Security Protocol Specification and Verification with $\textit{AnBx}$

Michele Bugliesi · Stefano Calzavara · Sebastian Mödersheim · Paolo Modesti

Abstract Designing distributed protocols is complex and requires actions at very different levels: from the design of an interaction flow supporting the desired application-specific guarantees, to the selection of the most appropriate network-level protection mechanisms. To tame this complexity, we propose $\textit{AnBx}$, a formal protocol specification language based on the popular $\textit{Alice & Bob}$ notation. $\textit{AnBx}$ offers channels as the main abstraction for communication, providing different authenticity and/or confidentiality guarantees for message transmission. $\textit{AnBx}$ extends existing proposals in the literature with a novel notion of forwarding channels, enforcing specific security guarantees from the message originator to the final recipient along a number of intermediate forwarding agents.

We give a formal semantics of $\textit{AnBx}$ in terms of a state transition system expressed in the AVISPA Intermediate Format. We devise an ideal channel model and a possible cryptographic implementation, and we show that, under mild restrictions, the two representations coincide, thus making $\textit{AnBx}$ amenable to automated verification with different tools. We demonstrate the benefits of the declarative specification style distinctive of $\textit{AnBx}$ by revisiting the design of two existing e-payment protocols, $\textit{iKP}$ and $\textit{SET}$.

Keywords Protocol specification, protocol verification, model-checking, e-payment

1 Introduction

The $\textit{Alice & Bob}$ notation, also known as protocol narrations, is a popular device which has been widely adopted in the literature as the basis of several security protocol specification frameworks [1, 2, 3, 4, 5]. In such frameworks, the semantics of the specification languages is defined by a translation into lower level formats, amenable to model-checking and automated verification. Besides making verification possible, the translation semantics provides for a clean separation between the abstract specification of the protocol structure and the details of its implementation, which may be generated directly from the specification [6, 7, 8, 9, 10]. This separation has a beneficial impact on both the specification and the implementation: on the one hand, it helps focusing on application-level properties, staying away from unnecessary low-level details; on the other hand, it contributes to strengthen the implementation and to ensure the protocol end-to-end security, by delegating to the compiler the selection of the most adequate core implementation components.

Channel abstractions make a further step in the same direction: they help designing distributed applications irrespective of the crypto-
graphic mechanisms needed to protect communication, by interpreting channels as a secure communication medium with built-in protection against certain attacks (e.g., on confidentiality). How these properties are actually ensured represents a different design aspect, which might not be a concern of the application designer at all, and may be left to the compiler.

**Related work.** Several papers in the literature have taken this approach, and developed it along different directions. First, there are papers that propose the definition and implementation of different channel types, based on cryptographic realizations and interaction patterns. Abadi et al. propose a process calculus with native constructs for authentication and discuss a possible cryptographic implementation [11]. Adão and Fournet design a variant of the pi-calculus with secure communication and describe its computationally sound compilation into a concrete implementation [12]. Other authors explore the idea of compiling secure protocols for distributed sessions from convenient ML abstractions based on session types [13,14].

Another line of research, instead, is more focused on reasoning about channels and their ideal behaviour in an abstract way. Dillaway and Lowe present a hierarchy of secure channels and discuss their relative strengths [15]. Bugliesi and Focardi devise secure channel abstractions in a process algebraic setting and reason about the relative power of a low-level adversary [16]. Armando et al. model different channel types using set-rewriting and linear temporal logic [17]. Kamil and Lowe adapt the Strand Spaces model to deal with secure channels, providing different security guarantees [18]. Mödersheim and Viganò consider both an abstract characterization and a concrete realization of channels, showing that both characterizations coincide; the paper defines also the notion of channels as goals and proves a related compositionality result [19]. Finally, Sprenger and Basin consider a refinement approach where cryptographic protocols are synthesised from high-level security goals; one of the steps of the refinement process builds on the usage of channel abstractions [20].

**Contributions.** In the present paper we develop channels one step further, generalizing them to capture the notion of forwarding channel, an abstraction that is not accounted for in existing protocol narration frameworks and is instead critical for designing and reasoning about complex protocols involving three or more parties. A typical scenario for such protocols is represented by e-commerce transactions, in which a customer requires a merchant to certify that her payment has been cleared out, and the merchant provides that evidence by forwarding to the customer the notification she received from the credit card issuer. Similarly, single sign-on protocols usually involve an authenticity-preserving forwarding of access tokens from a trusted third-party to different clients.

We develop the novel concept of forwarding channel as part of AnBz, a formal specification language that we introduce by conservatively extending the semantics of the AnB language [4]. AnBz includes modes for all kinds of message forwarding, where all or some of the properties of the original transmission are preserved upon relaying. In our characterization, we provide both an abstract interpretation of channels that captures their ideal behavior, and a cryptographic implementation, and we prove a formal correspondence between the two characterizations. Both interpretations are based on a translation to the AVISPA Intermediate Format, hence AnBz is directly available for automated verification with the different tools that adopt such format, such as OFMC [21].

We demonstrate the practical effectiveness of our approach by an analysis and re-engineering of two real-life e-payment protocols: iKP (Internet Keyed Payment [22,23]), and SET (Secure Electronic Transaction [24,25,26]). Though both protocols could be expressed in their full complexity in AnBz, we rely on the abstract channels available in the language to factor out the cryptographic aspects almost entirely. The resulting protocols are more concise, easier to understand and, interestingly, more efficient to verify than the original versions. In addition, the AnBz formulations outperform the original specifications, in that they enjoy stronger security goals and properties. As a byproduct of our comparative analysis, we also find a (to the best of our knowledge) new flaw in the original specification of iKP, and propose an amended version that rectifies the problem.

**Plan of the paper.** Section 2 introduces the basics of AnBz. Section 3 focuses on the semantics of the language and presents our formal results. Sections 4-6 discuss our case studies. Section
Protocol: Diffie-Hellman

Types:
- Agent A, B;
- Number g, X, Y, Msg;

Knowledge:
- A: A, B, g;
- B: A, B, g;

Actions:
- A → B, (A|B|−) : exp(g, X)
- B → A, (B|A|−) : exp(g, Y)
- A → B, (−|−|−) : |{[A, Msg]|exp(exp(g, Y), X)}

Goals:
- B authenticates A on Msg
- Msg secret between A, B

Figure 1 Diffie-Hellman specification in AnBx

7 concludes the presentation. The AnBx implementation, together with its analytical tool and the scripts employed in the case studies, is available online.

New contents. This paper integrates and extends the results reported in [27] and [4]. Section 3 is novel: in previous work the semantics of AnBx was defined by a direct translation to AnB, based on a cryptographic implementation. Here we recast our cryptographic implementation within the AVISPA Intermediate Format IF, and provide an alternative IF characterization, based on the ideal channel behaviour. We then prove that the cryptographic implementation conforms with the ideal semantics. Besides representing a valuable theoretical contribution, the semantics correspondence has practical value, as it makes both characterizations equally viable for automatic analysis within any verification framework supporting IF. The SET case study in Section 6 is new.

2 AnBx Protocol Specifications

AnBx is a formal protocol specification language based on the popular (informal) Alice & Bob notation. AnBx conservatively extends the AnB specification language [4] with a richer notion of communication channel.

2.1 Protocol Types and Agent Knowledge

Protocol narrations in AnBx are built around an underlying signature of typed identifiers that include protocol variables, constants, and function symbols. Variables are noted with uppercase initials and represent values that are determined dynamically, at each protocol run. Constants, in turn, are noted by lower-case identifiers and represent values and functions that are invariant across different protocol executions. As an example, consider the AnBx specification of the Diffie-Hellman key exchange protocol in Figure 1. Variables of type agent are roles: here we have the roles A and B, which get instantiated to arbitrary concrete agents when executing the protocol. The numbers g, X and Y, in turn, are the (constant) group generator and the (variable) random exponents of the Diffie-Hellman key exchange.

For each role, the protocol specification describes the knowledge that an agent playing that role needs to execute the protocol: this indirectly specifies what the intruder will know when playing one of the roles of the protocol. Only variables of type agent may be part of the initial knowledge. All other variables represent values that are chosen randomly by the participant who first uses them, e.g., in the example A chooses X and B chooses Y.

2.2 Protocol Actions

The core of an AnBx specification consists of the message exchanges between the participants in an ideal, unattacked run of the protocol. Every action has either of the two forms below:

A → B, η : M or A ⊸ B, η : M,

noting standard and fresh exchanges, respectively. In both cases, an agent playing role A communicates message M to the agent playing role B, along a communication channel that conveys the security guarantees specified by the exchange mode η. The AnBx modes are triples:

(Auth|Verifiers|Conf),

whose components may be set to an agent name (a list of names for the Verifiers field), or unset, in which case they are filled with the distinguished symbol “-”. When the Conf field is set, the action represents a confidential exchange, which guarantees that only the agent named in the field has access to the message. When the Auth field is set, the action identifies an authentic exchange, which guarantees that
the message originates from the agent named in the field; the Verifiers field must be set if and only if the Auth field is set, to include a non-empty list of agents that are entitled to verify the authenticity of the message. Authenti-
cic exchanges may further specify that the mes-
gage being exchanged is freshly communicated by the agent referenced in the Auth field: the notation \( A \rightarrow B, \pi : M \) serves that purpose. None of the modes conveys any guarantee that the intended recipients will eventually receive the message.

Though the intended purpose of the channel modes is to hide low-level communication details, we remark that AnBx conservatively extends the AnB notation, making it possible to freely intermix abstract exchanges and cryptographic terms. Note, in particular, that the first two actions in the Diffie-Hellman specification in Figure 1 employ the channel modes to express the authentic exchange of the two “half keys”, while the third describes the exchange of message \( Msg \) encrypted under the new key.

