \mathcal{DY}(IK)}{nonsense \in \mathcal{DY}(IK)}.$$

A considerable part of existing formalisms is just concerned with working around this tacit assumption of the standard Dolev-Yao model.

5.3 Formal Model

In this section, we introduce our formal model of a guessing intruder. We call our extension of the standard Dolev-Yao intruder the *guessing Dolev-Yao intruder* (DYG). Given an intruder knowledge IK, we define the messages derivable by the DYG intruder using a dictionary $\mathcal{D}$ as the smallest set $\mathcal{DYG}_{\mathcal{D}}(IK)$ closed under the rules we present in this section and the subsequent one. Naturally, all elements of IK itself are also elements of $\mathcal{DYG}_{\mathcal{D}}(IK)$. That is,

$$\frac{m \in IK}{m \in \mathcal{DYG}_{\mathcal{D}}(IK)}\ axiom.$$

We assume a set of simplifying algebraic equations on our term algebra of messages. To simplify the discussion, we focus here on pairing and encryption, though extending our set of equations to incorporate other operations (for instance, exponentiation or XOR) is not difficult. For the discussion here, we adopt the following set of equations:

$$\begin{aligned} \{\{M\}_K\}_{K^{-1}} &\approx M & (1)\\ (K^{-1})^{-1} &\approx K & (2)\\ \{|\{|M|\}_K|\}_K &\approx M & (3)\\ \pi_1(\langle m_1, m_2\rangle) &\approx m_1 & (4)\\ \pi_2(\langle m_1, m_2\rangle) &\approx m_2\ . & (5) \end{aligned}$$

Our algebra of message terms is then defined as the quotient algebra $\mathcal{T}/_{\approx}$. That is, the free algebra of message terms $\mathcal{T}$ modulo the equational theory defined by (1) - (5).

The standard Dolev-Yao model includes both synthesis rules for composing messages and analysis rules for decomposing them. Examples of the latter type are, for instance, decryption and the projection of a paired message into its two components. Given the equations above, we no longer need special analysis rules. Rather, we view everything (including decomposition of messages) as the application of an operator. Decryption is achieved using the encryption operator twice with corresponding keys as shown in (1) and (3). The projection of pairs is achieved by the application of the operators π_1 and π_2 as in (4) and (5). In this setting, we do not need to distinguish analysis from synthesis rules. The following intruder rule suffices:

$$\frac{m_1 \in \mathcal{DYG}_{\mathcal{D}}(IK) \quad \ldots \quad m_n \in \mathcal{DYG}_{\mathcal{D}}(IK)}{op(m_1, \ldots, m_n) \in \mathcal{DYG}_{\mathcal{D}}(IK)} \; compose \; (op \in \mathcal{O})$$

where $\mathcal{O}$ represents the set of operators that the intruder can perform, in this case $\mathcal{O} = \{\{\cdot\}_\cdot, \{|\cdot|\}_\cdot, \langle\cdot,\cdot\rangle, \cdot(\cdot), \pi_1(\cdot), \pi_2(\cdot)\}$. This rule plus the algebraic equations given above subsume the analysis rules of the standard Dolev-Yao intruder as described above. It is therefore straightforward to show that for all IK, $\mathcal{DY}(IK) \subseteq \mathcal{DYG}_{\mathcal{D}}(IK)$.

We now have "analysis rules" that can be applied even with incorrect decryption keys. The term resulting from a decryption attempt by the intruder, i.e. $\{|\{|m|\}_k|\}_{k'}$, can be rewritten to the corresponding plain-text m if and only if the intruder guessed correctly, i.e. $k' = k$. This models precisely the intuition behind guessing attacks. Furthermore, this gives us a straightforward way to deal with the problems that arise in the DY model that we discussed in the previous subsection. We need not, for instance, split cases and define $\pi_1(\langle m_1, m_2\rangle)$ in addition to defining $\pi_1(nonsense)$.

Note that, except for the perfect cryptography assumption and the assumptions on the intruder's control over channels, the capabilities of the intruder are defined by the selection of the set $\mathcal{O}$. We emphasise that, although we illustrate our technique by augmenting the standard DY intruder with guessing capabilities, the technique is in fact a general one that could be applied to alternate intruder models. One would simply need to define the set $\mathcal{O}$ accordingly and include equations corresponding to the operators contained therein.

Thus far, we have presented a relatively standard intruder model. We now augment this model with the construct necessary for guessing attacks: dictionaries, which we call *maps*. A map is a set of tuples with the following conditions. There is a message m, called the *pattern* of the map, which contains variables $v_1, \ldots, v_n$. The set $\{v_1, \ldots, v_n\}$ is called the *domain* of the map. Every tuple in the map has the form $((g_1, \ldots, g_n), m[v_1 \mapsto g_1, \ldots, v_n \mapsto g_n])$ where the vector $(g_1, \ldots, g_n)$ is an n-tuple of values from a dictionary $\mathcal{D}$. Moreover we require that any n-tuple of values in $\mathcal{D}$ appears as the first component of a map. A map thus describes a message-pattern which is parametrised over a set of things to be guessed, and the intruder's computation of the outcome of this message-pattern for every possible value of the guesses.

Maps are very expressive constructs; indeed, we can view all messages as maps. Messages with no guessed subterms are represented as maps with an empty domain (so they have just one entry $((), m)$). Moreover, the dictionary $\mathcal{D}$ itself is just a map with the domain v and the pattern v. This allows us to consider a formal model where the intruder knowledge consists only of maps.

