AnBx - Security Protocols Design and Verification*

Michele Bugliesi and Paolo Modesti

Università Ca' Foscari Venezia
Dipartimento di Informatica
{bugliesi,modesti}@dsi.unive.it

Abstract Designing distributed protocols is challenging, as it requires actions at very different levels: from the choice of network-level mechanisms to protect the exchange of sensitive data, to the definition of structured interaction patterns to convey application-specific guarantees. Current security infrastructures provide very limited support for the specification of such guarantees. As a consequence, the high-level security properties of a protocol typically must often be hard-coded explicitly, in terms of low-level cryptographic notions and devices which clutter the design and undermine its scalability and robustness.
To counter these problems, we propose an extended *Alice & Bob* notation for protocol narrations (`AnBx`) to be employed for a purely declarative modelling of distributed protocols. These abstractions provide a compact specification of the high-level security guarantees they convey, and help shield the design from the details of the underlying cryptographic infrastructure. We discuss an implementation of the abstractions based on a translation from the `AnBx` notation to the AnB language supported by the OFMC [1,2] verification tool. We show the practical effectiveness of our approach by revisiting the iKP e-payment protocols, and showing that the security goals achieved by our declarative specification outperform those offered by the original protocols.

1 Introduction

On-line transactions represent an important share of the overall world trade and security constitutes a major concern in this kind of applications, as agreeing, on the terms of a transaction in a distributed and open environment like the internet, requires protection against threats from intruders and/or from the potential misbehavior of other participants. Establishing the desired safeguards is challenging as it involves actions at different levels: from the choice of core, network-level mechanisms to protect the exchange of sensitive data, to the definition of structured, application-specific measures to enforce the high-level behavioral invariants of the participants. Current security infrastructures offer effective abstractions only for the core mechanisms, based on tools such as TLS/SSL [3] to provide tunneling support for communication. On the other hand, little to no support is provided for the specification of more structured interaction patterns, so that high-level security invariants must typically be expressed,

* Work partially supported by MIUR Projects SOFT "*Security Oriented Formal Techniques*" and IPODS "*Interacting Processes in Open-ended Distributed Systems*".

and hard-coded explicitly, in terms of low-level cryptographic notions such as salting, nonces, keyed-hashing, encryptions, signature schemes, and compositions thereof. As a result, the application code and data structures get intertwined with low-level code that not only gets in the way of a clear understanding of the applications' business logic, but also undermines its scalability and robustness.

To counter these problems, various papers in the recent literature (see, e.g., [4,5,6]) have advocated a programming discipline based on (*i*) high-level security abstractions and mechanisms for composing them to support structured interaction patterns [7,8], and (*ii*) automatic techniques to build defensive implementations on top of well-established cryptographic infrastructures and tools.

Following this line of research, in the present paper we isolate a core set of channel and data abstractions to be employed for a purely declarative modelling of distributed protocols. Our abstractions are part of `AnBx`, a dialect of the well-known *Alice & Bob* (AnB) notation for protocol narrations, which supports various mechanisms for securing remote communications based on abstract security *modes*, without any reference to explicit cryptography. The `AnBx` abstractions are readily translated into corresponding public-key cryptographic protocols described by standard AnB narrations. This provides an abstract, yet effective implementation of the `AnBx` specification language, to be employed as the basis for the development of fully-fledged implementation.

Main contributions and results. We developed a compiler for the automatic translation from `AnBx` to the AnB notation used in the symbolic model-checker OFMC (Open-source Fixed-point Model Checker [1,2]), and verified the soundness of our implementation with OFMC itself. The translation allows the verification of `AnBx` protocols with any OFMC-interoperable verification tool [9]. To experiment and validate the practical effectiveness of the `AnBx` approach to protocol design, we revisited the *iKP* e-payment protocol family (Internet Keyed Payment Protocol [10,11]) as a case study, and contrasted the security goals achieved by our version with those offered by the original protocol.

Interestingly, our `AnBx` versions of the iKP protocols outperform the original protocols (for all *i*'s), i.e. they satisfy stronger security goals and properties. This is largely a consequence of the declarative nature of the specification style supported by `AnBx`: being defined as channel-level abstractions, the `AnBx` primitives convey protection on *all* message components, not just on some components as in the original iKP specification, yielding stronger encapsulation mechanisms, and consequently, stronger and more scalable security guarantees. As a byproduct of our comparative analysis, we also found a (to the best of our knowledge) new flaw in the original specification of {2,3}KP, and proposed an amended version that rectifies the problem.

Plan of the paper. In Section 2 we introduce the `AnBx` specification language together with our high-level security abstractions; in Section 3 we outline a translation of the abstractions into a low-level, cryptographic language. In Section 4 we present the `AnBx` compiler that implements that translation using the AnB• cryptographic language supported by OFMC as a target, and discuss the soundness of the translation. In Section 5 we show the abstractions at work on the specification of the iKP protocols, and discuss their properties. Section 6 concludes the presentation.

Table 1 `AnBx` communication modes

η	*mode*	*semantics*
$(-,-)$	plain	conveys no security guarantee
$(A,-)$	from A	a public, but *authentic* exchange which provides the receiver with a guarantee on the origin of the message
$(@A,-)$	fresh from A	$(A,-)$ with the additional guarantee that the message is fresh, and may not be replayed
$(-,B)$	secret for B	a *secret* transmission, providing guarantees that only the intended receiver B will be exposed to the message payload: an intruder may become aware of the existence of an output, but not of the message contents
(A,B)	from A, secret for B	combines the guarantees of modes $(A,-)$ and $(-,B)$
$(@A,B)$	fresh from A, secret for B	combines the guarantees of modes $(@A,-)$ and $(-,B)$
$\uparrow \eta_0$	forward	conveys some of the guarantees given by η_0 and signals that the message did not originate at the sender's; η_0 can not be a forward mode itself

2 `AnBx`: declarative protocol narrations

Looking at the existing protocols for electronic transactions, such as e-payment, e-cash and e-voting, one notices that they are characterized by very specific interaction patterns, expressed by few messaging primitives and data structures. In this section, we isolate a core set of these primitives, and encode them in terms of (i) different *modes* for remote communication, and of (ii) a hiding transformer on data. We inject these modes and the data transformer into an extended version of the familiar AnB specifications for security protocols, whose syntax may be defined as follows:

$A : \alpha$	local action performed by A
$i.\ A \to B, \eta : m$	A sends a message m to B in mode η (symmetrically $\leftarrow$receives from)

The action statements $A : \alpha$ may be employed to specify operations performed by a principal, such as the generation of a new key or a test evaluation, as well as to declare the initial knowledge that we assume available to the principal. In the exchange statements $i.\ A \to B, \eta : m$, i is an index labelling a protocol step: the mode η and the format of the message m are discussed below.

