Reasoning About Some Security Protocol Implementation Details revision 4

Alfredo Pironti and Riccardo Sisto

Politecnico di Torino
Dip. di Automatica e Informatica
c.so Duca degli Abruzzi 24, I-10129 Torino (Italy)
e-mail:{alfredo.pironti, riccardo.sisto}@polito.it
phone: +390115647073 fax: +390115647099

Abstract. In formal methods, security protocols are usually modeled at a high level of abstraction. In particular, data encoding and decoding transformations are often abstracted away. However, in real applications, errors in these protocol components could be exploited to break protocol security.
In order to address this issue, this paper formally proves that, under some constraints checkable on sequential code, if an abstract protocol model is secure under a Dolev-Yao intruder, then a refined model, which takes into account a wide class of possible implementations of the encoding/decoding operations, is implied to be secure too under the same intruder model. The paper also indicates possible exploitations of this result in the context of methods based on formal model extraction from implementation code and of methods based on automated code generation from formally verified models.

1 Introduction

In the last years, several techniques based on formal methods have been developed to analyze abstract models of security protocols. These models, initially introduced by Dolev and Yao [1], represent messages as instances of high level abstract data types. One question that arises is how to get assurance about the fact that the logical correctness of an abstract protocol is indeed preserved when concrete versions of the protocol are defined and when their implementations are developed using programming languages. In general, security faults not present in an abstract protocol specification might be introduced when adding implementation details.

Recently, some work has been started in the direction of bridging the gap that exists between abstract formal models and their concrete counterparts. For example, some researchers have been working towards refining the abstract formal models used for data types, so as to represent the additional properties of certain cryptographic operations, such as for example those based on the use of exponentiation or exclusive-or functions (e.g. [2]). Other researchers have been working at reconciling the abstract formal models of cryptography with computational models, which are closer to implementations (e.g. [3, 4]). Another research line, instead, focuses on the concrete implementation of the protocol logic, rather than on the concrete implementation of cryptography. The addressed problem is ensuring that the formal model of a protocol role used for protocol analysis is a safe abstraction of the protocol role implementation, under the assumption that the cryptographic and communication libraries used by the implementation behave as specified by their ideal (Dolev-Yao) abstract models. In this case, the final aim of the analysis is to discover, and hence avoid, errors in the protocol role implementation that may cause violations of security properties, regardless of the encryption scheme used, i.e. even under the assumption of perfect cryptography, and regardless of how the other protocol roles are implemented, i.e. even under the assumption that the other protocol roles are implemented correctly. For example, if one of the prescribed checks on one of the received messages is missing in the protocol role implementation, the latter will behave differently from the abstract protocol specification, possibly leading to a violation of some security property.

Two different strategies have been proposed in order to cope with this problem: automatic code generation from abstract models ([5, 6]) and automatic model extraction from implementation code ([7–10]).

Methods based on automatic code generation start from a high-level, formally verified, specification of the protocol, which abstracts away from details about cryptographic and communication operations and binary data representations, and fill the semantic gap between formal specification and implementation, guided by implementation choices provided by the user. To our knowledge, up to now no formal proof has been given that security properties are indeed preserved in the proposed refinement processes.

Methods based on automatic model extraction start from an already existing, full blown implementation code, from which an abstract model is extracted and formally verified. In this case, a formal soundness proof has been given for the method presented in [7].

One of the things that can be observed by looking at the results reported in [7], [8] and [10], is that the part of the extracted formal model that describes data encoding and decoding operations can be quite complex, much more complex than the abstract protocol model itself. This occurs even though in [7] and [10] the implementations of some low-level library operations, such as those for basic XML manipulation, are not included in the model but rather assumed to correctly refine their symbolic counterpart. Note that taking data transformations into account when a protocol implementation is analyzed is important, because the wrong implementation of such transformations may be responsible for security faults that can go undetected when they are abstracted away. For example, an incorrect implementation of a marshalling function could unwillingly leak secret data, or the function that encodes some data before applying a hash function to them could erroneously transform part of the data (e.g. a nonce) into a constant, thus enabling replay attacks. Note that these errors do not necessarily infringe interoperability, so they may be difficult to discover by testing.

The aim of this paper is to formally state and prove sufficient conditions under which the detailed models of data transformations, such as the ones extracted from protocol code in [10], can be avoided and substituted by much simpler models or assumptions, that can be checked on sequential code and in isolation (i.e. without considering the behavior of the intruder), while obtaining the same kind of security assurance on the protocol implementation.

The first step in our methodology is the definition of simple formal models of data encoding and decoding transformations. Such models are general, in the sense that they do not describe specific implementations, but rather they capture only some general assumptions that are being made on implementations. Verifying a concrete protocol implementation can thus be reduced to verifying that the protocol implementation fulfils the assumptions made about data encoding and decoding transformations and verifying that the protocol model built according to such assumptions satisfies the required security properties. This approach makes verification modular, according to an assume-guarantee style.

The second step that is made is to show that the models built in this way can be further simplified, using fault-preserving transformations like the ones introduced in [11]. The result that is finally obtained is that, under some further simple assumptions, the protocol models including implementation details can be transformed back into the original abstract protocol models without implementation details, and the classical security faults (i.e. violations of secrecy and authentication) are preserved in this transformation. This means that, provided all the assumptions we made are verified on a given implementation, the formal model of the implementation details can be safely abstracted away in verifying the desired security properties, under the Dolev-Yao modelling approach.

It is worth noting that this kind of result can be exploited both when using the model extraction approach and when using code generation. In the former case, the assumptions made on data encoding and decoding transformations must be checked on the (sequential) code that implements them. If they hold, this code can be safely abstracted during model extraction.

In the latter case, if the starting point is an already verified abstract protocol model, the results given in this paper formally prove that the same security properties still hold in a refined model where code is generated so as to satisfy our assumptions. Then, such assumptions can be regarded as requirements on how the code must be generated, and the formal proofs given in this paper can be used as a basis for proving the soundness of refinement processes in methods based on code generation.

The remainder of the paper is organized as follows. Section 2 introduces the notation and the modelling approach, based on CSP, that is used to reason on security protocols throughout the paper. Then, a distinction is made between two kinds of data encoding and decoding transformations, because they need different assumptions. Section 3 focuses on encoding and decoding transformations that are applied to the messages when they are sent and received on a communication channel. While showing the results, a generalization of the fault-preserving transformations introduced in [11] is also presented. Section 4 instead deals with the encoding of key material and of data on which cryptographic operations, such as encryption and hashing, are applied. Then, section 5 discusses about the application of the results, using as examples the protocols for secure web services and the

SSH transport protocol. Finally, section 6 concludes with an overview of the achievements that have been reached and their practical impact.

2 Abstract Protocol Models and Notation

The formalism used in this paper is based on CSP [12, 13], and the datatype definitions and protocol models are an extension of the ones used in [11]. Essentially, they follow the Dolev-Yao approach [1]. It is believable that the extended datatype proposed in this paper can be enough to abstractly model the most common security protocols. Nevertheless, further extensions or modifications can be made to the datatype. The results presented here will still be valid, provided the new datatype satisfies some properties explicitly stated in this paper.

The main extension that we introduce w.r.t. [11] is an added support for non-atomic keys. This extension enables modelling protocols where the key is constructed from non-atomic data. The new datatype is defined as

$$\begin{aligned}
Message ::= \; & \textsc{Atom}\ Atom \mid \\
& \textsc{Pair}\ Message\ Message \mid \\
& \textsc{ShKey}\ Message \mid \\
& \textsc{PubKey}\ Message \mid \\
& \textsc{PriKey}\ Message \mid \\
& \textsc{ShKeyEncrypt}\ Message\ \textsc{ShKey}\ Message \mid \\
& \textsc{PubKeyEncrypt}\ Message\ \textsc{PubKey}\ Message \mid \\
& \textsc{PriKeyEncrypt}\ Message\ \textsc{PriKey}\ Message \mid \\
& \textsc{Hash}\ Message.
\end{aligned}$$

This definition has been developed using the following guidelines:

- Each key is typed. It is possible to obtain a key from generic material (that is, any generic *Message*). It is not possible to use raw material directly as a key; instead, the material must first be fed to a key construction operator.
- There is no longer need (as in [11]) for the inverse K^{-1} of a key K. Indeed, the key construction operators PubKey and PriKey fulfil this role.
- No new types are added in order to represent encoding parameters or encoded data, because the idea is to have a single datatype that can be used to model protocol data at different detail levels.

In this paper, M, N, O and K range over *Message*, U and S over $2^{Message}$, A and B over honest protocol agents, P, Q and R over processes. When not explicitly quantified, all of these variables are assumed to be universally quantified over their assigned ranges.

In order to get better reading for processes, the following syntactic sugar is also provided:

Message	Representation
$\textsc{Pair}\ M\ M'$	(M, M')
$\textsc{ShKey}\ M$	$M^{\sim}$
$\textsc{PubKey}\ M$	M^{+}
$\textsc{PriKey}\ M$	M^{-}
$\textsc{ShKeyEncrypt}\ M\ \textsc{ShKey}\ K$	$\{M\}_{K^{\sim}}$
$\textsc{PubKeyEncrypt}\ M\ \textsc{PubKey}\ K$	$\{[M]\}_{K^{+}}$
$\textsc{PriKeyEncrypt}\ M\ \textsc{PriKey}\ K$	$[\{M\}]_{K^{-}}$
$\textsc{Hash}\ M$	$H(M)$

Once the datatype is defined, it is also necessary to update the intruder knowledge derivation relation $\vdash$ which models the intruder data derivation capabilities ($U \vdash M$ means that M can be derived from U). Eleven rules are defined for this new datatype:

member $M \in U \Rightarrow U \vdash M$
pairing $U \vdash M \wedge U \vdash M' \Rightarrow U \vdash (M, M')$
splitting $U \vdash (M, M') \Rightarrow U \vdash M \wedge U \vdash M'$
key derivation $U \vdash K \Rightarrow U \vdash K^{\sim} \wedge U \vdash K^{+} \wedge U \vdash K^{-}$

shared key encryption $U \vdash M \land U \vdash K^{\sim} \Rightarrow U \vdash \{M\}_{K^{\sim}}$
public key encryption $U \vdash M \land U \vdash K^{+} \Rightarrow U \vdash \{[M]\}_{K^{+}}$
private key encryption $U \vdash M \land U \vdash K^{-} \Rightarrow U \vdash [\{M\}]_{K^{-}}$
shared key decryption $U \vdash \{M\}_{K^{\sim}} \land U \vdash K^{\sim} \Rightarrow U \vdash M$
public key decryption $U \vdash \{[M]\}_{K^{+}} \land U \vdash K^{-} \Rightarrow U \vdash M$
private key decryption $U \vdash [\{M\}]_{K^{-}} \land U \vdash K^{+} \Rightarrow U \vdash M$
hashing $U \vdash M \Rightarrow U \vdash H(M)$

The following lemma about the relation $\vdash$ is needed:

Lemma 1.

$$U \vdash M \land U \subseteq U' \Rightarrow U' \vdash M, \tag{1}$$
$$U \vdash M \land U \cup \{M\} \vdash M' \Rightarrow U \vdash M'. \tag{2}$$

It has been proven in [11] for the datatype defined there, and can be proven to hold for the definition of $\vdash$ presented above, by structural induction. In order to apply the results given in this paper to an extension of this datatype, it is necessary to ensure that the lemma holds for the new datatype.

Honest agents and the intruder remain unchanged from [11]. For completeness, they are briefly recalled here.

A honest agent can take part in a protocol by using the following events:

$send.A.B.M$ agent A sends message M, with intended recipient B;
$receive.A.B.M$ agent B receives message M, apparently from agent A;
$claimSecret.A.B.M$ A thinks that M is a secret shared only with B; if B is not the intruder, then the intruder should not learn M;
$running.A.B.M$ A thinks it is running the protocol with B; M is a message, recording some details about the run in question.
$finished.A.B.M$ A thinks it has finished a run of the protocol with B; M is a message, recording some details about the run in question.

The *send* and *receive* events can also be treated as channels, used by agents to exchange data; the remaining events are used to formally define the desired security properties of the protocol. $Honest$ is the set of all honest agents.

The intruder acts as the medium, thus being allowed to see, modify, forge or drop any message. It uses its knowledge derivation relation $\vdash$ to forge new messages from the previously learnt messages. The set of messages it can derive from a knowledge S is defined as

$$deds(S) \triangleq \{M \in Message \mid S \vdash M\}.$$

Finally, the formal definition of the intruder is

$$\begin{aligned} INTRUDER(S) \triangleq \quad & \Box_{M \in Message} send?A?B!M \rightarrow INTRUDER(S \cup \{M\}) \\ & \Box\, \Box_{M \in deds(S)} receive?A?B!M \rightarrow INTRUDER(S) \\ & \Box\, \Box_{M \in deds(S)} leak.M \rightarrow INTRUDER(S) \end{aligned}$$

where *send* and *receive* are the communication channels, and $leak.M$ is the event that signals that the intruder can derive M from its current knowledge. The set of all agents is defined as $Agent = Honest \cup \{INTRUDER\}$.

Like in [11], attacks are specified as trace properties. A trace specification $SPEC(tr)$ is a predicate whose free variable tr represents a trace. A process satisfies a specification if the $SPEC(tr)$ predicate is true for all the traces of the process:

$$P \textbf{ sat } SPEC \Leftrightarrow \forall\, tr \in traces(P) \cdot SPEC(tr).$$

Two predicates, namely secrecy and injective authentication (or simply authentication), define the two most common properties.

Secrecy states that if agent A believes that message M is shared only with honest agent B, then the intruder must not be able to derive M from its knowledge:

$$Secrecy(tr) \triangleq \forall\, A \in Agent; B \in Honest \cdot\ claimSecret.A.B.M \text{ in } tr \Rightarrow \neg leak.M \text{ in } tr$$

In order to define authentication, a formal definition of $AgreementSet$ is first needed.

$$\begin{array}{l} M \in AgreementSet \Leftrightarrow \exists\, tr \in traces(P); A \in Agents; B \in Honest \cdot \\ \quad tr \downarrow finished.A.B.M > 0 \vee tr \downarrow running.B.A.M > 0 \end{array} \tag{3}$$

where $tr \downarrow e$ is the number of events e occurring in trace tr. Informally, $AgreementSet$ is the set of all the possible messages upon which the agents should agree (e.g. if the agents should agree on a key and a nonce, the $AgreementSet$ includes pairs with the first item that is a key and the second one that is a nonce).

Authentication states that, for each protocol run that A thinks it has finished with B, B must have started a protocol run with A, and both A and B must agree on some message $M \in AgreementSet$:

$$\begin{array}{l} Agreement_{AgreementSet}(tr) \triangleq \forall\, A \in Agent; B \in Honest; M \in AgreementSet \cdot \\ \quad tr \downarrow finished.A.B.M \leq tr \downarrow running.B.A.M \end{array}$$

Weaker types of authentication have also been defined, for instance non injective authentication, where there is no one-to-one correspondence between the runs of actors A and B, or weak authentication, where there is no agreement on session data; they are described, for example, in [14]. It is believable that the results proven in this paper for injective authentication also hold for weaker forms of authentication.

3 Handling the Channel Encoding/Decoding Layer

In real applications, each protocol specification that aims to be interoperable, must explicitly specify the encodings applied to the data that are exchanged by protocol actors. That is, all the actors must exchange data with the same specified *external* representation. However, internally, each actor can use any representation that is suitable for its implementation, provided it can translate data from the *internal* representation to the *external* one, and vice versa. In this paper it is assumed that, as usual, such translations are implemented separately from the protocol logic. The functions that offer an interface in order to translate data representations are called here the "encoding/decoding layer".

In this section it is showed how the encoding/decoding layer of a generic protocol can be modeled externally from the protocol logic itself, so that the encoding/decoding layer interface is preserved. Then, it is formally shown that, under some constraints that can be checked on real applications, any incorrect implementation of the encoding/decoding layer cannot be more harmful than an intruder is, and can therefore be abstracted away.

Following the CSP modelling approach presented in [11], and recalled in section 2, an actor performs all of its inputs on the *receive* channel, and all of its outputs on the *send* channel. Moreover, the intruder acts as the medium, thus being allowed to see, modify, forge or drop any message. Then, for actors A and B, the abstract formal model of a protocol can be represented as in figure 1.

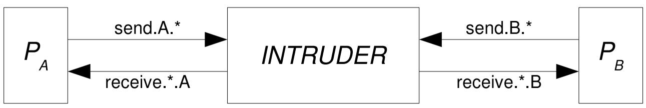

Fig. 1. Actors A and B with $INTRUDER$ in $SYSTEM$.