The intruder can compose new maps from known ones as follows: for all k-ary operators $op \in \mathcal{O}$, i.e. operations the intruder can perform himself, and given k maps with domains

$dom_1, \dots, dom_k$ and message-patterns $m_1, \dots, m_k$, the intruder can generate a map with the domain $\bigcup_{i=1}^{k} dom_i$ and the message pattern $op(m_1, \dots, m_n)$.

We illustrate with the first example from the previous section. In this case, the intruder hashed each element of the dictionary to try to find one entry that agreed with the message he had observed, $h(pw)$. He thus generates the map $\{((pw'), h(pw')) \mid pw' \in \mathcal{D}\}$. Since we know that $pw \in \mathcal{D}$ and $h(M) \approx h(pw)$ iff $M \approx pw$, there is exactly one entry $(pw, h(pw))$ in the map. This entry corresponds to the correct guess of the value of pw.

The intruder can find this entry and verify his guess by comparing two maps. If two maps correspond on exactly one entry, then this entry must be the right guess (of all guesses that the maps depend on). For instance, in the example, if the intruder finds one entry $((pw'), h(pw'))$ in the map such that $h(pw) \approx h(pw')$, then he has verified that $pw' \approx pw$ and pw is indeed the correct guess. Once the intruder has verified a guess, he knows its value in the classical sense.

In general, verifying a guess requires that the intruder construct two maps that correspond on exactly one entry. Thus, there is one uniquely determined assignment of values to the variables in the domains of the two maps which yields corresponding entries. This assignment of values corresponds to the correct guesses, and after verifying his guesses, the intruder learns these correct values for the guesses in the classical sense. Note that this simple definition avoids any notion of "different ways to construct the same message" as is required in all other formal approaches to guessing. Rather the definition implicitly prevents that the intruder can "cheat himself": suppose he tries to verify something by taking two copies of the same map; then either the domain is empty (so there is nothing guessable to obtain) or there is more than one entry on which the two maps correspond (actually, they correspond on all entries).

The formal model presented here only allows derivations that are indeed possible according to the intuition given in the previous subsection; on the other hand, all operations the intruder could perform according to this intuition have a counter-part in the formal model, since composition and decomposition of messages with guesses is described by respective operations on maps, and verification of guesses is described by the comparison of maps. (But, of course, one can only informally justify that a formal model captures the intuition.)

5.4 Rule set

In this subsection, we present the concrete set of rules with which we augment the intruder model presented above in order to incorporate our formal model of guessing. This rule set extends the $\mathcal{DYG}_{\mathcal{D}}$ intruder model presented in the previous section and gives our representation of an intruder model that includes maps.

We introduce a symbolic representation for a guessed term. For a guessable password pw, we write $[pw]$. This represents the map of possible values for the yet unverified guess: that is, $\{((pw'), pw') \mid pw' \in \mathcal{D}\}$. $[\cdot]$ thus represents a new operator in our message algebra[6],

[6] We note, however, that $[\cdot] \notin \mathcal{O}$, as the intruder may only apply it to terms in $\mathcal{D}$.

giving a formal symbolic representation to our extension of message terms to include maps.

Firstly, the *guess* rule states that the intruder may attempt to guess any value that he knows to be guessable. More precisely, for any guessable value g, the intruder may construct a map $[g]$, representing his unverified guess for the value of g.

$$\frac{g \in \mathcal{D}}{[g] \in \mathcal{DYG}_{\mathcal{D}}(IK)} \; guess$$

The second rule allows the intruder to verify a set of guesses. As described above, the intuition here is that if the intruder can construct two maps that agree on precisely one value (that is, the value where all guessed subterms were guessed correctly), then he has verified that the entries in the dictionary that led to this value must be the correct values for the guessed subterms.

More formally, we say that the intruder has successfully verified a set of guesses if he can derive two messages m_1 and m_2 that will be equal, modulo our equational theory described above, iff all the guesses were correct. If so much as one guess was incorrect, then $m_1 \neq m_2$.

We first define *guesses*, the set of all guesses in a given term, recursively as follows.

$$\begin{aligned} guesses([g]) &= \{g\} \\ guesses(op(m_1, \ldots, m_n)) &= \bigcup_{i=1}^{n} guesses(m_i) \end{aligned}$$

We now define a predicate *compare* to capture the notion described above. Two terms m_1 and m_2 are comparable if they are equal, modulo our equational theory, precisely when all guesses in both terms are correct. That is, precisely when the set of correct guesses is exactly $guesses(\langle m_1, m_2\rangle)$. However, if the set of correct guesses is merely a proper subset of $guesses(\langle m_1, m_2\rangle)$, then $compare(m_1, m_2)$ should be false. Our definition makes use of an operator $m_{|G}$ which should be interpreted as the value of term m if we assume that all the guesses in set G are correct. The type of this operator is thus $Message \times 2^{\mathcal{D}} \to Message$. We recall that atomic terms containing no guesses are equivalent to nullary operators and are a degenerate instance of the second case of the definition we give here. Clearly, for any such an atomic message t without guesses, we have $t_{|G} = t$ for all G. Our definition of *compare* is then as follows:

$$\begin{aligned} compare(m_1, m_2) &\equiv m_{1|guesses(\langle m_1, m_2\rangle)} \approx m_{2|guesses(\langle m_1, m_2\rangle)} \\ &\quad \wedge \forall G \subset guesses(\langle m_1, m_2\rangle) : m_{1|G} \not\approx m_{2|G}. \\ [g]_{|G} &= \begin{cases} g & \text{if } g \in G \\ wrong(g) & \text{otherwise} \end{cases} \\ op(t_1, \ldots, t_n)_{|G} &= op(t_{1|G}, \ldots, t_{n|G}) \end{aligned}$$

where *wrong* is a fresh unary function symbol (with $wrong \notin \mathcal{O}$).