Message formats. An important aspect of our abstractions is the choice of the message formats. A message may either be a tuple of names $(\tilde{n})$, or a reference to a message exchanged at a previous protocol step $(\uparrow i)$, or a message digest $[\tilde{m}]$. Notice that no explicit cryptographic operator is available for message formation, and the only operation on data is the creation of *digests* (or footprints) needed in most e-commerce and e-voting protocols: being able to form $[m]$ proves the knowledge of

Table 2 `AnBx` forward modes

$A \to B, \eta_0 : m$	$B \to C, \eta_1 : m$	*mode*	*semantics*
$(-, C)$	$\uparrow (-, C)$	blind	- Secrecy for C is preserved - B is not exposed to m
(A, C)	$\uparrow (A, C)$	blind	- Authenticity from A is preserved - Secrecy for C is preserved - B is not exposed to m
$(A, -)$ (A, B) $(@A, -)$ $(@A, B)$	$\uparrow (A, -)$	sighted	- Authenticity from A is preserved - Freshness (if present) is lost - B is exposed to m
$(A, -)$ (A, B) $(@A, -)$ $(@A, B)$	$\uparrow (A, C)$	sighted secret	- Authenticity from A is preserved - Freshness (if present) is lost - Secrecy for C is added to m, but C can not make any assumption on secrecy at origin A - B is exposed to m

m without leaking it. We assume digests to be resistant to chosen-plaintext attacks, hence presuppose an implementation based on a hashing scheme that packages m together with a randomized quantity known to the principals that possess m (the designated verifier of $[m]$), and is never leaked to any principals that do not have knowledge of m. To ease the implementation, we allow digests to be tagged with an annotation that specifies the intended verifier, as in $[\tilde{m} : B]$.

Communication modes. Following [17,18], our abstractions encompass two fundamental mechanisms for security, based on secrecy and authentication, and include the communication modes that result from their possible combinations. The mode η is encoded by a pair (*source*, *destination*) that qualifies the security guarantees conveyed at the two end-points of the remote communication. The structure and the informal reading of the communication modes are described in Table 1.

Forward modes. They can be used to model some three party exchanges, in n-party protocols where $n > 2$. The statement $B \to C, \eta_1 : m$ with $\eta_1 = \uparrow\eta$ expresses the intention of B to forward a message m to C. The statement is legal just in case B has been the target of a previous exchange $A \to B, \eta_0 : m$, where m is the same in both statements (or else $m = \uparrow i$ where i is the label identifying the $A \to B$ exchange).

When this is the case, the mode η used in the forward statement will preserve, and possibly extend, the security properties specified by η_0. There are additional coherence constraints that must be satisfied when specifying the mode of an exchange, to rule out unwanted effects of impersonation, and other inconsistencies that would result into unexecutable or broken protocols:

- the legal combination of η_0 and η_1 only are those in Table 2 and they specify possible forward modes;
- in $A \to B, \eta : m$, η may not specify a source other than A, unless η is a forward mode;

Table 3 AnB-like Intermediate Syntax Notation

syntax	*description*
(PK_A, PK_A^{-1})	long term (pub, priv) key pair used by principal A for encryption
(SK_A, SK_A^{-1})	long term (pub, priv) key pair used by A for signature
$\{m\}_{PK_A}$	message m encrypted with the public key P_{KA} of principal A
$\{m\}_{PK_A^{-1}}$	message m decrypted with the private key PK_A^{-1} by principal A
$\{m\}_K \quad \{m\}_{K^{-1}}$	message m encrypted (decrypted) with a symmetric key K
$\{H(m)\}_{SK_A^{-1}} \equiv sig_A(m)$	digital signature of the message m, signed by principal A, where H is the hash function specified by the digital signature scheme
$(m, sig_A(m)) \equiv S_A(m)$	message m digitally signed by principal A
n_A	nonce generated by principal A
$new\ K$	fresh symmetric key, randomly generated
$g^x\ mod\ p$ $g^y\ mod\ p$	fresh half keys generated during a discrete logarithm based agreement protocol (e.g. Diffie-Hellman), where p is prime and g is primitive root $mod\ p$
g^{xy}	fresh symmetric key, computed using a discrete logarithm based agreement protocol ($mod\ p$ is omitted in the notation)
$Hmac_K(m)$	HMAC of message m, computed with the key K
$A:\ \alpha$	action α performed by principal A
$A \rightarrow B:\ m$	A sends a message m to principal B (symmetrically $\leftarrow$)

– if η specifies a destination other than B, then m can be forwarded only blindly.

3 A cryptographic translation for `AnBx`

We outline a translation of the abstract specification language we just introduced into an intermediate representation based on few, well defined, building blocks: keys, cryptographic operations and primitives for remote communication. The intermediate notation, displayed in Table 3, is derived from the well known AnB security protocol notation, and represents the low-level counterpart of the `AnBx` abstract notation discussed in Section 2. The core resulting from the translation can then be further compiled against a specific and concrete target (operating system, programming language, cryptographic libraries, network protocols, etc) to produce the running code.

This translation is based on public key cryptography, and presupposes the existence of a Public Key Infrastructure (PKI) supporting a robust distribution (and use) of public keys and certificates (including their verification). We also assume that the underling PKI supports dual key pairs, for encryption and digital signatures [19].

Table 4 Exchange Modes Translation of $A \to B, \eta : m$

η	*mode*	*translation*	*certified*
$(-,-)$	plain	$A \to B :\ m$	-
$(A,-)$	from A	$A \to B :\ S_A(B,m)$	A
$(-,B)$	secret for B	$A :\ new\ K$ $A \to B :\ \{K\}_{PK_B}, \{m\}_K$	B
(A,B)	from A, secret for B	$A :\ new\ K$ $A \to B :\ \{K\}_{PK_B}, \{S_A(B,m)\}_K$	A, B
$(@A,-)$	fresh from A	$A \to B :\ A$ $A \leftarrow B :\ \{n_B, B\}_{PK_A}$ $A \to B :\ \{n_B, sig_A(B,m)\}_{PK_B}, S_A(B,m)$	A, B
$(@A,-)$	fresh from A with DH*	$A \to B :\ g^x$ $A \leftarrow B :\ \{g^y, n_B, B\}_{PK_A}$ $A \to B :\ \{n_B, sig_A(B,m)\}_{g^{xy}}, S_A(B,m)$	A
$(@A,B)$	fresh from A, secret for B	$A \to B :\ A$ $A \leftarrow B :\ \{n_B, B\}_{PK_A}$ $A :\ new\ K$ $A \to B :\ \{n_B, K\}_{PK_B}, \{S_A(B,m)\}_K$	A, B

* this translation is used only when the intended recipient B is not certified, and therefore the standard *fresh from* A can not be applied

Each `AnBx` principal may thus possess up to two pairs of certified encryption keys: if it does posses both key pairs, we say that the principal is *certified*.

