Design and analysis of distributed protocols with AnBx

Michele Bugliesi, Paolo Modesti

Dipartimento di Informatica, Università Ca’ Foscari di Venezia
Via Torino 155, 30172 Mestre–Venezia, Italy
Design and analysis of distributed protocols with AnBx

Michele Bugliesi and Paolo Modesti

Università Ca’ Foscari Venezia,
Dipartimento di Informatica,
Via Torino 155, 30172 Mestre (VE), Italy
E-mail: {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 interactions. As a consequence, the high-level security properties of a protocol typically must 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 extension of the Alice & Bob notation for protocol narrations (AnBx) to serve as a specification language for a purely declarative modelling of distributed protocols. AnBx is built around a set of communication and data abstractions which provide primitive support for the high-level security guarantees required in the design of distributed protocols, and help shield the specification from the details of the underlying cryptographic infrastructure. AnBx is implemented on top of the OFMC [7, 31] verification tool, by means of a translation to the AnB language supported by OFMC. As a result, AnBx serves not only for specification and design, but also as a powerful tool for the security analysis of distributed protocols. We demonstrate the practical effectiveness of our approach with the specification and analysis of two real-life e-payment protocols: iKP [12, 11] and SET [8, 9, 10]. As reported in the paper, the declarative nature of the AnBx abstractions pays off, and results in protocol specifications with stronger, and more scalable security guarantees than 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 these kind of applications, as agreeing on the terms of a transaction in distributed and open environments 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 [21] 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, 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., [2, 15, 5]) have advocated a programming discipline based on (i) high-level security abstractions and mechanisms for composing them to support structured interaction patterns [18, 13], and (ii) automatic techniques to build defensive implementations on top of well-established cryptographic infrastructures and tools.

Following this line of research, 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. AnBx is built around a set of communication and data abstractions which provide primitive support for the high-level security guarantees required in the design of distributed protocols, without any explicit reference to the details of the underlying cryptographic infrastructure. In addition to the channel abstractions for secrecy and integrity found in related formalisms [22, 23, 15, 31], AnBx provides new forward modes to support interaction patterns based on message relaying such as those available in multi-party protocols for e-commerce and e-payment. The AnBx abstractions may be implemented by means of a translation into corresponding public-key cryptographic protocols described by standard AnB narrations. As a result, AnBx serves not only for specification and design, but also as a powerful tool for the security analysis of distributed protocols.

Main contributions and results. We implemented AnBx on top of the OFMC [7, 31] verification tool, by a translation to the AnB language supported by OFMC, and we verified the soundness of our implementation with OFMC itself. We then experimented with the implementation by employing AnBx as a tool for specification and analysis on two real-life e-payment protocols: the iKP e-payment protocol family (Internet Keyed Payment Protocol [12, 11]), and the SET purchase protocol (Secure Electronic Transaction [8, 9, 10]). As we report in the paper, the results demonstrate the practical effectiveness of our approach. In fact, the declarative nature of the AnBx abstractions pays off, and results in protocol specifications with stronger, and more scalable security guarantees than those offered by the original protocols. In both cases, our declarative specifications turned out to be immune to the known flaws of the original versions of the protocols. In addition, 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. These are 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 and SET specification, yielding stronger encapsulation mechanisms, and consequently, stronger and more scalable security guarantees.

The present paper revises and extends our previous work in [17] by including in AnBx a new class of forward modes providing strong authenticity (i.e. freshness) guarantees for message relaying. The new modes integrate smoothly in the existing AnBx design, and contribute significantly to increase the expressive power of the existing abstractions. Specifically, the new modes provide a very effective tool to enforce non-repudiation in the presence of message-relaying, as we show by proposing an amended version of the SET purchase protocol. The new design fixes a known weakness of the original protocol specification, namely the absence of reliable evidence for non-repudiation of the transaction by all parties involved. By relying on the newly introduced primitives for message relaying, we can achieve non-repudiation, allowing the customer to gain evidence that the payment has been authorized by the acquirer.

Plan of the paper. Section 2 introduces the AnBx specification language and its our high-level security abstractions. Section 3 outlines a translation of the abstractions into a low-level, cryptographic language. In Section 4 we present the AnBx compiler that implements that translation on top of OFMC. Section 5 introduces a model for generic e-commerce protocols, that will be compared with the two case studies, iKP and SET, which are developed in Section 6 and Section 7 respectively. Section 8 concludes the presentation. A separate Appendix collects the AnBx scripts employed in the case studies.
<table>
<thead>
<tr>
<th>(\eta)</th>
<th>mode</th>
<th>semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>((\cdot, \cdot))</td>
<td>plain</td>
<td>conveys no security guarantee</td>
</tr>
<tr>
<td>((A, \cdot))</td>
<td>from (A)</td>
<td>a public, but authentic exchange which provides the receiver with a guarantee on the origin of the message</td>
</tr>
<tr>
<td>(@(A, \cdot))</td>
<td>fresh from (A)</td>
<td>((A, \cdot)) with the additional guarantee that the message is fresh, and may not be replayed</td>
</tr>
<tr>
<td>((\cdot, x))</td>
<td>secret for (x)</td>
<td>a secret transmission, providing guarantees that only the intended receiver (x) will be exposed to the message payload: an intruder may become aware of the existence of an output, but not of the message contents; (x) can be any principal, (B) included</td>
</tr>
<tr>
<td>((A, x))</td>
<td>from (A), secret for (x)</td>
<td>combines the guarantees of modes ((A, \cdot)) and ((\cdot, x))</td>
</tr>
<tr>
<td>(@(A, B))</td>
<td>fresh from (A), secret for (B)</td>
<td>combines the guarantees of modes (@(A, \cdot)) and ((\cdot, B)); when freshness is combined with secrecy, the only principal that can be specified in the destination field of (\eta) is (B) (freshness requires cooperation of the two end-points)</td>
</tr>
<tr>
<td>↑(\eta_1)</td>
<td>forward</td>
<td>conveys some of the guarantees given by (\eta_1) and signals that the message did not originate at the sender’s; (\eta_1) can be a forward mode itself, under some consistency rules</td>
</tr>
</tbody>
</table>

Table 1: AnBx communication modes - \(A \rightarrow B, \eta : m\)

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:

\[
A : \alpha \quad \alpha \text{ is a local action performed by principal } A
\]

\[
i A \rightarrow B, \eta : m \quad A \text{ sends a message } m \text{ to } B \text{ in mode } \eta
\]

(symmetrically \(\leftarrow\) receives from)

\[
i \text{ is an optional label naming a protocol step}
\]

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 is available to the principal. In the exchange statements \(i. A \rightarrow B, \eta : m, i\) is an index labelling a protocol step: the mode \(\eta\) 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 \(m\) without leaking it. We assume digests to be resistant to chosen-plaintext attacks,
A → B, η₀ : m  B → C, η₁ : m  mode  semantics

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(−, C)</td>
<td>↑ (−, C)</td>
<td>blind</td>
<td>- Secrecy for C is preserved</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- B is not exposed to m</td>
</tr>
<tr>
<td>(A, C)</td>
<td>↑ (A, C)</td>
<td>blind</td>
<td>- Authenticity from A is preserved</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Secrecy for C is preserved</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- B is not exposed to m</td>
</tr>
<tr>
<td>(A, −)</td>
<td>↑ (A, −)</td>
<td>sighted</td>
<td>- Authenticity from A is preserved</td>
</tr>
<tr>
<td>(A, B)</td>
<td></td>
<td></td>
<td>- Freshness (if present) is lost</td>
</tr>
<tr>
<td>(＠A, −)</td>
<td>↑ (A, −)</td>
<td>sighted</td>
<td>- B is exposed to m</td>
</tr>
<tr>
<td>(A, C)</td>
<td></td>
<td></td>
<td>as above</td>
</tr>
<tr>
<td>(＠A, B)</td>
<td>↑ (A, C)</td>
<td>secret</td>
<td>- Authenticity from A is preserved</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Freshness (if present) is lost</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- B is exposed to m</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- All exchanges are secret</td>
</tr>
</tbody>
</table>

Table 2: Examples of AnBx forward modes

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 principal 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 [ ˜m : B].

**Communication modes.** Following [28, 34], 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, dest) 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.

Note that there is a fundamental difference between fresh and non-fresh modes. Authenticity and secrecy are properties bounded to data. It is possible to verify several times a digital signature or the correspondence, given a key, of a ciphertext to a plaintext. Freshness is instead an “ephemeral” property that can be verified once, and only by the recipient of the message. Therefore when combining freshness and secrecy (fresh from, secret for mode) the only target of the exchange can be the intended recipient.

**Forward modes.** The forward modes are useful in the specification of message relaying in multi-party exchanges. For example the statement B → C, ↑ η₁ : m expresses the intention of B to forward a message m to C. The statement is legal in case B has been the target of a previous exchange A → B, η₀ : m, where m is the same in both statements (or else m = ↑i where i is the label identifying the A → B exchange). When this is the case, the mode η₁ used in the forward statement will preserve, and possibly extend, some of the security properties specified by η₀.