We can now give a formal rule for guess verification as shown here:

$$\frac{m_1 \in \mathcal{DYG}_{\mathcal{D}}(IK) \quad m_2 \in \mathcal{DYG}_{\mathcal{D}}(IK) \quad \mathit{compare}(m_1, m_2)}{\mathit{guesses}(\langle m_1, m_2 \rangle) \subseteq \mathcal{DYG}_{\mathcal{D}}(IK)}\ \mathit{verify}$$

If the intruder can verify his guesses by deriving two comparable maps, m_1 and m_2, then he learns, in the classical sense, the values of all guessed subterms in both. Note that this verification procedure inherently precludes the possibility that the intruder "cheats himself" by deriving two equivalent terms with which to verify his guesses. Assume, for instance, that we have $m_1 = m_2$. It is clear that, if all guesses are correct, we will have $m_1 \approx m_2$. However, if any of the guesses are incorrect, then we still have $m_1 \approx m_2$ and the condition $\mathit{compare}(m_1, m_2)$ is thus false. In such a case, the intruder may therefore not verify his guesses. This yields an intuitive formal model, and we need not introduce extra conditions to enforce that the intruder does not "cheat himself" as, for instance, in [30].

The main difference between this Dolev-Yao-style, rule-based model and the formal meta-model given in the previous section is that the maps are represented by the patterns they are based on and where the variables have already been replaced with the "correct" guesses in square brackets. Note that the meta-model does not contain a notion of right or wrong guesses (but rather of maps that correspond on only one entry). The guesses in square brackets of the rule-based model are thus a symbolic representation of the maps of the formal model, and we now give a proof sketch why these two models coincide.

First, observe that every term that can be constructed by using the composition rule *compose* with atomic messages in the intruder knowledge or from the *guess* rule correspond to maps the intruder can generate in the meta-model and vice-versa every map of the meta-model can be constructed using the rules. For correctness, everything the intruder can derive with the verification rule corresponds to two maps that coincide on exactly one entry. Vice-versa, for completeness, for two maps with exactly one corresponding entry, it can only be the "right" guesses according to the rule-based model: the only possible verification is based on guessing values that appear as submessages of the messages in the intruder knowledge (otherwise there can not be maps that correspond in only one entry). Thus, if there is only one correspondence between two maps, then this correspondence cannot involve "wrong guesses" as otherwise some guess does not play a role and consequently more entries correspond.

5.5 Examples

In this subsection, we present some examples of our technique. We begin with simple examples from [23] and [30], and conclude with a larger example based on one in [30]. We assume a single dictionary, $\mathcal{D}$, in the discussion of the examples below, and when we apply the *verify* rule of the previous section, it should be assumed that the precondition *compare* holds on the two messages m_1 and m_2.

Example 1: Assume the intruder has observed an execution of some protocol between honest participants and learned $IK = \{na, \{|na|\}_{pw}\}$; that is, a nonce na and this same nonce encrypted with a guessable password pw. Can the intruder verify a guess of the value of pw? We will show that he can indeed.

The intruder's first verifier can be generated as follows:

$$\dfrac{\dfrac{na \in IK}{na \in \mathcal{DYG}_{\mathcal{D}}(IK)}\ axiom \quad \dfrac{pw \in \mathcal{D}}{[pw] \in \mathcal{DYG}_{\mathcal{D}}(IK)}\ guess}{\{\!|na|\!\}_{[pw]} \in \mathcal{DYG}_{\mathcal{D}}(IK)}\ scrypt.$$

We note that the *scrypt* rule is simply an instance of the *op* rule given in §5.3. We adopt intuitive labels like *scrypt* and *apply* for the instances of this rule expressing symmetric encryption and function application, respectively.

His second verifier comes directly from his initial knowledge as shown here.

$$\dfrac{\{\!|na|\!\}_{pw} \in IK}{\{\!|na|\!\}_{pw} \in \mathcal{DYG}_{\mathcal{D}}(IK)}\ axiom$$

Based on these two values, the intruder can verify a guess of *pw* as shown here:

$$\dfrac{\{\!|na|\!\}_{[pw]} \in \mathcal{DYG}_{\mathcal{D}}(IK) \quad \{\!|na|\!\}_{pw} \in \mathcal{DYG}_{\mathcal{D}}(IK) \quad compare(\{\!|na|\!\}_{[pw]}, \{\!|na|\!\}_{pw})}{\{pw\} \subseteq \mathcal{DYG}_{\mathcal{D}}(IK)}\ verify.$$

This formalises the example presented in §5.2. Here, the intruder may verify his guess because the two terms $\{\!|na|\!\}_{pw}$ and $\{\!|na|\!\}_{[pw]}$ will be equal iff the intruder's guess for the value of *pw* is correct, and unequal otherwise. Thus, the precondition for the verification of guesses, $compare(\{\!|na|\!\}_{pw}, \{\!|na|\!\}_{[pw]})$, holds, and the intruder may learn the values of all guesses in both terms being compared (in this case, only *pw* is guessed).