The model representing all the honest agents and the intruder is called $SYSTEM$, and is formally defined as

$$SYSTEM \triangleq (|||_{A \in Honest} P_A) \parallel INTRUDER(IK_0)$$

where, for each $A \in Honest$, P_A is the CSP process that describes A's behavior, and IK_0 is the initial intruder knowledge. Here and in the rest of the paper, the parallel operator $\|$ without any subscripted set of events means synchronization on all the events that are in the intersection of the alphabets of the parallel processes; that is:

$$P \parallel Q \triangleq P \underset{\alpha P \cap \alpha Q}{\parallel} Q$$

Thus, in *SYSTEM* the intruder and the honest agents synchronize on the *send* and *receive* events.

It must be noticed that in *SYSTEM* the actors directly exchange the abstract representation of data with the intruder.

In order to model the encoding/decoding layer, a refined model $SYSTEM'$ is defined as depicted in figure 2 for actors A and B.

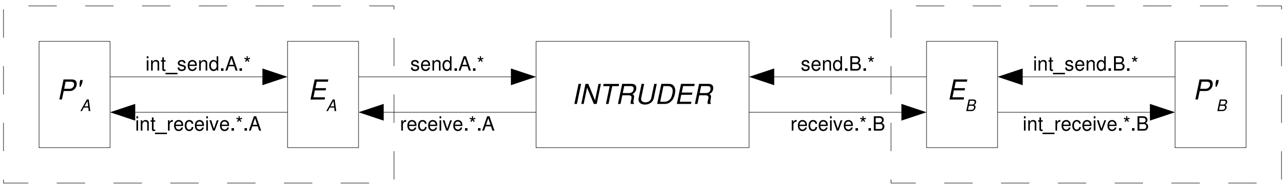

Fig. 2. Actors A and B with *INTRUDER* in $SYSTEM'$.

Basically, $SYSTEM'$ acts like *SYSTEM*, but it is explicitly modeled that the *external* representation of data is being sent over *send* and *receive*. More precisely, for each honest agent A, the coupled processes P'_A and E_A represent respectively the protocol logic and the encoding/decoding layer of a program. So each P'_A in $SYSTEM'$ acts *like* its corresponding P_A in *SYSTEM*, but it is explicitly modeled that it sends its *internal* representation to its coupled encoding layer E_A, which in turn sends the encoded data to the intruder, and vice versa.

This model can be described in CSP for all the honest agents as

$$SYSTEM' \triangleq (((|||_{A \in Honest} P'_A) \parallel (|||_{A \in Honest} E_A)) \setminus \{int\}) \parallel INTRUDER(IK'_0)$$

where $int = int\_send, int\_receive$. In this paper, set notation for events is sometimes abused so that a set defined as $\{a_1, \ldots, a_n\}$ is the set containing all the events that match one of the a_i forms. So, for instance, $int\_send.B.A.M \in \{int\}$, and $receive.C.A.M \in \{receive?X.A\}$. The context will make clear whether proper or abused set notation is being used.

It could be argued that, potentially, this model allows each honest agent to send messages to any encoding layer, and vice versa. However, the implementations of protocol logic and its coupled encoding layer are very often part of the same application, so errors that would lead honest agents or encoding layers to communicate with the wrong process are not realistic. For this reason, it is assumed that the P'_A model for the protocol logic is defined such that it will only exchange messages with its coupled encoding layer model E_A, and vice versa. Indeed, this assumption implies that such errors cannot happen in the model too. For the same reason, it is reasonable to hide the program internal communication channels *int_send* and *int_receive* from the intruder's view.

The initial intruder knowledge IK'_0 in $SYSTEM'$ has some relation with IK_0 in *SYSTEM*, however this relation now is irrelevant, and can be explained later.

Finally, the relation between each P_A and the corresponding P'_A and the formal definition of each E_A are given. The definitions of P_A and P'_A can be parameterized with respect to the channels used for communication. When not needed, process parameters will not be written. Thus, if s and r are the channel name parameters, $P'_A(s,r)$ can be written as P'_A, when parameters are irrelevant in the context.

For each P_A, P'_A can be built by refining P_A so as to model the information that the protocol agent must provide to the encoding/decoding layer for its proper working. More precisely, $P'_A(s,r)$ is obtained from $P_A(s,r)$ by replacing each s action $s.A.B.M$ in $P_A(s,r)$ with $s.A.B.(\text{ATOM } \mathcal{L}, (a, M))$, and each r action $r.B.A.M$ with $r.B.A.(\text{ATOM } \mathcal{L}, (a, M))$. Here, $\text{ATOM } \mathcal{L}$ is a special atom not present in the definition of P_A, whose only purpose is to tag the data exchanged on the internal channels, and a is such that $a \in \mathit{Encoding} \subseteq \mathit{Message}$ where *Encoding* is the set of messages that can be used as encoding/decoding parameters, i.e. additional information needed by the encoding/decoding layer when encoding and decoding operations are requested (e.g., an element of *Encoding* may include the name of the encoding algorithm to be applied and any related parameters, such as length of paddings etc.). Throughout the rest of the paper, a, b, c and d range over *Encoding*, and, unless explicitly quantified, they are assumed to be universally quantified over *Encoding*.

It is worth noting that an accurate model must set, for each message M that is sent or received, its correct encoding parameters a, according to the protocol specification documents. It can also be noted that the encoding parameters may or may not be already present in P_A. For example, if the encoding parameters a are being negotiated within the protocol logic, then a will be already present in P_A. Because of this, it is possible that the message (a, M) already exists in P_A. This is why we want to distinguish the messages (a, M) already present in P_A, from those added when deriving P'_A, which is achieved by the special label message $\textsc{Atom}\ \mathcal{L}$, which has the property of never appearing in P_A. Since $\textsc{Atom}\ \mathcal{L}$ is just a syntactic marker, it is assumed that neither P_A nor P'_A ever accept $\textsc{Atom}\ \mathcal{L}$ on inputs or send it on outputs, with the only exception when $\textsc{Atom}\ \mathcal{L}$ is explicitly needed as syntactic marker.

Each process E_A models the behavior of the encoding/decoding layer. Because of this, it can perform two kinds of actions:

- receive from its coupled process P'_A internal representations of data, along with encoding parameters, and send encoded data to the *INTRUDER* process;
- receive encoded data from the *INTRUDER* process, and send to its coupled process P'_A the internal representation, obtained using the decoding parameters specified by P'_A.

Apart from these assumptions on the possible interactions of E_A, it is assumed that internally E_A can behave in any way, thus even including erroneous implementations of data transformations. The only restriction is that E_A can access only the data explicitly provided from outside. This behavior can be represented by the following CSP process (where i_s and i_r represent the internal send and receive channels):

$$\begin{aligned}
&E_A(i_s, i_r, s, r) \triangleq \\
&\quad \mathop{\square}_{\substack{a \in \mathit{Encoding} \\ M \in \mathit{Message}}} i_s!A?B!(\textsc{Atom}\ \mathcal{L}, (a, M)) \rightarrow s!A!B!e_A(a, M) \rightarrow E_A(i_s, i_r, s, r) \\
&\square \\
&\quad \mathop{\square}_{y \in \mathit{Message}} r?B!A!y \rightarrow \\
&\qquad\qquad \mathop{\square}_{a \in \mathit{Encoding}} i_r!B!A!(\textsc{Atom}\ \mathcal{L}, (a, d_A(a, y))) \rightarrow E_A(i_s, i_r, s, r)
\end{aligned} \tag{4}$$

where $e_A(a, M)$ and $d_A(a, y)$ represent the result of the encoding and decoding operations implemented in actor A and are messages such that

$$e_A(a, M) \in \mathit{deds}(\{a, M\}) \ \wedge \ d_A(a, y) \in \mathit{deds}(\{a, y\}) \tag{5}$$

By this definition, it is possible to state the properties of the encoding/decoding layer model E_A. The result $e_A(a, M)$ of encoding M with parameters a can be anything that can be derived from M and a, thus accounting for arbitrary complex encoding schemes. Two aspects of this definition are particularly interesting:

- $e_A(a, M)$ can contain the same or less information than M.
- All information in $e_A(a, M)$ that is not present in M must be present in a.

That is, a possibly incorrect encoding function can loose some information on M, but can only use information that comes from the internal representation and from the encoding parameters. In order to model some information that is hard-coded into the encoding function implementation, it is needed to explicitly add that information to a.

The same reasoning applies to the result $d_A(a, y)$ of decoding y with parameters a, but the case when y is not recognized as a valid encoding for parameters a must be taken into account as well.

In the latter case, it is assumed that $d_A(a,y) = \textsc{Atom}\ \mathcal{E}$, where $\textsc{Atom}\ \mathcal{E}$ is a special atom that represents a decoding error code. Since this error code is part of the decoding function, it is assumed that $\textsc{Atom}\ \mathcal{E} \in \mathit{deds}(a)$ for any $a \in \mathit{Encoding}$. In the encoding/decoding layer model analyzed here, it is modeled that decoding error conditions are reported to the protocol logic, through the use of the special $\textsc{Atom}\ \mathcal{E}$ error code. An encoding/decoding layer model that gets stuck when a decoding operation fails, so that the protocol logic is never delivered the special $\textsc{Atom}\ \mathcal{E}$, is possible, and is actually a refinement of the model analyzed here. So the results obtained in this paper for the encoding/decoding layer model that reports the errors to the protocol logic, are also valid for the refined model of the encoding/decoding layer that immediately stops in case of error, and does not require the protocol logic to handle error conditions.

Another property implied by this model is that one computation of $e_A(a,M)$ and of $d_A(a,y)$ has no side effects and is memoryless. Encoding mechanisms with memory are not considered here for simplicity, but this model could be extended to include them.

It is worth noting that all the properties of the modeled encoding layer, namely that the only data accessed by the encoding/decoding functions, including hard-coded values, are their input parameters and that no side effect occurs, are information flow properties that can be verified on implementation code, by means of static sequential code analysis techniques.

3.1 Model Simplifications

The models of implementation details, introduced above, allow to refine an abstract SYSTEM into a more detailed, and complex, SYSTEM', which takes encoding/decoding functions into account. This section shows how, under some constraints involving the initial intruder knowledge IK_0, the detailed model SYSTEM' can be simplified back, still preserving the security faults that were present in SYSTEM'.

Two separate simplifications will be showed in this work. The first one is about removing the encoding/decoding layer; the second one is about removing the encoding parameters. They can obviously both be applied to the same system.

Removing the Encoding/Decoding Layer In order to remove the encoding/decoding layer from SYSTEM', a new process SYSTEM'' is defined as

$$\mathit{SYSTEM}'' \triangleq (|||_{A \in \mathit{Honest}} P'_A) \parallel \mathit{INTRUDER}(IK''_0)$$

where the initial intruder knowledge is now called IK''_0 and is assumed to be defined as

$$IK''_0 \triangleq IK'_0 \cup \mathit{Encoding} \cup \{\textsc{Atom}\ \mathcal{L}\} \tag{6}$$

The graphical representation of actors A and B in SYSTEM'' is given in figure 3.

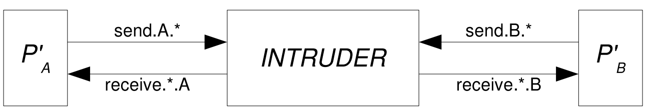

Fig. 3. Actors A and B with $\mathit{INTRUDER}$ in SYSTEM''.

A refinement relation between SYSTEM' and SYSTEM'' is expressed by:

Theorem 1. *Let* $\mathit{comm} = \{\mathit{send}, \mathit{receive}\}$, *then*

$$P\ \textbf{sat}\ \mathit{SPEC} \Leftrightarrow P \setminus \mathit{comm}\ \textbf{sat}\ \mathit{SPEC} \tag{7}$$
$$\Longrightarrow$$
$$\mathit{SYSTEM}''\ \textbf{sat}\ \mathit{SPEC} \Rightarrow \mathit{SYSTEM}'\ \textbf{sat}\ \mathit{SPEC}$$

That is, if the definition of attack does not involve the *send* and *receive* events, then all security properties defined on traces that are satisfied by $SYSTEM''$, are satisfied by $SYSTEM'$ too. Indeed, expression (7) formally states that the desired $SPEC$ property must not depend on the events in $comm$. In particular, the left-to-right implication states that, for any trace $tr \in traces(P)$ (that may contain *send* and *receive* events) and its corresponding trace $tr^- \in traces(P \setminus comm)$, if $SPEC(tr)$ holds, then $SPEC(tr^-)$ must hold too, because the *send* and *receive* events in tr do not affect the truth of $SPEC(tr)$. Conversely, the right-to-left implication states that if any number of *send* and *receive* events are added in any place of a trace of $P \setminus comm$, then $SPEC$ must still hold. Condition (7) is reasonable, because security properties are normally obtained by correct use of special events, such as $claimSecret$, $running$ or $finished$, and not directly by observing the sequence of messages exchanged on the communication channels.

Proof. The following statement is proven, that implies the theorem: for all traces tr' of $SYSTEM'$ such that an attack exists in tr', there exists a trace tr'' of $SYSTEM''$, such that an attack exists in tr'' too. Formally,

$$\begin{aligned} &\forall\, tr' \in traces(SYSTEM') \cdot \neg SPEC(tr') \Rightarrow \\ &\exists\, tr'' \in traces(SYSTEM'') \cdot \neg SPEC(tr'') \end{aligned}$$

In order to develop the proof, a new $SYSTEM'''$ is defined. Figure 4 shows the diagram for actors A and B in this system.

$$SYSTEM''' \triangleq (|||_{A \in Honest} P'_A) || E\_INTRUDER$$

where $E\_INTRUDER$ is the part inside the dashed box in figure 4. It is defined by

$$\begin{aligned} E\_INTRUDER \triangleq &(((|||_{A \in Honest} E_A(ext, int)) \\ &\parallel INTRUDER'(IK'_0)) \setminus \{int\}) \end{aligned} \tag{8}$$

where $ext = send, receive$ and $INTRUDER'(IK'_0) = INTRUDER(IK'_0)[[^{int}/_{ext}]]$.

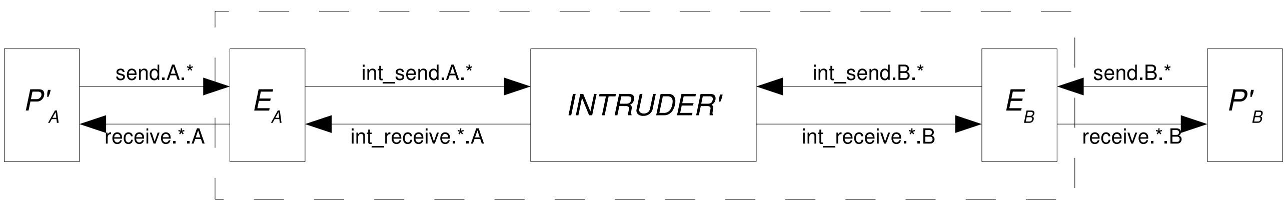

Fig. 4. Actors A and B with $E\_INTRUDER$ in $SYSTEM'''$.

A lemma is now introduced. It is needed to complete the proof of this theorem.

Lemma 2. *The intruder of $SYSTEM'''$ is a trace refinement of the intruder of $SYSTEM''$;*

$$INTRUDER(IK''_0) \sqsubseteq E\_INTRUDER$$

By lemma 2 and by refinement properties exposed in [13], it follows that

$$SYSTEM'' \sqsubseteq SYSTEM'''$$

Let us assume that a fault trace tr' exists in $traces(SYSTEM')$. It is worth noting that $SYSTEM'''$ is obtained from $SYSTEM'$, by swapping actual parameters of processes, and by injectively renaming communication channels of the $INTRUDER$ process. In particular, only the events *send* and *receive* are parameterized or renamed, and, since the renaming on $INTRUDER$ is injective, no non-determinism is introduced in the $INTRUDER'$ process. So, by condition (7), the trace

$tr''' \in traces(SYSTEM''')$, that is obtained by renaming channels in tr', is a fault trace too. Finally, since $SYSTEM'' \sqsubseteq SYSTEM'''$ holds, tr''' is also a (fault) trace in $SYSTEM''$. Then, having shown that faults are preserved from $SYSTEM'$ to $SYSTEM''$, the theorem is proven.
□

The proof of lemma 2 is given in appendix A.1.

Theorem 1 states that in a protocol specification where the encoding/decoding layer is modeled as previously described, only the protocol logic represented by P'_A is responsible for the security properties of the whole protocol, while any possible implementation of the encoding/decoding layer E_A of arbitrary complexity can be considered as part of the intruder, provided that the latter knows all required encoding schemes and parameters (because $\mathit{Encoding} \subset IK''_0$). It is also needed that the intruder knows the syntactic marker ATOM $\mathcal{L}$. This is not an issue, since it is assumed that ATOM $\mathcal{L}$ will only be treated as a marker by honest agents.