We also use symmetric encryption schemes, which are notoriously computationally more efficient than the asymmetric counterpart. Ideally perfect encryption is assumed and the hashing functions are expected to be collision-free and non-invertible. Practically the *secrecy* notion we expect is the standard computational secrecy, i.e. all polynomial time adversaries have negligible success probabilities. This allows to use real cryptographic protocol for a concrete implementation.

Communication Modes Translation. Table 4 summarizes the translation of the different `AnBx` communication modes into a sequence of AnB statements and shows which principals must be certified. In a real implementation we could include, for robustness, also some additional information (tags) such as protocol name, version, step number, process identifier, origin, destination, but at this level of abstraction we can skip these details.

The *authenticity* property enforced by the "*fresh from*" mode corresponds to Lowe's injective agreement [20] while the property enforced by the "*from*" mode corresponds to Lowe's non-injective agreement. In the former case we say that B authenticates A on m, while in the latter the authentication is only *weak*. We use mostly standard techniques, hence here we will underline only few additional points.

Table 5 Forward modes translation

η_0	η_1	*mode*	m_1	*certified*	*notes*
$(-, C)$	$\uparrow (-, C)$	blind	$\{K\}_{PK_C}, \{m\}_K$	C	$m_0 = m_1$
(A, C)	$\uparrow (A, C)$	blind	$\{K\}_{PK_C}, \{S_A(C, m)\}_K$	A, C	$m_0 = m_1$
$(A, -)$ $(@A, -)$	$\uparrow (A, -)$	sighted	$S_A(C, B, msg)$	A	$m_1 \subseteq m_0$ $m = C, msg$
(A, B) $(@A, B)$	$\uparrow (A, -)$	sighted	$S_A(C, B, msg)$	A, B	$m_1 \subseteq m_0$ $m = C, msg$
$(A, -)$ $(@A, -)$	$\uparrow (A, C)$	sighted secret	$\{K\}_{PK_C}, \{S_A(C, B, msg)\}_K$	A, C	$S_A(C, B, msg) \subseteq m_0$ $m = C, msg$
(A, B) $(@A, B)$	$\uparrow (A, C)$	sighted secret	$\{K\}_{PK_C}, \{S_A(C, B, msg)\}_K$	A, B, C	$S_A(C, B, msg) \subseteq m_0$ $m = C, msg$

`AnBx` specification: $A \rightarrow B, \eta_0 : m$; $B \rightarrow C, \eta_1 : m$

$m_1 \subseteq m_0$: message m_1 is a component of message m_0

- in *authentic* exchanges the identity of the intended recipient B is included to comply with the definition of the Lowe's non-injective agreement, (injective for *fresh* exchanges) [20];
- in *secret* exchanges we use an hybrid construction, with the fresh symmetric key K acting as a confounder. If both, key and data encapsulation schemes, are secure against adaptive chosen ciphertext attacks, then the hybrid scheme inherits that property as well [21];
- combining authenticity and secrecy, we first sign and then encrypt [22];
- in *fresh* exchanges we use a challenge-response technique based on nonces. We do not use timestamps, because they require synchronization (time servers), and this introduce more complexity (attacks, failures, etc);
- in *fresh from* exchanges, when the intended recipient is not certified, we use a combination of challenge-response and Diffie-Hellman key agreement protocol;

Forward Modes Translation. Table 5 outlines the translation of the forward modes. Each clause gives the translation of the statement $B \rightarrow C, \eta_1, m$ with reference to the statement $A \rightarrow B, \eta_0, m$ that originated m: we only give the translation for the legal combinations of η_0, η_1: in all other cases the translation is undefined.

Given that the forward modes in η_1 will never include the freshness tag @, the translation of $B \rightarrow C, \eta_1, m$ amounts to just one AnB step, $B \rightarrow C, m_1$ with m_1 as defined in Table 5.

Notice that m_1 is always defined in terms of m_0, which is the closing AnB message resulting from the translation of the reference `AnBx` statement $A \rightarrow B, \eta_0, m$. In the *sighted* modes, the presence of the identities B and C is necessary because both principals have to authenticate A on msg, which represents the real high level payload of the three-party exchange.

Table 6 Message Formats Translation (digest formed by principal A)

digest	*translation*	*certified*	*notes*
$[m : B]$	$A :\ new\ K$ $A \rightarrow B : Hmac_K(m), \{K\}_{PK_B}$	B	*Obfuscated verifiable digest* Can be verified only by B, if B knows m (beside the digest's creator A)
$[m : -]$	$A :\ new\ K$ $A \rightarrow B :\ Hmac_K(m)$	-	*Obfuscated unverifiable digest* Can not be verified by anyone (except the digest's creator A)
$[m]$	$\mathscr{H}(m)$	-	*Plain digest* Can be verified by everyone knowing m $\mathscr{H}$ cryptographic hash function
$[m : C]$	$p :\ new\ K$ $p \rightarrow C :\ \{\mathscr{H}(m_p)\}, \{K_p\}_{PK_C}$	C	*Digest verifiable by proxy* $p \in \{A, B\}$ C can notify to any principal if $\mathscr{H}(m_A) = \mathscr{H}(m_B)$ or not

Message Format Translation. (Table 6). As we observed, an implementation of a message digest $[m]$ should guarantee that the only principals entitled to verify that $[m]$ matches m are those who already know m, either as initial information, or provided explicitly by a legitimate source. In addition, a robust implementation should protect the digest against chosen-plaintext attacks, and additionally guarantee that even in case m is disclosed to a non-legitimate principal any time in the future, that principal should not be able to use m to match $[m]$.

These guarantees can be made by interpreting (and translating) the digest formation in terms of a hashing scheme that packages m together with a randomized quantity known to the principals that possess m, and never leaked to any principals that do not have knowledge of m. A hashing scheme with these properties is supported by what is known as keyed-Hash Message Authentication Code (HMAC).