Example 2: Here, we assume the intruder has learned $IK = \{\{\!|na|\!\}_{pw}, \{\!|\langle na, nb\rangle|\!\}_{pw}\}$, and that *pw* is guessable. We shows that the intruder can verify a guess of *pw*. His first verifier value is generated as follows:

$$\dfrac{\dfrac{\{\!|na|\!\}_{pw} \in IK}{\{\!|na|\!\}_{pw} \in \mathcal{DYG}_{\mathcal{D}}(IK)}\ axiom \quad \dfrac{pw \in \mathcal{D}}{[pw] \in \mathcal{DYG}_{\mathcal{D}}(IK)}\ guess}{\{\!|\{\!|na|\!\}_{pw}|\!\}_{[pw]} \in \mathcal{DYG}_{\mathcal{D}}(IK)}\ scrypt\ .$$

The second verifier is derived as shown here:

$$\dfrac{\dfrac{\dfrac{\{\!|\langle na, nb\rangle|\!\}_{pw} \in IK}{\{\!|\langle na, nb\rangle|\!\}_{pw} \in \mathcal{DYG}_{\mathcal{D}}(IK)}\ axiom \quad \dfrac{pw \in \mathcal{D}}{[pw] \in \mathcal{DYG}_{\mathcal{D}}(IK)}\ guess}{\{\!|\{\!|\langle na, nb\rangle|\!\}_{pw}|\!\}_{[pw]} \in \mathcal{DYG}_{\mathcal{D}}(IK)}\ scrypt.}{\pi_1(\{\!|\{\!|\langle na, nb\rangle|\!\}_{pw}|\!\}_{[pw]}) \in \mathcal{DYG}_{\mathcal{D}}(IK)}\ \pi_1$$

The intruder can then verify his guess of *pw* as illustrated in Fig. 1.

In this example, the algebraic equations presented in §5.3 come into play for the first time. Take, for instance, the left hand derivation tree. By equation 3, $\{\!|\{\!|na|\!\}_{pw}|\!\}_{[pw]} = na$ iff $[pw] = pw$: that is, if the intruder has guessed the value of *pw* correctly. Similarly, in the right hand derivation tree, $\pi_1(\{\!|\{\!|\langle na, nb\rangle|\!\}_{pw}|\!\}_{[pw]}) = na$ by equations (3) and (4) iff the intruder has guessed correctly. Therefore, the two will be equal iff the guesses involved (once again only pw) are correct, and not equal otherwise. We thus again have that *compare* of these two terms holds, and the intruder may verify his guess to learn pw.

Example 3: Consider the case in which $IK = \{\{\!|na|\!\}_{pw}\}$. Here, the intruder is not able to verify a guess of the value of na. There is simply not enough information to generate two comparable values which will be equal iff the intruder's guess is correct. We give a simple sketch of a proof by induction over the structure of messages to show this. It is clear that $\neg\ compare(\{\!|na|\!\}_{pw}, \{\!|na|\!\}_{pw})$. Since $\{\!|na|\!\}_{pw}$ is the only building block the intruder has, any messages he can generate will have the form $op(\{\!|na|\!\}_{pw})$ for some potentially nested message operation op. From the definition of *compare*, however, it follows immediately that

$$\neg\ compare(\{\!|na|\!\}_{pw}, \{\!|na|\!\}_{pw}) \rightarrow \neg\ compare(op(\{\!|na|\!\}_{pw}), op'(\{\!|na|\!\}_{pw}))$$

for arbitrary operations op and op'.

Example 4: Assume now that we have $IK = \{\{\!|\langle na, na\rangle|\!\}_{pw}\}$. The intruder derives his first potential verifier value as shown here:

$$\dfrac{\dfrac{\dfrac{\{\!|\langle na, na\rangle|\!\}_{pw} \in IK}{\{\!|\langle na, na\rangle|\!\}_{pw} \in \mathcal{DYG}_{\mathcal{D}}(IK)}\ axiom \quad \dfrac{pw \in \mathcal{D}}{[pw] \in \mathcal{DYG}_{\mathcal{D}}(IK)}\ guess}{\{\!|\{\!|\langle na, na\rangle|\!\}_{pw}|\!\}_{[pw]} \in \mathcal{DYG}_{\mathcal{D}}(IK)}\ scrypt}{\pi_1(\{\!|\{\!|\langle na, na\rangle|\!\}_{pw}|\!\}_{[pw]}) \in \mathcal{DYG}_{\mathcal{D}}(IK)}\ \pi_1 \quad .$$

In a symmetrical way, the intruder can derive $\pi_2(\{\!|\{\!|\langle na, na\rangle|\!\}_{pw}|\!\}_{[pw]})$. Given these, he can then verify a guess of the value of pw as shown in Fig. 2.

It is again clear that the two terms the intruder uses for verification will be equal iff the guessed value of pw was correct. This is in contrast to the previous example. Here, there is a structural property of the message that the intruder is able to exploit: namely, that the encrypted message should contain a pair whose two elements are the same.

Interesting to note here is that this example illustrates how our technique precludes the intruder from cheating himself as described in §5.4. For brevity, we abbreviate: let $t_1 = \pi_1(\{\!|\{\!|\langle na, na\rangle|\!\}_{pw}|\!\}_{[pw]})$. The intruder could attempt to generate a second verifier by constructing $t_2 = \pi_1(\langle t_1, t_1\rangle)$. Clearly, if the guess of pw is correct, we have $t_1 = t_2$. However, if the guess is wrong, then we have $t_1 = t_2$ all the same. Thus, the precondition $compare(t_1, t_2)$ does not hold, and the intruder may not verify his guess of pw in this way.

Example 5: A distinguishing feature of our approach to guessing is its ability to handle multiple guesses simultaneously. For this example, assume the intruder is trying to guess two passwords, pw_1 and pw_2, and has learned the set $IK = \{\{\!|k|\!\}_{pw_1}, f(\langle pw_2, k\rangle), f\}$.