Note that no assumption on the invertibility of encoding functions has been made, thus even erroneous specifications of encoding schemes are safe (thought not functional, of course). For instance, an erroneous specification that requires to collapse all nonces into a constant cannot be responsible for replay attacks, since it is protocol logic duty to check that the internal representation of the locally generated nonce is equal to the internal representation of the received unmarshalled nonce. Moreover, since no assumption on implementation correctness has been made, even erroneous implementations of the encoding scheme are safe, provided they satisfy the data flow assumptions made.

Removing the Encoding Parameters The last step that is useful to perform when dealing with the encoding/decoding layer, is to simplify each P'_A, so that the simplified process is equal to P_A. Since each P'_A is being built from P_A (plus other information), it is possible to find a simplifying transformation that can take from P'_A back to P_A.

Simplifying transformations have been introduced in [11]. The work proposed here extends that approach, and applies it in order to obtain new results. A fault-preserving simplifying transformation in its simplest form is a function

$$f : Message \to Message$$

that defines how messages in the original protocol are replaced by messages in the simplified protocol. The function f is then overloaded to take events, traces and processes, such that all messages in the events, traces or processes are replaced.

As stated in [11], let $SYSTEM^R$ be a protocol model with associated initial intruder knowledge IK_0^R, and $SYSTEM^A = f(SYSTEM^R)$ with associated initial intruder knowledge IK_0^A. If $f(\cdot)$ is a simplifying transformation that satisfies conditions

$$U \cup IK_0^R \vdash M \Rightarrow f(U) \cup IK_0^A \vdash f(M) \tag{9}$$

$$f(IK_0^R) \subseteq IK_0^A \tag{10}$$

then $SYSTEM^A$ **sat** $\mathit{Secrecy} \Rightarrow SYSTEM^R$ **sat** $\mathit{Secrecy}$. Note that (9) depends on the derivation relation $\vdash$, so this condition must be checked each time the datatype is updated.

The work in [11] also gives a sufficient condition about $f(\cdot)$ and $\mathit{AgreementSet}$, for $f(\cdot)$ to be a fault-preserving transformation with respect to authentication specifications. This condition requires $f(\cdot)$ to be injective for all messages belonging to $\mathit{AgreementSet}$, that is

$$\begin{aligned} &\forall\ M \in AgreementSet; M' \in Message \cdot \\ &\qquad M \neq M' \Rightarrow f(M) \neq f(M') \end{aligned}$$

However, it is very difficult to find the constraints under which a simplifying transformation satisfies this condition.

In this work, a new, weaker sufficient condition for fault preserving transformations with respect to authentication agreement specifications is provided. This condition requires less constraints on the simplifying transformation, so they are easier to be found.

The new sufficient condition states that

$$\forall\ M, M' \in AgreementSet\ \cdot\ M \neq M' \Rightarrow f(M) \neq f(M') \tag{11}$$

That is, $f(\cdot)$ must be *locally* injective. Note that it is possible that for some $M, M' \in Message$ with $M \neq M'$ and $M' \notin AgreementSet$, $f(M) = f(M')$, where M may or may not be in $AgreementSet$.

Theorem 2. *If condition* (11) *is satisfied, then if a particular trace tr constitutes a failure of authentication on the original protocol, then* $f(tr)$ *constitutes a failure of authentication on the simplified protocol:*

$$\neg Agreement_{AgreementSet}(tr) \Rightarrow \neg Agreement_{f(AgreementSet)}(f(tr))$$

Proof. Suppose tr constitutes a failure of agreement authentication on the original protocol:

$$\neg Agreement_{AgreementSet}(tr)$$

then, for some $A \in Agent$, $B \in Honest$, $M \in AgreementSet$, we have:

$$tr \downarrow finished.A.B.M > tr \downarrow running.B.A.M$$

If we denote with

$$Collidings(M) = \{M' \in Message \mid M' \notin AgreementSet \ \wedge \ f(M') = f(M)\}$$

the set of messages M' that are not in the $AgreementSet$ and that are colliding with the agreed message M, then, by condition (11) on $f(\cdot)$:

$$\begin{aligned} f(tr) \downarrow finished.A.B.f(M) &= tr \downarrow finished.A.B.M + \\ &\sum_{M' \in Collidings(M)} tr \downarrow finished.A.B.M' \\ f(tr) \downarrow running.B.A.f(M) &= tr \downarrow running.B.A.M + \\ &\sum_{M' \in Collidings(M)} tr \downarrow running.B.A.M' \end{aligned}$$

However, by the definition (3) of $AgreementSet$,

$$\begin{aligned} &\forall\, M' \in Message \cdot \\ &\quad M' \notin AgreementSet \Rightarrow \forall\, tr \in traces(P); A \in Agents; B \in Honest \\ &\quad tr \downarrow finished.A.B.M' = 0 \wedge tr \downarrow running.B.A.M' = 0 \end{aligned}$$

so, since no message in $Collidings(M)$ is in the $AgreementSet$,

$$\begin{aligned} &\sum_{M' \in Collidings(M)} tr \downarrow finished.A.B.M' = \\ = &\sum_{M' \in Collidings(M)} tr \downarrow running.B.A.M' = \\ &= 0 \end{aligned}$$

From this it follows that

$$f(tr) \downarrow finished.A.B.f(M) > f(tr) \downarrow running.B.A.f(M)$$

□

From theorem 2, the following corollary is obtained.

Corollary 1. *For a renaming transformation f and an AgreementSet that satisfies* (11),

$$f(SYSTEM)\ \textbf{sat}\ Agreement_{f(AgreementSet)} \Rightarrow SYSTEM\ \textbf{sat}\ Agreement_{AgreementSet}$$

A fault-preserving renaming transformation that transforms P'_A into P_A, and thus $SYSTEM''$ into $SYSTEM$ is now introduced. This transformation collapses a pair (M, M') into its first item M. This transformation is similar in purpose to the one described in [11]. However, it is worth to point out some differences:

- The definition of the renaming function is updated to the new datatype.
- Different constraints are given for $f(\cdot)$ to be fault-preserving with respect to secrecy.
- For the first time this kind of function is proven to be fault-preserving with respect to authentication on *any* message sequence (and not only with respect to message sequences made of atoms, as in [11]), and the required constraints for this result to hold are formally specified.

Let $Pairs$ be the set of pairs (M, M') that must be coalesced to M. The updated definition of $f(\cdot)$ is

$$\begin{aligned}
f(\text{ATOM } A) &= \text{ATOM } A,\\
f(M, M') &= \begin{cases} f(M), & \text{if } (M, M') \in Pairs \land \neg isPair(M'),\\ (f(M), f(M')) & \text{if } (M, M') \notin Pairs \land \neg isPair(M'),\end{cases}\\
f(M, (M', M'')) &= \begin{cases} f(M, M'') & \text{if } (M, M') \in Pairs,\\ (f(M), f(M', M'')) & \text{otherwise},\end{cases}\\
f(\{M\}_K) &= \{f(M)\}_{f(K)}\\
f(\{[M]\}_K) &= \{[f(M)]\}_{f(K)}\\
f([\{M\}]_K) &= [\{f(M)\}]_{f(K)}\\
f(H(M)) &= H(f(M))\\
f(K^*) &= f(K)^*
\end{aligned}$$

where K^* ranges over $\{K^{\sim}, K^+, K^-\}$.

In order to preserve secrecy, $f(\cdot)$ must satisfy conditions (9) and (10). If we set

$$IK_0^A = f(IK_0^R) \cup \{f(M') | (M, M') \in Pairs\} \tag{12}$$

then, by induction on the relation $\vdash$, condition (9) is proven to hold (the proof is very similar to the one given in [11]), and condition (10) is clearly satisfied too. Equation (12) states that the intruder must already know the information that is going to be collapsed.

In order to preserve agreement, $f(\cdot)$ must satisfy condition (11). In order to achieve this, only one additional constraint is required:

$$\begin{aligned}&\forall\, M \in AgreementSet; subM \in subterms(M)\ \cdot\\ &\quad isPair(subM) \Rightarrow subM \notin Pairs\end{aligned} \tag{13}$$

where $subterms(M)$ is the set containing M and all its subterms. Constraint (13) means that no subterm of any $M \in AgreementSet$ that is a pair must be in the $Pairs$ set, that is if agreement is required on a pair, then that pair must not be collapsed.

Now it is possible to show how the coalescing pairs function $f(\cdot)$ can be safely used to transform P'_A into P_A, and thus $SYSTEM''$ into $SYSTEM$. It is worth reminding that P'_A has been obtained from P_A by replacing each sent or received message M with $(\text{ATOM } \mathcal{L}, (a, M))$. Then, the following two steps are required in order to obtain back P_A from P'_A:

1. $P_A^{tmp} = f(P'_A)$, with $Pairs = \{(\text{ATOM } \mathcal{L}, a) | a \in Encoding\}$
2. $P_A = f^{sym}(P_A^{tmp})$, with $Pairs = \{(\text{ATOM } \mathcal{L}, M) | M \in Message\}$

where $f^{sym}(\cdot)$ is the symmetric function of $f(\cdot)$, that coalesces pairs of the form (M, M') into their second item M'.

In step 1, the syntactic marker ATOM $\mathcal{L}$ is used to find and remove all encoding parameters that have been added to represent the encoding/decoding layer. Then, step 2 removes the syntactic marker, finally obtaining P_A.

Each one of these transformations preserves secrecy and authentication if the required sufficient conditions (12) and (13) hold.

In step 1, by setting $IK_0^{tmp} = f(IK_0'') \cup f(Encoding) = f(IK_0') \cup f(Encoding) \cup \{\text{Atom } \mathcal{L}\}$, that is, by requiring that the intruder already knows all encoding schemes and parameters, condition (12) is clearly satisfied. As stated above, Atom $\mathcal{L}$ in the intruder knowledge is not an issue.

Moreover, condition (13) holds because $Pairs \cap subterms(AgreementSet) = \emptyset$. Indeed, in step 1 each element in $Pairs$ has the form (Atom $\mathcal{L}, a$); but Atom $\mathcal{L}$ can never appear in any *running* or *finished* event, and thus in any subterm of the $AgreementSet$, because it is assumed that no honest agent will ever input or internally generate the Atom $\mathcal{L}$ value, except when the syntactic marker is explicitly needed.

In step 2, condition (12) is clearly satisfied if we set $IK_0 = IK_0^{tmp}$; condition (13) holds because of the same reasoning used for step 1.

It is worth noting that, by theorem 1, it follows that $SYSTEM''$ satisfies all the security properties that can be checked on $SYSTEM'$ (provided they are not defined on the *send* or *receive* events, and the intruder knows the encoding parameters). However, when passing from $SYSTEM''$ to $SYSTEM$, the faults that are currently proven to be preserved are only those regarding secrecy and authentication specifications. That is, if $SYSTEM$ is verified instead of $SYSTEM'$, then secrecy and authentication properties are also verified for all possible implementation details represented in $SYSTEM'$. On the basis of the results achieved in this paper, it is still required to explicitly verify $SYSTEM'$ (or, equivalently, $SYSTEM''$) in order to verify other security properties.

4 Handling the Encoding of Data to be Ciphered and Key Material

Cryptographic protocols must define, for interoperability, not only the encoding of messages that are sent and received on communication channels, but also the encoding of data on which cryptographic operations are applied. Since the layered approach presented in the previous section does not apply to the latter encodings, a similar but different model is now introduced in order to represent them. For simplicity, in this section encodings of messages sent and received on communication channels are disregarded, but of course the two models can be combined together, as it will be shown in section 5.

Given an abstract protocol model $SYSTEM$, where the encoding of data to be ciphered and of key material are abstracted away, a refined $SYSTEM'$ that takes these details into account can be built as follows. The $INTRUDER$ process is left untouched, while each process P_A is transformed into a process P'_A, and is coupled with a decoding process DEC_A. Each pair of P'_A and DEC_A processes internally communicates by the $priv\_send_A$ and $priv\_receive_A$ hidden dedicated channels. $SYSTEM'$ with two actors A and B is depicted in figure 5.

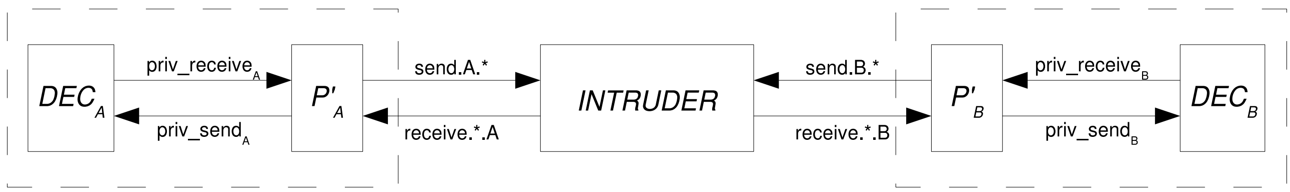

Fig. 5. Actors A and B with $INTRUDER$ in $SYSTEM'$.

The formal definition of $SYSTEM'$ is

$$SYSTEM' \triangleq INTRUDER(IK_0') \;\parallel (|||_{A \in Honest}((P'_A \parallel DEC_A) \setminus priv\_send \cup priv\_receive))$$

where $priv\_send = \{priv\_send_A | A \in Honest\}$ and $priv\_receive = \{priv\_receive_A | A \in Honest\}$

The P'_A process incorporates the capability of encoding data before applying cryptographic operations on them, and delegates to DEC_A the task of decoding data after decryptions. Accordingly, in order to obtain P'_A from P_A, each $send.A.B.M$ action in P_A must be changed into a $send.A.B.M'$ in P'_A, where M' is obtained from M by adding the required data and key encoding details. More precisely:

– each subterm of M taking the form $\{N\}_K$ or $\{[N]\}_K$ or $[\{N\}]_K$ will take the form $\{e_A(a, N)\}_K$, $\{[e_A(a, N)]\}_K$, $[\{e_A(a, N)\}]_K$ respectively in M';

- each subterm of M taking the form $H(N)$ will take the form $H(e_A(a,N))$ in M';
- each subterm of M taking the form of a key K^* will take the form $e_A(a,K)^*$ in M';
- each other subterm of M will remain unchanged.

Like in the previous section, $a \in \mathit{Encoding}$ represents encoding parameters, and $e_A(a,M)$ represents the result of the encoding transformation of M using parameters a. Like in the modelling of the encoding/decoding layer, a is added as to comply with the protocol specification documents.

The meaning of this refinement is that, when sending ciphered data, the encoded plaintext is ciphered, instead of its internal representation. Note that the encoding of key material is taken into account, because K^* messages are transformed into $e_A(a,K)^*$ messages. The same refinement is applied to the *claimSecret*, *running*, and *finished* actions.