The idea to structure protocol specifications around abstract mechanisms for secure commu-
nications is certainly not new, as we discussed in Section 1. Among the various approaches in the literature, the closest to ours is the “bullet” notation supported by AnB• [19], a specification language providing support for confidential and authenticated channels. Every exchange mode available in AnB• can be easily encoded in AnBx, as shown in Table 1 below; however, AnBx provides additional expressiveness, as we discuss in the next section.

<table>
<thead>
<tr>
<th>AnB•</th>
<th>AnBx</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plain</td>
<td>( A \rightarrow B )</td>
</tr>
<tr>
<td>AUTHENTIC</td>
<td>( A \leftrightarrow B )</td>
</tr>
<tr>
<td>CONFIDENTIAL</td>
<td>( A \leftrightarrow B )</td>
</tr>
<tr>
<td>SECURE</td>
<td>( A \leftrightarrow B )</td>
</tr>
</tbody>
</table>

Table 1 Encoding of AnB• in AnBx

2.3 Forwarding Modes

In addition to the standard AnB• exchanges, the AnBx modes allow additional generality. Specifically, AnBx provides primitive support for message forwarding, a feature which is not made available by existing proposals, and that instead constitutes a recurrent communication pattern in practical applications. We will provide examples of concrete uses of forwarding in our case studies; for the moment, we just illustrate the concept with some simple examples.

The first example shows how authenticity can be preserved upon forwarding:

\[
A \rightarrow B, (A | B, C | -) : M \\
B \rightarrow C, (A | B, C | -) : M
\]

The first action denotes an authentic exchange that originates from \( A \) and is meant to be delivered to both \( B \) and \( C \). Upon receiving \( M \), agent \( B \) forwards it to \( C \) in the second action, preserving the authenticity guarantees by \( A \). Notice that the mode \((A | B, C | -)\) in the second exchange still mentions \( A \) as the source of the communication, even though the message is sent by \( B \). This pattern cannot be encoded in the AnB• notation, since authentic messages are always assumed to be originated by the agent specified on the tail of the arrow.

Forwarding modes can be used also to implement a form of “blind” delivery, arising when an agent relays messages that are intended to remain confidential for a third party:

\[
A \rightarrow B, (- | - | C) : M \\
B \rightarrow C, (- | - | C) : M
\]

Here, \( A \) sends \( M \) to \( C \) confidentially, relying on \( B \) to deliver the message. As in the previous case, this protocol cannot be expressed in the AnB• notation, in this case because secret messages are always intended to be disclosed to the agent specified on the head of the arrow.

Message forwarding is also available for fresh exchanges, in various combinations. Assume message \( M \) is sent freshly from \( A \) to \( B \):

\[
A \overset{\alpha}{\rightarrow} B, (A | B, C | -) : M
\]

Then both the following actions:

\[
B \rightarrow C, (A | B, C | -) : M
\]

and:

\[
B \overset{\alpha}{\rightarrow} C, (A | B, C | -) : M
\]

are legal. With the first action, \( M \) is forwarded to \( C \) without any freshness guarantee, whereas the second action allows \( C \) to verify the freshness of the transmission.
2.4 Protocol Goals

AnBx protocol specifications are analyzed and validated against a set of security goals, that specify the properties expected of the protocol. Like its predecessors, AnBx supports three standard kinds of goals, which we briefly review below, referring the reader to [4] for full details.

- **Weak Authentication** goals have the form:
  \[ B \text{ weakly authenticates } A \text{ on } M, \]
  and are defined in terms of non-injective agreement on the runs of the protocol [28];

- **Authentication** goals have the form:
  \[ B \text{ authenticates } A \text{ on } M, \]
  and are defined in terms of injective agreement on the runs of the protocol, assessing the freshness of the exchange;

- **Secrecy** goals have the form:
  \[ M \text{ secret between } A_1, \ldots, A_k, \]
  and are intended to specify which agents are entitled to learn message \( M \) at the end of the protocol run.

3 AnBx Semantics

Following previous proposals [4, 19], we define the semantics of AnBx in terms of a translation to the AVISPA Intermediate Format IF [29]. IF is a set-rewriting calculus in which the semantics of a protocol is described in terms of a set of facts that encode the knowledge of the honest agents and the intruder at the different protocol steps, and a set of rewriting rules describing the state transitions of the participants and the intruder during the protocol execution. The rewriting rules for honest participants are generated from the AnBx protocol specification, while the capabilities available to the intruder are modelled by protocol-independent rules, i.e., the intruder is not forced to follow the protocol specification.

We define the translation from AnBx to IF in several steps, conveniently exploiting the existing AnB2IF compiler [4] as a black box. Given an AnBx specification, we translate it into a corresponding AnB specification, in which the AnBx modes are expressed as message tags (Section 3.1). The resulting AnB specification is fed to the AnB2IF compiler, which extracts from the narration the actions associated with the protocol agents, and renders them as IF rewriting rules (Section 3.2). The resulting IF rules still include the tags from the annotated AnB narration: a further transformation step (Sections 3.3 and 3.4) completes the translation, exploiting the tags to produce a cryptographic IF specification and an ideal IF specification. We refer to these two constructions as the Cryptographic Channel Model (CCM) and the Ideal Channel Model (ICM) respectively. The two models are contrasted in Section 3.5.

3.1 From AnBx to AnB

The first step of the translation transforms each action in the AnBx narration into a corresponding AnB action bearing additional annotations, which drive the later stages of the translation.

The AnBx-to-AnB translation is conceptually simple, though the presence of the fresh modes and their interaction with the forward modes hide a few subtleties. Our characterization of freshness relies on a simple mechanism by which the sender generates a fresh nonce and the recipient caches every nonce it receives, telling fresh messages from replicas by checking whether the received nonce is in the cache. In case of forwarding of a fresh message we reuse the same nonce generated at the step which introduced the message being forwarded.

In order to ensure that newly generated nonces are indeed fresh, the AnBx-to-AnB translation keeps track in a store \( \xi \) of all the protocol variables introduced to represent the different nonces created along the protocol steps. Note that the store \( \xi \) does not have any counterpart in the translated protocols, and it is just an artifact used in the translation to AnB, as described below:

- At each fresh exchange which is not a forward, we first select a nonce identifier \( N \) that does not occur in \( \xi \) and then store in \( \xi \) the 4-tuple \((A, \tilde{V}, M, N)\) which includes the name \( A \) of the source agent, a (non-empty) list \( \tilde{V} \) of intended verifiers, the message \( M \) exchanged in the AnBx specification, and the identifier \( N \); the additional information enables the reuse of the nonce \( N \) in all the possible future forwards of message \( M \).
- At each authentic forward action, we lookup the store in search of a tuple whose first
three components match the Auth and the Verifiers components of the mode, as well as the message being forwarded; if such a tuple exists, the action is a forward of a fresh exchange, and we include the nonce generated at that exchange among the components of the forwarded message, irrespective of whether the forward is fresh or not (this choice is technically convenient in the definition of the translation). If the tuple does not exist, then the source action must be non-fresh, thus no nonce is included in the forward of the generated message.

- When more than one entry in $\xi$ matches the required side-conditions, the most recent match is used.
- The translation is undefined when a fresh forward is performed, but no matching tuple was found in the store.

In a practical implementation, one would of course either use timestamps or sequence numbers, in order to limit the amount of data that the receiver has to store. We remark, however, that these realizations are isomorphic to our formal model$^2$.

A further subtlety in the translation arises from blind forwards, i.e., when the recipient $A$ of a message $M$ differs from the final intended receiver $B$, and the message $M$ should not be exposed to $A$. To capture the desired effect, we wrap $M$ inside the constructor $\text{blind}_B$, to denote that it should be readable only by $B$.

The translation clauses are listed in Table 2, where we do not explicitly track the updating of $\xi$ for the sake of readability. The tags $\text{plain}$, $\text{ctag}$, $\text{atag}$ and $\text{fstag}$ are just as in [19]; in addition, we include the new tags $\text{fata}g$ and $\text{fsta}g$ to account for freshly authenticated channels. All the tags are public constants in the target AnB specification, while $\text{blind}_X$ is a public function symbol for any $X$, so all this information is available to every agent (including the intruder). In addition, the specification is extended with private function symbols $\text{unblind}_X$, parameterized over the agents identity, which are used to extract the confidential messages$^3$.

As a first, simple illustration, below we give the annotated AnB narration that results from applying the translation to the AnBx specification of the protocol in Figure 1:

$$A \rightarrow B : \text{atag}, A, B, \exp(g, X)$$
$$B \rightarrow A : \text{atag}, B, A, \exp(g, Y)$$
$$A \rightarrow B : \text{plain}, ([\exp(g)] \exp(\exp(g, Y), X))$$

As a further example, consider the following variant of the blind forward protocol examined earlier on:

$$A \rightarrow B : \text{ctag}, \text{blind}_C(M)$$
$$B \rightarrow C : \text{ctag}, \text{blind}_C(M)$$

where we assume that the three agents use token as a known tag marking their exchanges. The resulting AnB narration is as follows:

$$A \rightarrow B : \text{ctag}, \text{blind}_C(M)$$
$$B \rightarrow C : \text{ctag}, \text{blind}_C(M)$$

Error conditions. If none of the clauses in Table 2 applies, the translation is undefined and an error is reported. Errors signal unexecutable specifications, which expect the protocol participants to send messages they are unable to compose, since they lack some of the required information bits. One such error condition arises when an agent is expected to execute a fresh forward action for a message it received without any freshness guarantee, as in the following specification:

$$A \rightarrow B : (A | B, C | -) : M$$
$$B \xrightarrow{\text{msg}} C, (A | B, C | -) : M$$

Further cases of unexecutable specifications are identified by a subsequent translation step, specifically during the AnB-to-IF translation. Indeed, the $\text{blind}_X(M)$ construction for confidential messages has precisely the purpose to signal to the AnB2IF compiler that message $M$ can only be seen by $X$, so that a protocol turns out to be unexecutable if such a blinded message needs to be read by another agent. Consequently, a sequence of AnBx actions like the one below is translated successfully to AnB, but the AnB2IF compiler will reject it as non-executable, since after the first exchange $B$ has access to $\text{blind}_C(M)$ and not to $M$:

$$A \rightarrow B : (\text{A} | \text{B}, \text{C} | \text{X}) : M$$
$$B \rightarrow C : (\text{X} | \text{C} | \text{Y}) : M$$

the current implementation of AnB2IF does not support user-defined algebraic theories. Namely, we let $\text{bind}_X(M) \equiv ([M]_{b(X)})$, where $b(\cdot)$ is a public function symbol and $\text{inv}(b(X))$ is known only to $X$. 