The legal combinations of communication and forward modes involving three principals are shown in Table 2.

It is possible to forward a message previously forwarded, provided that some of the security properties specified in η₀, the communication mode of the originating exchange, are preserved. Note that the originating and the forwarded exchanges can be consecutive or not. To keep the notation simple, ↑ η is used as an abbreviation for ↑↑ η.

One may wonder if it is really necessary to use explicitly the symbol ↑ to denote the forward modes, as in some cases the forward operation can be understood from the context. Let’s consider the blind forward of a secret message: if we omit ↑ : A → B, (−, C) : m , B → C, (−, C) : m,
A → B₁, η₀ : m Bᵢ → Bᵢ₊₁, ηᵢ : m mode semantics

<table>
<thead>
<tr>
<th>Mode</th>
<th>Semantics</th>
</tr>
</thead>
</table>
| (−, Bₙ) ↑ (−, Bₙ) blind | - Secrecy for Bₙ is preserved  
- Bᵢ is not exposed to m, for i < n |
| (A, Bₙ) ↑ (A, Bₙ) blind | - Authenticity from A is preserved  
- Secrecy for Bₙ is preserved  
- Bᵢ is not exposed to m, for i < n |
| (A, −) ↑ (A, −) or (A, Bᵢ₊₁) sighted | - Authenticity from A is preserved  
- Freshness (if present) is lost  
- All Bᵢ are exposed to m  
- Some exchanges in the chain are not secret:  
  ∃ j such that ηᵢ = (A, −) j ∈ [0, n − 1]  
  or η₀ = (@A, −) |
| (A, B) ↑ (A, Bᵢ₊₁) sighted | - Authenticity from A is preserved  
- Freshness (if present) is lost  
- All Bᵢ are exposed to m  
- All exchanges are secret |

Table 3: AnBx forward modes - i ∈ [1, n − 1]

looking at the second exchange in isolation, it seems that B is composing a (standard) secret message for C. Someone reading the protocol may conclude that B knows m. But what we are really expressing with a blind forward is the fact that B is not exposed to m, since the originator A composes a secret message for C. B just delivers to C what she has received from A without being aware of the content of the message (in fact B cannot decrypt a message secret for C). Therefore omitting ↑ can undermine the readability of the protocol in such cases; hence we prefer to use ↑ always explicitly.

In general, 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 combinations of η₀ and ηᵢ are just those in Table 3;
- unless η is a forward mode, in A → B, η : m η may not specify a source other than A;
- 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 4, 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 underlying PKI supports dual key pairs, for
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 P\_KA\^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 \text{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
\(m, \text{sig}_A(m)\) \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 | fresh half keys generated during a discrete logarithm based agreement protocol (e.g. Diffie-Hellman), where p is prime and g is a primitive root \(mod\ p\) (\(mod\ p\) is omitted in the notation)
g\^y | fresh symmetric key, computed using a discrete logarithm based agreement protocol
H\text{mac}_K(m) | HMAC of message m, computed with the key K
A : \alpha | action \(\alpha\) performed by principal A
A \rightarrow B : m | A sends a message m to principal B (symmetrically \(\leftarrow\))

| Table 4: AnB-like Intermediate Syntax Notation |

encryption and digital signatures [25]. Each AnBx principal may thus possess up to two pairs of certified encryption keys: if it does possess 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 us to use real cryptographic protocols for a concrete implementation. Correctness of these solutions is discussed in Section 4.

### 3.1 Communication Modes

Table 5 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 [26] 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. The most relevant points in our translation are:
Table 5: Exchange Modes Translation of $A \rightarrow B, \eta : m$

<table>
<thead>
<tr>
<th>$\eta$</th>
<th>mode</th>
<th>translation</th>
<th>certified</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(-, -)$</td>
<td>plain</td>
<td>$A \rightarrow B : m$</td>
<td>-</td>
</tr>
<tr>
<td>$(A{\tilde{v}}, -)$</td>
<td>from $A$</td>
<td>$A \rightarrow B : S_A(\tilde{v}, m)$</td>
<td>$A$</td>
</tr>
<tr>
<td>$(-, x)$</td>
<td>secret for $x$</td>
<td>$A : \text{new } K$</td>
<td></td>
</tr>
<tr>
<td>$(A{\tilde{v}} , x)$</td>
<td>from $A$, secret for $x$</td>
<td>$A \rightarrow B : {K}_{PK_A}, {S_A(\tilde{v}, m)}_K$</td>
<td></td>
</tr>
<tr>
<td>$(@A{\tilde{v}}, -)$</td>
<td>fresh from $A$</td>
<td>$A \rightarrow B : {K}_{PK_A}, {S_A(\tilde{v}, m)}_K$</td>
<td></td>
</tr>
<tr>
<td>$(@A{\tilde{v}}, -)$</td>
<td>with DH*</td>
<td>$A \rightarrow B : g^x$</td>
<td></td>
</tr>
<tr>
<td>$(@A{\tilde{v}}, B)$</td>
<td>fresh from $A$, secret for $B$</td>
<td>$A \rightarrow B : {n_B, B}_{PK_A}$</td>
<td></td>
</tr>
</tbody>
</table>

$\tilde{v}$: list of intended verifiers
* this translation is used only when the intended recipient $B$ is not certified, and therefore the standard fresh from $A$ can not be applied

- in authentic exchanges the list of the identities of the intended verifiers $\tilde{v}$ is included to comply with the definition of the Lowe’s non-injective agreement, (injective for fresh exchanges) [26]. Therefore the compiler can be instructed enriching the source field with an additional optional parameter verifiers;
- in secret exchanges we use a 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 [20];
- when combining authenticity and secrecy, we first sign and then encrypt [4];
- 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 introduces more complexity;
- in fresh from exchanges, when the intended recipient is not certified, we use a combination of challenge-response and Diffie-Hellman key agreement protocol.

### 3.2 Forward Modes

Table 6 outlines the translation of the forward modes involving three principals. Each clause gives the translation of the statement $B \rightarrow C, \eta_1 : m$ with reference to the originating statement $A \rightarrow B, \eta_0 : m$. We only give the translation for the legal combinations of $\eta_0$ and $\eta_1$: in all other cases the translation is undefined.

When $\eta_1$ does not 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 $m_1$ defined as in Table 6: we remark 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 \( A \rightarrow B, \eta_0 : m \).

The simplest case is the blind forward: the secrecy (to \( C \)) of the high-level message \( m \) implies that the low-level message \( m_0 \) is opaque to \( B \). Therefore the only action \( B \) can perform is to forward \( m_0 \) to \( C \) blindly.

The sighted forward mode deserves some more attention. As mentioned, the translation of an authentic exchange must include the identity of the intended verifiers to comply with the definition of the Lowe’s agreement [26]. However the statement \( A \rightarrow B, (A, -) : m \) can directly translated in \( A \rightarrow B : S_A(B, m) \), without the need to explicit the verifier \( B \) because the compiler can clearly understand that \( B \) is the target of the exchange.

If the same message is afterwards forwarded to another principal \( C \), how can \( C \) verify the authenticity of a message originating from \( A \)? According to the Lowe’s agreement definition, the identity of \( C \) must be included in the payload, hence \( A \) should compose the message specifying the list of the intended verifiers \( \{C, B\} \). The compiler can then be instructed in one sense, enriching the source field with the additional optional parameter verifiers (vers): \( \eta = (src \{vers\}, dest) \).

For example, writing \( A \rightarrow B, (A(C, B), B) : m \) means that \( A \) sends an authentic and secret message to \( B \) and the authenticity can be verified by \( B \) and \( C \). Its combination with a subsequent forward to \( C \) is translated as:

\[
\begin{align*}
\text{AnBx} & : A \rightarrow B, (A(C, B), B) : m & \text{AnB} & : A \rightarrow B : \{K\}_{PK_B} \cdot \{S_{A(C, B, m)}\}_{K_B} \\
B \rightarrow C, & (A(C, B), -) : m & A \rightarrow B : S_A(C, B, m)
\end{align*}
\]

Again, in the list of verifiers the identity of \( B \) can be omitted, since \( B \) is the target of the first exchange. Hence we can simply write \( A \rightarrow B, (A(C), -) : m \). Noting that \( A \rightarrow B, (A, -) : C, m \) compiles exactly in the same way as \( A \rightarrow B, (A(C), -) : m \), we conclude that the expression \( A \rightarrow B, (src, dest) : vers, m \) is equivalent to \( A \rightarrow B, (src\{vers\}, dest) : m \), provided that \( B \) is automatically included among the verifiers by the compiler.