A different refinement is needed instead for each *receive.B.A.M* action, which becomes *receive.B.A.M′*, followed by zero or more pairs of $\mathit{priv\_send}_A.(y,a) \to \mathit{priv\_receive}_A.N'$ actions, that represent interaction with the DEC_A process. More precisely:

- if M is an encryption $\{N\}_K$, and P_A can decrypt it because it knows the appropriate key, then M' will take the form $\{y\}_{K'}$, where y is a free variable and K' is obtained from K by the same procedure described for the *send* action. Moreover, a pair of $\mathit{priv\_send}_A.(y,a) \to \mathit{priv\_receive}_A.N'$ is added in order to represent the interaction with the DEC_A process, needed for decoding y. Here N' is obtained by recursively applying this procedure, which is being explained, to N, thus possibly appending further $\mathit{priv\_send}_A$, $\mathit{priv\_receive}_A$ action pairs, depending on the form of N. Then, by the $\mathit{priv\_send}_A.(y,a)$ action, the encoded plaintext y and the decoding parameters a are sent to the DEC_A process, which returns the decoded representation of y. If $d_A(a,y)$ denotes the result of decoding y using parameters a, by the $\mathit{priv\_receive}_A.N'$ action P'_A is forcing the match between $d_A(a,y)$ and N', that is the expected decoded representation of the plaintext. Note that N' may differ from N because N must be refined as well, so it is necessary to instantiate this procedure over N, thus obtaining N'.
  The same reasoning applies to public key encryptions $\{[N]\}_K$ and private key ones $[\{N\}]_K$.
  For example, if an abstract process P_A is ready to perform the $\mathit{receive.B.A.}\{[\{\textsc{Atom}\ A\}_{K_1}]\}_{K_2^+}$ action, where K_1 is a shared key known by P_A (in this example K_1 is opaque, i.e. P_A has received it as an opaque message), and K_2^+ is a public key for which the corresponding private key K_2^- is known by P_A, then the refined process P'_A, coupled with its decoding process DEC_A, must be ready to perform the following sequence of action prefixes

$$\begin{aligned} \mathit{receive.B.A.}\{[y]\}_{e_A(a,K_2)^+} &\to \mathit{priv\_send}_A.(y,b) \to \mathit{priv\_receive}_A.\{x\}_{K_1} \to \\ &\mathit{priv\_send}_A.(x,c) \to \mathit{priv\_receive}_A.\textsc{Atom}\ A \end{aligned}$$

  where $a,b,c \in \mathit{Encoding}$ are the encoding parameters prescribed by the protocol specification documents. Moreover, in order to be able to perform decryption operations, P'_A must know the symmetric key K_1 and the private key $e_A(a,K_2)^-$. In this example, P'_A receives an encoded message y that is deciphered by using the known refined asymmetric key $e_A(a,K_2)^-$ (note that private key is required to decrypt public-key ciphered data). Then, the encoded message y along with the decoding parameters b is sent to the coupled DEC_A process, which returns the decoded representation. This message must match the $\{x\}_{K_1}$ message, where x is again the encoded form of the plaintext, as required by the cryptographic algorithm, and K_1 is the opaque symmetric key. Note that in this model the value of K_1 will be the refined form of the symmetric key. However, since this key is opaque and known in its abstract form by P_A, it remains opaque and known in its refined form by P'_A. Finally, P'_A sends the encoded message x and the decoding parameters c to DEC_A, obtaining the decoded plaintext, which is forced to match $\textsc{Atom}\ A$.
- if M is a pair (N,O), then this procedure is applied to N and O. This case is needed to handle messages where encryptions are not top-level messages, but they are contained into, possibly nested, pairs. We make the assumption that this procedure is applied depth-first to N, then to O.
- In all the other cases, M' is obtained from M by the same procedure described for the *send* action, and no pairs of $\mathit{priv\_send}_A$, $\mathit{priv\_receive}_A$ actions are added. By this case, it is modelled that all the data that cannot be decrypted, for example hashed data, are generated locally, taking encodings into account. Then, locally generated data are compared with received data.

For example, if an abstract process P_A is ready to perform the $receive.B.A.\{H(\text{ATOM } A)\}_{K^\sim}$ action, then the refined process P'_A, coupled with its decoding process DEC_A, is ready to perform the sequence of action prefixes

$$receive.B.A.\{y\}_{e_A(a,K)^\sim} \to priv\_send_A.(y,b) \to priv\_receive_A.H(e_A(c, \text{ATOM } A)) \tag{14}$$

where $a, b, c \in Encoding$ are the encoding algorithms prescribed by the protocol specification documents. In this example, P'_A receives a message that is deciphered by using the refined symmetric key $e_A(a,K)^\sim$, which is known by P'_A. The obtained plaintext y should be the encoding, required by the cryptographic algorithm, of the original message $H(e_A(c, \text{ATOM } A))$. Here y is treated by P'_A as an opaque message and it is sent along with the encoding parameters b to the coupled decoding process DEC_A. The latter returns the internal representation of y, which must equal $H(e_A(c, \text{ATOM } A))$. Again, P'_A expects to receive $H(e_A(c, \text{ATOM } A))$, and not $H(\text{ATOM } A)$, because it is taken into account that ATOM A must be encoded before being passed to the hashing algorithm.

Each DEC_A process is formally defined as follows:

$$\begin{aligned} DEC_A \triangleq\ & \Box_{\substack{y \in Message \\ a \in Encoding}}\ priv\_send_A!(y,a) \to \\ & (priv\_receive_A!d_A(a,y) \to DEC_A) \between d_A(a,y) \neq \text{ATOM } \mathcal{E} \between STOP \end{aligned} \tag{15}$$

where $P \between b \between Q$ means *if* b *then* P *else* Q.

Note that, like in previous section, for $e_A(a,M)$ and $d_A(a,y)$, condition (5) holds, and absence of side effects and of memory between calls is assumed. Moreover, an error in decoding y with parameters a is modeled by $d_A(a,y) = \text{ATOM } \mathcal{E}$, where ATOM $\mathcal{E}$ is the special atom representing a decoding error code, and ATOM $\mathcal{E} \in deds(a)$. In this section, however, it is assumed that the decoding process stops immediately if decoding fails, which is the most realistic behaviour for the kind of encoding considered in this section.

4.1 Abstracting the Refined Model

In this section it is shown that, under some conditions, if $SYSTEM$ does not have security flaws, then $SYSTEM'$ does not have any either.

Like in the previous section, although by a technically different reasoning, it is shown in a first step that under some assumptions the refined $SYSTEM'$ meets any security property that is satisfied by a more abstract, intermediate $SYSTEM^*$. Note that this relation holds for any security property that can be defined on traces, provided that it is not defined on the *send* or *receive* events that may appear in a trace. Then, in a second step, it is showed that under further assumptions the intermediate $SYSTEM^*$ can be further simplified to the original abstract $SYSTEM$, by applying a newly introduced simplifying transformation, that still preserves secrecy and authentication.

In order to obtain the intermediate $SYSTEM^*$, let us define

$$priv_A \triangleq \{priv\_send_A, priv\_receive_A\}$$

and the function $f(\cdot)$ that transforms P'_A into $P^*_A \triangleq f(P'_A)$. Informally, function $f(\cdot)$ removes events on the channels in $priv_A$ without changing the external behaviour of the process. Formally, $f(\cdot)$ can be defined as a function on CSP processes that distributes over any CSP operator ω but the action prefix operator, on which $f(\cdot)$ acts by removing the events in $priv_A$, i.e.

- for any CSP operator ω, with any arity n, except action prefix:

$$f(\omega(P_1, \ldots, P_n)) = \omega(f(P_1), \ldots, f(P_n))$$

- for the action prefix operator $ev \to P$: if

$$\begin{gathered}(ev \notin \{receive?B.A\} \cup priv_A) \lor \\ (ev \in \{receive?B.A\} \ \land\ \forall\, pev \in priv_A \cdot P \neq pev \to P')\end{gathered}$$

then

$$f(ev \to P) = ev \to f(P)$$

else

$$f(receive.B.A.M \to priv\_send_A.(y,a) \to priv\_receive_A.N \to P) =$$
$$f\left(receive.B.A.M\left[{}^{e_A(a,N)}/_{y}\right] \to P\right)$$

Although function $f(\cdot)$ is defined for any CSP process, in order to keep the proofs simpler, from now on it will be assumed that P'_A is a sequential process. This assumption does not narrow the generality of our results, since multi-threaded implementations of protocol logics can be simulated by corresponding sequential implementations.

The intermediate $SYSTEM^*$ is formally defined as

$$SYSTEM^* \triangleq |||_{A \in Honest} P^*_A \parallel INTRUDER(IK^*_0)$$

where IK^*_0 is the initial intruder knowledge in the intermediate system. It is worth noting that, in $SYSTEM^*$, the DEC_A processes have been removed. Indeed, each intermediate P^*_A process "embeds" decoding capabilities, by expecting to receive only the encoded form of data. For example, if P'_A can perform the action sequence in (14), then P^*_A will be able to perform the action $receive.B.A.\{e_A(b, H(e_A(c, \text{ATOM } A)))\}_{e_A(a,K)^{\sim}}$.

Now that a formal relation between $SYSTEM'$ and $SYSTEM^*$ has been defined, the following theorem, needed to get to the final result, can be formulated:

Theorem 3.

$$d_A(a,y) \neq \text{ATOM } \mathcal{E} \Rightarrow e_A(a, d_A(a,y)) = y \tag{16}$$
$$\wedge\ IK^*_0 \supseteq IK'_0 \tag{17}$$
$$\Longrightarrow$$
$$SYSTEM^* \setminus comm \sqsubseteq SYSTEM' \setminus comm \tag{18}$$

Condition (16) requires that, for each actor, the implementation of the encoding function $e_A(a,\cdot)$ is the inverse of the implementation of the decoding function $d_A(a,\cdot)$. Condition (17) simply requires that at the beginning of the protocol run, the intruder in the intermediate system knows at least the same messages known by the intruder in the refined system. Note, however, that no assumption on implementation correctness with respect to any encoding scheme specification is made, and no relationship is being assumed between encoding scheme implementations of different actors. That is, condition (16) can be checked in isolation on every implementation alone, without referring to any encoding scheme specification.

Also note that canonicalization schemes are neglected, because they transform data between two different items of the same equivalence class, which are normally all represented by a single term in a formal CSP model. Indeed, as stated in [15] too, canonicalization does not impact security properties, as long as all elements of the same canonicalization equivalence class have the same meaning (which actually is the aim of canonicalization). Moreover, representing canonicalization operations in abstract models would introduce non injective functions, whose interaction with some security properties (e.g. authentication) would be rather complex, despite not so significant.

The proof of theorem 3 is given in appendix A.2.

Now, let $SPEC(tr)$ be a generic security property that can be defined on traces.

Corollary 2. *If theorem 3 holds, and $SPEC(tr)$ is such that condition* (7) *holds, then*

$$SYSTEM^* \textbf{ sat } SPEC \Rightarrow SYSTEM' \textbf{ sat } SPEC \tag{19}$$

As previously explained, condition (7) formally states that the desired $SPEC$ property must not depend on the events in *comm*. It is worth recalling that the given condition is reasonable, because security properties are normally obtained by correct use of special events, such as $claimSecret$, *running* or $finished$, and not directly by observing the sequence of messages exchanged on the communication channels.

Proof. Trace refinement (18) implies

$$SYSTEM^* \setminus comm \ \mathbf{sat} \ SPEC \Rightarrow SYSTEM' \setminus comm \ \mathbf{sat} \ SPEC$$

Then, by using the $\Leftarrow$ side of (7), it follows that (19) holds.
□

Summing up the results of theorem 3 and corollary 2, if the intruder knowledge in the abstract system is no less than the intruder knowledge in the refined system, and the implementation of the encoding function of each actor is the inverse of the implementation of the decoding function of the same actor, then the more abstract $SYSTEM^*$ can be verified instead of $SYSTEM'$, for any security property that does not depend on the *send* and *receive* events.

By this result, when a model extraction approach like the one presented in [7] is used, the verification of any security property can be safely divided into two distinct verifications. On one hand, the verification of the property on an abstract protocol model where all the decoding functions that are modeled in this work by the DEC_A processes are left out. On the other hand, the verification that the sequential code of each encoding procedure implements the inverse of the corresponding decoding function implementation.

Let us consider now the further simplification from $SYSTEM^*$ to $SYSTEM$. In order to define this transformation, it is required to ensure that, in each actor, the implementation of the encoding function $e_A(a, \cdot)$ is injective. Note that this condition, along with (16), which is already required by theorem 3, amounts to requiring that the implementations of encoding and decoding in each actor are the inverses of each other. Using injectivity of $e_A(a, \cdot)$, it is possible to introduce a fault preserving simplifying transformation $f_d(\cdot)$, defined as the identity function except for the following cases:

$$\begin{aligned}
f_d(M, M') &= (f_d(M), f_d(M')) \\
f_d(\{e_A(a, M)\}_K) &= \{f_d(M)\}_{f_d(K)} \\
f_d(\{[e_A(a, M)]\}_K) &= \{[f_d(M)]\}_{f_d(K)} \\
f_d([\{e_A(a, M)\}]_K) &= [\{f_d(M)\}]_{f_d(K)} \\
f_d(H(e_A(a, M))) &= H(f_d(M)) \\
f_d(e_A(a, K)^*) &= (f_d(K))^*
\end{aligned}$$

With this definition, $P_A = f_d(P_A^*)$. Preservation of security properties in the refinement from $SYSTEM$ to $SYSTEM^*$ can now be expressed by the following theorems.

Theorem 4.

$$\begin{gathered}
IK_0 \supseteq f_d(IK_0^*) \\
\Longrightarrow \\
SYSTEM \ \boldsymbol{sat} \ Secrecy \Rightarrow SYSTEM^* \ \boldsymbol{sat} \ Secrecy
\end{gathered} \tag{20}$$

Proof. In order to prove that secrecy in the abstract system implies secrecy in the refined system, it is enough to show that $f_d(\cdot)$ is actually a fault preserving simplifying transformation that preserves secrecy, which amounts to check that conditions (9) and (10) are satisfied.

Satisfaction of condition (9) can be proven by induction over the knowledge derivation relation $\vdash$; while satisfaction of condition (10) can be proven by hypothesis (20), which is reasonable because it simply requires that the intruder in the abstract system knows at least the simplified form of messages that are known by the intruder in the refined system.
□

In order to prove that authentication is preserved when refining $SYSTEM$ into $SYSTEM^*$, some further conditions that must hold for messages in the *AgreementSet* need to be stated.

Let $e(a, M)$ and $d(a, y)$ denote the definitions of the encoding and decoding functions (in contrast with $e_A(a, M)$ and $d_A(a, y)$ that define their implementations in agent A). One of the additional conditions, needed in order to show authentication preservation, is

$$e_A(a, O) = e(a, O) \tag{21}$$

which means that the encoding implementation in each actor is correct with respect to the specification of the encoding scheme. In practice, if (21) holds, then all actor's implementations of encoding functions are equivalent, so that implementing actors can be ignored. So, the $e(a, O)$ symbolic form of encoding will be used from now on, regardless of the implementing actor. Although condition (21) is harder to check than (16), the example in section 5 shows that this check is viable at least for an important class of encoding schemes.

Let us introduce now the *symbolic expression* of a message, that is a term that represents the message but leaving all data encoding operations in their symbolic form $e(a, O)$ (in contrast to resolve them to the resulting term obtained by encoding message O with parameters a).

Using the symbolic expression concept, the $AgreementSet$ can be partitioned into equivalence classes. Two messages M and M' belong to the same equivalence class if their symbolic expressions are equal, modulo a renaming of the first argument of each encoding operation occurring in them (namely the a argument of $e(a, O)$). The $M \sim M'$ notation means that M and M' belong to the same equivalence class. For example, if $(H(e(a, O)), N)$ is the symbolic expression of M, and $(H(e(b, O)), N)$ is the symbolic expression of M', then $M \sim M'$ is true; in contrast, if $(H(e(b, O)), N')$ is the symbolic expression of M' and $N \neq N'$, then $M \nsim M'$, because their symbolic expressions also differ by the $N \neq N'$ terms. In other words, each equivalence class contains all the messages that can be obtained by applying encodings with different encoding parameters to the same unencoded message.

Theorem 5. *If* (20), (21), *and*

$$\begin{array}{c} \forall\ M, M' \in AgreementSet \cdot \\ M \sim M' \Rightarrow M = M' \end{array} \tag{22}$$

hold, then

$$\begin{array}{c} SYSTEM\ \textbf{\textit{sat}}\ Agreement_{f_d(AgreementSet)} \Rightarrow \\ SYSTEM^*\ \textbf{\textit{sat}}\ Agreement_{AgreementSet} \end{array}$$

Condition (22) states that each equivalence class must have only one element. In other words, it must never happen that $AgreementSet$ contains two messages that share the same symbolic expression, except for some encoding parameters. Indeed, this condition can be enforced in practice by explicitly including, within each message upon which agreement is required, the parameters that must be used to encode each part of the message itself. Agreement on the encoding parameters used to encode data on which agreement is required is an authentication property that holds if the negotiation algorithm used in the protocol to establish such parameters is logically correct in an unsafe environment. This can be verified as part of the formal protocol verification task on the abstract protocol, as shown in section 5.

Proof. In order to prove that $f_d(\cdot)$ preserves agreement, since conditions (9) and (10) have already been proven for corollary 4, it is enough to prove (11), that is $f_d(\cdot)$ is locally injective on $AgreementSet$. In other words, if by hypotheses (20), (21) and (22), $f_d(\cdot)$ satisfies condition (11), then, by theorem 2, this theorem is proven.

Now, function $f_d(\cdot)$ is shown to be locally injective on $AgreementSet$.

Let $M, M' \in AgreementSet$ with $M \neq M'$. If $M \nsim M'$, then M and M' are terms with different structures, or, if they have the same structure, there exist two subterms N and N' with $N \neq N'$, in the same position in M and M' respectively, that cause them to differ. In this case, $f_d(M) \neq f_d(M')$ because it can be easily shown, by structural induction over messages, that $f_d(\cdot)$ preserves message structure and does not alter subterms, except for removing symbolic encoding operations, and their first parameter, which is not what is making M and M' different in this case.

If $M \sim M'$, then, by (22), if follows that $M = M'$, which contradicts the hypothesis, so this case cannot happen.