The subtlety in our translation is in the way the secrets keys are made available to the legitimate principals, as the structure of the digest $[m]$ does not inform on who is intended to act as the digest's verifier. In certain cases, that information may be recovered by an analysis of the initial knowledge of the participants. However, in general, it requires the designer's intervention to tag each digest occurrence $[m]$ with an annotation $[m : B]$ that signals that B is the designated verifier of $[m]$ (a digest can also be always verified by its creator). Table 6 illustrates the translation scheme for digests based on these additional annotations proposing also alternatives when the digest can not directly verified, due to the lack of certification of some participant. When and how to use these alternatives should be carefully evaluated since it could allow an intruder to perform downgrade attacks.

Table 7 AnB OFMC specification of "*fresh from*" mode

```
Protocol: Fresh_From_A
Types:
  Agent A,B;
  Number Msg,N1;
  Function pk,sk
Knowledge:
  A: A,B,pk,sk,inv(pk(A)),inv(sk(A));
  B: A,B,pk,sk,inv(pk(B)),inv(sk(B))
Actions:
  A -> B: A
  B -> A: {N1,B}pk(A)
  A -> B: {N1,hash({B,Msg}inv(sk(A)))}pk(B),{B,Msg}inv(sk(A))
Goals:
  B authenticates A on Msg
```

4 Implementing the translation: the AnBx compiler

We have implemented the AnBx translation we just outlined using a subset of the AnB cryptographic language supported by the OFMC model checker as a target. In this section, we briefly outline the implementation and discuss its soundness.

An overview of OFMC. OFMC [1,2]) supports two protocol specification languages, the Intermediate Format (IF) and a dialect of the AnB style of protocol narrations, that we will refer to as OFMC-AnB. Details about of OFMC-AnB can be found in [23] and in documentation included in the OFMC software package, so we will give a very short description. A protocol specification comprises various sections:

- `Types`: describes the entities (agents/principals) involved in the protocol, as well as protocol data and data operators (constants, cryptographic functions, ... etc.)
- `Knowledge`: specifies the initial knowledge of each principal
- `Actions`: specifies the sequence of statements that constitute the ideal, unattacked run of the protocol;
- `Goals`: specifies the goals that the protocol is meant to convey.

A sample specification is reported in Table 7, where we give the AnB narration of the low-level translation for the "fresh-from" `AnBx` messaging primitive.
`Msg` is the message being authenticated, using `N1` to complete the nonce-exchange; `pk` and `sk`, generate the public keys for encryption and signing. Each principal is assumed to know its own and its partner's identities, the public key of all known principals and its own private keys. The specification is completed with the authentication goal for the transmitted message `Msg`. This goal already implies the freshness of the message, since in OFMC authenticity goals correspond to the requirements of Lowe's injective agreement.

`AnBx` and its compiler. `AnBx` is defined as variant of OFMC-AnB that supports the messaging modes discussed in §2 as well as few additional features in the preamble sections. A sample `AnBx` specification is in reported Table 8, where we give the `AnBx` specification of the "fresh from" messaging primitive. Notice that while in the action section we use `AnBx` syntax to specify the transmission modes, the protocol goals are expressed by means of OFMC-AnB goal statements.

Table 8 `AnBx` specification of "*fresh from*" mode

```
Protocol: Fresh_From_A
Types:
  Agent A,B;
  Certified A,B;
  Number Msg;
  Function id
Definitions
  Msg: Msg1,Msg2
Knowledge:
  A: A,B;
  B: A,B
Actions:
  A -> B,(@A,-): Msg
Goals:
  B authenticates A on Msg
```

Given an `AnBx` specification, the compiler generates a corresponding OFMC-AnB protocol specification that results from composing the translations of each `AnBx` statement into a corresponding OFMC-AnB protocol, as outlined in §3: thus, while `AnBx` supports abstract messaging primitives for secrecy and authentication, the code generated by the compiler only involves standard, "plain" messages exchanges, and relies of the cryptographic constructions of §3 to achieve the desired security guarantees. In addition to generating the AnB protocol steps, the compiler implements all the bookkeeping needed to generate the entries in the preamble sections (`Types` and `Knowledge`) required to make a consistent OFMC-AnB specification. To illustrate, the AnB protocol in table 7 is, in fact, the result of compiling the `AnBx` in Table 8.

One may wonder why we did not take advantage of the bulleted OFMC-AnB channels in our translation. The fact is that the combination of the three OFMC basic kinds of channel (authentic, confidential, secure) does not allow one to express communication patterns as those defined by the forward modes.

This can be easily seen if we try to build the following sighted forward exchange: $A \to B, (@A, -) : m$, $B \to C, \uparrow (A, -) : m$. If we attempt to use a bulleted channel to translate the first instruction, as in `A*->B:B,m`, there is no bulleted equivalent to express the second one. On the other hand, if we provide a translation over a plain channel for the second instruction as the following, `B->C:{B,C,Msg}inv(sk(A))`, OFMC will deem the protocol non-runnable. In fact OFMC works at symbolic level, and the term `{B,C,Msg}inv(sk(A))` can be used in the second instruction only if it is has appeared earlier in the network, because only principal A can have created such term, being the only one knowing `inv(sk(A))`. Therefore OFMC rejects a protocol where the term `{B,C,Msg}inv(sk(A))` appears for the first time in an exchange originating from B. In other words, there is no way to bind the bulleted and the unbulleted channels in order to satisfy the expected goals for the forward modes. Hence in order to provide a coherent translation of all `AnBx` modes we need the plain channel as target in all cases.

Soundness. To verify the soundness of the implementation, we coded in OFMC-AnB all the `AnBx` primitives listed in §3, defining the expected security goals. The full list is given in Appendix (Table 13). Then we ran OFMC to verify the safety of each

protocol. We tested, successfully, all the resulting protocols with OFMC, in one and two sessions (both typed and untyped) with the classical mode. Most protocols were also verified in classical typed mode up to four sessions. We checked also the new fix-point module for unbounded number of sessions, but limited to secrecy and weak authenticity goals, which are the only goals currently supported by OFMC 2009c.

A note on compositionality. Having validated the implementation of each `AnBx` primitive against its expected goals does not in itself provide any validation guarantee on the implementation of a structured `AnBx` specification. In fact, composing the implementation protocols resulting from the translation of each `AnBx` step may, in principle, break the security guarantees provided by each of the component protocols: this is an instance of the well-known compositionality problems in security. While we do not have, as yet, a formal compositionality proof for our translations, we are confident that such result can be proved: in fact, the implementation protocols satisfy all the static conditions, such as the use of encryption keys from disjoint key-spaces that constitute the standard sufficient conditions for compositionality.