This simplify the notation as we can see in the \( AnBx \) specification of the sighted secret forward mode. This mode requires the secrecy in both exchanges:

\[
\begin{align*}
\text{AnBx} & : A \rightarrow B, (A, B) : C, m & \text{AnB} & : A \rightarrow B : \{K_0\}_{PK_B} \cdot \{S_{A(C, B, m)}\}_{K_0} \\
B \rightarrow C, & (A, C) : C, m & A \rightarrow B : \{K_1\}_{PK_B} \cdot \{S_{A(C, B, m)}\}_{K_1}
\end{align*}
\]
Table 7: Message Formats Translation (digest formed by principal A)

<table>
<thead>
<tr>
<th>digest</th>
<th>translation</th>
<th>certified</th>
<th>notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>([m : B])</td>
<td>(A: \text{new } K) (A \rightarrow B: H_{\text{mac}}K(m), {K}_{PK_B})</td>
<td>(B)</td>
<td>Obfuscated verifiable digest</td>
</tr>
<tr>
<td>([m])</td>
<td>(\mathcal{H}(m))</td>
<td>(-)</td>
<td>Plain digest</td>
</tr>
</tbody>
</table>

The AnB structure is similar in the two exchanges. The authentic message is the same \(S_A(C, B, m)\), while the secrecy is achieved using two different symmetric keys \((K_0, K_1)\), which are encrypted with the public key of \(B\) and \(C\) respectively.

As stated in Section 2, the sighted secret forward mode can be used to enforce secrecy on the high-level message \(m\) among the three principals involved in this pattern.

### 3.3 Message Formats

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 (Table 7) is in the way the secret keys are made available to the legitimate principals, as the structure of the digest \([m]\) does not inform on whom 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 7 illustrates the translation scheme for digests based on these additional annotations also proposing alternatives when the digest cannot directly be 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.

The standard implementation of \([m : B]\) (obfuscated verifiable digest) is the following: a new key \(K\) is generated by the sender \(A\), \(H_{\text{mac}}K(m)\) is computed then sent to the verifier \(B\) together with the key \(K\) encrypted with the public key of the recipient. In this way the recipient is the only principal entitled to verify that the received \(H_{\text{mac}}K(m)\) was generated by a principal who knows \(m\), provided that the recipient also knows \(m\). Notice that this implementation does not prove anything about the identity of the originator of the \(H_{\text{mac}}K(m)\), so a proof of authenticity must necessarily be combined with one of the “from” modes presented in Table 5.

This solution works only if principal \(B\) is certified, and can be applied seamlessly because it requires a simple substitution of \([m : B]\) in AnBx with \(H_{\text{mac}}K(m), \{K\}_{PK_B}\) as subterm in the message payload.

If the designated verifier \(B\) is not certified, we propose a couple of alternatives:
A \rightarrow B, \eta_0 : m \quad B \rightarrow C, \eta_1 : m \quad \text{mode} \quad \text{semantics}

| (\eta_0, -) \ 
\begin{cases} 
\quad \text{(A, -)} \\
\quad \text{(A, B)} 
\end{cases} |
\quad \uparrow (\eta_0, -) 
\ 
\text{fresh} 
\ 
\text{sighted} 
\ 
\text{as above} 
\ 
\text{Authenticity from A is preserved} 
\ 
\text{Freshness is preserved} 
\ 
\text{B is exposed to m} 

| (\eta_0, C) \ 
\begin{cases} 
\quad \text{(A, -)} \\
\quad \text{(A, B)} 
\end{cases} |
\quad \uparrow (\eta_0, C) 
\ 
\text{fresh} 
\ 
\text{sighted} 
\ 
\text{secret} 
\ 
\text{as above} 
\ 
\text{Authenticity from A is preserved} 
\ 
\text{Freshness is preserved} 
\ 
\text{B is exposed to m} 
\ 
\text{All exchanges are secret}

Table 8: AnEx fresh forward modes

- **plain digest** \([m]\). It may be employed when chosen plaintext attacks may be tolerated as the secrecy of the message content is not required. In this case, the digest may directly be computed by the application of a hash function. For example if the protocol involves many uncertified principals, the security goals are so relaxed or weak that from a practical point of view, the resilience of the digest to chosen plaintext attacks is not the main concern;

- **obfuscated unverifiable digest**. It is built like the verifiable counterpart, but with the fundamental difference that the key \(K\) is not included in the transmitted message. This implies that the digest cannot be verified by anyone (except the digest’s creator \(A\)). This can be useful in some application were the same digest is disseminated among different principal that can, eventually, compare them. This is actually what the original iKP [11] protocol does.

An additional possibility is offered by the *digest verifiable by proxy* where verification is delegated to a (certified) trusted third party \(C\). Individually \(A\) and \(B\) can construct the digests (secret for \(C\)) \([m_A : C],[m_B : C]\), respectively, and then send these values to \(C\) (even as component of a wider message). \(p \rightarrow C : \{\mathcal{H}(m_p)\}^K_p, \{K_p\}^P_K\), where \(p \in \{A, B\}\). At a certain point \(C\) would verify and notify to any principal if the condition \(\mathcal{H}(m_A) = \mathcal{H}(m_B)\) is true or not. It should be noticed that the verifier \(C\) could run chosen plaintext attacks against \(\mathcal{H}(m)\), having a fundamental advantage with respect to any other principal. The protocol designer should be aware of it and evaluate if the application requirements can tolerate this.

In summary, when it is not possible to use the standard verifiable digest, the alternatives must be chosen according with the security goals the application wants to achieve. In general this will require some slight adaptation of the protocol, but we could also instruct the compiler to apply some strategy when the intended verifier is not certified. For example, instead of returning a error, it just can downgrade the digest mode from verifiable, to unverifiable, or even to plain (see Section 6 for an example of it), but as we mentioned earlier these alternatives should be carefully evaluated since they could allow an intruder to perform downgrade attacks.

### 3.4 Fresh Forward Modes

The fresh forward modes, those where \(\eta_1\) includes the freshness tag \(@\), require a translation involving more messages. An peculiar aspect, mentioned in Section 2, is that the freshness is an “ephemeral” property that can be verified once, and only by the recipient of the message.

An important design aspect is that we do not want to give to the first message \(A \rightarrow B, \eta_0 : m\) any special structure in order to accommodate the needs of the second message. For example, in the first message, we do not generate nonces that will be used in the second exchange. Thus principal \(A\) can compile the first exchange as usual with the standard translation rules given in Table 5. Obviously \(\eta_0\) can be only \((\eta_0, -)\) and \((\eta_0, B)\) because if the first message is not fresh, it does not make any sense to try to “forward” the freshness.
\[ \eta_0 \quad \eta_1 \quad \text{mode} \quad \text{translation of} \quad \text{translation of} \\
\text{(certified)} \quad A \rightarrow B : m_0 \quad B \rightarrow C, \eta_1 : m \]

(from Table 5)

\[
\begin{array}{llll}
(\@A, -) & \uparrow (\@A, -) & \text{fresh} & B \rightarrow C : S_A(C, B, m) \\
(\@A, B) & \quad & \text{sighted} & A \leftarrow C : \{n_C, C\}_{PK_A} \\
& & (A, C) \quad & A \rightarrow C : \{n_C, sig_A(C, B, msg)\}_{PK_C} \\
& & \text{or} & \\
& & (A, B, C) & \ni \\
\end{array}
\]

\[
\begin{array}{llll}
(\@A, -) & \uparrow (\@A, -) & \text{fresh} & B \rightarrow C : S_A(C, B, m) \\
& & \text{sighted} & A \leftarrow C : g^{x} \\
& & (A) \quad & A \rightarrow C : \{g^{y}, n_C, C\}_{PK_A} \\
& & \ni & \quad A \rightarrow C : \{n_C, sig_A(C, B, m)\}_{g^{xy}} \\
\end{array}
\]

\[
\begin{array}{llll}
(\@A, B) & \uparrow (\@A, C) & \text{fresh} & B \rightarrow C : \{S_A(C, B, m)\}_{K} \\
& & \text{sighted} & A \leftarrow C : \{n_C, C\}_{PK_A} \\
& & \{S_A(C, B, m)\} \ni & A \rightarrow C : \{n_C, K\}_{PK_C} \\
& & (A, B, C) & \\
\end{array}
\]

AnBx specification: \( A \rightarrow B, \eta_0 : m; B \rightarrow C, \eta_1 : m \)

* this translation is used only when the intended recipient \( C \) is not certified, and therefore the standard fresh forward can not be applied

Table 9: Fresh Forward Modes Translation

Another requirement, for efficiency reasons, is to avoid retransmission of information that is already possessed by the principals, in particular the payload \( m \). Hence in the fresh sighted forward, see Table 9, in the first step of the translation, \( B \) sends to \( C \) a message that is equal to sighted forward (without the freshness). The subsequent exchanges implement the challenge-response between \( A \) and \( C \), in analogy to what we have done for the corresponding fresh communication modes.

The nonce exchange is performed by mean of public key cryptography if both \( A \) and \( C \) are certified; otherwise if \( C \) is not certified, the nonce exchange is performed using a Diffie-Hellman key agreement. If a direct connection is available between \( A \) and \( C \) they communicate directly, otherwise they can communicate using \( B \) as a mediator, who will blindly forward the messages exchanged among them.

The fresh forward sighted mode can be seen as a standard forward sighted mode, followed by a nonce exchange between the originator and the final recipient in order to give evidence of the freshness.