□

Note that if extensions of the proposed datatype are used, structural inductions used in the proofs of theorems 4 and 5 must be checked to hold for the new datatype.

5 Applications and Examples

This section shows some practical applications of the results illustrated in this paper. First, a class of concrete encoding schemes of the family discussed in section 4, that includes XML encodings, is considered. Regarding this class, all the conditions required by theorems 3, 4 and 5 for enabling the safe application of abstractions on an implementation of XML encodings like the one described in [10] are analyzed.

Then, it is showed how incidentally a particular instance of the modelling framework proposed in section 4 can be used for a somewhat different purpose, i.e. to give sufficient conditions under which cryptographic algorithms and parameters can safely be abstracted away in Dolev-Yao models.

Finally, the modelling of an SSH Transport Layer Protocol client is presented. Both an abstract model and a refined one are provided. The refined model includes alltogether marshalling functions, as explained in section 3, encodings of data to be ciphered, as illustrated in section 4, and cryptographic algorithms and parameters, as explained in the example in section 5.2. Upon the results showed in this paper, the abstractions that can be made on this particular model are showed, along with the conditions that must be checked or assumed on the sequential code that implements the various encoding and decoding functions, in order to safely apply abstractions.

5.1 A Class of Data Encodings including XML Encodings

The encoding schemes considered in this example simply add a header to each part of the message being encoded, and do not alter the message content itself. Formally:

$$\begin{aligned} e(a, M) &= (head(a, M), M) \quad ,\text{if } \neg isPair(M) \\ e(a, (M, M')) &= (e(a, M), e(a, M')) \end{aligned} \tag{23}$$

where $head(a, M) \in deds(\{a, M\})$ is an header that may depend on parameters a and on message M. The peculiarity of this kind of encoding function is that it distributes headers across pairs. It is possible to define the symmetric encoding function, that adds a trailer, or padding, to data; it is furthermore possible to combine the two.

This class of encoding schemes is general enough to include, for example, XML encodings. Then, it can be used to model the data encodings used in the protocols of the WS-Security [16] standard.

In [10], an implementation of the XML encoding scheme where XML fragments are internally represented as F# (an ML dialect) records is described. For example, the XML security header specified in the WS-Security standard is internally stored as:

```
type security = {
    timestamp: ts;
    utoks: utok list;
    xtoks: xtok list;
    ekeys: encrkey list;
    dsigs: dsig list }
```

Then, a set of `gen*` and `parse*` functions, implemented in F#, is used to translate internal records to and from XML, for example when an XML fragment must be encrypted.

In the CSP modelling framework for cryptographic protocols used in this paper, an F# record (i.e. the internal data representation) can be modeled by means of nested pairs. The F# functions translating to and from XML, are actually encoding functions that add some XML header and trailer to each element of the record, that is to the nested pairs. Thus, the `gen*` and `parse*` functions behave like the encoding scheme defined in (23).

In order to check that the implementation of these functions satisfies theorem 3, it is enough to check that, on one hand, the `gen*` functions only add a tagged fragment before and after any record element, leaving the record element unchanged (with the exception of canonicalization operations, which are abstracted in the model). Note that the added tagged fragment may be anything that can be correctly recognized in the decoding phase; for theorem 3 to hold, correctness of implementation w.r.t. the XML specification is not required. On the other hand, it is enough to check that the `parse*` functions match some expected tagged header and trailer, that must comply with the expected record type and value, and that they store the data between header and trailer into the proper record field without further modification (with the exception of canonicalization operations).

The injectivity property required by theorem 4 is implied by the encoding definition (23). Then, once it has been verified that the scheme expressed by (23) is correctly implemented by the `gen*` functions, injectivity is automatically implied too.

For applying theorem 5, the correctness of the encoding functions implementation w.r.t. their specification is additionally needed. This amounts to the extra check that the tagged data added by the encoding `gen*` functions, and recognized by the decoding `parse*` functions, actually comply with the XML and WS-Security standards.

In [10], a model extraction approach for security protocols is proposed. That is, a formal Dolev-Yao model is extracted from the implementation code of the protocol, and then security properties are checked on the extracted model. In that work, each time a model extraction is performed, a model is extracted from the `gen*` and `parse*` function implementations too, and a global protocol model including the model of these encoding functions is formally analyzed. Moreover, messages are modeled in their encoded, complex form.

By using the results presented in this work, once correctness of the sequential `gen*` and `parse*` functions has been proven *una tantum*, it is possible to exclude these functions from the model extraction process, and to use the simpler abstract representation of messages, thus reducing model complexity and required verification resources, while keeping the same security assurance. This is possible, because it has already been proven, by theorems 3, 4 and 5, that secrecy and authentication faults cannot arise from the encoding functions, or because of using encoded forms of messages, provided the stated conditions hold.

A similar result can be obtained when using a code generation approach. If formally verified implementations of `gen*` and `parse*` functions are available, they can be used to correctly refine abstract models into concrete implementations while keeping the same security w.r.t. a Dolev-Yao intruder.

5.2 Conditions for Abstracting Cryptographic Algorithms and Parameters

The goal of this example is to show how cryptographic algorithms and parameters can be modelled and safely abstracted away in protocol models, by reusing the results presented in section 4. For this reason, protocol role implementations are less relevant in this context.

The abstract datatype defined in section 2, intentionally represents the result of encryption as a function of a plaintext message and a key. The ciphertext has the property of being restored to plaintext only if the appropriate key is known. In the same way, the hash function is simply a non invertible representation of a message.

However, different real cryptographic algorithms obtain these properties in different, incompatible ways. For example, in a real application, if a message M is ciphered with the key K^+, by using the RSA algorithm, obtaining $\{[M]\}_{K^+}$, then the decryption of $\{[M]\}_{K^+}$ using the key K^- will not get M, if not exactly using the RSA algorithm. An interesting example is given by protocols that require agreement on the value of an hashed message, which, for example, is then used as the material to build a shared session key. If both actors obtain the same message M, and compute $H(M)^\sim$, it is a prerequisite to key agreement that they have previously agreed on the same hashing algorithm and key construction algorithm, otherwise they could obtain the same logical value (a shared key obtained from a non invertible representation of M), but different concrete values. Then, key agreement may fail in the concrete world, even if the actors agree on the abstract $H(M)^\sim$.

In order to faithfully describe this issue, cryptographic algorithms and their parameters are sometimes introduced in abstract descriptions. For example, in [11, 7], encryption functions are distinguished according to the algorithm and parameters they use.

In this work, in order to handle cryptographic algorithms and parameters, one could extend the datatype and the associated derivation relation $\vdash$ shown in section 2. The obtained datatype would be similar to the one presented in [11]. However, this approach would not give us the opportunity to easily discuss about the conditions under which abstracting these details is safe.

Fortunately, it turns out that the modelling approach in section 4 can be used in a particular way, such that the models of cryptographic algorithms and parameters can be represented as encodings in the current datatype, rather than being explicitly added to it. This approach has the advantage that the sufficient abstraction conditions already shown in section 4 can be directly reused in order to abstract cryptographic algorithms and parameters too.

So, referring to the modelling approach presented in section 4, let us define the following encoding.

$$\begin{aligned} e((\textsc{Atom}\ \mathcal{L}, a), M) &= (\textsc{Atom}\ \mathcal{L}, (a, M)) \\ d((\textsc{Atom}\ \mathcal{L}, a), (\textsc{Atom}\ \mathcal{L}, (a, M))) &= M \\ d((\textsc{Atom}\ \mathcal{L}, a), N) &= \textsc{Atom}\ \mathcal{E},\ \text{if } N \text{ does not take the form } (\textsc{Atom}\ \mathcal{L}, (a, M)) \end{aligned} \tag{24}$$

These definitions clearly represent functions that satisfy equation (16). As previously explained, $\textsc{Atom}\ \mathcal{L}$ is a syntactic marker used to tag added data, and $a \in \mathit{Encoding}$ are the cryptographic algorithms and parameters chosen according to the protocol specification documents.

The refined model that is obtained in this way corresponds to the one used in [11, 7] where different encryption, decryption and hashing functions are used for each different choice of algorithms and parameters.

For example, the refined encryption

$$\{(\textsc{Atom}\ \mathcal{L}, (\mathit{DES\_encrypt}, M))\}_{(\textsc{Atom}\ \mathcal{L}, (\mathit{DES\_key}, K))^{\sim}} \tag{25}$$

expressed in the modelling framework being presented here, corresponds, using an approach like the one presented in [11, 7], to a term like $\mathit{DESEncrypt}(\mathit{DESKeyBuild}(K), M)$, i.e. the encryption of M performed using the $\mathit{DES\_encrypt}$ cryptographic algorithms and parameters, and a shared key constructed from the material K and the $\mathit{DES\_key}$ algorithms and parameters.

It is worth noting that this refined model can fully handle the case where both encryption and key construction have their own, possibly different parameters. Also, the refined hash

$$H(\textsc{Atom}\ \mathcal{L}, (\mathit{SHA1}, M))$$

would correspond, using an approach like the one presented in [11, 7], to a term like $\mathit{SHA1Hash}(M)$, which means that the hashing of M is performed using the SHA-1 algorithm.

Another property that is shared between the refined model used in this paper and the datatype presented in [11], is that in both approaches it is impossible to achieve "security through obscurity". For example, in the refined encryption (25), it is enough for the intruder to know the key $(\textsc{Atom}\ \mathcal{L}, (\mathit{DES\_key}, K))^{\sim}$, or key material K and parameters $\mathit{DES\_key}$, in order to get both the plaintext M and the cryptographic parameters $\mathit{DES\_encrypt}$. Similarly, in [11, 7], the intruder can apply any encryption and decryption function, thus being able to deduce which algorithm and parameters have been used for generating an encrypted term. This is a strong assumption, however it is realistic. Many cryptographic algorithms put enough redundancy in the key (and even in the ciphertext), that it is possible to guess what algorithm has been used. Moreover algorithms are a few number, so even brute force attack on all known algorithms is feasible. One can argue that some symmetric encryption algorithms use an initialization vector, which is difficult to guess. This is true, however the assumption that the intruder can guess the initialization vector represents the worst case analysis, which is acceptable, and sometimes even desirable, when dealing with security protocols implementations.

Now, sufficient conditions, so that the model of cryptographic algorithms and parameters that has been encoded into the current datatype can be abstracted away, can be stated, by reusing the results given in section 4.

First of all, the DEC_A decoding process can be safely removed in this case, because condition (16) is satisfied by the definition (24) of this specific encoding scheme and condition (17) naturally holds.

Moreover, since the encoding defined in (24) is injective too, the encoding that represents cryptographic algorithms and parameters can be completely abstracted away in secrecy verifications, provided that condition (20) holds, i.e. provided that the intruder is assumed to know cryptographic algorithms and parameters, which is a reasonable assumption. Then, the first result is that, by appropriately setting the intruder knowledge when verifying an abstract model, secrecy is implied in the refined model for any possible security protocol, regardless of its logic.

For what concerns the verification of authentication, the model of cryptographic algorithms and parameters can be completely abstracted away if conditions (21) and (22) hold too. Since in this example we are modelling cryptographic algorithms and parameters for all actors by the same encoding and decoding functions, it is safe to assume that (21) holds. Nevertheless, unlike (20), the truth of (22), and thus agreement preservation, depends on the particular protocol logic that is being verified: as stated in the previous section, condition (22) can be enforced by specifying (and verifying) agreement on all the cryptographic algorithms and parameters used to build data on which agreement is required. Then, the second result is that cryptographic parameters and algorithms can be safely abstracted away in authentication verification provided that all the cryptographic algorithms and parameters used to build data on which agreement is required are explicitly included in the agreement statements to be verified.

5.3 Modelling an SSH Transport Layer Protocol Client

In this example, another syntactic sugar is added: lists of n messages are reduced to nested pairs. So, for example, (M, M', M'') is equal to $(M, (M', M''))$.

The SSH Transport Layer Protocol [17] (SSH-TLP) is part of the SSH three protocols suite [18]; in particular it is the first protocol that is used in order to establish an SSH connection between client and server. SSH-TLP gives server authentication to the client, and establishes a set of session shared secrets.

The following is a possible fully abstract model of an SSH-TLP client:

$$\begin{array}{l} SSHClient(IDC, me, you, CAlgs, TrustedKeys) = \\ \quad send!me!you!IDC \to \\ \quad receive!you!me?IDS \to \\ \quad \sqcap_{cookieC \in Cookies}\, send!me!you!(cookieC, CAlgs) \to \\ \quad receive!you!me?(cookieS, SAlgs) \to \\ \quad g := Negotiate(CAlgs, SAlgs, `g') \in CryptoParameter; \\ \quad p := Negotiate(CAlgs, SAlgs, `p') \in CryptoParameter; \\ \quad \sqcap_{x \in DHSecrets}\, send!me!you!EXP(g, x, p) \to \\ \quad receive!you!me?(PubKeyS, DHPublicS, [\{sshash\}]_{PriKeyS}) \to \\ \quad GO(EXP(DHPublicS, x, p), sshash, PubKeyS, TrustedKeys) \end{array} \tag{26}$$

Note that this model is part of the abstract $SYSTEM$, where one or more instances of $SSHClient$ can partake together with one or more instances of the server model for SSH-TLP, and the intruder.

The $SSHClient$ process starts a protocol session with the server by sending it the client identification string denoted IDC. The server responds with IDS, the server identification string. Then the client generates and sends a nonce $cookieC$, followed by the client lists of supported algorithms $CAlgs$. The server responds sending a nonce $cookieS$, followed by the server lists of supported algorithms $SAlgs$. The client then computes the value of the Diffie Hellman (DH) parameters g and p by computing the $Negotiate(CAlgs, SAlgs, Param)$ function, which returns the requested negotiated algorithm parameter named $Param$, obtained from the supported client and server algorithms $CAlgs$ and $SAlgs$. $CryptoParameter \subseteq Message$ is the set of messages that can be used as cryptographic algorithms or parameters. Once g and p have been obtained, the client generates a random private key x and sends $EXP(g, x, p)$ (its DH public key, as explained below), which is a message representing the result of the modular exponentiation $e = g^x \bmod p$. This message is added to the datatype and is defined as

$$\textsc{Exp}\ Message, Message, Message$$

along with the syntactic sugar $\textsc{Exp}\ g, x, p = EXP(g, x, p)$. From the point of view of the intruder knowledge derivation relation $\vdash$, the $EXP(\cdot)$ message is a non invertible function like an hash. Accordingly, only a single **exponentiation** rule is needed, defined as

exponentiation $B \vdash g \wedge B \vdash x \wedge B \vdash p \Rightarrow B \vdash EXP(g, x, p)$

This rule is similar to the **hashing** one. For this reason, all previously proven results still hold.

The following additional property of the $EXP(\cdot)$ function must be modelled

$$EXP(EXP(g, y, p), x, p) = EXP(EXP(g, x, p), y, p) \tag{27}$$

so that two messages satisfying equation (27) are considered equal by all parties, both protocol actors and the intruder.

When the EXP function is used for the DH key exchange algorithm, x and y can be considered as DH private keys, $EXP(g,x,p)$ and $EXP(g,y,p)$ as DH public keys, and the expression in (27) as the DH shared key that can be obtained by each actor. Finally, g and p are the DH group parameters. In the next step of the protocol, the client receives a message containing the opaque server public key $PubKeyS$, the opaque server DH public key $DHPublicS$ and the server signed final hash $[\{sshash\}]_{PriKeyS}$. The server computes its DH public key as $DHPublicS = EXP(g,y,p)$, where y is the server's DH private key. However the client is modeled to receive an opaque $DHPublicS$ message, because the server DH public key is an opaque value from the client's point of view. Analogous reasoning holds for public and private server keys $PubKeyS$ and $PriKeyS$. The server final hash $sshash$ is the value upon which agreement is required, and contains all the relevant data of a protocol session, i.e.

$$
\begin{aligned}
sshash = H(H(&IDC, IDS, (cookieC, CAlgs), (cookieS, SAlgs), PubKeyS,\\
&EXP(g,x,p), DHPublicS, EXP(DHPublicS, x, p)))
\end{aligned}
$$

Note the double hashing $H(H(\cdot))$. The internal hash is the final hash, computed on all relevant session data, prescribed by the protocol specification, while the external hash is the one required by the signature algorithm, which prescribes to hash, and then cipher, the value that must be signed. The $EXP(DHPublicS, x, p)$, that is used inside the final hash, is the DH shared key as computed by the client. This is the session secret shared between the client and the server. Finally, the $GO(\cdot)$ process is defined as

$$
\begin{aligned}
&GO(DHKey, sshash, PubKeyS, TrustedKeys) =\\
&\quad (claimSecret.me.you.DHKey \rightarrow finished.me.you.sshash)\\
&\not< PubKeyS \in TrustedKeys \not> STOP
\end{aligned}
$$

That is, if the server public key $PubKeyS$ is in the $TrustedKeys$ set, the set of server public keys trusted by the client, then the protocol run ends well, and all security properties can be claimed, namely the secrecy of the DH shared key $DHKey$, and the agreement on the server signed final hash, $sshash$. Note that agreement on $sshash$ implies agreement on all the data items on which the final hash is computed. However, since the final hash is used later on with other data in order to establish a set of session keys, it is required that actors agree explicitly on $sshash$, and not only on its content.