The intruder now attempts to guess both pw_1 and pw_2 simultaneously. We thus have:

$$\dfrac{pw_1 \in \mathcal{D}}{[pw_1] \in \mathcal{DYG}_{\mathcal{D}}(IK)}\ guess \qquad \dfrac{pw_2 \in \mathcal{D}}{[pw_2] \in \mathcal{DYG}_{\mathcal{D}}(IK)}\ guess.$$

The intruder uses the first guess to construct the following encrypted term:

$$\dfrac{[pw_1] \in \mathcal{DYG}_{\mathcal{D}}(IK) \quad \{\!|k|\!\}_{pw_1} \in \mathcal{DYG}_{\mathcal{D}}(IK)}{\{\!|\{\!|k|\!\}_{pw_1}|\!\}_{[pw_1]} \in \mathcal{DYG}_{\mathcal{D}}(IK)}\ scrypt.$$

Using this, the intruder can derive his first verifier as follows:

$$\dfrac{[pw_2] \in \mathcal{DYG_D}(IK) \quad \{\!|\{\!|k|\!\}_{pw_1}|\!\}_{[pw_1]} \in \mathcal{DYG_D}(IK) \quad \dfrac{f \in IK}{f \in \mathcal{DYG_D}(IK)}\,axiom}{f(\langle [pw_2], \{\!|\{\!|k|\!\}_{pw_1}|\!\}_{[pw_1]}\rangle) \in \mathcal{DYG_D}(IK)}\,apply \quad .$$

For the second verifier, he simply uses the term $f(\langle pw_2, k\rangle)$ from his knowledge and verifies his guess as shown in Fig. 3.

Again, these two terms fulfil our verification criterion in that *compare* holds on them. In particular, one can see that *both* the guesses pw_1 and pw_2 must be correct in order for the two terms to be equal by our equational theory. If either or both is incorrect, then the two will be unequal. Verification, in this case, allows the intruder to learn the values of both the guesses.

Example 6: EKE. Here, we present a more involved example based on one in [30]. In [12], Bellovin and Merritt introduce the Encrypted Key Exchange protocol (EKE), which is intended to be secure against dictionary attacks. This protocol, in Alice & Bob notation, is shown below:

$$\begin{array}{lll}
1. & A \rightarrow B: & \langle A, \{\!|Pk|\!\}_{pw}\rangle \\
2. & B \rightarrow A: & \{\!|\{K\}_{Pk}|\!\}_{pw} \\
3. & A \rightarrow B: & \{\!|Na|\!\}_K \\
4. & B \rightarrow A: & \{\!|\langle Na, Nb\rangle|\!\}_K \\
5. & A \rightarrow B: & \{\!|Nb|\!\}_K
\end{array}$$

Here, pw is a guessable password shared by A and B, Pk is an asymmetric key whose inverse is possessed by A, and K is a symmetric session key. Lowe analysed EKE in [30] and found no guessing attacks. He then poses the interesting question as to whether or not Pk must be an asymmetric key, or if would suffice to use a symmetric key in its place. He finds that there would be an attack were this the case, and we show here how this attack can be derived in our model.

We assume the intruder has observed a run of the protocol between honest agents and we have $IK = \{\langle A, \{\!|Pk|\!\}_{pw}\rangle, \{\!|\{\!|K|\!\}_{Pk}|\!\}_{pw}, \{\!|Na|\!\}_K, \{\!|\langle Na, Nb\rangle|\!\}_K, \{\!|Nb|\!\}_K\}$.

Recall that we consider the hypothetical case in which Pk is a symmetric key. The intruder can attempt to obtain Pk based on a guess of pw as shown here:

$$\dfrac{\dfrac{pw \in \mathcal{D}}{[pw] \in \mathcal{DYG_D}(IK)}\,guess \quad \dfrac{\{\!|Pk|\!\}_{pw} \in IK}{\{\!|Pk|\!\}_{pw} \in \mathcal{DYG_D}(IK)}\,axiom}{\{\!|\{\!|Pk|\!\}_{pw}|\!\}_{[pw]} \in \mathcal{DYG_D}(IK)}\,scrypt \quad .$$

For brevity, let $\bar{Pk} = \{\!|\{\!|Pk|\!\}_{pw}|\!\}_{[pw]}$. Using this, the intruder may attempt to obtain K itself:

$$\dfrac{\bar{Pk} \in \mathcal{DYG_D}(IK) \quad \dfrac{[pw] \in \mathcal{DYG_D}(IK) \quad \{\!|\{\!|K|\!\}_{Pk}|\!\}_{pw} \in \mathcal{DYG_D}(IK)}{\{\!|\{\!|K|\!\}_{Pk}|\!\}_{[pw]} \in \mathcal{DYG_D}(IK)}\,scrypt}{\{\!|\{\!|\{\!|\{\!|K|\!\}_{Pk}|\!\}_{pw}|\!\}_{[pw]}|\!\}_{\bar{Pk}} \in \mathcal{DYG_D}(IK)}\,scrypt \quad .$$

Again, we introduce the abbreviation $\bar{K} = \{|\{|\{|\{|K|\}_{Pk}|\}_{pw}|\}_{[pw]}|\}_{\bar{Pk}}$. If the intruder's guess of pw is correct, then we have $\bar{Pk} = Pk$ and it follows that $\bar{K} = K$; that is, the intruder could find out the session key. Thus, this would represent a serious attack on the protocol.