---

$^2$ A further possible alternative is to use challenge-response protocols, but these generate additional network traffic, which in turn would considerably complicate our exposition of the two channel models and their relationship, as well as the practical model-checking problems induced in our tool.

$^3$ In our implementation we actually rely on the OFMC facility for asymmetric cryptography, since
The AVISPA Intermediate Format IF [29] is a low-level language for specifying transition systems using set rewriting. We refer the reader to [4] for full details on the translation from AnB to IF; here, we just provide an informal overview to make the paper self-contained.

An IF specification \( I = (I, R, G) \) consists of an initial state \( I \), a set of transition rules \( R \) for the protocol participants and the intruder, and a set of goals \( G \) that determine which states count as attack states. A protocol is safe when no attack state is reachable from the initial state using the transition rules.

An IF state is a set of ground facts, separated by dots ("."), which encode the knowledge of the different protocol agents. We distinguish two kinds of facts: \( \text{iknows}(m) \), which denotes that the intruder knows the term \( m \), and \( \text{state}_{\sigma}(A, m_1, \ldots, m_n) \), which characterizes the local state of an honest agent during the protocol execution by the terms \( A, m_1, \ldots, m_n \). The constant \( \sigma \) identifies the role of the agent, and, by convention, the first message \( A \) is the name of that agent\(^4\). Our formalization of the intruder also includes a further class of facts of the form \( \text{dishonest}(A) \) to identify the dishonest agents participating in the protocols. While many tools assume that there is only a single dishonest agent \( i \) (the "intruder"), our model supports any number of collaborating dishonest agents – one may still think of one intruder who has compromised several agents and can now use their identities.

\[ [A \rightarrow B, (-|-): M]_I = A \rightarrow B : \text{plain}, M \]
\[ [A \rightarrow B, (-|-B): M]_I = A \rightarrow B : \text{ctag}, \text{blind}_B(M) \]
\[ [A \rightarrow B, (\bar{A}V|-): M]_I = A \rightarrow B : \text{atag}, \bar{A}V, M, N \]
\[ = A \rightarrow B : \text{atag}, \bar{A}V, M \]
\[ \text{if } \bar{A} \neq A \text{ and } (\bar{A}, \bar{V}, M, N) \in \xi \]
\[ \text{otherwise} \]
\[ [A \rightarrow B, (\bar{A}V|\bar{B}): M]_I = A \rightarrow B : \text{stg}, \text{blind}_B(\bar{A}V, M, N) \]
\[ = A \rightarrow B : \text{stg}, \text{blind}_B(\bar{A}V, M) \]
\[ \text{if } \bar{A} \neq A \text{ and } (\bar{A}, \bar{V}, M, N) \in \xi \]
\[ \text{otherwise} \]
\[ [A \rightarrow B, (\bar{A}V|\bar{B}): M]_I = A \rightarrow B : \text{fsg}, \text{blind}_B(\bar{A}V, M, N) \]
\[ \text{if } \bar{A} = A \text{ (with N chosen fresh in } \xi) \]
\[ \text{or } \bar{A} \neq A \text{ and } (\bar{A}, \bar{V}, M, N) \in \xi \]

| Table 2 | Translation from AnB to AnB |

3.2 From AnB to IF

We now discuss how the initial state is generated from an AnB specification. Let \( n \) denote a bounded number of protocol sessions and let \( \sigma_1, \ldots, \sigma_n \) be corresponding mappings from the protocol roles \( R_1, \ldots, R_m \) to concrete agent names. Let \( K_j \) stand for the initial knowledge of the role \( R_j \), then the initial state is:

\[ \bigcup_{1 \leq i < k \leq n, 1 \leq j \leq m} \left\{ \{ \text{state}_{\sigma_j}(K_j, \sigma_i) \} \right\} \quad \text{if } R_j \sigma_i \neq i \]
\[ \left\{ \{ \text{iknows}(K_j, \sigma_i), \text{dishonest}(i) \} \right\} \quad \text{if } R_j \sigma_i = i \]

where \( i \) is a reserved constant denoting the identity of the intruder. The initial state thus consists of the local states of the honest agents and the initial knowledge of the intruder, which is determined by the compromised agents; a \( \text{dishonest}(i) \) fact is introduced when at least one of the agents is compromised.

The transitions of an IF specification are of the form \( L \mid Cond = \{ \mathcal{X} \} \Rightarrow R \) where \( L \) and \( R \) are states, \( \mathcal{X} \) is a set of fresh variables (representing fresh values generated at run-time), and \( Cond \) is a set of conditions, expressed as (in)equalities and negated predicates. The semantics of an IF rule is defined by the state transitions it enables: from a state \( S \) the rule enables a transition to a state \( S' \) iff there exists a substitution \( \sigma \) of the variables of \( L \) and \( \mathcal{X} \) such that \( L \sigma \subseteq S, S' = (S \setminus L \sigma) \cup R \sigma \), and \( \mathcal{X} \sigma \) are fresh constants not occurring in \( S \); moreover, the conditions \( Cond \sigma \tau \) are true in \( S \) for every substitution \( \tau \) of the variables in \( Cond \) that do not occur in \( L \). We assume the \( \text{iknows}(\cdot) \) and the \( \text{dishonest}(\cdot) \) facts to be persistent, i.e., to be always propagated to the right-hand side of any transition.

The semantics of AnB is just defined by the translation from an AnB specification to IF. The main point of the translation is to define...
the behavior of the honest agents in terms of IF transition rules, by identifying in particular what checks must be performed on the messages they receive, and how they construct the messages they send out. The behavior of the intruder, in contrast, is defined by protocol-independent rules modelling a Dolev-Yao attacker, such as:

\[ \text{iknows}(M) \cdot \text{iknows}(K) \Rightarrow \text{iknows}(\{M\}_K) \]
\[ \text{iknows}(\{M\}_K) \cdot \text{iknows} (\text{inv}(K)) \Rightarrow \text{iknows}(M) \]

The first rule describes both asymmetric encryption and signing, while the second one expresses that the payload of a ciphertext can be retrieved if the corresponding decryption key is known. We use inv(\cdot) as a private function symbol, employed, e.g., to represent the secret component of a given key-pair. All the messages exchanged by honest agents are always assumed to be mediated by the intruder, i.e., every communication happens through \text{iknows}(\cdot) facts.

We illustrate the translation from AnB to IF with an example. Specifically, we give the IF transitions for roles A and B from the AnB translation of the protocol in Figure 1. The IF transition rules are in Figure 2 below, where for the sake of readability we do not explicitly represent the public tags in the state facts and we turn the side-conditions of the transitions into pattern matching:

\[
\text{state}_A(A, B, C, g) = [X] \Rightarrow \\
\text{state}_A(A, B, g, X).\text{iknows}(\text{atag}, A, B, \text{exp}(g, X)) \\
\text{state}_A(A, B, g, X).\text{iknows}(\text{atag}, A, B, GY) = [\text{msg}] \Rightarrow \\
\text{state}_A(A, B, g, X).\text{iknows}(\text{plain}, \{A, \text{msg}\}_{\text{exp}(GY, X)}) \\
\text{state}_{iB}(B, A, g).\text{iknows}(\text{atag}, A, B, GX) = [Y] \Rightarrow \\
\text{state}_{iB}(B, A, g, GX, Y).\text{iknows}(\text{atag}, A, B, \text{exp}(g, Y)) \\
\text{state}_{iB}(B, A, g, GX, Y).\text{iknows}(\text{plain}, \{A, \text{msg}\}_{\text{exp}(GX, Y)}) \Rightarrow \text{state}_{iB}(B, A, g, GX, Y, \text{msg})
\]

**Figure 2** IF translation of the example of Figure 1

Notice in the second clause that A accepts any value \(GY\) from the network, not necessarily the result of a correct Diffie-Hellman exponentiation, and applies it to encrypt the last message of the protocol. Conversely, in the fourth clause, B checks that the first encrypted message component is indeed the identity of A, but it cannot check anything about \(\text{msg}\), since it is freshly generated by another participant.

### 3.3 From IF to CCM

The Cryptographic Channel Model realizes the AnBz channel modes by means of digital signatures and public-key encryptions, represented in a simple symbolic model of cryptography.

**Honest agents.** The translation of the honest agents is based on the IF-to-CCM mapping defined in Table 3. For rules generated by the AnB2IF compiler, the corresponding CCM rule results from applying the mapping in the table. In the CCM code, we additionally associate two key-pairs \((\text{pk}(A), \text{inv}(\text{pk}(A)))\) for encryption/decryption, and \((\text{sk}(A), \text{inv}(\text{sk}(A)))\) for verification/signature with every agent \(A\) acting as the source of an authentic message or as the target of a confidential exchange.