Now that the abstract model of the client has been introduced, it is possible to show how this model can be refined, according to the modelling approach presented in this paper. Three kinds of details that have been studied in isolation, namely the encoding/decoding layer, as modeled in section 3, the encoding of data to be ciphered or hashed and key material, as modeled in section 4, and the cryptographic algorithms and parameters, as modeled in section 5.2, are applied alltogether in this model.

It is worth pointing out that the three different kinds of refinements must be applied on the same system in the correct order, so as to avoid improper interactions: first, data to be ciphered or hashed and key material should be refined; then cryptographic algorithms and parameters should be added; finally the encoding/decoding layer should be introduced.

For better clarity, we will no longer refer to a single set of parameters $Encoding \subseteq Message$. Instead, a different set is defined for each kind of parameter used for refinement. Specifically, $MarshEncoding$ will denote messages used as the encoding/decoding layer parameters; $CryptoEncoding$ will denote messages used as encoding parameters for data to be ciphered or hashed and for key material; $CryptoParameter$ will denote messages used as cryptographic algorithms and related parameters.

The refined model for the SSH-TLP client includes, in addition to a refined model of the client role, an encoding/decoding layer model E_A, and a decoding process model DEC_A. Most of the complexity of the protocol indeed stands in E_A and DEC_A.

The refined client role model can be written as follows:

$$
\begin{array}{l}
SSHClientRef(IDC, me, you, CAlgs, TrustedKeys) = \\
\quad send!me!you!(\text{ATOM } \mathcal{L}, (string, IDC)) \rightarrow \\
\quad receive!you!me?(\text{ATOM } \mathcal{L}, (string, IDS)) \rightarrow \\
\quad \sqcap_{cookieC \in Cookies}\, send!me!you!(\text{ATOM } \mathcal{L}, (KEX, (cookieC, CAlgs))) \rightarrow \\
\quad receive!you!me?(\text{ATOM } \mathcal{L}, (KEX, (cookieS, SAlgs))) \rightarrow \\
\quad g := Negotiate(CAlgs, SAlgs, \text{`}g\text{'}) \in CryptoParameter; \\
\quad p := Negotiate(CAlgs, SAlgs, \text{`}p\text{'}) \in CryptoParameter; \\
\quad FinalHashAlg := Negotiate(CAlgs, SAlgs, \text{`}FinalHashAlg\text{'}) \in CryptoParameter; \\
\quad SignHashAlg := Negotiate(CAlgs, SAlgs, \text{`}SignHashAlg\text{'}) \in CryptoParameter; \\
\quad SignKeyType := Negotiate(CAlgs, SAlgs, \text{`}SignKeyType\text{'}) \in CryptoParameter; \\
\quad SignMode := Negotiate(CAlgs, SAlgs, \text{`}SignMode\text{'}) \in CryptoParameter; \\
\quad SignPadding := Negotiate(CAlgs, SAlgs, \text{`}SignPadding\text{'}) \in CryptoParameter; \\
\quad \sqcap_{x \in DHSecrets}\, send!me!you!(\text{ATOM } \mathcal{L}, ((bin\_pack, mpint), EXP(g, x, p))) \rightarrow \\
\quad receive!you!me?(\text{ATOM } \mathcal{L}, ((bin\_pack, string, mpint, string), \\
\qquad (PubKeyS, DHPublicS, [\{(\text{ATOM } \mathcal{L}, ((SignMode, SignPadding), \\
\qquad y))\}]_{PriKeyS}))) \rightarrow \\
\quad int\_send_{me}!(y, (SignMode, SignPadding)) \rightarrow int\_receive_{me}!sshash\_enc \rightarrow \\
\quad GO(EXP(DHPublicS, x, p), sshash\_enc, PubKeyS, TrustedKeys)
\end{array}
\tag{28}
$$

where:

$$
\begin{aligned}
KEX &= (bin\_pack, bytes, namelists) \\
sshash\_enc &= H(\text{ATOM } \mathcal{L}, (SignHashAlg, \\
&\quad e_{me}(SignHashAlg, H(\text{ATOM } \mathcal{L}, (FinalHashAlg, \\
&\quad e_{me}(FinalHashAlg, (IDC, IDS, (cookieC, CAlgs), (cookieS, SAlgs), \\
&\quad PubKeyS, EXP(g, x, p), DHPublicS, \\
&\quad EXP(DHPublicS, x, p))))))))
\end{aligned}
$$

Here, *string* denotes the SSH string encoding, *bytes* denotes a raw encoding (a sequence of bytes), *mpint* denotes the SSH encoding of a multiple precision integer and *namelists* denotes the SSH encoding of a list of lists of strings. The *bin_pack* parameter specifies that the SSH binary packet encoding must be used, with a payload made up of as many fields as the number of subsequent parameters, each of which in turn specifies the encoding to be applied to each field. So, for example, KEX specifies the encoding of an SSH KEX packet, which is an SSH binary packet with a payload that includes a sequence of bytes followed by a list of strings. The $FinalHashAlg$, $SignHashAlg$, $SignKeyType$, $SignMode$, and $SignPadding$ variables represent the negotiated cryptographic algorithms and parameters (in addition to p and g, which were already present in the abstract model).

Note that the $e_{me}(\cdot)$ function in the expression for $sshash\_enc$ represents the implementation of encoding transformations. Its detailed description is omitted here for simplicity.

Let us show now how, by the results presented in this paper, this refined model can be simplified to perform formal verifications of security properties, and what checks are needed on the sequential code that implements encoding/decoding functions in order to safely apply each simplification.

In order to remove the models of the encoding/decoding layer E_A from the refined system model $SYSTEM'$, it is needed that the desired security properties do not depend on the send and receive events and that the intruder knowledge used for verification in the abstract system includes encoding parameters (already shown to be reasonable constraints). Moreover, the implementations of the marshalling functions must be checked to be memoryless, and to access no external data but their input parameters (which amounts to check an information flow property on sequential code). Since the properties to be verified for this protocol are secrecy and authentication, even the ATOM $\mathcal{L}$ term and the messages in $MarshEncoding$ that have been added by the encoding/decoding layer refinement procedure can be abstracted away without need of further checks.

The models of cryptographic algorithms and parameters only affect data terms including cryptographic operations. For verifying the secrecy property, they can be abstracted away with no additional check. When verifying authentication, they can be abstracted away if condition (22) holds, which can be ensured by explicitly adding the negotiated cryptographic algorithms and parameters to the internal actions used to specify agreement. For example, the finished action should become

$$finished.me.you.(sshash\_enc, FinalHashAlg, SignHashAlg))$$

Note that this condition is needed even though the negotiated algorithms are already included in $CAlgs$ and $SAlgs$, because it is necessary to ensure that the specific negotiated algorithms are agreed, rather than the sets of algorithms from which they are selected.

Finally, in order to abstract the decoding processes DEC_A away, condition (16) must be checked on the sequential code of the implementation of the encoding/decoding functions $e_{me}(\cdot)$ and $d_{me}(\cdot)$, along with the same data flow properties already specified for the marshalling and unmarshalling functions. When verifying secrecy, even data encodings can be abstracted, by removing the $e_{me}(\cdot)$ functions from the client role model, provided the implementation of $e_{me}(\cdot)$ is shown to be injective. The same simplification can be applied when verifying authentication, but in this case correctness of the implementation of encoding/decoding functions must be checked on their sequential code with respect to their specification, and condition (22) must be guaranteed to hold. The latter condition can be ensured as already explained for the abstraction of cryptographic parameters.

6 Conclusions

The work presented in this paper is a useful step towards the verification of refined security protocol models that take encoding and decoding data transformations into account, thus allowing formal verification to get closer to protocol code written in a programming language.

The main contribution of the paper is the formulation of a set of sufficient conditions under which the models of data encoding and decoding functions can be safely simplified or even completely abstracted away under the Dolev-Yao assumption, and its formal justification.

It has been shown that different conditions apply to different kinds of encoding and decoding operations. Specifically, two kinds of operations can be distinguished.

For what concerns marshalling and unmarshalling operations applied when sending and receiving messages on public channels, a refined protocol model corresponding to a typical layered implementation of such operations has been defined. It has been proven that, in order to verify secrecy or authentication properties on the refined model, it is enough to verify those properties on the corresponding abstract model, provided that the intruder knows the encoding parameters, which is a reasonable assumption. Alternatively, in order to check a generic security property defined on protocol traces, still a simplified refined model, that excludes explicit models of marshalling and unmarshalling functions but preserves the parameters of such operations, can be used in place of the full one.

The model of encoding schemes that has been developed in this work is general enough to take into account a widely used class of encoding schemes and implementations, namely the memoryless and side effect free ones. Moreover, on the marshalling and unmarshalling operations, no assumption has been made about invertibility, nor about implementation correctness; instead, it is only required that the implementation of encoding and decoding functions satisfies some data flow properties, that can be checked by standard static analysis techniques. The consequence of this result is that, if such data flow properties are satisfied on implementation code, then even erroneous specifications or implementations of encoding schemes cannot be more harmful than a Dolev-Yao intruder is.

For what concerns instead encoding and decoding operations made on key material or on data on which cryptographic operations are applied, a similar general model supporting a wide class of data encoding schemes has been introduced. On this model, it has been shown that the models of the decoding operations can be safely omitted when verifying security properties, but under stricter constraints: it is required that the implementations of encoding functions are the inverses of the corresponding decoding functions for each actor. If secrecy is being verified, then the encoded form of messages can also be safely abstracted away, by additionally checking injectivity of the encoding function implementation. If instead authentication is being verified, then the encoded form of messages can be abstracted if it is checked that the implementation of the encoding and decoding functions is correct w.r.t. their specification, and if all actors agree on the same encoding parameters. The latter condition can be checked as part of the abstract protocol verification.

Therefore, an important distinction in criticality has been shown to exist between channel marshalling and unmarshalling transformations, for which neither invertibility nor correctness are needed, and encoding and decoding operations applied to key material or to data on which cryptographic operations are applied, for which they are needed in order to use the fully abstract model.

A previous work related to our approach is [15], where it is showed that any Dolev-Yao model of a WS-Security protocol, where the XML encoding is embedded into the datatype, can be simplified into a more abstract protocol, still preserving secrecy and agreement, by abstracting away XML tag encodings and just keeping the contents of XML elements. It turns out that the results being presented here generalize in two directions the work in [15]. On the one hand, the work presented here takes the implementation of encoding functions into account, and is not limited to consider only how they are specified. On the other hand, the results proposed here apply to any possible encoding scheme, XML and WS-Security being just a particular instance of it.

Although some of the results presented in this paper are probably not so surprising, all of them have been formally proved for the first time in this paper, and they find application in improving the development of formally verified implementation code of security protocols, both using the code generation approach or the model extraction approach.

For example, for any abstract protocol model where the intruder is assumed to know the encoding parameters used in marshalling and unmarshalling operations on transmitted and received data, if this abstract model has already been verified to be secure with respect to secrecy or authentication under a Dolev-Yao intruder, then any corresponding refined model taking into account the encoding and decoding transformations as specified in this paper is now implied to be secure too, with no other effort required. This result is especially useful in code generation, because the developer only needs to write and verify the abstract protocol model, and the code generation engine can take care of ensuring that the generated code meets the data flow assumptions made about the refined model. So, an important step towards a formally safe refinement process in methods based on code generation has been made.

A similar approach can be used to deal with encoding and decoding operations made on data on which cryptographic operations are applied. In this case, we can safely analyze secrecy and authentication on the abstract model only, provided that we can show that the implementation of the encoding and decoding functions is correct with respect to their definition and that they satisfy an information flow property. Alternatively, an intermediate model where encoding operations are explicitly modeled while decoding ones are not can be safely used under the weaker constraint of ensuring or assuming that the implementation of encoding is the inverse of the implementation of decoding for that specific actor. By the way, it has been shown that, at least for an important class of cases, the check on the correctness of the encoding and decoding functions is viable.

When adopting a model extraction approach, the results presented in this paper make it possible to avoid extracting from code a complex model that represents all the implementation details of encoding and decoding functions. Instead, a simpler model can be extracted, provided that some static checks are first made on the implementation code.

We can expect that, by abstracting away implementation details from the model, a reduction in the time needed for formal verification and the possibility to analyze more complex protocols are finally achieved. It may be argued that, in order to abstract details away, some properties have to be verified on the implementation code. This is true, however, the implementation code to be checked is sequential and independent from the intruder, and thus simpler to be checked in isolation than it is checking a full protocol model, which is a concurrent system that includes a detailed model of the encoding and decoding functions and an intruder.

Moreover, it should be considered that sometimes the source code that implements encoding and decoding functions may not be available (e.g. for libraries). In such cases, code extraction is even impossible. The results presented in this paper show what are the requirements on this code, which can be used to derive specific testing procedures in order to get a good assurance level. In order to obtain the results achieved in this paper, some general extensions to the results presented in [11] have been made. While these extensions are immediately useful in this work, they are also stand alone achievements that can be used as a starting point for future works.

Some issues on the topics presented in this paper are still open for future work. In particular, it can be interesting to explore under which conditions the final transformation back to the original abstract model is safe for other security trace properties or for security properties not defined on traces (e.g. strong secrecy). Another interesting further work is to consider verification with computational models instead of verification with Dolev-Yao models.

APPENDIX

A Proofs

This appendix contains the proofs of lemma 2 (section 3) and theorem 3 (section 4). In order to avoid ambiguities, it is worth reminding that each proof refers to the notation used in the section where the corresponding theorem or lemma is stated. For instance, in the two sections, the symbol $SYSTEM'$ refers to two different systems, and so do the corresponding proofs.

A.1 Proof of lemma 2

Proof. In order to carry out the proof, the states reachable from $E\_INTRUDER$ must be written explicitly. For this reason, let E_i be defined as a generic state reachable from $|||_{A \in Honest} E_A$. Moreover, for each state E_i, let S_{E_i} be the set of all the encoded messages ready to be delivered to the $INTRUDER'$ component of $E\_INTRUDER$, but not yet dispatched to it.

Formally, S_{E_i} is a set of messages that can be defined inductively. In the initial state

$$E_0 = |||_{A \in Honest} E_A$$

we have $S_{E_0} = \emptyset$. The evolution of S_{E_i} after an event e can be represented by an extension of the $\xrightarrow{e}$ state transition relation as follows: $S_{E_i} \xrightarrow{e} S_{E_j}$ means that the occurrence of e in a state where the set of encoded messages ready to be delivered to $INTRUDER'$ is S_{E_i} leads to a new state where the new set is S_{E_j}. Now, since the events e occurring in E_i can only take 4 different forms, the relation $\xrightarrow{e}$ on sets of messages can be defined by enumeration. By the definition of process E_A in (4), it follows that

$$\begin{array}{lcl}
S_{E_i} & \xrightarrow{send.A.B.(\textsc{Atom}\ \mathcal{L},(a,M))} & S_{E_i} \cup \{e_A(a, M)\} \\
S_{E_i} & \xrightarrow{receive.A.B.(\textsc{Atom}\ \mathcal{L},(a,M))} & S_{E_i} \\
S_{E_i} & \xrightarrow{int_send.A.B.e_A(a,M)} & S_{E_i} \setminus \{e_A(a, M)\} \\
S_{E_i} & \xrightarrow{int_receive.A.B.y} & S_{E_i}
\end{array}$$

A generic state of $E\_INTRUDER$ then takes the form

$$E\_INTRUDER_i \triangleq (E_i \parallel INTRUDER'(S'_i)) \setminus \{int\}$$

where S'_i is the current intruder knowledge. The initial state is

$$E\_INTRUDER_0 = E\_INTRUDER$$

i.e.

$$E\_INTRUDER_0 = (E_0 \parallel INTRUDER'(S'_0)) \setminus \{int\}$$

where S'_0 is defined to be the set IK'_0.

Finally, let us define the total knowledge S_{T_i} associated with state $E\_INTRUDER_i$ as

$$S_{T_i} \triangleq S_{E_i} \cup S'_i$$

It is worth noting that, in state $E\_INTRUDER_i$, by lemma 1, $deds(S'_i) \subseteq deds(S_{T_i})$.