The intruder can generate a first verifier value for his guess of pw as follows:

$$\frac{\bar{K} \in \mathcal{DYG_D}(IK) \quad \{|Na|\}_K \in \mathcal{DYG_D}(IK)}{\{|\{|Na|\}_K|\}_{\bar{K}} \in \mathcal{DYG_D}(IK)}\ scrypt \quad .$$

The second verifier can be generated based on message 4:

$$\dfrac{\dfrac{\bar{K} \in \mathcal{DYG_D}(IK) \quad \{|\langle Na, Nb\rangle|\}_K \in \mathcal{DYG_D}(IK)}{\{|\{|\langle Na, Nb\rangle|\}_K|\}_{\bar{K}} \in \mathcal{DYG_D}(IK)}\ scrypt}{\pi_1(\{|\{|\langle Na, Nb\rangle|\}_K|\}_{\bar{K}}) \in \mathcal{DYG_D}(IK)}\ \pi_1 \quad .$$

The *compare* precondition holds on these two verification terms: they are equal iff the guess of pw was correct. More specifically, when the intruder guesses right, then both are equal to Na. Therefore, the intruder has verified his guess, as Fig. 4 illustrates.

As discussed, this represents a serious guessing attack on our hypothetical variant of EKE. From this example, it is clear that Pk must indeed be an asymmetric key, as it is then impossible to obtain K from Pk without Pk^{-1}.

5.6 Extending the Lazy Intruder for Guessing

The guessing rules have a similar problem as the normal Dolev-Yao rules: the expansion of every non-empty initial intruder knowledge is infinite. The lazy intruder technique used both in the CL-Atse [40] and OFMC [10] back-ends can be used to address this kind of infinity by exploring in a demand-driven way the set of messages a Dolev-Yao intruder can generate from a given knowledge, resulting in a correct, complete, and terminating procedure for constraints over the Dolev-Yao model.

With guessing, we just add two additional intruder capabilities, namely that he may generate maps and use these maps to verify guesses, which are then added to his knowledge. The set of maps that the intruder can generate is infinite, but as in the standard case, we can explore this infinity (partially) in a demand-driven way. The verification of new guesses is bounded by the amount of constants appearing in the messages that are guessable.

We can thus describe the ability of guessing as an additional analysis rule of the intruder: each guess that he somehow verifies with whatever maps is an analysis step on his knowledge, adding a new guess. In particular, it is possible to normalise the intruder knowledge until no further messages can be learned or guesses verified. The maps produced play only a role during this analysis process (and may be removed from the intruder knowledge after analysis).[7]

[7]At least as long as we do not consider online guesses where such maps may occur in sent messages.

$$\dfrac{\begin{array}{l} \{\!|\{\!|na|\!\}_{pw}|\!\}_{[pw]} \in \mathcal{DYG}_{\mathcal{D}}(IK) \\ \pi_1(\{\!|\{\!|\langle na, nb\rangle|\!\}_{pw}|\!\}_{[pw]}) \in \mathcal{DYG}_{\mathcal{D}}(IK) \quad \mathit{compare}(\{\!|\{\!|na|\!\}_{pw}|\!\}_{[pw]}, \pi_1(\{\!|\{\!|\langle na, nb\rangle|\!\}_{pw}|\!\}_{[pw]})) \end{array}}{\{pw\} \subseteq \mathcal{DYG}_{\mathcal{D}}(IK)}\ \mathit{verify}$$

Figure 1: Derivation Tree for Example 2

$$\dfrac{\begin{array}{l} \pi_1(\{\!|\{\!|\langle na, na\rangle|\!\}_{pw}|\!\}_{[pw]}) \in \mathcal{DYG}_{\mathcal{D}}(IK) \\ \pi_2(\{\!|\{\!|\langle na, na\rangle|\!\}_{pw}|\!\}_{[pw]}) \in \mathcal{DYG}_{\mathcal{D}}(IK) \quad \mathit{compare}(\pi_1(\{\!|\{\!|\langle na, na\rangle|\!\}_{pw}|\!\}_{[pw]}), \pi_2(\{\!|\{\!|\langle na, na\rangle|\!\}_{pw}|\!\}_{[pw]})) \end{array}}{\{pw\} \subseteq \mathcal{DYG}_{\mathcal{D}}(IK)}\ \mathit{verify}$$

Figure 2: Derivation Tree for Example 4

$$\dfrac{\begin{array}{l} f(\langle [pw_2], \{\!|\{\!|k|\!\}_{pw_1}|\!\}_{[pw_1]}\rangle) \in \mathcal{DYG}_{\mathcal{D}}(IK) \\ f(\langle pw_2, k\rangle) \in \mathcal{DYG}_{\mathcal{D}}(IK) \qquad \mathit{compare}(f(\langle [pw_2], \{\!|\{\!|k|\!\}_{pw_1}|\!\}_{[pw_1]}\rangle), f(\langle pw_2, k\rangle)) \end{array}}{\{pw_1, pw_2\} \subseteq \mathcal{DYG}_{\mathcal{D}}(IK)}\ \mathit{verify}$$

Figure 3: Derivation Tree for Example 5

$$\dfrac{\begin{array}{l} \{\!|\{\!|Na|\!\}_{K}|\!\}_{\bar{K}} \in \mathcal{DYG}_{\mathcal{D}}(IK) \\ \pi_1(\{\!|\{\!|\langle Na, Nb\rangle|\!\}_{K}|\!\}_{\bar{K}}) \in \mathcal{DYG}_{\mathcal{D}}(IK) \quad \mathit{compare}(\{\!|\{\!|Na|\!\}_{K}|\!\}_{\bar{K}}, \pi_1(\{\!|\{\!|\langle Na, Nb\rangle|\!\}_{K}|\!\}_{\bar{K}})) \end{array}}{\{pw\} \subseteq \mathcal{DYG}_{\mathcal{D}}(IK)}\ \mathit{verify}$$

Figure 4: Derivation Tree for Example 6

The question if two maps can be constructed such that they can be verified and offer a guess that the intruder has not yet found out, can now be addressed as follows. We let the intruder "eagerly" create all maps that represent the analysis of messages using guesses. Then, for every map m_1, we try to "lazily" find all ways to construct a map m_2 for the same value (whatever guesses are used in that) and check whether or not this pair $\langle m_1, m_2 \rangle$ is indeed a verifier for the contained guesses, i.e. satisfies the predicate $\mathit{compare}(m_1, m_2)$.