The message \(\text{msg}\) occurring in all clauses in Table 3 may be an arbitrary tuple. The last clause is the exception, as it only applies to variables: this clause handles the case of agents that are expected to execute blind forward actions for confidential or (fresh) secure messages. In the AnB2IF translation, such agents receive the messages to be forwarded as terms of the form \((t, X)\) for some variable \(X\), as they are going to accept any message at such steps, without inspecting it: therefore, to obtain the corresponding CCM code, we just remove the tag.

To illustrate, consider again the annotated AnB blind-forward example we examined in Section 3.1:

\[
A \rightarrow B: \text{ctag}, \text{blind}_C(\text{msg}, \text{token}) \\
B \rightarrow C: \text{ctag}, \text{blind}_C(\text{msg}, \text{token})
\]

Though \(\text{token}\) is assumed to be known to all agents, the forward action by \(B\) is performed irrespectively of the actual content of the message it receives, since \(B\) is not able to perform any check on a confidential message for \(C\). This is shown by the IF code produced by the translation of the exchange to the CCM:

\[
\text{state}_A(A, B, C, \text{token}) = [\text{msg}] \Rightarrow \\
\text{state}_A(A, B, C, \text{token}, \text{msg}).\text{iknows}(\{\text{msg}, \text{token}\}_{\text{pk}(C)}) \\
\text{state}_B(B, C, A, \text{token}).\text{iknows}(X) \Rightarrow \text{state}_B(B, C, A, \text{token}, X) \\
\text{state}_C(C, A, B, \text{token}).\text{iknows}(\{\text{msg}, \text{token}\}_{\text{pk}(C)}) \Rightarrow \text{state}_C(C, A, B, \text{token}, \text{msg})
\]

In the second transition rule, \(B\) accepts every variable \(X\) provided by the intruder. (Recall
that the \( \text{iknows}(X) \) fact is not reported explicitly on the right-hand side of the arrow, since such facts are persistent.) Conversely, in the third rule \( C \) can verify that the second component of the encryption is indeed the expected \( \text{token} \) available in her knowledge.

An additional measure is needed for translating to the \( \text{CCM} \) the transitions expecting a fresh message on input. These transitions are easily identified in the annotated \( \text{AnB} \) code, as they have an occurrence of \( \text{fstag/fstag} \) in their incoming message. For any such transition, let \( B \) be the receiver, and \( N \) the nonce associated with the fresh message. Now, to implement the nonce-checking mechanism of replay protection we discussed in Section 3.1, it is enough (\( i \)) to predicate the transition to the side condition \( \text{not(seen}(B,N)) \), and (\( ii \)) to introduce the fact \( \text{seen}(B,N) \) to the right-hand side of the same transition. For instance, for the sender of the message \( A \xrightarrow{\text{cnfCh}} B, (A|B|-) : \text{Msg} \), the \( \text{CCM} \) will comprise a transition of the form:

\[
\ldots \Rightarrow \text{iknows}((B,\text{Msg},N)_{\text{inv}(\text{pk}(A))}) \ldots
\]

with \( N \) fresh. Correspondingly, on the receiver side, the transition in the \( \text{CCM} \) will be structured as follows:

\[
\ldots \text{iknows}((B,\text{Msg},N)_{\text{inv}(\text{pk}(A))}) \mid \text{not(seen}(B,N)) \Rightarrow \text{seen}(B,N) \ldots
\]

As a result, message \( M \) is received only if the nonce \( N \) was never seen before by the receiver: if that is the case, and the message is accepted, the receiver adds \( N \) to its cache of seen nonces.

### Intruder rules

The symbols \( \text{sk}(-) \) and \( \text{pk}(-) \) introduced earlier on are public functions. Consequently, every agent, including the intruder, can obtain the public keys of every other agent as soon as their name is known: that gives the intruder the full power of the Dolev-Yao model.

The function \( \text{inv}(-) \), providing the ability to construct signing and decryption keys, is private, and each agent \( A \) knows only her own private keys \( \text{inv}(\text{sk}(A)) \) and \( \text{inv}(\text{pk}(A)) \). Notice that private keys of dishonest agents are available to the intruder, according to the definition of the \( \text{IF} \) initial state in Section 3.2.

### 3.4 From \( \text{IF} \) to \( \text{ICM} \)

The Ideal Channel Model provides for a direct representation of the communication modes in terms of corresponding \( \text{IF} \) state facts that encode the types of channel involved in the exchanges. In particular, the ideal semantics draws on the constructors \( \text{athCh} \), \( \text{cnfCh} \) and \( \text{secCh} \), around which we define persistent state facts that track the protocol exchanges. Protocol-independent rewriting rules, in turn, characterize the intended behaviour of the ideal channels.

#### Honest agents.

The translation of the honest agents is based on the \( \text{IF} \)-to-\( \text{ICM} \) mapping defined in Table 4. For each rule generated by the \( \text{AnB2IF} \) compiler, the corresponding \( \text{ICM} \) rule results from applying the mapping in the table. Similarly to the \( \text{CCM} \) translation, the last case in the table handles a blindly forwarding agent who cannot check anything about the message being forwarded.

For our blind forwarding example, the translation to the \( \text{ICM} \) generates the following \( \text{IF} \) transition rules:

\[
\text{state}_{\text{IF}}(A,B,C,\text{token}) \Rightarrow \text{state}(A,B,C,\text{token}) = [\text{Msg}]
\]

\[
\text{state}_{\text{IF}}(A,B,C,\text{token},\text{Msg}),\text{cnfCh}(C;\text{Msg},\text{token})
\]

\[
\text{state}_{\text{IF}}(B,C,A),\text{iknows}(X)
\]

\[
\Rightarrow \text{state}(B,C,A,\text{token},X)
\]

\[
\text{state}_{\text{IF}}(C,A,B),\text{cnfCh}(C;\text{Msg},\text{token})
\]

\[
\Rightarrow \text{state}(C,A,B,\text{token},\text{Msg})
\]

<table>
<thead>
<tr>
<th>IF</th>
<th>CCM</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{iknows}(\text{plain},M) )</td>
<td>( \text{iknows}(M) )</td>
</tr>
<tr>
<td>( \text{iknows}(\text{ctag},\text{blind}_{B}(M)) )</td>
<td>( \text{iknows}({M}_{\text{pk}(B)}) )</td>
</tr>
<tr>
<td>( \text{iknows}(\text{atag},A,\dot{V},M) )</td>
<td>( \text{iknows}({\dot{V},M}_{\text{inv}(\text{sk}(A))}) )</td>
</tr>
<tr>
<td>( \text{iknows}(\text{atag},\text{blind}_{B}(A,\dot{V},M)) )</td>
<td>( \text{iknows}({{\dot{V},M}<em>{\text{inv}(\text{sk}(A))}}</em>{\text{pk}(B)}) )</td>
</tr>
<tr>
<td>( \text{iknows}(\text{fstag},A,\dot{V},M,N) )</td>
<td>( \text{iknows}({{\dot{V},M,N}<em>{\text{inv}(\text{sk}(A))}}</em>{\text{pk}(B)}) )</td>
</tr>
<tr>
<td>( \text{iknows}(\text{fstag},\text{blind}_{B}(A,\dot{V},M,N)) )</td>
<td>( \text{iknows}(X) )</td>
</tr>
<tr>
<td>( \text{iknows}(\text{r},X) ) ( t \in {\text{ctag, stag, fstag}} )</td>
<td>( \text{iknows}(X) )</td>
</tr>
</tbody>
</table>

Table 3 Translation from \( \text{IF} \) to \( \text{CCM} \)
Intruder rules. The intruder rules constitute the key component of the ideal semantics, as it is through these rules that we define the actual interpretation of our channel facts. The rules are reported in Table 5 below.

\[
\begin{align*}
&\text{ICM:} \\
&\text{ICM:} \\
&\text{ICM:} \\
&\text{ICM:} \\
&\text{ICM:} \\
&\text{ICM:} \\
\end{align*}
\]

An intruder can forge a message over an authentic channel only if the associated sender identity is compromised, while he can learn every message sent over an authentic channel. Dually, an intruder can send over a confidential channel every message he can compose, but he can learn a message sent over a confidential channel only if the associated receiver identity is compromised.

In addition, we give the intruder two more abilities for secure channels, corresponding to those available in the CCM. Specifically, the last two transitions in Table 5 provide the intruder with the ability to secure authentic channels, and to drop confidentiality from secure channels shared with compromised agents. The two transitions reflect the corresponding intruder capabilities available in the CCM, where an intruder can upgrade a message on a (fresh) authentic channel to one on a (fresh) secure channel, by decrypting it, and dually access the contents of a secure message directed to a compromised receiver, by decrypting it.

3.5 Relating ICM and CCM

We complete our formalization of the AnBx semantics by analyzing the relationship between the ICM and the CCM characterizations. As a first step, we define a correspondence relation \( \sim \) between ICM and CCM states, that relates states that only differ in their encoding of channels. Intuitively, two \( \sim \)-correspondent states encode the same local knowledge for each protocol agent and the intruder.

The formal definition of \( \sim \) is given below: it relies on the simple mapping from ICM states to CCM states shown in Table 6.

<table>
<thead>
<tr>
<th>ICM</th>
<th>CCM</th>
</tr>
</thead>
<tbody>
<tr>
<td>iknows(plain,M)</td>
<td>cfnCh(C;M)</td>
</tr>
<tr>
<td>iknows(ctag,blind_B(M))</td>
<td>cnfCh(B;M)</td>
</tr>
<tr>
<td>iknows(atag,A,\overline{V},M)</td>
<td>athCh(A;\overline{V};M)</td>
</tr>
<tr>
<td>iknows(stag,B,\overline{V},M,N)</td>
<td>secCh(A;\overline{V};B;M)</td>
</tr>
<tr>
<td>iknows(fstag,A,\overline{V},M,N)</td>
<td>secCh(A;\overline{V};B;M,N)</td>
</tr>
<tr>
<td>iknows(cnfCh(A,\overline{V},M,N))</td>
<td>athCh(A;\overline{V};M,N)</td>
</tr>
<tr>
<td>iknows(\overline{X},X)</td>
<td>iknows(X)</td>
</tr>
</tbody>
</table>

Table 4 Translation from ICM to CCM

The only significant difference with respect to the CCM is that the encrypted message for C is replaced by a cfnCh(C;\cdot) channel fact.