In the case of the fresh forward secret mode the main difference is that \( S_A(C, B, msg) \) is forwarded from \( B \) to \( C \), encrypted with the fresh key \( K \), generated by \( A \) to compose the secret message for \( B \). If the challenge-response succeeds, \( C \) receives from \( A \) the key \( K \) necessary to unlock the encrypted message.

4 Implementing the translation: the AnBx compiler

We have implemented the AnBx translation previously 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.
Protocol: Fresh_From_A
Types:
Agent A,B;
Number Msg,N1;
Function pk,sk,hash
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

Table 10: AnB OFMC specification of “fresh from” mode

Protocol: Fresh_From_A
Types:
Agent A,B;
Certified A,B;
Number M1,M2;
Function id
Definitions
Msg: M1,M2
Knowledge:
A: A,B;
B: A,B
Actions:
A -> B,(@A, -): Msg
Goals:
B authenticates A on Msg

Table 11: AnEx specification of “fresh from” mode

4.1 An overview of OFMC

OFMC [7, 31]) 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 OFMC-AnB can be found in [30] and in the 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,...);
- **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 10, where we give the AnB narration of the low-level translation for the “fresh-from” AnEx 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.
4.2 AnBx and its compiler

AnBx is defined as a variant of OFMC-AnB that supports the messaging modes discussed in Section 2 as well as few additional features in the preamble sections. A sample AnBx specification is reported in Table 11, 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.

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 Section 3. Thus, while AnBx supports abstract messaging primitives for secrecy and authentication, the code generated by the compiler only involves standard, “plain” message exchanges, and relies on the cryptographic constructions of Section 3 to achieve the desired security guarantees. In addition to generating the AnB protocol steps, the compiler creates the entries in the preamble sections (Types and Knowledge) required to make a consistent OFMC-AnB specification. To illustrate, the AnB protocol in Table 10 is, in fact, the result of compiling the AnBx code in Table 11.

In section Types, every principal is modeled as an Agent. OFMC distinguishes between variable and constant agents, written in capital and small letters respectively. A new variable type Certified is introduced to list all the certified agents, i.e. principals who possess dual key pairs. The compilation process adds the private keys inv(pk(.)) and inv(sk(.)) to the knowledge of each agent (. is replaced by the agent’s name):

A new section named Definitions contains the variable declarations (macros) for messages. To be well-formed, the left side must include literals that have been already declared in the Types section or be themselves declarations preceding the current one. Therefore declarations that have dependencies must be written in an order that does never leave any declaration undefined. At compile time, the left side is replaced by the right side in any occurrence of Actions and Goals. In this example we say that Msg is composed by two message components Msg1 and Msg2, two variables of type Number.

Since our implementation presupposes the existence of a PKI (Section 3) this implies the availability of a set of functions and values which must be added to the Types section: functions pk, sk for encryption and signing and hash, hmac for the digests. The digests, for the moment, are modeled using hash and hmac functions or the generic dig function. The constant numbers g (the primitive root of the Diffie-Hellman agreement) and empty (used to model the empty message applied in messages whose aim is, typically, just transferring initiative from one principal to another), are added too. inv and exp are built-in functions of AnB and should not be declared explicitly. id (identity function) is declared here just because in AnB the function declaration cannot be empty, but it not used.

In Actions, we added the support of the new channel types as specified by AnBx (Section 2). Each AnBx statement is compiled to one or more actions in AnB, according to the translation scheme given in Section 3. The only target channel for AnBx channels is the plain channel, while the OFMC-AnB “bulleted” channels - secret ->*, authentic *-> and secure *--> - are left untouched. These channels guarantee the expected security properties without referring to a specific implementation.

If the compilation introduces new variables (nonces or fresh keys), they are added as Numbers in Types; the Goals sections accepts any goal expressed in AnB, including those containing terms declared in the Definitions section.

In summary, the AnBx compiler (anbxc) allows to write protocols in AnBx and translates them in the OFMC AnB variant. The AnBx language is an extension of AnB, i.e. any protocol written in AnB is a correct protocol in AnBx.

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 channel types (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:
Table 12: Map of AnBx communication modes

<table>
<thead>
<tr>
<th>exchange</th>
<th>mode</th>
<th>OFMC goals</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \rightarrow B, (-, -)$ : $Msg$</td>
<td>plain</td>
<td>-</td>
</tr>
<tr>
<td>$A \rightarrow B, (A, -)$ : $Msg$</td>
<td>from $A$</td>
<td>$B$ weakly authenticates $A$ on $Msg$</td>
</tr>
<tr>
<td>$A \rightarrow B, (@A, -)$ : $Msg$</td>
<td>fresh from $A$</td>
<td>$B$ authenticates $A$ on $Msg$</td>
</tr>
<tr>
<td>$A \rightarrow B, (-, B)$ : $Msg$</td>
<td>secret for $B$</td>
<td>$A \rightarrow B$: $Msg$</td>
</tr>
<tr>
<td>$A \rightarrow B, (A, B)$ : $Msg$</td>
<td>secret for $B$</td>
<td>$B$ weakly authenticates $A$ on $Msg$</td>
</tr>
<tr>
<td>$A \rightarrow B, (@A, B)$ : $Msg$</td>
<td>fresh from $A$, secret for $B$</td>
<td>$Msg$ secret between $A, B$</td>
</tr>
</tbody>
</table>

1. $A \rightarrow B, (@A, -)$ : $m$
2. $B \rightarrow C, \uparrow (A, -)$ : $m$

If we attempt to use a bulleted channel to translate the first instruction, as in $A \rightarrow B, B, m$, 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, $B \rightarrow C: \{B, C, Msg\}^{inv}(sk(A))$, OFMC will deem the protocol non-runnable.

In fact OFMC works at the symbolic level, and the term $\{B, C, Msg\}^{inv}(sk(A))$ can be used in the second instruction only if it has appeared earlier in the network, because only principal $A$ could have created such a 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 to use the plain channel as target in all cases.

4.3 Soundness

To verify the soundness of the implementation, we coded in OFMC-AnB all the AnBx abstractions protocols outlined Section 3, defining the expected security goals (see Tables 12 and 13), and 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 also checked the new fixpoint module for an unbounded number of sessions, but limited to secrecy and weak authenticity goals, which are the only goals currently supported by OFMC 2009c.

4.4 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 yet have 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
A → B, (−, C) : Msg
B → C, ↑ (−, C) : Msg