In addition, although compositionality is certainly an interesting and useful property, it does not represent a primary concern for our present endeavour. Our main interest is in making `AnBx` interoperable with existing verification tools like OFMC rather than in defining a new tool. Thus, as long as the security goals of a given `AnBx` protocol narration provide an adequate specification of the security properties we expect of the protocol, we may safely content ourselves with validating the AnB protocol resulting from our translation, rather than the `AnBx` specification itself.

A similar approach is gaining popularity in the literature on typed process calculi targeted at the specification of distributed protocols and systems (see, e.g., [24,25]). In these papers, the typed calculi provide for idealized specifications which are implemented by translations into low-level, but still typed, cryptographic languages. Rather than showing the translations sound, the process calculi specifications are shown secure by directly proving that *the result of the translations* are secure, just as we propose here: the difference is that in a typed calculus, security is by (well) typing, while in `AnBx` we prove security by model-checking.

5 A case study: e-payment systems

We show `AnBx` at work on the specification of systems for e-payment. We start by outlining a general e-payment scheme which captures the essential ingredients of most of the existing e-payment protocols, among which IBM's iKP [10,11], SET [26] and 3-D Secure [27] adopted by VISA.

Each principal has an initial knowledge shared with other participants. In particular, since most e-commerce protocols describe only the payment transaction, we assume

that customer and merchant have agreed on a *contract*, that includes an *order description* (*desc*) and a *price*. We also assume that payments are based on existing credit-card systems operated by an acquirer who shares with the customer the *customer's account number (can)* comprising the credit card number and the associated PIN. In summary, the initial knowledge of the parties is the following:

- Customer C: *price, desc, can*
- Merchant M: *price, desc*
- Acquirer A: *can*

To make each transaction univocally identified, the merchant generates a unique transaction ID (*tid*) and associates the transaction also with a *date* (or any appropriate time information). Both pieces of information must be communicated to the other parties. Summarizing, the transaction is characterized by the the tuple *(price, tid, date, can, desc)* which also constitutes the payment order information: if customer and merchant reach an agreement on this tuple, and they can prove their agreement to the acquirer, then the transaction can be completed successfully.

However, two security concerns arise here: on the one hand, customers typically wish to avoid leaking credit-card information to the merchant; on the other hand, customers and the merchant would not let the acquirer know the details of the order or the services involved in the transaction. Both these requirements can be enforced by protecting the exchange of *can* and *desc* with the digests we introduced in Section 2 and implemented as described in Table 6. The tuple that represent the *contract* among the three parties may thus be represented as $(price, tid, date, [can][desc])$, while *auth* is the transaction authorization result returned by the acquirer.

Revisiting iKP. The *i*KP protocol family {i=1,2,3} was developed at IBM Research [10,11,28] 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, and vary in the number of parties that own individual public key-pairs to generate digital signatures: this is reflected by the name of the different protocols – 1KP, 2KP, 3KP – which offer increasing levels of security. The structure of all the protocols can be specified by means of the `AnBx` messaging primitives as follows:

1. $C \rightarrow M, \eta_1 : [can : A], [desc : M]$
2. $C \leftarrow M, \eta_2 : price, tid, date, [contract]$
3. $C \rightarrow M, \eta_3 : price, tid, can, [can : A], [contract]$
4. $M \rightarrow A$
   (a) $M \rightarrow A, \eta_{4a} : price, tid, can, [can : A], [contract]$
   (b) $M \rightarrow A, \eta_{4b} : price, tid, date, [desc : M], [contract]$
5. $M \leftarrow A, \eta_5 : auth, tid, [contract]$
6. $C \leftarrow M, \eta_6 : auth, tid, [contract]$

The digests $[can]$ and $[desc]$ are annotated with their intended verifiers, A and M respectively (in 1KP $[desc]$ deserves some more care, as we will discuss shortly). Correspondingly, *contract* is the tuple $price, tid, date, [can : A], [desc : M]$ and its

Table 9 Exchange modes for the revisited iKP e-commerce protocol

mode/step	$\rightarrow$	1KP	2KP	3KP
η_1	$C \rightarrow M$	$(-,-)$	$(-,M)$	$(@C,M)$
η_2	$C \leftarrow M$	$(-,-)$	$(@M,-)$	$(@M,C)$
η_3	$C \rightarrow M$	$(-,A)$	$(-,A)$	(C,A)
η_{4a}	$M \rightarrow A$	$\uparrow(-,A)$	$\uparrow(-,A)$	$\uparrow(C,A)$
η_{4b}	$M \rightarrow A$	$(-,A)$	$(@M,A)$	$(@M,A)$
η_5	$M \leftarrow A$	$(@A,-)$	$(@A,M)$	$(@A,M)$
η_6	$C \leftarrow M$	$\uparrow(A,-)$	$\uparrow(A,-)$	$\uparrow(A,C)$
certified		A	M,A	C,M,A

digests can be left as [*contract*] as all the sensitive data is already protected against chosen plaintext attacks by the nested digests.

By instantiating the exchange modes η_i one may generate different versions of the protocol, achieving different security guarantees. In Table 9 we report the protocols resulting from the choice of the exchange modes η_i that offer the strongest level of security for a given number (i) of participants having public key-pairs at their disposal. The resulting protocols constitute the `AnBx` counterpart of the iKp protocol family.

Security Verification. We verified the `AnBx` specifications of {1,2,3}KP by compiling them into OFMC-AnB and running OFMC on the generated protocols against the strongest possible goals, for each of the protocols.
We also carried out a corresponding analysis of the original specifications, as reported in [11] and later amended in [29]. Below, we refer to this amended version as the "original" iKP, to be contrasted with the "revised", `AnBx` version discussed above. In both versions, we run our tests assuming that the acquirer is trusted (technically, modelled as a constant in the OFMC specification). This appears reasonable in an e-commerce application, since the acquirer is always certified. Furthermore, to compare the results of the analysis we treated the messages *common* and *contract*, in the original and in the revised version, respectively, as equivalent (indeed, they do provide the same abstraction conceptually).

On each protocol, we ran OFMC in classic mode with 1 and 2 sessions (typed and untyped): with 2 sessions we were sometimes unable to complete the test due to search space explosion. We also ran intensive tests limiting the depth of the search space to remain within the available memory space (2Gb). Below, we report on the results of such tests. For 3KP, the `AnBx` code for the revised and the original versions is shown in the Appendix (Tables 11 and 12).