For instance, consider the initial intruder knowledge $\mathit{IK}_0 = \{\{\vert g_2 \vert\}_{g_1}, h, h(g_3)\}$, where $g_1, g_2, g_3 \in \mathcal{D}$. First we close this set under all analysis steps (involving guessing) that are possible, adding the message $\{\vert\{\vert g_2\vert\}_{g_1}\vert\}_{[g_1]}$. Next, we generate for each message additional ways to obtain them:

m_1	m_2	$\mathit{compare}(m_1, m_2)$?
$\{\vert g_2\vert\}_{g_1}$	$\{\vert g_2\vert\}_{g_1}$	no
	$\{\vert\{\vert\{\vert g_2\vert\}_{g_1}\vert\}_{[g_1]}\vert\}_{[g_1]}$	no
$\{\vert\{\vert g_2\vert\}_{g_1}\vert\}_{[g_1]}$	$\{\vert\{\vert g_2\vert\}_{g_1}\vert\}_{[g_1]}$	no
	$[g_2]$	yes $\Rightarrow \{g_1, g_2\} \subseteq \mathcal{DYG}_{\mathcal{D}}(\mathit{IK}_0)$
$h(g_3)$	$h(g_3)$	no
	$h([g_3])$	yes $\Rightarrow g_3 \in \mathcal{DYG}_{\mathcal{D}}(\mathit{IK}_0)$
...	...	...

Note that for every term, it is trivially possible to take the term itself as a possible verifier, which, however, always results in the answer no. We thus define the new lazy intruder rule as (a) rules to obtain new maps that represent analysis using guesses and (b) rules that find for a given map other ways to generate a map for the same term. The formal definition of this extension to the lazy intruder is work in progress.

6 Conclusion

We have presented four techniques spanning three different classes of environmental assumptions. The compound typing approach described in §2 allows us to explicitly specify how honest principals interprets message parts that they cannot decrypt. Experimental results gathered to date show this technique yields a significant performance gain and can therefore extend the scope of tools like SATMC.

Alternative channel models, discussed in §3, are essential for modelling networks comprised of heterogeneous communication channels which may be characterised by different intruder models.

Oracle rules (§4) are a useful means of extending the intruder's deductive powers. As shown, this can be employed to better model fine-grained properties of cryptographic operations, bringing vulnerabilities to attacks like low-exponent attacks on RSA into the scope of our analyses.

Finally, §5 presents a novel approach to analysing guessing attacks in which the intruder exploits poorly chosen passwords. This models an important real-world vulnerability and improves on previous approaches in that it is simpler, more declarative, and closer to general intuition regarding guessing attacks. We have given a concise comparison with some of these other approaches here and intend to expand on this with a more detailed treatment in future work.

These techniques and others for modelling and reasoning about a variety of environmental settings in which a protocol may execute allow us to more faithfully model modern network architectures and the vulnerabilities to which they may be subject. Moreover, analysis under different sets of assumptions upon the environment can yield a more precise understanding of what security goals are ensured by a given protocol and under what circumstances.

References

[1] N. Asokan, V. Shoup, and M. Waidner. Asynchronous protocols for optimistic fair exchange. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 86–99, Oakland, CA, USA, May 1998. IEEE Computer Society Press.

[2] AVISPA. Deliverable 2.1: The High-Level Protocol Specification Language. Available at `http://www.avispa-project.org`, 2003.

[3] AVISPA. Deliverable 2.2: Algebraic Properties. Available at `http://www.avispa-project.org`, 2003.

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

[5] AVISPA. Deliverable 4.4: AVISPA tool v.1. Available at `http://www.avispa-project.org`, 2003.

[6] AVISPA. Deliverable 5.1: Abstractions. Available at `http://www.avispa-project.org`, 2003.

[7] AVISPA. Deliverable 6.1: List of selected problems. Available at `http://www.avispa-project.org`, 2003.

[8] AVISPA. Deliverable 4.5: AVISPA tool v.2. Available at `http://www.avispa-project.org`, 2004.

[9] D. Balfanz, D. K. Smetters, P. Stewart, and H. C. Wong. Talking to strangers: Authentication in ad-hoc wireless networks. In *Proceedings of Network and Distributed System Security Symposium 2002 (NDSS'02)*, 2002.

[10] D. Basin, S. Mödersheim, and L. Viganò. An On-The-Fly Model-Checker for Security Protocol Analysis. In E. Snekkenes and D. Gollmann, editors, *Proceedings of ESORICS'03*, LNCS 2808, pages 253–270. Springer-Verlag, 2003. Available at `http://www.avispa-project.org`.

[11] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *Advances in Cryptology – EUROCRYPT ' 2000*, LNCS 1807, pages 139–155. Springer-Verlag, 2000.

[12] S. M. Bellovin and M. Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 72–84, May 1992.

[13] S. Bradner, A. Mankin, and J. I. Schiller. A framework for purpose built keys (PBK). Internet draft, June 2003.