A → B, (A, C) : Msg
B → C, ↑ (A, C) : C, Msg

A → B, (A, −) : C, Msg
B → C, ↑ (A, −) : C, Msg

A → B, (A, B) : C, Msg
B → C, ↑ (A, C) : C, Msg

A → B, (A, −) : C, Msg
B → C, ↑ (A, C) : C, Msg

A → B, (@A, −) : C, Msg
B → C, ↑ (A, −) : C, Msg

A → B, (@A, B) : C, Msg
B → C, ↑ (A, C) : C, Msg

A → B, (A, −) : C, Msg
B → C, ↑ (A, C) : C, Msg

A → B, (@A, B) : C, Msg
B → C, ↑ (@A, C) : C, Msg

A → B, (@A, −) : C, Msg
B → C, ↑ (@A, −) : C, Msg

A → B, (@A, B) : C, Msg
B → C, ↑ (@A, −) : C, Msg

A → B, (@A, B) : C, Msg
B → C, ↑ (@A, C) : C, Msg

A → B, (@A, −) : C, Msg
B → C, ↑ (@A, C) : C, Msg

Table 13: Map of AnBx forward modes

<table>
<thead>
<tr>
<th>exchange</th>
<th>mode</th>
<th>OFMC goals</th>
</tr>
</thead>
<tbody>
<tr>
<td>A → B, (−, C) : Msg</td>
<td>blind</td>
<td>A→C: Mag</td>
</tr>
<tr>
<td>B → C, ↑ (−, C) : Msg</td>
<td></td>
<td>Msg secret between A,C</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C weakly authenticates A on Mag</td>
</tr>
<tr>
<td>A → B, (A, C) : Msg</td>
<td>blind</td>
<td>C weakly authenticates A on Mag</td>
</tr>
<tr>
<td>B → C, ↑ (A, C) : C, Msg</td>
<td></td>
<td>B weakly authenticates A on Msg</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C weakly authenticates A on Msg</td>
</tr>
<tr>
<td>A → B, (A, −) : C, Msg</td>
<td>sighted</td>
<td>B weakly authenticates A on Msg</td>
</tr>
<tr>
<td>B → C, ↑ (A, −) : C, Msg</td>
<td></td>
<td>C weakly authenticates A on Msg</td>
</tr>
<tr>
<td>A → B, (A, B) : C, Msg</td>
<td>sighted</td>
<td>B weakly authenticates A on Msg</td>
</tr>
<tr>
<td>B → C, ↑ (A, C) : C, Msg</td>
<td></td>
<td>C weakly authenticates A on Msg</td>
</tr>
<tr>
<td>A → B, (A, −) : C, Msg</td>
<td>sighted</td>
<td>B weakly authenticates A on Msg</td>
</tr>
<tr>
<td>B → C, ↑ (A, C) : C, Msg</td>
<td></td>
<td>C weakly authenticates A on Msg</td>
</tr>
<tr>
<td>A → B, (@A, −) : C, Msg</td>
<td>sighted</td>
<td>B weakly authenticates A on Msg</td>
</tr>
<tr>
<td>B → C, ↑ (A, −) : C, Msg</td>
<td></td>
<td>C weakly authenticates A on Msg</td>
</tr>
<tr>
<td>A → B, (@A, B) : C, Msg</td>
<td>sighted</td>
<td>B weakly authenticates A on Msg</td>
</tr>
<tr>
<td>B → C, ↑ (A, C) : C, Msg</td>
<td></td>
<td>C weakly authenticates A on Msg</td>
</tr>
<tr>
<td>A → B, (A, −) : C, Msg</td>
<td>sighted</td>
<td>B weakly authenticates A on Msg</td>
</tr>
<tr>
<td>B → C, ↑ (A, C) : C, Msg</td>
<td></td>
<td>C weakly authenticates A on Msg</td>
</tr>
<tr>
<td>A → B, (@A, B) : C, Msg</td>
<td>sighted</td>
<td>B weakly authenticates A on Msg</td>
</tr>
<tr>
<td>B → C, ↑ (@A, −) : C, Msg</td>
<td></td>
<td>C weakly authenticates A on Msg</td>
</tr>
<tr>
<td>A → B, (@A, B) : C, Msg</td>
<td>fresh</td>
<td>B authenticates A on Msg</td>
</tr>
<tr>
<td>B → C, ↑ (@A, C) : C, Msg</td>
<td>secret</td>
<td>C authenticates A on Msg</td>
</tr>
<tr>
<td>A → B, (@A, −) : C, Msg</td>
<td>fresh</td>
<td>B authenticates A on Msg</td>
</tr>
<tr>
<td>B → C, ↑ (@A, −) : C, Msg</td>
<td>sighted</td>
<td>C authenticates A on Msg</td>
</tr>
<tr>
<td>A → B, (@A, B) : C, Msg</td>
<td>fresh</td>
<td>B authenticates A on Msg</td>
</tr>
<tr>
<td>B → C, ↑ (@A, −) : C, Msg</td>
<td>sighted</td>
<td>C authenticates A on Msg</td>
</tr>
<tr>
<td>A → B, (@A, B) : C, Msg</td>
<td>fresh</td>
<td>B authenticates A on Msg</td>
</tr>
<tr>
<td>B → C, ↑ (@A, C) : C, Msg</td>
<td>secret</td>
<td>C authenticates A on Msg</td>
</tr>
<tr>
<td>A → B, (@A, B) : C, Msg</td>
<td>fresh</td>
<td>B authenticates A on Msg</td>
</tr>
<tr>
<td>B → C, ↑ (@A, −) : C, Msg</td>
<td>sighted</td>
<td>C authenticates A on Msg</td>
</tr>
<tr>
<td>A → B, (@A, B) : C, Msg</td>
<td>fresh</td>
<td>B authenticates A on Msg</td>
</tr>
<tr>
<td>B → C, ↑ (@A, C) : C, Msg</td>
<td>secret</td>
<td>C authenticates A on Msg</td>
</tr>
</tbody>
</table>

Table 13: Map of AnBx forward modes

<table>
<thead>
<tr>
<th>exchange</th>
<th>mode</th>
<th>OFMC goals</th>
</tr>
</thead>
<tbody>
<tr>
<td>A → B, (−, C) : Msg</td>
<td>blind</td>
<td>A→C: Mag</td>
</tr>
<tr>
<td>B → C, ↑ (−, C) : Msg</td>
<td></td>
<td>Msg secret between A,C</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C weakly authenticates A on Mag</td>
</tr>
<tr>
<td>A → B, (A, C) : Msg</td>
<td>blind</td>
<td>C weakly authenticates A on Mag</td>
</tr>
<tr>
<td>B → C, ↑ (A, C) : C, Msg</td>
<td></td>
<td>B weakly authenticates A on Msg</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C weakly authenticates A on Msg</td>
</tr>
<tr>
<td>A → B, (A, −) : C, Msg</td>
<td>sighted</td>
<td>B weakly authenticates A on Msg</td>
</tr>
<tr>
<td>B → C, ↑ (A, −) : C, Msg</td>
<td></td>
<td>C weakly authenticates A on Msg</td>
</tr>
<tr>
<td>A → B, (A, B) : C, Msg</td>
<td>sighted</td>
<td>B weakly authenticates A on Msg</td>
</tr>
<tr>
<td>B → C, ↑ (A, C) : C, Msg</td>
<td></td>
<td>C weakly authenticates A on Msg</td>
</tr>
<tr>
<td>A → B, (A, −) : C, Msg</td>
<td>sighted</td>
<td>B weakly authenticates A on Msg</td>
</tr>
<tr>
<td>B → C, ↑ (A, C) : C, Msg</td>
<td></td>
<td>C weakly authenticates A on Msg</td>
</tr>
<tr>
<td>A → B, (@A, −) : C, Msg</td>
<td>sighted</td>
<td>B weakly authenticates A on Msg</td>
</tr>
<tr>
<td>B → C, ↑ (A, −) : C, Msg</td>
<td></td>
<td>C weakly authenticates A on Msg</td>
</tr>
<tr>
<td>A → B, (@A, B) : C, Msg</td>
<td>sighted</td>
<td>B weakly authenticates A on Msg</td>
</tr>
<tr>
<td>B → C, ↑ (A, C) : C, Msg</td>
<td></td>
<td>C weakly authenticates A on Msg</td>
</tr>
<tr>
<td>A → B, (A, −) : C, Msg</td>
<td>sighted</td>
<td>B weakly authenticates A on Msg</td>
</tr>
<tr>
<td>B → C, ↑ (A, C) : C, Msg</td>
<td></td>
<td>C weakly authenticates A on Msg</td>
</tr>
<tr>
<td>A → B, (@A, B) : C, Msg</td>
<td>fresh</td>
<td>B authenticates A on Msg</td>
</tr>
<tr>
<td>B → C, ↑ (@A, −) : C, Msg</td>
<td>secret</td>
<td>C authenticates A on Msg</td>
</tr>
<tr>
<td>A → B, (@A, −) : C, Msg</td>
<td>fresh</td>
<td>B authenticates A on Msg</td>
</tr>
<tr>
<td>B → C, ↑ (@A, −) : C, Msg</td>
<td>sighted</td>
<td>C authenticates A on Msg</td>
</tr>
<tr>
<td>A → B, (@A, B) : C, Msg</td>
<td>fresh</td>
<td>B authenticates A on Msg</td>
</tr>
<tr>
<td>B → C, ↑ (@A, C) : C, Msg</td>
<td>secret</td>
<td>C authenticates A on Msg</td>
</tr>
<tr>
<td>A → B, (@A, B) : C, Msg</td>
<td>fresh</td>
<td>B authenticates A on Msg</td>
</tr>
<tr>
<td>B → C, ↑ (@A, −) : C, Msg</td>
<td>sighted</td>
<td>C authenticates A on Msg</td>
</tr>
<tr>
<td>A → B, (@A, B) : C, Msg</td>
<td>fresh</td>
<td>B authenticates A on Msg</td>
</tr>
<tr>
<td>B → C, ↑ (@A, C) : C, Msg</td>
<td>secret</td>
<td>C authenticates A on Msg</td>
</tr>
</tbody>
</table>

Table 13: Map of AnBx forward modes
interoperable with existing verification tools like OFMC rather than in defining a new tool. Thus, as long as the security goals of a given \textsc{AnBx} protocol narration provide an adequate specification of the security properties we expect of the protocol, we may safely content ourselves with validating the \textsc{AnB} protocol resulting from our translation, rather than the \textsc{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., \cite{6, 19}). 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 \textit{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 \textsc{AnBx} we prove security by model-checking.

5 Design and Analysis of e-commerce protocols

We show \textsc{AnBx} at work on the specification of e-payment systems. In general their design and specification is complex and their analysis challenging. 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 \textsc{iKP} \cite{11, 12}, \textsc{SET} \cite{8, 9, 10} and \textsc{3-D Secure} \cite{36} adopted by VISA.

Such scheme helps the designer to focus on the business logic of the protocol, specifying a protocol template in which the security properties enforced at every exchange are parametrized. The instantiation of these parameters, by means of the \textsc{AnBx} modes, produces a concrete version of the protocol.

Inspired by such idea, the first case study (\textsc{iKP}) gives a sample of the \textsc{AnBx} design philosophy showing in particular that one of the key features of \textsc{iKP} - the increasing levels of security according to the number of certified principals - can be achieved in a very elegant and effective way. As a byproduct we found a new flaw in the original \textsc{iKP} specification and propose a fix.

The second case study illustrates a revised version of \textsc{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 \textsc{SET} and show that our revised versions are immune to such defects. Moreover we outline some weaknesses of the dual signature, and give directions on how to address them in \textsc{AnBx}.