Main results. In general, our implementation of the iKP protocols outperforms the original version, i.e. it satisfies more and stronger security goals, for all i's. This is largely due to the declarative nature of `AnBx` messaging primitives, which being defined as channel abstractions provide strong encapsulation mechanisms on the messages they exchange. One of the design goals of iKP was scalability, to support

Table 10 Security goal satisfied by Original and Revised iKP

	1KP		2KP		3KP	
Goal	O	R	O	R	O	R
can secret between *C,A*	-	-	-	-	+	+
$C \rightarrow *a : can$	+	+	+	+	+	+
A authenticates *C* on *can*	-	-	-	-	-	+
desc secret between *C,M*	+	+	+	+	+	+
Auth secret between *C,M,A*	-	-	-	-	-	+
M authenticates *A* on *Auth*	-	+	+*	+	+*	+
C authenticates *A* on *Auth*	-	+	+*	+	+*	+
tid secret between *C,M,A*	-	-	-	-	-	+
price secret between *C,M,A*	-	-	-	-	-	+
[*contract*] secret between *C,M,A*	-	-	-	-	-	+
a authenticates C on [*contract*]	-	-	-	+	+	+
a authenticates M on [*contract*]	-	-	-	+	+	+

* goal satisfied only fixing the definition of Sig_A

increasing security with the increase of the number of certified principals. In the original version, scalability is achieved by including extra components (e.g. signatures) in the messages exchanged in the different instances of the protocols. Conversely, the `AnBx` versions are naturally scalable, as the different instances of the protocol are simply encoded by tuning the exchange modes employed at the different steps (9). The only price to pay with respect to the original iKP is that we need to split step 4 in two substeps, plus one additional step to return the initiative to the merchant after step 4a. (Obviously the compiled AnB code involves more additional steps in order to achieve the freshness when required).

As we mentioned earlier, the `AnBx` specification are not just more scalable: they provide stronger security guarantees (cf. Table 13). During the analysis of the original 2KP and 3KP we found what, to the best of our knowledge, is a new flaw. It is related with the authenticity of the Authorization response (*auth*) that is generated by the acquirer and then sent to the other principals at step 5 and 6. The authenticity of the response is very important for the correct termination of the transaction because otherwise, in case of controversy customer and merchant could not rely on a robust proof of payment. The starred goals in Table 13 are met only after fixing this flaw that is in relation with how injective agreement is defined (and to its notion of authenticity). The change we propose is to add the identities of merchant and customer in Sig_A, adding the identities of merchant and customer (in 2KP this can be done with an ephemeral identity derived from the credit card number). Therefore in the original specification [11]: $Sig_A : \{hash(Auth, hash(Common))\}_{SK_A^{-1}}$ should be replaced by $Sig_A : \{hash(C, M, Auth, hash(Common))\}_{SK_A^{-1}}$.

Our revisited 2KP performs almost as good the original 3KP. The only goal not satisfied is *can* secret between *C,A*. This does not means that the credit card is leaked

but only that the credit card number is not strongly authenticated by the acquirer. In 3KP the credit card number can be signed by C, whereas this is not possible in 2KP and 1KP (neither in the original, nor in the revised versions). In this case, the acquirer weakly authenticates the customer by means of the credit card number, which is a shared secret among the two parties.

We mentioned earlier that some care must be taken in the revisited version of 1KP to compute the digest $[desc{:}M]$. In fact M is not certified, so we cannot calculate the digest verifiable by M (Table 6). Therefore we must adopt one of the possible alternatives and this requires a slight modification of the protocol. If we decide to use the unverifiable digest, as done by the original iKP, it is advisable to have M generate the digest rather than C, because *[can:A]* it already generated by C, and since the digest generation implies the usage of fresh keys as confounders, it is appropriate that those two values are generated by two different principals. In 2KP and 3KP, $[desc{:}M]$ is verifiable by M and this concern is not present. Finally, we notice that in 1KP the only information really protected is the credit card number (Goal $C \rightarrow * \, a : can$). Since this is one of the main concerns of all e-commerce user, this is indeed good news.

6 Related Work and Conclusion

The benefits of a programming discipline based on high-level abstractions for security are increasingly being recognized as important in the design of distributed systems. We believe the experience with `AnBx` we have reported in this paper provides further evidence of the advantages of the approach: as we have illustrated, using adequate abstractions results non only in simpler design, but also in more robust code and stronger security. In fact, being defined as channel-level abstractions, the `AnBx` primitives convey protection on *all* message components, yielding stronger encapsulation mechanisms, and consequently, stronger and more scalable security guarantees.

Our approach shares motivations and ideas with various papers on design and implementation of high-level process calculi for distributed protocols and systems [4,12,13,6,14]. While related in their initial motivations, these approaches differ significantly from our present endeavour in their technical development. In addition, the abstractions we discuss here capture more expressive security mechanisms such as those needed in the design of structured e-commerce protocols. Work more closely related to ours has been carried out in [15,16,2]. Guttman [15] has proposed a protocol design process organized around the authentication tests, a method for protocol verification based on the strand space theory. Guttman, Herzog, Ramsdell, and Sniffe [16] attached trust management assertions to protocol actions, constraining the behavior of a principal to be compatible with its own trust policy, and proposed the CPPL language for design and verification. However their language still make use of cryptographic primitives while we avoid any reference to explicit cryptography

Mödersheim and Viganò [2] described security protocols with three basic kinds of channels (authentic, confidential, secure). An Ideal and a Cryptographic Channel Model, describe the ideal functionality of the channels and the concrete cryptographic messages on insecure channels. The meaning of channels is defined as goals proving

that, under certain restrictions, composing individually secure protocols results in a secure protocol. We used their OFMC tool [1,2] to verify the `AnBx` compiled protocols. Our set of channel modes has a wider extension with respect to the OFMC bulleted channels and in §4 we showed that our forward modes (and their associated goals) can not specified only by means of the bulleted OFMC channels.

Future plans include work along several directions. Specifically, we are interested in developing a typed version of our primitives to provide static support for security, according to the design principles that are often referred to as *language-based security*. The resulting typed abstractions could then be integrated into a programming language. Meanwhile we are coding a Java library to be used to experiment with real applications. Also, it would be interesting to investigate the effectiveness of our approach in expressing other properties, such as anonymity and privacy, and work with the corresponding application domains such as e-cash and e-voting.