Two comments are in order for the ICM translation. First, nonces are implicitly included in the payload of the message when freshness is lost upon forwarding: this choice reflects the corresponding behavior in the CCM, where nonces cannot be removed from digitally signed packets. Second, given that the state channel facts employed in the ICM are persistent, we need additional measures to protect against replicas in all transitions expecting a fresh message on input. For that purpose, we rely on the very same mechanism described earlier for the cryptographic model, based on the seen(\cdot,\cdot) facts to tell replicas apart. Even though we could define additional non-persistent channel facts to model fresh channels, this choice simplifies the definition of the correspondence between the two channel models and the related proof.
S1 and S2 contain the same facts besides channel facts and intruder knowledge facts,
the intruder knowledge in S1 and S2 is the same when removing all messages that contain CCM material; and
the channel facts in S1 and intruder knowledge of crypto-encodings in S2 are equivalent modulo the mapping in Table 6.

Based on this definition, we now turn to the problem of establishing a semantic equivalence between the ICM and the CCM, proving a one-to-one correspondence between attack states. Given an AnBx specification P, let CCM(P) and ICM(P) stand for its translation to the CCM and to the ICM, respectively.

**Theorem 1 (Soundness)** Let P be an AnBx specification. For each state S1 reachable from ICM(P) there exists a state S2 reachable from CCM(P) such that S1 \sim S2.

**Proof** We proceed by induction on the number of steps performed. The initial states are equivalent modulo \sim by definition of our translation. Let us assume, by induction hypothesis, that S1 \sim S2 for some reachable ICM state S1 and some reachable CCM state S2. Let S1 be an ICM state reachable from S1 in one step: we show that there exists a CCM state S2' such that S2' \sim S2.

We proceed by a case analysis on the transition rule r applied to rewrite S1 into S1'. The easiest case is when r is an intruder rule, which does not involve any channel fact (e.g., a rule like \texttt{iknows(M) \cdot iknows(K) \Rightarrow iknows(\{M\}K)}). In this case the very same rule can be applied also in the CCM to obtain an equivalent state. A similar reasoning applies for honest agents rules, given the definition of our translation. The most interesting possibility is when r is an intruder rule involving channel facts.

We show the cases for authentic channels as representative of all other cases:
- Let \( r = \texttt{athCh(A; V; M)} \). Then, it can be shown that if \( S_1 \sim S_2 \), the intruder knows \( M \) also in \( S_2 \). The CCM encoding of the channel fact on the right side of the rule is \( \texttt{iknows(V, M)} \). This term can be constructed by the intruder in the CCM, since \texttt{dishonest(A)} implies that \texttt{inv(sk(A))}. Therefore, there is a reachable state \( S'_1 \) such that \( S'_1 \sim S'_2 \). Let \( r = \texttt{athCh(A; V; M)} \Rightarrow \texttt{iknows(M)} \cdot \texttt{iknows(V)} \). Since \( S_1 \sim S_2 \), the intruder knows \( M \) in \( S_2 \). The CCM encoding of the channel fact, i.e., we have \( \texttt{iknows(V, M)} \). The intruder can thus learn \( M \) and \( V \) by verification of the signature, using \( \texttt{sk(A)} \). Therefore, an \( S'_2 \) with \( S'_1 \sim S'_2 \) is reachable.

The proof for confidential and secure channels proceeds along the same lines, identifying a 1:1 correspondence between a transition in the ICM and a transition in the CCM.

Theorem 1 shows that the CCM is a sound implementation of the ICM: if we verify a protocol in the CCM, then the protocol is also secure in the ICM. The opposite direction, which we call completeness, does not hold in general, since there is an unbounded number of reachable CCM states which do not have any counterpart in the ICM. Still, for verification purposes, we are interested in attack states, and we can in fact prove a formal result about them. Carrying out such proof is challenging, since in principle the intruder can abuse channel encodings inside CCM states for mounting attacks which would not work in the ICM, where such cryptographic messages are not present at all.

The insight is interpreting such abuses as a special case of “type-flaw” attacks, as the intruder is actually fooling the honest agents in improperly using cryptographic material related to the channel encodings. Interestingly, it is well-known that type-flaw attacks can be systematically prevented by good protocol design, when all message components are annotated with sufficient information to enforce a unique interpretation [30,31]. These “typing results” do not keep the intruder from sending ill-typed messages (e.g., sending an encrypted message in place of a nonce); rather, they ensure that every message (part) has a unique interpretation. Then, it can be shown that if an attack exists, also a well-typed attack exists – hence it never helps the intruder to use ill-typed messages. Considering only well-typed attacks is a convenient proof strategy for our completeness.
Typeable protocols. We presuppose a finite set of basic type symbols $B$ (like nonce, agent, etc.). We define the set $T$ of composed types as the least set that contains $B$ and that is closed under the following property: if $\tau_1, \ldots, \tau_n \in T$ and $f$ is a function symbol of arity $n$, then also $f(\tau_1, \ldots, \tau_n) \in T$.

We note with $\Gamma$ typing environments, binding constants and variables of a protocol specification to types, so that $\Gamma(c) \in T$ for every constant $c$ and $\Gamma(X) \in T$ for every variable $X$. We extend $\Gamma$ to a function on arbitrary terms as follows:

$$\Gamma(f(t_1, \ldots, t_n)) = f(\Gamma(t_1), \ldots, \Gamma(t_n)).$$

**Definition 2 (Typeable Protocol)** Consider a CCM protocol specification $P$ with the standard operators for symmetric and asymmetric encryption, and such that communication occurs only via $\mathsf{iknows}(\cdot)$ facts (i.e., the transition rules of the protocol agents operate on disjoint facts for disjoint agents).

Let the set $\mathsf{MP}(P)$ of message patterns of $P$ be defined as the set of all terms of the form $\mathsf{iknows}(m)$ in the initial state and the transition rules of the honest agents; we assume here that variables occurring in $\mathsf{MP}(P)$ are $\alpha$-renamed in such a way that no two distinct elements have a common variable ($\alpha$-renaming is assumed to be type consistent). Let then finally:

$$\mathsf{SMP}(P) = \{ s \mid s \subseteq t \in \mathsf{MP}(P) \land s \notin \mathcal{T} \}$$

$$\cup \{ \mathsf{inv}(k) \mid \{ m \} k \subseteq t \in \mathsf{MP}(P) \},$$

be the the non-variable subterms of message patterns as well as all decryption keys, again under $\alpha$-renaming ($\subseteq$ denotes the subterm relationship.)

We say that the protocol $P$ is typeable in a typing environment $\Gamma$ if for all $s, t \in \mathsf{SMP}(P)$ one has $\Gamma(s) = \Gamma(t)$ iff $s$ and $t$ have a unifier. We omit $\Gamma$ when clear from the context. $\square$

**Theorem 2 (Well-typed Attacks)** If there is an attack against a typeable protocol, then there is a well-typed one, i.e., where every variable $X$ is instantiated with a term $t$ such that $\Gamma(X) = \Gamma(t)$.

**Proof** A simple adaptation of the proof in [32]. See Appendix A for details. $\square$

As usual, the notion of typing we adopt rules out as non-typeable many specifications that are actually perfectly alright. This happens when several messages have similar formats. In this case, we cannot apply the theorem on well-typed attacks (and invoke the completeness theorem below). Fortunately, there is a systematic way to make all protocols typeable, by adding tags to tell different messages apart, a practice which is not expensive in the implementation and does not destroy any standard authentication and secrecy property.

We are finally ready to state and prove the completeness result for our typed model. We conjecture that such result may hold true also for arbitrary attacks on any given protocol, but we do not see any viable proof strategy for this more general setting.

**Theorem 3 (Completeness)** Let $P$ be an AnBx specification and let us assume a well-typed attack in CCM($P$) that leads to the attack state $S_2$. Then there exists a reachable attack state $S_1$ in ICM($P$) such that $S_1 \sim S_2$.

**Proof** First observe that an honest agent can only receive messages that are a well-typed instance of a message in $\mathsf{MP}(P)$ for the CCM variant of $P$. Thus, there is no point for the intruder to construct any such messages unless they can be used for decryption, in which case they are also subterms of a well-typed instance in $\mathsf{MP}(P)$ or $\mathsf{inv}(\cdot)$ thereof.

Further, observe that the key functions $\mathsf{sk}(\cdot)$ and $\mathsf{pk}(\cdot)$ also are typed, and may occur only in the channel encodings in the CCM and not in the AnBx protocol specification. Hence, the intruder can use the channel keys for composition of messages only as it is intended by the protocol, e.g., we can exclude double encryption with the channel key $\mathsf{pk}(\cdot)$.

Now, we prove a stronger statement, namely that any well-typed attack trace in CCM($P$) has a corresponding attack trace in ICM($P$) with the same length such that every state in the first trace corresponds (in the sense defined by $\sim$) to its matching state in the second trace. We proceed by induction on the length of the trace. If the trace is empty, then the conclusion is immediate by definition of our translation. Otherwise, assume the attack trace in CCM($P$) includes a transition from a state $S_2$ to a state $S'_2$. By inductive hypothesis, there exists a reachable state $S_1$ in ICM($P$) such that $S_1 \sim S_2$. We show that there exists an ICM state
Thus, if \( M \) or \( K \) contain \( \text{pk}(\cdot) \) or \( \text{sk}(\cdot) \) as subterms, i.e., they are not related to channel material. Then by definition of \( \sim \) we have that \( \{ \text{iknows}(M), \text{iknows}(K) \} \subseteq S_1 \) and so the same step is possible in the ICM.