We apply the fresh forward mode introduced in Section 2 to \textsc{SET} case study proposing a solution to the issue outlined in \cite{35}: even a successful and completed \textsc{SET} protocol run does not give the parties enough evidence to prove certain important transaction features, such as a non-repudiable proof of payment authorization. By means of the fresh forward mode, we can achieve non-repudiation in \textsc{SET}, allowing the customer to have evidence that the payment has been authorized by the acquirer.

Interestingly, in both case studies, our versions outperform, in term of security guarantees, the original ones, giving evidence that using adequate abstractions results not only in simpler design, but also in a more robust code and a stronger security. This is largely a consequence of the declarative nature of the specification style supported by \textsc{AnBx}: being defined as channel-level abstractions, the \textsc{AnBx} primitives convey protection on \textit{all} message components, not just on some components as in the original \textsc{iKP} and \textsc{SET} specification, yielding stronger encapsulation mechanisms, and consequently, stronger and more scalable security guarantees.

We verified the \textsc{AnBx} specifications of \{1,2,3\}\textsc{KP} and \textsc{SET} by compiling them into OFMC-\textsc{AnB} and running OFMC on the generated protocols against the strongest possible goals, for each of the protocols.

The original versions of \textsc{iKP} and \textsc{SET} were also model checked, and compared with the revised versions. Below, we report on the results of such tests. 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).
### 5.1 A general e-payment scheme

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.

The process to complete the transaction can be schematized by the following actions:

1. $C \rightarrow M$: *Initiate*
2. $C \leftarrow M$: *Invoice*
   - The Customer and the Merchant exchange information necessary to compose the payment messages.
3. $C \rightarrow M$: *Payment Request*
4. $M \rightarrow A$: *Authorization Request*
   - The Customer sends a *Payment Request* message to the Merchant. The Merchant uses the information received to compose an *Authorization Request* asking authorization from the Acquirer.
5. $M \leftarrow A$: *Authorization Response*
6. $C \leftarrow M$: *Confirm*
   - 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. The Acquirer returns an *Authorization Response* to the Merchant, indicating success or failure. The Merchant then produces a Payment-Response (*Confirm*) message and sends it to the Customer.

Beside the initial knowledge, other information needs to be exchanged. First, to make each transaction univocally identified, the Merchant generates a unique transaction ID (`tid`). Second, the Merchant associates to the transaction also a *date* or any appropriate time information. Both pieces of information must be communicated to the other parties. The core information describing a transaction is then characterized by the tuple $(\text{price, tid, date, can, desc})$ which also constitutes the payment order information. It can be seen as a *contract* among the three parties: 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. *auth*, the transaction authorization result, is then returned by the Acquirer, and communicated to the two other participants.

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 7.

The structure of the payment protocol template can be specified by means of the AnBx messaging primitives as follows:
1. $C \rightarrow M, \eta_1 : [\text{can} : A], [\text{desc} : M]$

2. $C \leftarrow M, \eta_2 : \text{price, tid, date, [contract]}$

3. $C \rightarrow M, \eta_3 : \text{price, tid, can, [can : A], [contract]}$

4. $M \rightarrow A$
   
   (a) $M \rightarrow A, \eta_{4a} : \text{price, tid, can, [can : A], [contract]}$
   
   (b) $M \rightarrow A, \eta_{4b} : \text{price, tid, date, [desc : M], [contract]}$

5. $M \leftarrow A, \eta_5 : \text{auth, tid, [contract]}$

6. $C \leftarrow M, \eta_6 : \text{auth, tid, [contract]}$

The digests [can] and [desc] are annotated with their intended verifiers, A and M respectively (if M is not certified [desc] deserves some more care, as we will discuss shortly). Correspondingly, contract is the tuple ($\text{price, tid, date, [can : A], [desc : M]}$) and its digest 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 $\eta_i$ one may generate different versions of the protocol, achieving different security guarantees. In Table 14 we report the protocols resulting from the choice of the exchange modes $\eta_i$ that offer the strongest level of security for a given number ($i$) of participants having public key-pairs at their disposal. This scalability is a key feature of the iKP protocol family, and therefore the protocols resulting by the instantiation of the exchange modes $\eta_i$ can be considered the AnBx “revised” counterpart of $\{1,2,3\}$KP protocol family. We will be compare the original and the revised version in our first case study. This protocol template also captures most of the essential features of the SET purchase protocol, our second case study.

The (common) narration of the three resulting protocols is:

- Step 1 and 2: Customer and Merchant exchange data that let them build independently the contract. They must tell their “own version of the story” to the acquirer. C declares [can : A], the credit card that will be used in contract, sending the protected digest of can to M. The Customer also informs (and agrees with) the Merchant on the digest of desc they will use to define the contract. 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). M can verify the integrity of [desc : M], form the contract, and then compute its digest. The tuple ($\text{price, tid, date, [contract]}$) 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 it stops.

<table>
<thead>
<tr>
<th>mode/step</th>
<th>→</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>(−,M)</td>
<td>(@C,M)</td>
</tr>
<tr>
<td>$\eta_2$</td>
<td>$C \leftarrow M$</td>
<td>(−,−)</td>
<td>(−,A)</td>
<td>(@M,C)</td>
</tr>
<tr>
<td>$\eta_3$</td>
<td>$C \rightarrow M$</td>
<td>(−,A)</td>
<td>(−,A)</td>
<td>(C,A)</td>
</tr>
<tr>
<td>$\eta_{4a}$</td>
<td>$M \rightarrow A$</td>
<td>↑ (−,A)</td>
<td>↑ (−,A)</td>
<td>↑ (C,A)</td>
</tr>
<tr>
<td>$\eta_{4b}$</td>
<td>$M \rightarrow A$</td>
<td>(−,A)</td>
<td>(−,A)</td>
<td>(M,A)</td>
</tr>
<tr>
<td>$\eta_5$</td>
<td>$M \leftarrow A$</td>
<td>(@A,−)</td>
<td>(@A,M)</td>
<td>(@A,M)</td>
</tr>
<tr>
<td>$\eta_6$</td>
<td>$C \leftarrow M$</td>
<td>↑ (A,−)</td>
<td>↑ (A,−)</td>
<td>↑ (A,C)</td>
</tr>
</tbody>
</table>

Table 14: Exchange modes for the revisited iKP e-commerce protocol
• 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 in general an e-commerce protocol does not allow direct interaction between Customers and Acquirers, but only through Merchant mediation. We assume that $M$ is cooperating in delivering messages. Hence $M$ receives an opaque message, and all he can do is to blindly forward it to $A$ (step 4a).

• Step 4: the Merchant sends the tuple $\langle price, tid, date, [desc : M], [contract] \rangle$ to the Acquirer (step 4b). This information is necessary to complete the payment. In particular $date$ and $[desc : M]$ are required by $A$ to compute independently the digest of $contract$. Upon reception of the two versions of $[contract]$ originating from the two other principals, $A$ can also compute the same value autonomously. If all three match, the transaction can be authorized, since this is the proof of the complete agreement between the customer and the merchant.

• Step 5: the Acquirer 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 the information, and produce a proof that the payment has been authorized for that specific transaction and that specific contract.

• Step 6: the Merchant forwards the message received from the Acquirer to the Customer. This is a notification of the result of the transaction. In this way $C$ receives, via $M$, a proof of payment from $A$. Since the message is signed by $A$, $M$ cannot alter the message without being discovered.

Protocol Goals After completion of the protocol, each participant ideally should have non-repudiable proof of the transaction authorization by the other two parties. In this case they should have sufficient proofs to convince a verifier of the results of the transaction in which they were involved. The lack of some of these provable authorizations does not necessary make the protocol insecure, but it makes disputes between the parties more difficult, relying on evidence provided by other parties, or requiring off-line information.

In our general e-payment protocol, a key role is played by $contract$. It is required that all the parties agree on $contract$ (actually on its digest). In terms of OFMC goals this requires that each participant can authenticate the other two parties on $[contract]$. Moreover the Acquirer should provide a proof that the payment has been authorized and the associated transaction performed. In OFMC this can be represented by the fact that $M$ and $C$ can authenticate $A$ on $Auth$. Ideally the proofs we want to achieve are:

1. $M$ authenticates $C$ on $[contract]$, gives evidence (to $M$) that $C$ has authorized the payment to $M$.

2. $C$ authenticates $M$ on $[contract]$, gives evidence (to $C$) on the terms ($desc$, $price$, etc) of the purchase that $M$ has promised to $C$.

3. $A$ authenticates $C$ on $[contract]$, gives evidence (to $A$) that $C$ authorized $A$ to transfer the money from her account to $M$’s account.

4. $A$ authenticates $M$ on $[contract]$, gives evidence (to $A$) that $M$ has requested the transfer of the money to her account.

5. $C$ authenticates $A$ on $[contract],Auth$, gives evidence (to $C$) that $A$ authorized the payment and performed the transaction.