References

1. Basin, D., Mödersheim, S., Viganò, L.: OFMC: A symbolic model checker for security protocols. International Journal of Information Security **4**(3) (2005) 181–208
2. Mödersheim, S., Viganò, L.: The open-source fixed-point model checker for symbolic analysis of security protocols. In: Foundations of Security Analysis and Design V, Springer-Verlag (2009) 194
3. Dierks, T., Allen, C.: Rfc2246: The TLS protocol version 1.0. Internet RFCs (1999)
4. Abadi, M., Fournet, C., Gonthier, G.: Authentication primitives and their compilation. In: Proceedings of the 27th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, ACM New York, NY, USA (2000) 302–315
5. Bugliesi, M., Focardi, R.: Language Based Secure Communication. In: Computer Security Foundations Symposium, 2008. CSF'08. IEEE 21st. (2008) 3–16
6. Adao, P., Fournet, C.: Cryptographically Sound Implementations for Communicating Processes. Lecture Notes in Computer Science **4052** (2006) 83–94
7. Corin, R., Dénielou, P.M., Fournet, C., Bhargavan, K., Leifer, J.J.: Secure implementations of typed session abstractions. In: CSF 2007, IEEE (2007) 170–186
8. Bhargavan, K., Corin, R., Dénielou, P.M., Fournet, C., Leifer, J.J.: Cryptographic protocol synthesis and verification for multiparty sessions. In: CSF 2009. (2009)
9. Armando, A., Basin, D., Boichut, Y., Chevalier, Y., Compagna, L., Cuellar, J., Drielsma, P., Héam, P., Kouchnarenko, O., Mantovani, J., et al.: The AVISPA tool for the automated validation of internet security protocols and applications. In: CAV. Volume 5., Springer (2005) 281–285
10. Bellare, M., Garay, J., Hauser, R., Herzberg, A., Krawczyk, H., Steiner, M., Tsudik, G., Waidner, M.: iKP A Family of Secure Electronic Payment Protocols. In: Proceedings of the 1st USENIX Workshop on Electronic Commerce. (1995)
11. Bellare, M., Garay, J., Hauser, R., Herzberg, A., Krawczyk, H., Steiner, M., Tsudik, G., Van Herreweghen, E., Waidner, M.: Design, implementation, and deployment of the iKP secure electronic payment system. IEEE Journal on selected areas in communications **18**(4) (2000) 611–627
12. Abadi, M., Fournet, C., Gonthier, G.: Secure implementation of channel abstractions. Information and computation(Print) **174**(1) (2002) 37–83

13. Abadi, M., Fournet, C.: Private authentication. Theor. Comput. Sci. **322**(3) (2004) 427–476
14. Bugliesi, M., Giunti, M.: Secure implementations of typed channel abstractions. In Hofmann, M., Felleisen, M., eds.: POPL, ACM (2007) 251–262
15. Guttman, J.: Security protocol design via authentication tests. In: In Proceedings of 15th IEEE Computer Security Foundations Workshop. IEEE Computer. (2002)
16. Guttman, J., Herzog, J., Ramsdell, J., Sniffen, B.: Programming cryptographic protocols. Lecture notes in computer science **3705** (2005) 116
17. Maurer, U., Schmid, P.: A calculus for secure channel establishment in open networks. In: Computer Security-ESORICS 94: Third European Symposium on Research in Computer Security, Brighton, United Kingdom, November 7-9, 1994. Proceedings, Springer (1994) 175
18. van Doorn, L., Abadi, M., Burrows, M., Wobber, E.: Secure network objects. In: Secure Internet Programming. (1999) 395–412
19. Kelsey, J., Schneier, B., Wagner, D.: Protocol interactions and the chosen protocol attack. Lecture Notes in Computer Science (1998) 91–104
20. Lowe, G.: A hierarchy of authentication specifications, IEEE Computer Society Press (1997) 31–43
21. Cramer, R., Shoup, V.: Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. SIAM journal on computing(Print) **33**(1) (2004) 167–226
22. Abadi, M., Needham, R.: Prudent engineering practice for cryptographic protocols. In: 1994 IEEE Computer Society Symposium on Research in Security and Privacy, 1994. Proceedings. (1994) 122–136
23. Mödersheim, S.: Algebraic properties in alice and bob notation. Availability, Reliability and Security, International Conference on **0** (2009) 433–440
24. Backes, M., Hritcu, C., Maffei, M.: Type-checking zero-knowledge. In Ning, P., Syverson, P.F., Jha, S., eds.: ACM Conference on Computer and Communications Security, ACM (2008) 357–370
25. Corin, R., Deniélou, P.M., Fournet, C., Bhargavan, K., Leifer, J.J.: A secure compiler for session abstractions. Journal of Computer Security **16**(5) (2008) 573–636
26. Bella, G., Massacci, F., Paulson, L.: An overview of the verification of SET. International Journal of Information Security **4**(1) (2005) 17–28
27. Visa: Visa 3-D Secure Specifications. Technical report (2002)
28. O'Mahony, D., Peirce, M., Tewari, H.: Electronic payment systems for e-commerce. Artech House Publishers (2001)
29. Ogata, K., Futatsugi, K.: Formal analysis of the iKP electronic payment protocols. Lecture notes in computer science (2003) 441–460

Appendix

Revised iKP narration

The ideal narration of the revised iKP protocol given in Section 5 is, basically, the following:

- Step 1 and 2: customer and merchant exchange data that let them build independently the *contract*. They have to tell their "own version of the story" to the acquirer A. Customer C declares $[can : C]$, the credit card is going to use in *contract*, sending the protected digest of *can* to the merchant M. The customer also informs (and agrees with) the merchant on the digest of *desc* they will use to define the *contract*. The merchant M generates a (fresh) transaction ID (*tid*) and the *date* of transaction (C and M already had agreed on *price* and *desc*, being part of their initial knowledge). C can verify the integrity of $[desc : M]$, form the *contract*, and then compute its digest. The tuple $(price, tid, date, [contract])$ is sent to C which, upon receiving it, can compute $[contract]$ and verify that it matches the digest provided by M. If the match succeeds the protocol execution continues, otherwise stops.
- Step 3: the customer prepares a secret message for the acquirer containing the information necessary to complete the transaction: *contract*, credit card number (*can*), amount of the transaction (*price*) and transaction ID *(tid)*. However this message is sent to the merchant and not to the acquirer, because this protocols does not allow direct interaction between customers and acquirer, but only through merchant mediation. We assume that the merchant M is cooperating in delivering messages. Hence M receives an opaque message, and the only thing can do is to blindly forward it to the acquirer A (step 4a).
- Step 4: the merchant M sends the tuple $price, tid, date, [desc : M], [contract]$ to the acquirer (step 4b). This information is necessary to complete the payment. In particular *date* and $[desc : M]$ are required by the acquirer to compute independently the digest of *contract*. Upon reception of the two version of $[contract]$ originating from the two other principals, the acquirer can also compute the same value autonomously. If all three match together, the transaction can be authorized, since this is the proof of the complete agreement between the customer and the merchant.
- Step 5: the acquirer A sends the authorization response to the merchant, within a fresh authentic message, containing also *tid* and $[contract]$. This is done in order to bind all those information, and produce a proof that the payment has been authorized for that specific transaction and that specific contract.
- Step 6: the merchant M forwards the message received from the acquirer A to the customer C. This is a notification of the result of the transaction. In this way C receives, via the merchant, a proof of payment from the acquirer. Since the message is signed by the acquirer, the merchant can not alter the message without to be discovered.