- \( K = \text{pk}(B) \) and \( M \) does not contain \( \text{pk}(\cdot) \) or \( \text{sk}(\cdot) \). Then by definition of \( \sim \) we have \( \text{iknows}(M) \in S_1 \). The result here corresponds to proving \( \text{cnfCh}(B; M) \in S_1 \), which can be generated by the rule \( \text{iknows}(B). \text{iknows}(M) \Rightarrow \text{cnfCh}(B; M) \) in the ICM.

- \( K = \text{pk}(B) \) and \( M = \{ \hat{V}, M_0 \} \text{inv}(\hat{a}(A)) \), i.e. the intruder turns an authentic message from \( A \) for verifiers \( \hat{V} \) into a secure message for \( B \). Since \( \text{iknows}(M) \in S_2 \), by definition of \( \sim \) we have \( \text{athCh}(A; \hat{V}; M_0) \in S_1 \), and thus we can reach the corresponding \( \text{secCh}(A; \hat{V}; B; M_0) \in S_1 \) by the rule \( \text{athCh}(A; \hat{V}; M). \text{iknows}(B) \Rightarrow \text{secCh}(A; \hat{V}; B; M) \) in the ICM.

All other encryptions steps would produce messages that cannot be received and we excluded these redundant steps above.

The cases for signing, analysis, and the transitions of honest agents similarly have a 1:1 correspondence in the ICM. \( \Box \)

Theorems 2 and 3 can be combined as follows. Given a protocol \( P \), we verify that its CCM translation satisfies the assumptions of the typing result: note that the conditions to check are purely syntactical and can be effectively mechanized. We then know that, if \( P \) has an attack, then it has a well-typed one, so our completeness result shows that there is also an attack on the ICM. Thus, if ICM(\( P \)) is secure, then so is CCM(\( P \)).

Conceptually, the ICM is the preferential definition of our channels, as it is independent of the specific implementation details and it focuses solely on formalizing the behaviour of channels, and as such is amenable for protocol design. Moreover, for tools like ProVerif [33] and SATMC [34], the ideal model is easier for verification, since it is free of most of the typing problems such as those discussed above. On the other hand, the CCM is more convenient in conjunction with other model-checking tools like the ones of AVISPA [21], where CCM specifications may be verified directly. Collectively, our results have thus relevant practical consequences for automating security verification, with several different tools.

4 Case Study: e-Payment Protocols

We demonstrate AnBz at work on the specification of a wide and interesting class of protocols, namely e-payment protocols, showing how it naturally accounts for all the necessary primitives to provide, and reason about, the required high-level security guarantees.

4.1 Introducing the Case Studies

The first case study we propose is the \( iKP \) e-payment protocols family, showing how AnBz lends itself to a robust and modular design that naturally captures the increasing level of security enforced by the different protocols in the \( iKP \) family, depending on the number of principals possessing a certified signing key. Interestingly, as a byproduct of our design and verification efforts, we isolate a new flaw in the original \( iKP \) specification and propose a fix.

The second case study illustrates a revised version of SET, a protocol that for its complexity is considered a benchmark for protocol analysis. Here, we shift our attention to some known security flaws of the protocol and show that our \( AnBz \) variant is immune to such defects. Notably, the case study employs fresh forward terms of security guarantees. This was largely expected, since the \( AnBz \) channel abstractions convey protection on all message components; however, we believe that our exercise of revisiting existing protocols provides evidence about the value of employing adequate channel abstractions for protocol design. In fact, our revised protocols have a much simpler structure of their original specification and, in principle, a robust implementation can be automatically synthesised from their \( AnBz \) narration, yielding stronger and more scalable security guarantees with limited effort.
We postpone a detailed discussion on the verification setup until Section 4.4, and turn now to the details of the e-payment protocols specification in **AnBx**.

### 4.2 A Basic e-Payment Scheme

We outline the bare-bone specification of an e-payment protocol, exposing the protocol structure and the message formats common to both our case studies.

We presuppose three principals: a Customer \( C \), a Merchant \( M \) and an Acquirer \( A \), i.e., a financial institution entitled to process the payment. In our model, each principal starts with an initial knowledge shared with other participants. Indeed, since most e-payment protocols describe only the payment transaction and do not consider any preliminary phase, we assume that the Customer and the Merchant have already agreed on the details of the transaction, including an order description (\( \text{desc} \)) and a price. We also assume that the Acquirer shares with the Customer a customer’s account number (\( \text{can} \)) comprising a credit card number and the related PIN. The initial knowledge of the three parties can thus be summarized as follows: \( C \) knows \( \text{price, desc, can} \); \( M \) knows \( \text{price, desc} \); and \( A \) knows \( \text{can} \).

The transaction can be decomposed into the following steps:

1. \( C \rightarrow M : \text{Initiate} \)
2. \( C \leftarrow M : \text{Invoice} \)
   (In steps 1 and 2 the Customer and the Merchant exchange all the information which is necessary to compose the next payment messages.)
3. \( C \rightarrow M : \text{Payment Request} \)
4. \( M \rightarrow A : \text{Authorization Request} \)
   (In steps 3 and 4 the Customer sends a payment request to the Merchant. The Merchant uses this information to compose an authorization request for the Acquirer and tries to collect the payment.)
5. \( M \rightarrow A : \text{Authorization Response} \)
6. \( C \leftarrow M : \text{Confirm} \)
   (In steps 5 and 6 the Acquirer processes the transaction information, and then relays the purchase data directly to the issuing bank, which actually authorizes the sale in accordance with the Customer’s account. This interaction is not part of the narration. The Acquirer returns a response to the Merchant, indicating success or failure of the transaction. The Merchant then informs the Customer about the outcome.)

Interestingly, steps (4) and (6) involve forwarding operations, since the Customer never communicates directly with the Acquirer, but some credit-card information from the Customer must flow to the Acquirer through the Merchant to compose a reasonable payment request, while the final response from the Acquirer must flow to the Customer through the Merchant to provide evidence of the transaction.

Besides some elements of the initial knowledge, other information needs to be exchanged in the previous protocol template. First, to make transactions univocally identifiable, the Merchant generates a fresh transaction ID (\( \text{tid} \)) for each different transaction. Second, the Merchant associates to the transaction also a date or any appropriate timestamp. Both pieces of information must be communicated to the other parties. The transaction is then identified by a contract, which comprises most of the previous information: if Customer and Merchant reach an agreement on it, and they can prove this to the Acquirer, then the transaction can be completed successfully. The details on the structure of the contract vary among different protocols. At the end of the transaction, the authorization \( \text{auth} \) is then returned by the Acquirer, and communicated to the two other participants.

**Message formats.** Our protocol templates presuppose the exchange of three kinds of messages: either simple names, \( m \), or tuples of messages (\( \langle M \rangle \)), or else message digests.

We represent digest creation simply as a term \( [M] \) by which an agent may prove the knowledge of a message \( M \) without leaking it to the recipient: this is implemented in OFMC through a non-invertible function symbol. We consider also digests which are resistant to dictionary attacks, hence presuppose an implementation based on a hashing scheme that combines the message \( M \) with a shared key known only to the principal which must verify the digest. We note with \( [M:A] \) a digest of a message \( M \) which is intended to be verified only by \( A \). The symbolic implementation of this HMAC primitive is standard, full details can be found in the scripts employed for our case studies.
4.3 Protocol Goals

We provide a brief overview of our security properties of interest for e-payment protocols. Further details about the validated protocol goals are later reported for each case study.

A first goal we would like to meet for an e-payment system is that all the principals agree on the contract they sign. In terms of OFMC goals, this corresponds to requiring that each participant can authenticate the other two parties on the contract. Moreover, the Acquirer should be able to prove to the other two parties that the payment has indeed been authorized and the associated transaction performed: in OFMC this can be represented by requiring that $M$ and $C$ can authenticate $A$ on the authorization auth.

A stronger variant of the goals described above requires that, after completion of a transaction, each participant provide a non-repudiable proof of the effective agreement by the other two parties on the terms of the transaction. In principle, each principal may wish to have sufficient proofs to convince an external verifier that the transaction was actually carried out as she claims. The lack of some of these provable authorizations does not necessarily make the protocol insecure, but it makes disputes between the parties difficult to settle, requiring to rely on evidence provided by other parties or to collect off-line information.

Finally, we are also interested in some secrecy goals, like verifying that the Customer’s credit card information can is kept confidential, and transmitted only to the Acquirer. In general, we would like to keep the data exchanged by the principals secret among the parties who strictly need to access them for protocol functionality.

4.4 Experimental Setup

We verified the AnBx specifications of iKP and SET by compiling them into their cryptographic implementation, using our tool, and running OFMC on the generated CCM translation against the described security goals. We also encoded and verified the original versions of iKP and SET, and compared the results with those of the revised versions. In the following we report on the results of such tests.

For all the tests we ran OFMC in classic mode with one and two symbolic sessions, using both the typed and the untyped mode: with two sessions we were sometimes unable to complete the verification due to search space explosion. We ran intensive tests with a fixed number of sessions. This bounds how many protocol executions the honest agents can engage in, while the intruder is left unbounded thanks to the symbolic lazy intruder technique in OFMC. In the following we say that a goal is met only if it is satisfied in all the considered settings.