6. $M$ authenticates $A$ on $[contract],Auth$, gives evidence (to $M$) that $A$ authorized the payment and performed the transaction.
<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>O</td>
<td>R</td>
<td>O</td>
</tr>
<tr>
<td>$C \rightarrow A$: can</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>$M$ authenticates $C$ on $can$</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>desc secret between $C,M$</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>$Auth$ secret between $C,M,A$</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>$tid$ secret between $C,M,A$</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>price secret between $C,M,A$</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$[contract]$ secret between $C,M,A$</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$A$ authenticates $C$ on $[contract]$</td>
<td>-</td>
<td>-</td>
<td>-</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>w</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 $\text{Sig}_A$

$w \equiv$ only weak authentication

Table 15: Security goals satisfied by Original and Revised iKP

Since authentication is realized by means of the digital signature, it should be clear that if a principal is not certified it cannot provide the evidence required. However being certified is not a sufficient condition, but just a necessary one.

Besides the non-repudiable proof, we are also interested on some secrecy goals. First, the Customer’s credit card, or equivalent payment information, should be confidential, and transmitted only to the payment Acquirer. Second, we would like to keep the information exchanged among the principals secret, depending on their encryption capabilities.

6 The iKP protocol family

The iKP protocol family \{i=1,2,3\} was developed at IBM Research [12, 11, 33] 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.

Security analysis

We verified the AnBx protocols described in Section 5 and carried out a corresponding analysis of the original specifications of \{1,2,3\}KP, as reported in [11] and later amended in [32]. 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 ran 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, conceptually they do provide the same abstraction).

For 3KP, the AnBx code for the revised and the original version is shown in the Appendix (Tables 17 and 18).

As we mentioned earlier, the AnBx specification are not just more scalable: they provide stronger security guarantees (cf. Table 15). 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 the 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 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 (Table 14). 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).

An important aspect is that after the completion of our version of iKP payment protocol, each party has non-repudiable proofs of transaction authorization by the other two parties. These are exactly the six goals described in Section 5. Therefore our version achieves all the goals that can ideally be requested, according to the number of certified principals. On the contrary even the original 3KP, the strongest original version, fails in one goal: A can authenticate M on [contract] only weakly. This implies that A needs some information from C, to solve a dispute with M, because only C has evidence that M signed a fresh [contract].

During the analysis of the original 2KP and 3KP we found, to the best of our knowledge, 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 cannot rely on a robust proof of payment. The starred goals in Table 15 are met only after fixing this flaw, which is related with the definition of Lowe’s injective agreement is defined (and 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, since the customer is not certified, this can be done with an ephemeral identity derived from the credit card number). Therefore in the original specification [11]: \( \text{Sig}_A : \{\text{hash}(\text{Auth}, \text{hash}(\text{Common}))\}_{SK_A^{-1}} \) should be replaced by \( \text{Sig}_A : \{\text{hash}(C, M, \text{Auth}, \text{hash}(\text{Common}))\}_{SK_A^{-1}} \).

Our revisited 2KP performs almost as good the original 3KP. A goal that is not satisfied is “can secret between C,A”. This does not mean that the credit card is leaked but only that 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. Moreover the fact the C is not certified prevents in 2KP the possibility to authenticate this principal on [contract].

We mentioned earlier that some care must be taken in the revised version of 1KP to compute the digest \([\text{desc}:M]\). In fact M is not certified, so we cannot calculate the digest verifiable by M (Table 7). 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 let M generate the digest rather than C, because [\text{can}:A] is already generated by C. This because the digest generation implies the usage of fresh keys as confounders and it is appropriate that those two values are generated by two different principals. Note that 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 (the goal is $C \rightarrow *:can$). Since this is one of the main concerns of all e-commerce users, it is indeed a good news.

7 Secure Electronic Transaction

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. The main aim of SET was to enable customers to make purchases, having guarantees of authenticity of the transaction while keeping the customer’s account details secret from the Merchant and his choice of goods secret from the Acquirer (payment gateway).

Despite being backed by the major players of the IT and financial industry, SET failed to become the standard de facto for electronic payments. Reasons of the lack of adoption include the fact that the infrastructure required by SET is complex and requires cooperation of many parties to be established. Moreover as we will see SET fails to meet some of the desired protocol goals.

Nevertheless SET is still an interesting case study, as its complex structure makes it a benchmark for security protocols design and verification. It is a real-world protocol, arising from the industry, and its documentation comprises more than 1000 pages. Such a large protocol needs tool support to be verified effectively. Bella et al. [8, 9, 10] made a major endeavor analyzing and abstracting the SET protocol and we take their work as reference for the protocol specification.

In the present case study we consider the SET purchase protocol as outlined in [10]. SET uses many optional data and, depending on which are taken into account, we may obtain different alternative versions of the purchase phase. The most difficult task is to find a version that is both simple and relatively close to reality. Following a common idea in literature [14, 22, 10, 24, 29, 35], we consider a single transaction involving no optional data. We assume that the Merchant registration and the optional cardholder registration have been successfully completed. In this case the fact that agents $C$ and $M$ are certified. The certification of $C$ is optional, in analogy to 2KP, where only Customer and Acquirer are certified. This variant of the protocol is called the “unsigned” version of SET, in contrast with the “signed” version where all principals are certified.

Introduction to SET

To ease the comparison with other works on SET, in this presentation the information exchanged is denoted with the names commonly used in the SET specification. Now we introduce some basic concepts of the protocol and provide a mapping of the data to the general e-commerce template presented in section 5.

As is customary in the e-payment protocol specifications, we assume that Customer and Merchant have already completed the initial shopping agreement, and they agree on the order description ($OrderDesc$) and purchase amount ($PurchAmt$). This is not part of the SET specification and it can be done by any mean, even out of band, as long as the secrecy of $OrderDesc$ and $PurchAmt$ is guaranteed. The primary account number, $pan(C)$, is an abstraction of the credit card number belonging to the Customer. If the cardholder registration (a subprotocol of SET) is completed, $pan(C)$ is a public-key certificate that includes the hash of the credit card number and an optional PIN ($PanSecret$). Otherwise $pan(C)$ could be the credit card number itself. In any case the Acquirer should be able to verify the validity of the payment information presented by the Customer. In the case of the “unsigned” version, the payment information may also be employed as a weak proof of identity of $C$, since $C$ cannot digitally sign.

The initial knowledge of each participant is:

- **Customer $C$**: $PurchAmt, OrderDesc, pan(C)$;
- **Merchant $M$**: $PurchAmt, OrderDesc$;
• Acquirer $A : \text{pan}(C)$.

During the protocol run the principals generate some identifiers: $LIDM$ is a local transaction identifier that the Customer sends to the Merchant in order to identify the transaction, while the Merchant generates $XID$ which is used in the rest of the protocol as a transaction ID. There is debate whether $XID$ can be considered globally unique or not [14]; we will return later to this issue. In analogy with the SET syntax we use some abbreviations, but compared with the original specification (see [10, 14] and Table 20 in appendix) they are less and simpler:

- $Oldata: \text{OrderDesc}$;
- $PIdata: \text{pan}(C)$;

In our specification $Oldata$ and $PIdata$ just collapse to the order description and the primary account number respectively. We just keep these names to maintain a notation similar to the original SET. $HOD$ structure is quite different, and is composed by two digests. The first digest $[Oldata:M]$ is verifiable by the Merchant while the second $[PIdata:A]$ is verifiable by the Acquirer.

Moreover we find convenient, in order to make clearer the specification, to denote $TID: LIDM, XID$.

As it can be easily seen, we can map the names used in SET with the names of our e-payment template protocol presented in Section 5:

- $\text{OrderDesc} = \text{OIData} = \text{desc}$;
- $\text{pan} = \text{PIData} = \text{can}$;
- $\text{PurchAmt} = \text{price}$;
- $LIDM, XID = TID = \text{tid}$;
- $HOD = [\text{desc}:M],[\text{can}:A]$.

**Dual Signature**

A key idea introduced in SET is the dual signature. Its purpose is to let several parties agree on a transaction without giving any of them full view of the details. Consider when a certified Customer, as in the “signed” SET, needs to send the order information ($Oldata$) to the Merchant and the payment information ($PIdata$) to the Acquirer. In order to process an order, the Merchant does not need the customer’s credit card number; 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 Customer’s order, he just needs evidence that a particular payment has been asked for a specific order.

As showed in [10], a fragment of an e-commerce protocol using the dual signature, similarly to what SET does, may be the following:

1. $C \rightarrow M : \{Oldata, \mathcal{H}(PIdata)\}_{PK_M}, sig_C(M, A, \mathcal{H}(Oldata), \mathcal{H}(PIdata)), \{PIdata\}_{PK_A}$
2. $M \rightarrow A : \{\mathcal{H}(Oldata)\}_{PK_A}, sig_C(M, A, \mathcal{H}(Oldata), \mathcal{H}(PIdata)), \{PIdata\}_{PK_A}$

The digests of $Oldata$ and $PIdata$ are independently computed by the Customer by means of an hash function $\mathcal{H}$. Then $C$ computes the digital signature of the concatenation of the two hashes - recall that $sig_C(m) = (\mathcal{H}(m))_{SK_C}$ - obtaining the dual signature:

$\text{sig}_C(M, A, \mathcal{H}(Oldata), \mathcal{H}(PIdata))$.