Let us preliminarily prove the following

Lemma 3.

$$\begin{array}{c}
E_INTRUDER_i \xrightarrow{\tau} E_INTRUDER_j \\
\Longrightarrow \\
S_{T_i} = S_{T_j}
\end{array}$$

Proof. There are two possible cases: τ comes from an *int_send* event, or τ comes from an *int_receive* event.

if τ comes from event $int\_send.A.B.e_A(a,M)$**:** then, by the definition of each process E_A, a corresponding $send.A.B.(\textsc{Atom}\ \mathcal{L},(a,M))$ must have previously occurred. So, by the definition of S_{E_i},

$$e_A(a,M) \in S_{E_i}$$

and

$$S_{E_j} = S_{E_i} \setminus \{e_A(a,M)\}$$

and, by the definition of $INTRUDER'$,

$$S'_j = S'_i \cup \{e_A(a,M)\}$$

so $S_{T_i} = S_{T_j}$.

if τ comes from event $int\_receive.A.B.y$**:** then, by the definition of S_{E_i},

$$S_{E_j} = S_{E_i}$$

and, by the definition of $INTRUDER'$,

$$S'_j = S'_i$$

so $S_{T_i} = S_{T_j}$.

□

Now that lemma 3 is proven, the following property can be proven for each trace tr, which implies lemma 2:

$$\begin{gathered}(E\_INTRUDER_0 \stackrel{tr}{\Longrightarrow} E\_INTRUDER_f)\\ \Longrightarrow \\ \exists\ IK''_f \mid (INTRUDER(IK''_0) \stackrel{tr}{\Longrightarrow} INTRUDER(IK''_f) \wedge \\ deds(S_{T_f}) \subseteq deds(IK''_f))\end{gathered}$$

The proof is based on induction on the length of trace tr.

Base ($length(tr) = 0$) Since tr is the empty trace

$$E\_INTRUDER_0 \stackrel{\tau}{\longrightarrow}^* E\_INTRUDER_f$$

Then, by lemma 3, we have that

$$S_{T_f} = S_{T_0} = S_{E_0} \cup S'_0 = IK'_0$$

Moreover, $INTRUDER(IK''_0)$ cannot execute internal events, so if we take

$$IK''_f = IK''_0$$

then, considering that by definition (6) it follows that

$$IK'_0 \subset IK''_0$$

we can conclude $deds(S_{T_f}) \subseteq deds(IK''_f)$.

Induction ($length(tr) = n + 1$) The trace tr is then composed of a subtrace tr' of length n, followed by the $n + 1^{th}$ event. There are three possible cases: the $n + 1^{th}$ event is a *send*, *receive* or *leak* event.

case $n+1^{th}$ **event is a** *send* **event:** by the definitions of processes and by inductive hypotheses

$$E\_INTRUDER_0 \stackrel{tr'}{\Longrightarrow} E\_INTRUDER_i$$

Moreover:

$$\stackrel{\tau}{\longrightarrow}^* \begin{array}{c} E\_INTRUDER_i \\ \stackrel{send.A.B.(\textsc{Atom}\ \mathcal{L},(a,M))}{\longrightarrow} \\ E\_INTRUDER_f \end{array} \stackrel{\tau}{\longrightarrow}^*$$

By lemma 3, S_{T_i} will remain unchanged for each τ transition before the *send* event, and S_{T_f} will remain unchanged for each τ transition after the *send* event. Moreover, by definition,

$$S_{T_f} = S_{T_i} \cup \{e_A(a, M)\}$$

At the other side, by inductive hypotheses, there exists IK''_i such that

$$INTRUDER(IK''_0) \stackrel{tr'}{\Longrightarrow} INTRUDER(IK''_i)$$

and

$$INTRUDER(IK''_i) \stackrel{send.A.B.(\textsc{Atom}\ \mathcal{L},(a,M))}{\longrightarrow} INTRUDER(IK''_f)$$

where $IK''_f = IK''_i \cup \{(\textsc{Atom}\ \mathcal{L}, (a, M))\}$
By inductive hypotheses, $deds(S_{T_i}) \subseteq deds(IK''_i)$; by definition (5) of $e_A(a, M)$ and by lemma 1, it follows that $deds(e_A(a, M)) \subseteq deds((\textsc{Atom}\ \mathcal{L}, (a, M)))$. So

$$deds(S_{T_i} \cup \{e_A(a, M)\}) \subseteq deds(IK''_i \cup \{(\textsc{Atom}\ \mathcal{L}, (a, M))\})$$

thus $deds(S_{T_f}) \subseteq deds(IK''_f)$.

case $n+1^{th}$ **event is a** *receive* **event:** by process definitions and by inductive hypotheses

$$E\_INTRUDER_0 \stackrel{tr'}{\Longrightarrow} E\_INTRUDER_i$$

Moreover:

$$\stackrel{\tau}{\longrightarrow}^* \begin{array}{c} E\_INTRUDER_i \\ \stackrel{receive.A.B.(\textsc{Atom}\ \mathcal{L},(a,d_A(a,y)))}{\longrightarrow} \\ E\_INTRUDER_f \end{array} \stackrel{\tau}{\longrightarrow}^*$$

By lemma 3, S_{T_i} will remain unchanged for each τ transition before the *receive* event, and S_{T_f} will remain unchanged for each τ transition after the *receive* event. Moreover

$$S_{T_f} = S_{T_i}$$

At the other side, by inductive hypotheses, there exists IK''_i such that

$$INTRUDER(IK''_0) \stackrel{tr'}{\Longrightarrow} INTRUDER(IK''_i)$$

Then, we need to show that

$$INTRUDER(IK''_i) \stackrel{receive.A.B.(\textsc{Atom}\ \mathcal{L},(a,d_A(a,y)))}{\longrightarrow} INTRUDER(IK''_f)$$

where $IK''_f = IK''_i$, because the intruder knowledge does not change on *receive* events.
Since, by inductive hypotheses, $deds(S_{T_i}) \subseteq deds(IK''_i)$, and it has been shown that both $S_{T_f} = S_{T_i}$ and $IK''_f = IK''_i$ hold, it is possible to conclude that $deds(S_{T_f}) \subseteq deds(IK''_f)$ holds too. In order to complete the proof of this case, it is enough to show that the *receive* event indeed can happen in $INTRUDER(IK''_i)$, that is, in order to send $(\textsc{Atom}\ \mathcal{L}, (a, d_A(a, y)))$ on

the *receive* channel, $INTRUDER(IK''_i)$ must be able to derive the required message from its knowledge.
If a and y are derivable from IK''_i, then, by definition (5), $d_A(a, y)$ is derivable too, and, by applying the **pairing** rule, $(a, d_A(a, y))$ is derivable too. Message a is derivable from IK''_i with the **member** rule, since

$$a \in Encoding \subset IK''_0 \subseteq IK''_i$$

Message y is derivable from IK''_i because, since the *receive* event can happen in $E\_INTRUDER_i$, then

$$y \in deds(S_{T_i})$$

and, by inductive hypotheses,

$$deds(S_{T_i}) \subseteq deds(IK''_i)$$

thus $y \in deds(IK''_i)$.
Finally, if $\textsc{Atom}\ \mathcal{L}$ is derivable from IK''_i, then, by applying the **pairing** rule, the required $(\textsc{Atom}\ \mathcal{L}, (a, d_A(a, y)))$ message can be obtained. The message $\textsc{Atom}\ \mathcal{L}$ is derivable from IK''_i with the **member** rule, because

$$\textsc{Atom}\ \mathcal{L} \in IK''_0 \subseteq IK''_i$$

case $n + 1^{th}$ **event is a** *leak* **event:** by process definitions and by inductive hypotheses

$$E\_INTRUDER_0 \stackrel{tr'}{\Longrightarrow} E\_INTRUDER_i$$

Moreover:

$$\begin{gathered} E\_INTRUDER_i \\ \stackrel{\tau}{\longrightarrow}^* \stackrel{leak.M}{\longrightarrow} \stackrel{\tau}{\longrightarrow}^* \\ E\_INTRUDER_f \end{gathered}$$

By lemma 3, S_{T_i} will remain unchanged for each τ transition before the *leak* event, and S_{T_f} will remain unchanged for each τ transition after the *leak* event. Moreover, the *leak* event is not engaged by the E_i process, so $S_{E_f} = S_{E_i}$, and, by the definition of intruder, $S_f = S_i$, so $S_{T_f} = S_{T_i}$.
At the other side, by inductive hypotheses, there exists IK''_i such that

$$INTRUDER(IK''_0) \stackrel{tr'}{\Longrightarrow} INTRUDER(IK''_i)$$

and

$$INTRUDER(IK''_i) \stackrel{leak.M}{\longrightarrow} INTRUDER(IK''_f)$$

where $IK''_f = IK''_i$. So, by inductive hypotheses, it follows that $deds(S_{T_f}) \subseteq deds(IK''_f)$.
Finally, the $leak.M$ event can indeed happen in $INTRUDER(IK''_i)$ because, by hypotheses, the $leak.M$ event can happen in $E\_INTRUDER_i$, which implies $M \in S_{T_i}$. Since, by inductive hypotheses, $deds(S_{T_i}) \subseteq deds(IK''_i)$, it follows that $M \in IK''_i$ too, so the $leak.M$ event can happen in $INTRUDER(IK''_i)$.

□

A.2 Proof of theorem 3

Proof. In order to lighten notation, let us define

$$comm_A \triangleq \{send.A, receive?B.A\}$$
$$P_A^{dec-} \triangleq (P'_A \parallel DEC_A) \setminus priv_A$$

Since P_A^{dec-} has been introduced, an equivalent expression for $SYSTEM'$ is the following:

$$SYSTEM' = |||_{A \in Honest} P_A^{dec-} \parallel INTRUDER(IK'_0)$$

In order to complete the proof, the following lemma is needed. It is proven below.

Lemma 4. *If* (16) *and* (17) *hold, then, for any $A \in Honest$,*

$$(P^*_A \parallel INTRUDER(IK^*_0)) \setminus comm_A \sqsubseteq (P^{dec-}_A \parallel INTRUDER(IK'_0)) \setminus comm_A$$

Lemma 4 states that, under the same assumptions used by theorem 3, if communication channels are hidden, then one abstract protocol logic P^*_A acting with the intruder, is refined by its more detailed protocol logic P^{dec-}_A, that explicitly models the decoding process, acting with the intruder.

The proof idea for theorem 3 is to refine all abstract protocol logics P^*_A in $SYSTEM^*$ into their refined counterparts P^{dec-}_A, one at a time, thus refining the whole $SYSTEM^*$ to $SYSTEM'$. Note that it is not possible to trivially infer this result directly from lemma 4. Indeed, it is true that, for any processes A and R, and for any context $C[\,]$,

$$A \sqsubseteq R \Rightarrow C[A] \sqsubseteq C[R]$$

So, let us consider that lemma 4 holds for honest actor H. Then, setting the context to

$$C[X] = (|||_{A \in Honest \setminus \{H\}} P^*_A \parallel X) \setminus comm$$

leads to a process

$$(|||_{A \in Honest \setminus \{H\}} P^*_A \parallel ((P^*_H || INTRUDER(IK^*_0)) \setminus comm_H)) \setminus comm$$

which is not trivially proven to have all and the same traces of $SYSTEM^*$.

The proof steps of trace refinement reported here use two CSP operator properties that are introduced now.

The first property states that, for any processes P, Q, R, if $\alpha P \cap \alpha Q = \emptyset$, then

$$P \parallel (Q \parallel R) = (P \mathbin{|||} Q) \parallel R$$

Remember that parallel operator without any subscripted event means synchronization on the intersection of the alphabets.

Informally, this property means that if P and Q cannot communicate directly (because, being the intersection of their alphabets empty, they cannot synchronize on any event), then, on one hand, P communicating with $Q \parallel R$, is actually only communicating with R; on the other hand, Q is only communicating with R, and never with P. Thus, R acts as a proxy between P and Q, while the latter two processes can execute in interleaving.

The second property states that, for any processes P, Q, R, if $\alpha P \cap \alpha Q = \emptyset$, then

$$(P \parallel (Q \parallel R)) \setminus (\alpha Q \cap \alpha R) \cup (\alpha P \cap \alpha R) = (P \parallel ((Q \parallel R) \setminus (\alpha Q \cap \alpha R))) \setminus (\alpha P \cap \alpha R)$$

Informally, this property means that, since P and Q never communicate directly, and R is their proxy, from P's view it is irrelevant wether communication between Q and R is observable or not, thus allowing to put P in parallel either with $Q \parallel R$ or with $(Q \parallel R) \setminus (\alpha Q \cap \alpha R)$. However, from an observer point of view, communication between Q and R must always be hidden, so if it is not hidden with the $(Q \parallel R) \setminus (\alpha Q \cap \alpha R)$ process, it must be hidden at the top level process.

Indeed, in the used Dolev-Yao approach, any pair of protocol actors has disjoint alphabets, because the intruder is the only proxy between any pair of actors, and they can never communicate directly. Thus, these properties directly apply when P and Q are two processes representing interleaved actors and R is the intruder.

Trace refinement is proven by induction over the number of protocol logics that are step by step refined in $SYSTEM^*$.

base it will be proven that $SYSTEM^*$, where all protocol logics are abstract, is refined by a process where one protocol logic is refined, that is

$$\forall X \in Honest$$
$$(|||_{A \in Honest} P^*_A \parallel INTRUDER(IK^*_0)) \setminus comm \sqsubseteq \left(\left(|||_{A \in Honest \setminus \{X\}} P^*_A ||| P^{dec-}_X\right) \parallel INTRUDER(IK'_0)\right) \setminus comm$$

The proof steps are:

$$(|||_{A\in Honest}P_A^* \parallel INTRUDER(IK_0^*)) \setminus comm$$

$$= \left((|||_{A\in Honest\setminus\{X\}}P_A^* \ ||| \ P_X^*\right) \parallel INTRUDER(IK_0^*)) \setminus comm$$

$$= \langle \text{by letting } Others = |||_{A\in Honest\setminus\{X\}}P_A^* \rangle$$

$$((Others \ ||| \ P_X^*) \parallel INTRUDER(IK_0^*)) \setminus comm$$

$$= \langle \text{by property of } ||| \text{ and } \parallel, \text{ and by } \alpha(Others) \cap \alpha P_X^* = \emptyset \rangle$$

$$(Others \parallel (P_X^* \parallel INTRUDER(IK_0^*))) \setminus comm$$

$$= \langle \text{by hiding property; letting } comm^- = comm \setminus comm_X \rangle$$

$$(Others \parallel (P_X^* \parallel INTRUDER(IK_0^*) \setminus comm_X)) \setminus comm^-$$

$$\sqsubseteq \langle \text{consider the context } Others \text{ surrounding the process refined in lemma 4} \rangle$$

$$\left(Others \parallel \left(\left(P_X^{dec-} \parallel INTRUDER(IK_0')\right) \setminus comm_X\right)\right) \setminus comm^-$$

$$= \langle \text{by hiding property} \rangle$$

$$\left(Others \parallel \left(P_X^{dec-} \parallel INTRUDER(IK_0')\right)\right) \setminus comm$$

$$= \langle \text{by property of } ||| \text{ and } \parallel, \text{ and by } \alpha(Others) \cap \alpha P_X^* = \emptyset \rangle$$

$$= \left(\left(Others \ ||| \ P_X^{dec-}\right) \parallel INTRUDER(IK_0')\right) \setminus comm$$

$$= \langle \text{by definition of } Others \rangle$$

$$= \left(\left(|||_{A\in Honest\setminus\{X\}}P_A^* \ ||| \ P_X^{dec-}\right) \parallel INTRUDER(IK_0')\right) \setminus comm$$

induction it will be shown that if the refinement relation holds when n actors have been refined, then it keeps holding when the $n+1^{th}$ actor is refined too. Let Ref be the set of already refined actors, then $\forall\ X \in Honest \setminus Ref$

$$\left(\left(|||_{A\in Honest\setminus Ref}P_A^* \ ||| \ |||_{B\in Ref}P_B^{dec-}\right) \parallel INTRUDER(IK_0')\right) \setminus comm$$

$$= \left(\left(|||_{A\in Honest\setminus(Ref\cup\{X\})}P_A^* \ ||| \ |||_{B\in Ref}P_B^{dec-} \ ||| \ P_X^*\right) \parallel INTRUDER(IK_0')\right) \setminus comm$$

$$\sqsubseteq \Big\langle \text{by setting } Others = |||_{A\in Honest\setminus(Ref\cup\{X\})}P_A^* \ ||| \ |||_{B\in Ref}P_B^{dec-}; \text{by using the same steps as in base case}$$

$$\left(\left(|||_{A\in Honest\setminus(Ref\cup\{X\})}P_A^* \ ||| \ |||_{B\in Ref}P_B^{dec-} \ ||| \ P_X^{dec-}\right) \parallel INTRUDER(IK_0')\right) \setminus comm$$

$$= \left(\left(|||_{A\in Honest\setminus(Ref\cup\{X\})}P_A^* \ ||| \ |||_{B\in Ref\cup\{X\}}P_B^{dec-}\right) \parallel INTRUDER(IK_0')\right) \setminus comm$$

It is worth noting that, in the inductive step, the intruder knowledge in the abstract system (the one having n refined actors) is set to IK_0', and not IK_0^*. Indeed, after the first refinement step made in the base case, the intruder knowledge is "restricted" to IK_0', the refined one. This is not an issue during the inductive step, because condition (17) in lemma 4 requires, in the abstract system, that the intruder knowledge is a *weak* superset of the intruder knowledge in the refined system.