5 The iKP Protocol Family

The iKP protocol family was developed at IBM Research [22, 23, 36] to support credit card-based transactions between customers and merchants (under the assumption that payment clearing and authorization may be handled securely off-line). All protocols in the family are based on public-key cryptography. The idea is that, depending on the number of parties that own certified public key-pairs, we can achieve increasing levels of security, as reflected by the name of the different protocols (1KP, 2KP, and 3KP).

5.1 Protocol Narration

Despite the complexity of iKP, by abstracting from cryptographic details, we can isolate a common communication pattern underlying all the protocols of the family. Namely, a common template can be specified as follows:

1. $C \rightarrow M, \eta_1 : [\text{can}:A], [\text{desc}:M]$
2. $C \leftarrow M, \eta_2 : \text{price}, \text{tid}, \text{date}, [\text{contract}]$
3. $C \rightarrow M, \eta_3 : \text{price}, \text{tid}, \text{can}, [\text{can}:A], [\text{contract}]$
4. $M \rightarrow A$ (decomposed in two steps to specify different communication modes)
   (a) $M \rightarrow A, \eta_{4a} : \text{price}, \text{tid}, \text{can}, [\text{can}:A], [\text{contract}]$
   (b) $M \rightarrow A, \eta_{4b} : \text{price}, \text{tid}, \text{date}, [\text{desc}:M], [\text{contract}]$
5. $M \leftarrow A, \eta_5 : \text{auth}, \text{tid}, [\text{contract}]$
6. $C \leftarrow M, \eta_6 : \text{auth}, \text{tid}, [\text{contract}]$

with $\text{contract} \triangleq (\text{price}, \text{tid}, \text{date}, [\text{can}:A], [\text{desc}:M])$.

By instantiating the exchange modes $\eta_j$ in the previous scheme, one may generate the AnBx variants of the different protocols in the iKP family, achieving different security guarantees: this is exactly what we do in Table 7. Notice that all the considered protocols rely on blind
forwarding at step 4 to communicate sensitive payment information from the Customer to the Acquirer, without disclosing them to the Merchant. Moreover, a forwarding operation is employed at step 6 to preserve the authenticity of the response by the Acquirer.

5.2 Main Results of 3KP Security Verification

We verified the AnBx protocols described above and carried out a corresponding analysis of the original specifications of \{1,2,3\}KP, as amended in [37]. Below we refer to this amended version as the “original” iKP, to be contrasted with the “revised” AnBx version in Table 7. In both cases, we ran our tests assuming that the Acquirer is trusted, i.e., encoded as a concrete agent a rather than as a role A; this is often a reasonable assumption in e-payment applications. As we mentioned earlier, the AnBx specifications are not just more scalable and remarkably simpler, but they also provide stronger security guarantees, which are detailed in Table 8 and commented further below.

During the analysis of the original 2KP and 3KP we found a (to the best of our knowledge) new flaw. It is related to the authenticity of the Authorization response auth that is generated by the Acquirer and then sent to the other principals at steps 5 and 6. In particular, the starred goals in Table 8 are met only after changing the protocol by adding the identities of Merchant and Customer inside the signature of the Acquirer in the original specification. In 2KP, since the Customer is not certified, this can be done with an ephemeral identity derived from the credit card number.

It is worth noticing that, after the completion of the revised and the amended original 3KP, each party has evidence of transaction authorization by the other two parties, since the protocol achieves all the authentication goals that can ideally be requested, according to the number of certified principals. Moreover, our revised 3KP, with respect to the original version, provides the additional guarantee of preserving the secrecy of the authorization response Auth.

In contrast, the original 3KP protocol, the strongest proposed version, fails in two authentication goal: A can only weakly authenticate M and C on [contract]. Luckily, if the transaction ID tid is unique, this is only a minor problem, since [contract] should also be unique, i.e., two different contracts cannot be confused.

6 SET Purchase Protocol

Secure Electronic Transaction (SET) is a family of protocols for securing credit card transactions over insecure networks. This standard was proposed by a consortium of credit card companies and software corporations led by Visa and MasterCard and involving companies like IBM, Microsoft, Netscape, RSA and Verisign.

In the present paper we consider the SET purchase protocol as outlined in [26]. In the following we distinguish a signed and an unsigned version of SET: in the former all the parties possess certified key-pairs, while in the latter the Customer does not.

6.1 Protocol Narration

Given the complexity of SET, to ease the comparison with other works on such protocol, in this presentation the information exchanged by the principals is denoted with the names commonly used in SET specifications. We intro-

<table>
<thead>
<tr>
<th>mode/step</th>
<th>1KP</th>
<th>2KP</th>
<th>3KP</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\eta_1$</td>
<td>$C \rightarrow M$</td>
<td>$(-</td>
<td>-) -$</td>
</tr>
<tr>
<td>$\eta_2$</td>
<td>$C \leftarrow M$</td>
<td>$(-</td>
<td>-) -$</td>
</tr>
<tr>
<td>$\eta_3$</td>
<td>$C \rightarrow M$</td>
<td>$(-</td>
<td>A)$</td>
</tr>
<tr>
<td>$\eta_4$</td>
<td>$M \rightarrow A$</td>
<td>$(-</td>
<td>A)$</td>
</tr>
<tr>
<td>$\eta_5$</td>
<td>$M \leftarrow A$</td>
<td>$@A(C,M)$</td>
<td>$(A</td>
</tr>
<tr>
<td>$\eta_6$</td>
<td>$C \leftarrow M$</td>
<td>$(A</td>
<td>C,M)$</td>
</tr>
</tbody>
</table>

Table 7 Exchange modes for the revised iKP e-payment protocol
duce some basic concepts of the protocol by simply providing a mapping of the exchanged data to the corresponding information in the bare-bone specification presented in Section 4: this should clarify the role of most of the elements. We can identify PurchAmt with price, OrderDesc with desc and pan with can. The initial knowledge of the three parties can then be summarized as follows: C knows PurchAmt, OrderDesc and pan; M knows PurchAmt and OrderDesc; A knows pan.

During the protocol run, the principals generate some identifiers: LIDM is a local transaction identifier that the Customer sends to the Merchant, while the Merchant generates another session identifier XID; we denote the pair (LIDM,XID) with TID. Finally, we complete our abstraction by stipulating Oldata = OrderDesc and Pldata = pan; we let HOD = (\{OData:M\}, \{PlData:A\}). The latter contains the evidence (digest) of the credit card that the Customer intends to use, and the evidence of the order description that will later be forwarded to the Acquirer. In our model HOD plays the role of the dual signature, a cryptographic mechanism central to SET, which is employed to let the Merchant and the Acquirer agree on the transaction without giving any of them full view of the details. Namely, as we said, the Merchant does not need the customer’s credit card number to process an order, but he only needs to know that the payment has been approved by the Acquirer. Conversely, the Acquirer does not need to be aware of the details of the order, but he just needs evidence that a particular payment must be processed.

Although many papers on SET [26,38,35] focus their attention on the signed version of the protocol, again we note that both versions expose a common pattern which allows for an easy specification in AnBx. The narration depicting the common structure of the protocols is reported below:

1. \( C \rightarrow M, \eta_1 : \text{LIDM} \)
2. \( M \rightarrow C, \eta_2 : \text{XID} \)
3. \( C \rightarrow M \) (decomposed in two steps to specify different communication modes)
   (a) \( C \rightarrow M, \eta_3 : \text{TID}, \text{HOD} \)
   (b) \( C \rightarrow M, \eta_3 : \text{TID}, \text{PurchAmt}, \text{HOD}, \text{Pldata} \)
4. \( M \rightarrow A \) (decomposed in two steps to specify different communication modes)
   (a) \( M \rightarrow A, \eta_4 : \text{TID}, \text{PurchAmt}, \text{HOD}, \text{Pldata} \)
   (b) \( M \rightarrow A, \eta_4 : \text{TID}, \text{PurchAmt}, \text{HOD} \)
5. \( A \rightarrow M, \eta_5 : \text{TID}, \text{HOD}, \text{AuthCode} \)
6. \( M \rightarrow C, \eta_6 : \text{TID}, \text{HOD}, \text{AuthCode} \)

Table 9 shows the communication modes we specify to instantiate the previous protocol template to our revised variants of the unsigned and signed versions of SET.

6.2 Main Results of SET Security Verification

We verified the AnBx specifications of the SET purchase protocol and carried out a correspond-

<table>
<thead>
<tr>
<th>Goal</th>
<th>1KP</th>
<th>2KP</th>
<th>3KP</th>
</tr>
</thead>
<tbody>
<tr>
<td>can secret between C,A</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>A weakly authenticates C on can desc</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>auth secret between C,M desc</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>price secret between C,M,A auth</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>M authenticates A on auth</td>
<td>+*</td>
<td>+</td>
<td>+*</td>
</tr>
<tr>
<td>C authenticates A on auth</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>A authenticates C on [contract]</td>
<td>-</td>
<td>-</td>
<td>w</td>
</tr>
<tr>
<td>M authenticates C on [contract]</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>A authenticates M on [contract]</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>C authenticates M on [contract]</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>C authenticates A on [contract],auth</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>M authenticates A on [contract],auth</td>
<td>+*</td>
<td>+</td>
<td>+*</td>
</tr>
</tbody>
</table>

* goal satisfied only fixing the definition of SigA [23]

w = only weak authentication

Table 8  Security goals satisfied by Original and Revised iKP
The revised protocol follows the general structure of the original SET protocol, with five phases: application protocols, data binding, exchange modes, and the payment phase.