[14] E. Bresson, O. Chevassut, and D. Pointcheval. Security proofs for an efficient password-based key exchange. In *Proceedings of CCS'03*, pages 241–250, 2003.

[15] P. Calhoun, J. Loughney, E. Guttman, G. Zorn, and J. Arkko. RFC 3588: Diameter Base Protocol, Sept. 2003. Status: Proposed Standard.

[16] Common Authentication Protocol Specification Language. URL: `http://www.csl.sri.com/~millen/capsl/`.

[17] Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. An NP Decision Procedure for Protocol Insecurity with XOR. In *Proceedings of the Logic In Computer Science Conference, LICS'03*, pages 261–270, 2003. Available at `http://www.avispa-project.org`.

[18] Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. Deciding the Security of Protocols with Diffie-Hellman Exponentiation and Products in Exponents. In *Proceedings of the Foundations of Software Technology and Theoretical Computer Science, FST TCS'03*, LNCS 2914. Springer-Verlag, 2003. Available at `http://www.avispa-project.org`.

[19] Y. Chevalier, R. Küsters, M. Rusinowitch, M. Turuani, and L. Vigneron. Extending the Dolev-Yao Intruder for Analyzing an Unbounded Number of Sessions. In M. Baaz, editor, *Proceedings of CSL'2003*, LNCS 2803. Springer-Verlag, 2003. Available at `http://www.avispa-project.org`.

[20] Y. Chevalier and L. Vigneron. Automated Unbounded Verification of Security Protocols. In E. Brinksma and K. G. Larsen, editors, *Proceedings of CAV'02*, LNCS 2404, pages 324–337. Springer-Verlag, 2002.

[21] E. Cohen. Proving protocols safe from guessing. In *Proceedings of Foundations of Computer Security '02*, pages 85–92.

[22] D. Coppersmith, M. Franklin, J. Patarin, and M. Reiter. Low-exponent RSA with related messages. In *In* Proceedings of the Advances in Cryptology – Eurocrypt '96 Conference, volume 1070, pages 1–??, 1996. Long version available as IBM Research Report RC 20318, December 27, 1995.

[23] R. Corin, S. Malladi, J. Alves-Foss, and S. Etalle. Guess what? here is a new tool that finds some new guessing attacks (extended abstract). In R. Gorrieri and R. Lucchi, editors, *Proceedings of IFIP WG 1.7 and ACM SIGPLAN Workshop on Issues in the Theory of Security (WITS)*, pages 62–71. Dipartimento di Scienze dell'Informazione, Università di Bologna, Italy, 2003. `http://www.ub.utwente.nl/webdocs/ctit/1/000000b2.pdf`.

[24] S. Creese, M. Goldsmith, B. Roscoe, and I. Zakiuddin. The attacker in ubiquitous computing environments: Formalising the threat model. In *Proceedings of the First*

International Workshop on Formal Aspects in Security and Trust, pages 83–97, Italy, 2003.

[25] S. Delaune and F. Jacquemard. A theory of guessing attacks and its complexity. Research Report LSV-04-1, Lab. Specification and Verification, ENS de Cachan, Cachan, France, Jan. 2004.

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

[27] J. Dyers. Exponent attacks, lattice reduction algorithm. `http://math.arizona.edu/~ura/022/McCallum_group/`, May 2002.

[28] J. Heather, G. Lowe, and S. Schneider. How to prevent type flaw attacks on security protocols. In *Proceedings of The 13th Computer Security Foundations Workshop (CSFW'00)*. IEEE Computer Society Press, 2000.

[29] Kerberos: The Network Authentication Protocol. URL: `http://web.mit.edu/kerberos/www/`.

[30] G. Lowe. Some new attacks upon security protocols. In *Proceedings of The 9th Computer Security Foundations Workshop (CSFW'96)*. IEEE Computer Society Press, 1996.

[31] G. Lowe. Casper: a Compiler for the Analysis of Security Protocols. *Journal of Computer Security*, 6(1):53–84, 1998. See `http://web.comlab.ox.ac.uk/oucl/work/gavin.lowe/Security/Casper/`.

[32] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, i. *Inf. Comput.*, 100(1):1–40, 1992.

[33] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, ii. *Inf. Comput.*, 100(1):41–77, 1992.

[34] R. Morris and K. Thompson. Passwork security: A case history. *Comm. of the ACM*, 22(11):594, Nov. 1979.

[35] R. M. Needham and M. D. Schroeder. Using Encryption for Authentication in Large Networks of Computers. Technical Report CSL-78-4, Xerox Palo Alto Research Center, Palo Alto, CA, USA, 1978. Reprinted June 1982.

[36] B. C. Neuman and T. Ts'o. Kerberos : An authentication service for computer networks. Technical Report ISI/RS-94-399, USC/ISI, 1994.

[37] J. Raskin and S. Kremer. Game analysis of abuse-free contract signing. In *15th IEEE Computer Security Foundations Workshop*, pages 206–220. IEEE Computer Society Press, June 2002.

[38] V. Shmatikov and J. C. Mitchell. Analysis of a fair exchange protocol. In *Proceedings of the 1999 FLoC Workshop on Formal Methods and Security Protocols*, Trento, Italy, 1999.

[39] V. Shmatikov and J. C. Mitchell. Finite-state analysis of two contract signing protocols. *Theoretical Computer Science*, 283(2):419–450, June 2002.

[40] M. Turuani. *Sécurité des Protocoles Cryptographiques: Décidabilité et Complexité.* PhD thesis, Université Henri Poincaré, Nancy, December 2003.