□

Now the proof of lemma 4 is given.

Proof. A weak simulation relation between the abstract process

$$P_A^* \parallel INTRUDER(IK_0^*) = f(P_A') \parallel INTRUDER(IK_0^*)$$

and the refined process

$$P_A^{dec-} \parallel INTRUDER(IK_0') = ((P_A' \parallel DEC_A) \setminus priv_A) \parallel INTRUDER(IK_0')$$

is first proven, which is then shown to imply the desired trace refinement. Briefly, a weak simulation relation binds the transitions between external states of an abstract process to the transitions between external states of a refined (also called *concrete* in other works) process, but each process is still allowed to perform any internal step in between two external states. More details about the weak refinement used here can be found, for example, in [19].

An external state of the refined protocol logic of actor A is defined as a generic state P'_{A_i} of the process P'_A, such that P'_{A_i} does not begin with an action in the $priv_A$ set; that is, P'_{A_i} does not take the form $ev \to Q$, with $ev \in priv_A$ (note that, by construction of P'_A, if an event $ev \in priv_A$ can occur, it is the only event that can occur). Accordingly, an external state of the refined process takes the form

$$SYSTEM'_{A_i} = ((P'_{A_i} \parallel DEC_A) \setminus priv_A) \parallel INTRUDER(IK'_i)$$

because, by construction of P'_A and DEC_A, the latter process will always be in its initial state, which is DEC_A, when the former is in an external state.

In the same way, an external state $P^*_{A_i}$ of the abstract protocol logic is defined as a generic state of $f(P'_A)$ that is not ready to perform an event that is in $priv_A$. With this definition it turns out that *any* state of $f(P'_A)$ is an external state, since all actions in $priv_A$ have been removed by $f(\cdot)$. Then, any external state of the abstract process takes the form

$$SYSTEM^*_{A_i} = P^*_{A_i} \parallel INTRUDER(IK^*_i)$$

The relation R that binds external states of the abstract process to external states of the refined one is formally defined as

$$\begin{gathered} R(SYSTEM^*_{A_i}, SYSTEM'_{A_i}) \\ \Leftrightarrow \\ P^*_{A_i} = f(P'_{A_i}) \ \wedge \ IK^*_i \supseteq IK'_i \end{gathered}$$

Note that relation R holds on the initial states of the abstract and refined processes. Indeed, P'_A must be an external state, because, by construction, it cannot begin with an action in $priv_A$. Moreover, by hypothesis (17), $IK^*_0 \supseteq IK'_0$ holds.

Let us denote with P/ev ("P after ev") the state of process P after it has engaged the event ev; if P/ev is not applicable, that is, P cannot engage the event ev, then $P/ev = P$. With this definition, the operator / ("after") is distributive with respect to the operator $\parallel$, which synchronizes on the intersection of the alphabets, that is

$$(P \parallel Q)/ev = P/ev \parallel Q/ev$$

The after operator is then overloaded to sequences of events in the obvious way. In this work, $\langle ev_1, ev_2 \rangle$ denotes the sequence of events ev_1 and ev_2.

In order to show that the weak simulation relation holds, it is enough to show that, for any external states $SYSTEM'_{A_i}, SYSTEM^*_{A_i}$, and event sequence ev_s:

$$\begin{gathered} R(SYSTEM^*_{A_i}, SYSTEM'_{A_i}) \ \wedge \ SYSTEM'_{A_i} \overset{ev_s}{\to} SYSTEM'_{A_i}/ev_s \\ \Longrightarrow \\ SYSTEM^*_{A_i} \overset{ev_s^*}{\to} SYSTEM^*_{A_i}/ev_s^* \ \wedge \ R(SYSTEM^*_{A_i}/ev_s^*, SYSTEM'_{A_i}/ev_s,) \end{gathered} \tag{29}$$

where $\overset{ev_s}{\rightarrow}$ denotes the concatenation of state transitions for all events in ev_s, and ev_s^* is the sequence of events obtained by stripping all τ events from ev_s.

Note that, by the definition of intruder, $INTRUDER(IK_i^*)/ev_s^* = INTRUDER(IK_j^*)$, where IK_j^* is the new intruder knowledge reached after ev_s^*, and, by the distribution property of the after operator

$$\begin{aligned} SYSTEM_{A_i}^*/ev_s^* &= (f(P'_{A_i}) \parallel INTRUDER(IK_i^*))/ev_s^* \\ &= f(P'_{A_i})/ev_s^* \parallel INTRUDER(IK_i^*)/ev_s^* = f(P'_{A_i})/ev_s^* \parallel INTRUDER(IK_j^*) \end{aligned}$$

The same reasoning applies in the refined model.

Now all the possible event sequences ev_s that can lead to one of the next external states must be considered. Note that ev_s cannot start with a τ step, because it starts from an external state. Also note that it is enough to prove (29) for event sequences ev_s such that no proper subsequence of ev_s leads to an external state because then the most general case descends by induction. Then, if ev_s starts with a *claimSecret*, *running*, *finished*, *leak* or *send* event, it is possible to consider only the case when it is composed of just one event, because after the first event an external state is reached.

Let us first consider these cases, where $ev_s = \langle ev \rangle = ev_s^*$ and let us show that (29) holds.

case $ev = claimSecret.A.B.M$ By hypothesis this event can happen in the refined system. If we let $P'_{A_j} = P'_{A_i}/ev$, we have

$$((P'_{A_i} \parallel DEC_A) \setminus priv_A)/ev = (P'_{A_i}/ev \parallel DEC_A/ev) \setminus priv_A = (P'_{A_j} \parallel DEC_A) \setminus priv_A$$

because $ev \notin priv_A$ and DEC_A is only engaged in the events in $priv_A$.

Since ev can occur in P'_{A_i}, and $P'_{A_i} \overset{ev}{\rightarrow} P'_{A_j}$, by the properties of $f(\cdot)$, it can be concluded that

$$f(P'_{A_i}) \overset{ev}{\rightarrow} f(P'_{A_j})$$

that is, ev can also happen in the abstract system, and the states of the abstract and refined protocol logics after ev are bound by $f(\cdot)$.

Moreover, after event ev, intruder knowledges remain unchanged in both systems, so $IK_j^* \supseteq IK'_j$ holds, and it can be concluded that R holds after ev.

The same reasoning also applies for the *running* and *finished* events.

case $ev = leak.M$ Since this event is engaged by the intruder process, and not by the protocol logic, if ev can happen in the refined system, then M must be in the intruder knowledge, that is $M \in IK'_i$. Consequently, by the assumption $IK_i^* \supseteq IK'_i$, it follows that $M \in IK_i^*$ too, and ev can happen in the abstract system too. Moreover, the state of the protocol logic and the intruder knowledge remain unchanged after this event in both refined and abstract processes, so relation R still holds after it.

case $ev = send.A.B.M$ If this event can happen in the refined system, it can also happen in the abstract system, because, as with the *claimSecret* event,

$$P'_{A_i} \overset{ev}{\rightarrow} P'_{A_j}$$

implies

$$f(P'_{A_i}) \overset{ev}{\rightarrow} f(P'_{A_j})$$

While the reasoning about the protocol logic states is the same as explained in the $claimSecret.A.B.M$ case, the reasoning about intruder knowledges changes. After the event ev happens in the refined system, we have

$$IK'_j = IK'_i \cup \{M\}$$

and similarly, in the abstract system,

$$IK_j^* = IK_i^* \cup \{M\}$$

Since, by hypothesis, $IK_i^* \supseteq IK'_i$ holds, then $IK_j^* \supseteq IK'_j$ holds too.

The cases that remain to be considered are when ev_s starts with a $receive.B.A.M$ event. This event is followed by the τ steps coming from the sequence of the pairs of $priv\_send_A.(y,a) \rightarrow priv\_receive_A.N$ actions that have been added during refinement after the *receive* action. Note that an external state is reached only after all these τ steps have been completed and, after a *receive* event, the protocol logic is obliged by construction to execute the sequence of private actions before any further external action. Then, there is a single external state that can be reached by the protocol logic after a *receive* event. Note also that, being the protocol logic synchronized with the intruder, the latter can execute only internal actions, i.e. *leak* actions, while the protocol logic is executing the internal steps that follow a *receive* event. These *leak* events do not change the global state. In conclusion, the only event sequences that start with a *receive* event and that lead to an external state are those that, after the *receive* event, have a sequence of τ events, corresponding to all the private actions that follow the *receive* action in the protocol logic, with interleaved *leak* events generated by the intruder. All these sequences lead to a single external state. If the refined protocol logic cannot execute one of the private actions that follow a *receive* action, the protocol logic gets stuck and no external state is ever reached. So, the only case that must be considered in order to check (29) is when all the private actions are executed.

By using the distributive property of the after operator, again we analyze protocol logics first, and then intruder knowledges.

In the refined system we have

$$((P'_{A_i} \parallel DEC_A) \setminus priv_A)/ev_s = (P'_{A_j} \parallel DEC_A) \setminus priv_A$$

Note that the state of DEC_A does not change because after each pair of $priv\_send_A$ and $priv\_receive_A$ events, it returns to its initial state, and we are under the hypothesis that all private events happen.

Let

$$\begin{aligned} receive.B.A.T \rightarrow priv\_send_A.(y_1,a_1) \rightarrow priv\_receive_A.T_1 \rightarrow \dots \\ \rightarrow priv\_send_A.(y_n,a_n) \rightarrow priv\_receive_A.T_n \rightarrow P'_{A_j} \end{aligned}$$

be the sequence of action prefixes that are executed in P'_{A_i} when ev_s occurs.

By the definition of DEC_A it follows that the data exchanged between the protocol logic and DEC_A in each pair of events $priv\_send_A.(M,a), priv\_receive_A.N$ must satisfy equations $N = d_A(a,M)$ and $N \neq \textsc{Atom}\,\mathcal{E}$. Then, if all private events can occur in the refined system, the previous $receive.B.A.T$ action must have bound variables in T in such a way that $T_i = d_A(a_i,y_i)$ for all $1 \leq i \leq n$. But, by (16), this also implies that

$$e_A(a_i,T_i) = e_A(a_i,d_A(a_i,y_i)) = y_i$$

By the definition of $f(\cdot)$, we have that the abstract protocol logic in state $f(P'_{A_i})$ is ready to execute

$$receive.B.A.T^* \rightarrow f(P'_{A_j})$$

where $T^* = T\left[{}^{e_A(a_1,T_1)}/_{y_1}\right]\dots\left[{}^{e_A(a_n,T_n)}/_{y_n}\right]$, i.e. where T^* is T with the same binding of variables that occurs in the concrete system when all private events occur.

Then, if in the refined system an event $receive.B.A.M$ followed by all the subsequent private events can occur, the same event can occur in the abstract system too, and

$$f(P'_{A_i}) \stackrel{receive.B.A.M}{\rightarrow} f(P'_{A_j})$$

With respect to the intruder, since the event $receive.B.A.M$ can happen in the refined system, it follows that $M \in IK'_i$. Since $IK^*_i \supseteq IK'_i$, it follows that the event can happen in the abstract system too, because $M \in IK^*_i$. Moreover, since after a *receive* event intruder knowledges remain unchanged, it also follows that $IK^*_j \supseteq IK'_j$.

In order to complete the proof of the simulation relation, we have to show that *leak* events interleaved in ev_s can happen in the abstract system too. This descends from the fact that neither the *receive* event nor the τ events change the intruder knowledge. Then, the occurrence of a $leak.N$ event in ev_s implies that $N \in IK'_i$, which, in turn, by the hypothesis $IK'_i \subseteq IK^*_i$, implies $N \in IK^*_i$, which finally means that $leak.N$ can be executed by the intruder in the abstract system too.

The weak simulation relation that has been proven implies that any trace of the refined system $P_A^{dec-} \parallel INTRUDER(IK_0')$ that leads to an external state is also a trace of the abstract system $P_A^* \parallel INTRUDER(IK_0^*)$. However it is still possible that a trace that leads to an internal state in the refined system is not a trace of the abstract system. By the cases that have just been analyzed, it can be realized that a trace of the refined system can lead to an internal state only when the last action performed by the protocol logic is a *receive*. Moreover, in this trace, the last *receive* event can be followed only by *leak* events. The longest prefix of this trace that leads to an external state is the one that is obtained by removing the last *receive* event and the subsequent *leak* events, and the proof previously given ensures that this is also a trace of the abstract system. Moreover, since the receive event does not change the intruder knowledge, we can conclude that any *leak* event following the receive event in the refined system could also occur, both in the refined and in the abstract systems, before the *receive* event. Then, we have that even when a trace tr of the refined system is not a trace of the abstract system, the abstract system can still execute a trace tr^* that differs from tr only in the last *receive* event. This lets us conclude that lemma 4, which involves processes with hidden *send* and *receive* events, holds.

□

References

1. Dolev, D., Yao, A.C.C.: On the security of public key protocols. IEEE Transactions on Information Theory **29**(2) (1983) 198–207
2. Millen, J.K., Shmatikov, V.: Symbolic protocol analysis with an abelian group operator or diffie-hellman exponentiation. Journal of Computer Security **13**(3) (2005) 515–564
3. Abadi, M., Rogaway, P.: Reconciling two views of cryptography (the computational soundness of formal encryption). Journal of Cryptology **15**(2) (2002) 103–127
4. Abadi, M., Jürjens, J.: Formal eavesdropping and its computational interpretation. In: Theoretical Aspects of Computer Software. (2001) 82–94
5. Pozza, D., Sisto, R., Durante, L.: Spi2java: Automatic cryptographic protocol java code generation from spi calculus. In: International Conference on Advanced Information Networking and Applications. (2004) 400–405
6. Tobler, B., Hutchison, A.: Generating network security protocol implementations from formal specifications. In: Certification and Security in Inter-Organizational E-Services, Toulouse, France (2004)
7. Bhargavan, K., Fournet, C., Gordon, A.D., Tse, S.: Verified interoperable implementations of security protocols. In: Computer Security Foundations Workshop. (2006) 139–152
8. Jürjens, J.: Verification of low-level crypto-protocol implementations using automated theorem proving. In: Formal Methods and Models for Co-Design. (2005) 89–98
9. Goubault-Larrecq, J., Parrennes, F.: Cryptographic protocol analysis on real C code. In: Verification, Model Checking, and Abstract Interpretation. (2005) 363–379
10. Bhargavan, K., Fournet, C., Gordon, A.D.: Verified reference implementations of WS-security protocols. In: Web Services and Formal Methods. (2006) 88–106
11. Hui, M.L., Lowe, G.: Fault-preserving simplifying transformations for security protocols. Journal of Computer Security **9**(1/2) (2001) 3–46
12. Hoare, C.A.R.: Communicating Sequential Processes. Prentice Hall (1985)
13. Roscoe, A.W.: The Theory and Practice of Concurrency. Prentice Hall (1997)
14. Lowe, G.: A hierarchy of authentication specifications. In: Computer Security Foundations Workshop, IEEE Computer Society Press (1997) 31–43
15. Kleiner, E., Roscoe, A.W.: On the relationship between web services security and traditional protocols. Electronic Notes in Theoretical Computer Science **155** (2006) 583–603
16. Nadalin, A., Kaler, C., Hallam-Baker, P., Monzillo, R.: OASIS web services security: SOAP message security 1.1 (WS-security 2004) (2006)
17. Ylonen, T., Lonvick, C.: The Secure Shell (SSH) Transport Layer Protocol. RFC 4253 (Proposed Standard) (January 2006)
18. Ylonen, T., Lonvick, C.: The Secure Shell (SSH) Protocol Architecture. RFC 4251 (Proposed Standard) (January 2006)
19. Schellhorn, G.: ASM refinement and generalizations of forward simulation in data refinement: a comparison. Theoretical Computer Science **336**(2-3) (2005) 403–435