### Table 9: Exchange modes for the revised SET e-payment protocol

<table>
<thead>
<tr>
<th>mode/step</th>
<th>unsigned SET</th>
<th>signed SET</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\eta_1$</td>
<td>$C \rightarrow M$</td>
<td>$\langle -</td>
</tr>
<tr>
<td>$\eta_2$</td>
<td>$C \leftarrow M$</td>
<td>$\langle M</td>
</tr>
<tr>
<td>$\eta_{3a}$</td>
<td>$C \rightarrow M$</td>
<td>$\langle -</td>
</tr>
<tr>
<td>$\eta_{3b}$</td>
<td>$C \rightarrow M$</td>
<td>$\langle -</td>
</tr>
<tr>
<td>$\eta_{4a}$</td>
<td>$M \rightarrow A$</td>
<td>$\langle -</td>
</tr>
<tr>
<td>$\eta_{4b}$</td>
<td>$M \rightarrow A$</td>
<td>$\langle (M</td>
</tr>
<tr>
<td>$\eta_5$</td>
<td>$M \leftarrow A$</td>
<td>$\langle (A</td>
</tr>
<tr>
<td>$\eta_6$</td>
<td>$C \leftarrow M$</td>
<td>$\langle (A</td>
</tr>
</tbody>
</table>

*certified agents*: $M,A$ and $C,M,A$

### Table 10: Security goals satisfied by Original and Revised SET purchase protocol

<table>
<thead>
<tr>
<th>Goal</th>
<th>unsigned SET</th>
<th>signed SET</th>
</tr>
</thead>
<tbody>
<tr>
<td>pan secret between $C,A$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$A$ weakly authenticates $C$ on pan</td>
<td>$-$</td>
<td>$-$</td>
</tr>
<tr>
<td>OrderDesc secret between $C,M$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>PurchAmt secret between $C,M,A$</td>
<td>$-$</td>
<td>$-$</td>
</tr>
<tr>
<td>AuthCode secret between $C,M,A$</td>
<td>$-$</td>
<td>$-$</td>
</tr>
<tr>
<td>$M$ authenticates $A$ on AuthCode</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$C$ authenticates $A$ on AuthCode</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$C$ authenticates $M$ on AuthCode</td>
<td>$+$*</td>
<td>$+$</td>
</tr>
<tr>
<td>$A$ authenticates $C$ on contract</td>
<td>$-$</td>
<td>$-$</td>
</tr>
<tr>
<td>$M$ authenticates $C$ on contract</td>
<td>$-$</td>
<td>$-$</td>
</tr>
<tr>
<td>$A$ authenticates $M$ on contract</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$C$ authenticates $M$ on contract</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$C$ authenticates $A$ on contract,AuthCode</td>
<td>$-$</td>
<td>$+$</td>
</tr>
<tr>
<td>$M$ authenticates $A$ on contract,AuthCode</td>
<td>$+$</td>
<td>$+$</td>
</tr>
</tbody>
</table>

*goal satisfied only fixing step 5 as in [26]*

$w$ = only weak authentication

for revised SET: $contract = \text{PriceAmt},\text{TID},[\text{PIData}:A], [\text{OIData}:M]$

for original SET: $contract = \text{PriceAmt},\text{TID},\text{hash}([\text{PIData}],\text{hash}([\text{OIData}]))$

Table 10 summarizes the security goals satisfied by the Original and Revised SET purchase protocol.
may allow a dishonest Customer to cheat on an honest Merchant when collaborating with another dishonest Merchant. The attack is based on the fact that neither LIDM nor XID can be considered unique, so they cannot be used to identify a specific Merchant. Therefore the customer can start a parallel purchase with an accomplice, playing the role of another merchant, and make the Acquirer authorize the payment in favor of the accomplice. Here, again the goal "C authenticates M on AuthCode" fails.

During our analysis we also verified that both the original specifications [26,38] fail to verify the goals “C authenticates A on AuthCode” and “C authenticates M on contract.AuthCode”. To overcome this problem the protocol must be fixed in the sixth (and final) step, as already outlined in [35]. This issue also leads us to more interesting considerations on how to prove the authorization of the transaction.

Proving authorization of the transaction. The previous problem arises from the fact that the Customer does not have an evidence of the origin of AuthCode by the Acquirer and she rather relies only on information provided by the Merchant. For example, giving to the Customer a proof that the Acquirer authorized the payment requires substantial modification of the sixth step of the protocol. In fact, instead of letting the Merchant sign a message for the Customer, we exploit the AnBx forward mode to bring to the Customer the authorization of the payment signed directly by the Acquirer. It is worth noticing that, employing a fresh forward mode in the sixth step, we can achieve the desired strong authenticity goal on the pair, even though the transaction identifier is not unique.

We can then confirm the results outlined in [35], showing that, while iKP meets all the non-repudiation goals, the original specification of SET does not. It is important to notice that, to achieve non-repudiation, each participant must have sufficient proofs to convince an external verifier that the transaction was actually carried out as she claims. A way to obtain this is to assume that the authentication is obtained by means of digital signatures computed with keys which are valid within a Public Key Infrastructure and are issued by a trusted third party (Certification Authority). Although this limits the way authentic channels in AnBx could be implemented, in practice it does not represent a significant restriction, since in the considered protocols digital signature is the standard way meant to achieve authentication.

7 Conclusions

We presented AnBx, the currently most expressive Alice & Bob-style language. The distinguishing key-features of the language is a powerful concept of channels that includes forwarding. We analysed the formal details related to the definition of the language, and we proved a connection between the ideal behaviour of our channels and a simple cryptographic implementation. We showed the amenability of the language on two case studies from the e-payment area, namely iKP and SET, and we argue that the abstraction from low-level security mechanisms turns out to be helpful for protocols designers. Our compiler from AnBx to IF is available online along with the related documentation and the source code of both our case studies.

Acknowledgements This work was partially supported by the MIUR Projects SOFT (Security Oriented Formal Techniques), IPOS (Interacting Processes in Open-ended Distributed Systems) and CINA (Compositionality, Interaction, Negotiation and Autonomy), and by FP7-ICT-2007-1 Project no. 216471, AVANTSSAR (Automated Validation of Trust and Security of Service-oriented Architectures). The authors thank Luca Viganò, David Basin, and Benedikt Schmidt for helpful comments.

References


5 http://www.dais.unive.it/~modesti/anbx/
A Proof of Theorem 2

The idea behind the proof is to abuse a popular verification technique as a proof argument, namely the symbolic constraint-based approach that we call “the lazy intruder” [39,40,41,32]. The intuition behind the lazy intruder is as follows. Every trace can be seen as an instance of a symbolic trace, i.e., a sequence of transition rule applications where we delay the unification of left-hand side knows(m) facts and leave variables in there uninstantiated. Instead, we keep a constraint M ⪰ m, where M is the set of messages the intruder knows at that state. Such a constraint expresses that the intruder must be able to generate the message m from knowledge M. Thus, these constraints before reduction contain only messages m, or instances thereof, for which knows(m) occurs in the IF specification of the protocol P. It can be shown that, if there is an attack trace, then there is a corresponding symbolic trace with satisfiable intruder constraints, hence in the proof we can focus without loss of generality on such symbolic traces.

The lazy intruder technique is based on a calculus of constraint reduction rules for checking their satisfiability (and, if satisfiable, determine a solution). There are three constraint reduction rules: GENERATE (to compose new messages from public function symbols), ANALYZE (to obtain all subterms of known messages by decryption and projection) and UNIFY, which states that a possible solution to the constraint M ⪰ s exists if there is a t ∈ M, both s and t are not variables, s and t have the most general unifier σ, and all other constraints are satisfiable under σ. The formal constraint reduction rules are reported below, where we let φ range over M ⪰ s constraints or conjunctions thereof. For the ANALYZE rule we give only the example of asymmetric decryption, other rules are similar.

\[
\begin{align*}
\text{UNIFY} & \quad (σ \in mgu(s,t) \text{ and } s,t \notin V) \\
& \quad φσ \\
& \quad φ \land (\{s\} \cup M \vdash t)
\end{align*}
\]

\[
\begin{align*}
\text{GENERATE} & \quad (f \text{ public}) \\
& \quad φ \land (M \vdash t_1) \land \ldots \land (M \vdash t_n) \\
\text{ANALYZE} & \quad ([m]_k \in M) \\
& \quad φ \land (M \vdash t) \land M \vdash \text{inv}(k) \\
& \quad φ \land M \vdash t
\end{align*}
\]

We can finally prove the theorem.

Restatement of Theorem 2 If there is an attack against a typeable protocol, then there is a well-typed one, i.e., where every variable X is instantiated with a term t such that Γ(X) = Γ(t).

Proof Assume an arbitrary attack trace and consider its corresponding symbolic trace; we show that all instantiations of variables of its (satisfiable) intruder constraints are well-typed, hence the existence of an attack implies the existence of a well-typed one. Technically, we actually need to prove a stronger result by induction over the entire constraint reduction: we prove that every message occurring in the constraints, and any arbitrary subterm of it, is either a variable or an instance of a message in SMP(P), and that all variables are only instantiated in a well-typed way.

Let us first consider a protocol P such that no element of MP(P) is a variable, i.e., P does not involve any step where a “bare value” is transmitted, but all messages are composed terms or constants. In this case, MP(P) ⊆ SMP(P), i.e., the union of the initial intruder knowledge and the messages exchanged in P is included in SMP(P). The GENERATE and ANALYZE cases are straightforward to handle, since, in particular, such rules do not instantiate any variable. In the UNIFY case, both s and t must be instances of elements of SMP by induction, since they are not variables. Given that s and t have a unifier, the typability assumption implies Γ(s) = Γ(t), hence also all corresponding subterms of s and t must have the same types by Definition 2, and the substitution σ is hence well-typed.

Finally, we extend the proof to any protocol P we excluded above, i.e., such that there exists a variable in MP(P). Let P′ be a modification of P where every “bare variable” X is replaced by the composed term (t.X) for some fresh tag t that is initially known to the intruder. Assume now that P has an attack, then there is a well-typed attack on P′, but it is immediate that such well-typed attack works also on P when removing the tag t.