Table 11 AnBx specification of revised 3KP

```
Protocol: Revised_3KP
Types:
  Agent C,Me,a;
  Certified C,Me,a;
  Number TID,Auth,Desc,Price;
  Function can
Definitions:
  Contract: Price,TID,dig(can(C),a),dig(Desc,Me)
Knowledge:
  C: C,Me,a,can(C);
  Me: C,Me,a;
  a: C,Me,a
Actions:
  # 0. Setup/Initial Knowledge
  C *->*Me: Price,Desc
  Me -> C: empty
  C -> Me,(@C,Me): dig(can(C),a),dig(Desc,Me)
  Me -> C,(@Me,C): TID,dig(Contract)
  C -> Me,(C,a): Price,TID,can(C),dig(can(C),a),dig(Contract)
  Me -> a,^(C,a): Price,TID,can(C),dig(can(C),a),dig(Contract)
  a -> Me: empty
  Me -> a,(@Me,a): Price,TID,dig(Desc,Me),dig(Contract)
  a  -> Me,(@a,Me): C,Auth,TID,dig(Contract)
  Me -> C,^(a,C): C,Auth,TID,dig(Contract)
Goals:
  can(C) secret between C,a
  a authenticates C on can(C)
  Desc secret between C,Me
  Auth secret between C,Me,a
  Me authenticates a on Auth
  C authenticates a on Auth
  TID secret between C,Me,a
  Price secret between C,Me,a
  dig(Contract) secret between C,Me,a
```

Table 12 Portion of the AnBx specification of original 3KP

```
Protocol: Original_3KP
Types:
  Agent C,Me,A;
  Certified C,Me,A;
  Number TID,Auth,empty,Desc,Price,ID,SALTC,SALTMe,V,VC,NONCE,RC;
  Function pk,sk,hash,hmac,can
Definitions:
  IDC: hmac(RC,can(C));
  Common: Price,ID,TID,NONCE,hmac(RC,can(C)),hmac(SALTC,Desc),hash(V),hash(VC);
  Clear: ID,TID,NONCE,hash(Common),hash(V),hash(VC);
  Slip: Price,hash(Common),can(C),RC,SALTMe;
  EncSlip: {Slip}pk(A);
  SigMe: {hash(Common),EncSlip}inv(sk(Me));
  SigC: {hash(EncSlip,hash(Common))}inv(pk(C));
  SigA: {C,Me,hash(Auth,hash(Common))}inv(sk(A))
Knowledge: [...]
Actions:
  C *->* Me: Price,Desc
  Me -> C: empty
  C -> Me: SALTC,IDC
  Me -> C: Clear,SigMe
  C -> Me: EncSlip,SigC
  Me -> A: Clear,hmac(SALTC,Desc),EncSlip,SigMe,SigC
  A -> Me: Auth, SigA
  Me -> C: Auth,SigA,V,VC
Goals: [...]
```

Table 13 Map of `AnBx` communication and forward modes

exchange	*mode*	*OFMC goals*
Communication Modes - Tables 1,4		
$A \to B, (-,-) : Msg$	plain	-
$A \to B, (A,-) : Msg$	from A	`B weakly authenticates A on Msg`
$A \to B, (@A,-) : Msg$	fresh from A	`B authenticates A on Msg`
$A \to B, (-,B) : Msg$	secret for B	`A->*B: Msg`
$A \to B, (A,B) : Msg$	from A, secret for B	`B weakly authenticates A on Msg` `Msg secret between A,B`
$A \to B, (@A,B) : Msg$	fresh from A, secret for B	`B authenticates A on Msg` `Msg secret between A,B`
Forward Modes - Tables 2,5		
$A \to B, (-,C) : Msg$ $B \to C, \uparrow (-,C) : Msg$	blind	`A->*C: Msg`
$A \to B, (A,C) : Msg$ $B \to C, \uparrow (A,C) : Msg$	blind	`Msg secret between A,C` `C weakly authenticates A on Msg`
$A \to B, (A,-) : C, Msg$ $B \to C, \uparrow (A,-) : C, Msg$	sighted	`B weakly authenticates A on Msg` `C weakly authenticates A on Msg`

$A \to B, (A,B) : C, Msg$
$B \to C, \uparrow (A,-) : C, Msg$
	sighted	`B weakly authenticates A on Msg`
		`C weakly authenticates A on Msg`
		`A->*B: Msg`

$A \to B, (@A,-) : C, Msg$ $B \to C, \uparrow (A,-) : C, Msg$	sighted	`B authenticates A on Msg` `C weakly authenticates A on Msg`

$A \to B, (@A,B) : C, Msg$
$B \to C, \uparrow (A,-) : C, Msg$
	sighted	`B authenticates A on Msg`
		`C weakly authenticates A on Msg`
		`A->*B: Msg`

$A \to B, (A,-) : C, Msg$ $B \to C, \uparrow (A,C) : C, Msg$	sighted secret	`B weakly authenticates A on Msg` `C weakly authenticates A on Msg`
$A \to B, (@A,-) : C, Msg$ $B \to C, \uparrow (A,C) : C, Msg$	sighted secret	`B authenticates A on Msg` `C weakly authenticates A on Msg`

$A \to B, (A,B) : C, Msg$
$B \to C, \uparrow (A,C) : C, Msg$
	sighted
	secret
		`B weakly authenticates A on Msg`
		`C weakly authenticates A on Msg`
		`Msg secret between A,B,C`

$A \to B, (@A,B) : C, Msg$
$B \to C, \uparrow (A,C) : C, Msg$
	sighted
	secret
		`B authenticates A on Msg`
		`C weakly authenticates A on Msg`
		`Msg secret between A,B,C`