However, in contrast to the example given in [10], the identities of the intended verifiers $(M, A)$ are here included to comply with the Abadi and Needham’s [4] explicitness principle, and, as a consequence, to meet the definition of Lowe’s weak agreement. The Customer sends the dual
signature to the Merchant along with two other submessages: the first $OIdata, H(PIdata)$ - secret for the Merchant -, the second $PIdata$ - secret for the Acquirer. The first submessage allows the Merchant to have evidence of the digest of $PIdata$ without seeing the $PIdata$ itself (the payment information). Similarly in the second step the Merchant sends the digest of $OIdata$ to the Acquirer. Obviously this mechanism works only if Merchant and Acquirer do not cooperate to cheat the Customer, exchanging their information. We assume that the Acquirer, typically a financial institution, is trusted and respects the privacy of the Customers.

Model checking the AnB protocol with OFMC, it is possible to verify that the protocol fragment satisfies the following goals:

- $OIdata$ secret between $C,M$
- $PIdata$ secret between $C,A$
- $A$ weakly authenticates $C$ on $hash(OIdata), hash(PIdata)$
- $M$ weakly authenticates $C$ on $hash(OIdata), hash(PIdata)$

As a consequence, we can say that if all the principals are certified, the dual signature works as expected. Mimicking the same protocol fragment in AnBx, results in a protocol satisfying the same goals, but requiring more steps.

1. $C \rightarrow M, (C,M) : [OIdata], [PIdata], OIdata$
2. $M \rightarrow C, (−,−) : empty$
3. $C \rightarrow M, (C,A) : [OIdata], [PIdata], PIdata$
4. $M \rightarrow A, ↑(C,A) : [OIdata], [PIdata], PIdata$
5. $A \rightarrow M, (−,−) : empty$
6. $M \rightarrow A, (M,A) : [OIdata],[PIdata]$

The extra communications depend by the fact that in AnBx the security properties are specified at the channel level, rather than on the single components of the payload. In fact we cannot build messages having components with different security attributes, with the only exception of the verifiable digests (Section 3).

An issue arises when considering the “unsigned” SET, where the Customer is not able to sign messages. In this case, the protocol fragment involving the dual signature becomes:

1. $C \rightarrow M, (C,M) : [OIdata],[PIdata], OIdata$
2. $M \rightarrow A, (−,−) : H(OIdata), H(PIdata)$, $[PIdata]$ $PK_A$

The term $H(OIdata), H(PIdata)$, binding two hashes, is built using a hash function rather than a digital signature. This implies the intruder could attempt a chosen plaintext attack against the hashes, while previously the intruder was not be able to forge the Customer’s signature. The order information is vulnerable since the “universe” of possible orders can be in some cases quite limited, and breaking the secrecy of the order information could disclose some sensitive data about the Customer. In general attacks against hash functions could undermine some of the goals of the protocol: the confidentiality of the order and payment information.

To counter this problem and make more robust our protocol we can use the AnBx protected digests (Section 3), which have the nice property to be verifiable only by a predefined principal and therefore are immune from chosen plaintext attacks (the encryption key acts as a confounder). Instead of the dual signature, the Customer can build the pair $([OIdata : A],[PIdata : M])$, in which the two components are verifiable by the Acquirer and the Merchant respectively. The components are bound together by the secret messages that $C$ sends to $M$. Thus we can write:

1. $C \rightarrow M, (−,M) : [OIdata : M],[PIdata : A], OIdata$
2. $M \rightarrow C, (−,−) : empty$
achieving in this way a stronger robustness even if the customer is not certified (as in the unsigned SET). In fact every message exchanged is encrypted and the goals satisfied are:

\[ \text{C} \rightarrow \text{M}, (-,A) : [\text{OI data} : \text{M}], [\text{PI data} : \text{A}], \text{PI data} \]

\[ \text{M} \rightarrow \text{A}, ↑(-,A) : [\text{OI data} : \text{M}], [\text{PI data} : \text{A}], \text{PI data} \]

\[ \text{A} \rightarrow \text{M}, (-,-) : \text{empty} \]

\[ \text{M} \rightarrow \text{A}, (M,A) : [\text{OI data} : \text{M}], [\text{PI data} : \text{A}] \]

With some slight adaptation this fragment of protocol will be the central part (steps 3 and 4) of the revised version of SET we are going to present. For scalability this solution can also be applied in the “signed” version just tuning the exchange modes according to the Customer ability to digitally sign, i.e. setting \((C,M),(C,A),↑(C,A)\) in steps 1,3,4.

### Revisiting SET

Many papers on SET ([10, 14, 35]) focused their attention mostly on the “signed” version of SET. Therefore, to make easier the comparison, we will present the same version, provided that the “unsigned” version is obtained by the following specification, just removing \(C\) in source field of the exchange modes:

1. \(C \rightarrow M, (@C, M) : \text{LIDM}\)
2. \(M \rightarrow C, (@M, C) : \text{XID}\)
3. # Payment Request
   - (a) \(C \rightarrow M, (@C, M) : \text{TID}, \text{PurchAmt}, \text{HOD}\)
   - (b) \(M \rightarrow C : \text{empty}\)
   - (c) \(C \rightarrow M, (C,A) : \text{TID}, \text{PurchAmt}, \text{HOD}, \text{PI data}\)
4. # Authorization Request
   - (a) \(M \rightarrow A, ↑(C,A) : \text{TID}, \text{PurchAmt}, \text{HOD}, \text{PI data}\)
   - (b) \(A \rightarrow M : \text{empty}\)
   - (c) \(M \rightarrow A, (M,A) : \text{TID}, \text{PurchAmt}, \text{HOD}\)
5. \(A \rightarrow M, (@A, M) : C, \text{TID}, \text{AuthCode}\)
6. \(M \rightarrow C, ↑(@A, C) : C, \text{TID}, \text{AuthCode}\)

Step by step the narration of the protocol is:

- **Step 1 and 2 - Purchase Initialization Request/Response:** the Customer sends the local transaction identifier LIDM, then the Merchant replies with the transaction identifier XID. These steps have the sole aim to let C and M exchange data that is needed to build the messages exchanged in the following steps. In these steps the exchanges are secret, authentic and fresh.

- **Step 3 - Payment Request:**
(a) the Customer prepares a fresh authentic message secret for the Merchant containing TID and HOD and the price PurchAmt. TID includes information to identify the transaction, while HOD contains the evidence (digest) of the credit card the Customer intends to use, and the evidence of the order description that will later forward to the Acquirer. Being part of an authentic message, HOD binds its components similarly to what happens with the dual signature. The Merchant can verify the digest \( OIdata:M \) generated by the Customer, while TID contains information he already knows and that can trivially be checked. If the verification fails the protocol can stop since there is no possibility of authorizing a payment if Customer and Merchant disagree on OrderDesc or TID or PurchAmt (see also step 5)

(c) the Customer composes a fresh authentic and secret message for the Acquirer. Beside the information provided in 3.(a), the message includes the payments information \((PIdata)\) which will be processed by the Acquirer in order to authorize the payment. The message is delivered to the Merchant, since the protocol does not allow direct interaction between customers and acquirers. Note that, being included in a secret message, the payment information is not captured by the Merchant.

- **Step 4 - Authorization Request:**
  - (a) the message received at 3.(a) by the Merchant is blindly forwarded to the Acquirer. The Merchant is not exposed to its content.
  - (c) the Merchant sends an authentic and secret message to the Acquirer containing TID, PurchAmt, HOD, telling in this way “his own version of the story”
  - the Acquirer checks if the two versions of TID, PurchAmt, HOD, provided independently by the Customer and the Merchant, are the same. If this happens the Acquirer can be sure that the two principals had agreed on the terms of the transaction and then can verify the payment information \((PIdata)\) originated by the customer. If the verification succeeds the payment can be authorized and the authorization code \((AuthCode)\) is generated, otherwise the request is rejected \((AuthCode\) will include an error message explaining the reason of the failure).

- **Step 5 - Authorization Response:** the Acquirer sends, in an authentic and secret message to the Merchant, the authorization code along with TID which is necessary to identify the transaction. The identity of the Customer is included to allow the forwarding of the message at step 6, maintaining the authenticity guarantee.

- **Step 6 - Purchase Response:** the Merchant forwards the message received from the Acquirer to the Customer. 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 cannot alter the message without being discovered. The structure of this message is quite different from the original SET. This will help to guarantee the non-repudiation goals that the original SET does not provide. Details are given below.

### 7.1 Main results of SET security verification

We verified the AnBx specifications of “signed” and “unsigned” SET purchase protocol and carried out a corresponding analysis of the original specifications, as reported in [10]. As in iKP we ran our tests assuming that the Acquirer is trusted. Indeed, though SET does not rely on such assumption, each customer can stop the protocol if she does not want to proceed with an Acquirer she does not trust. Therefore we can assume that, if the run of the protocol is completed, the customer decided to trust the Acquirer. Moreover to compare the results of the analysis we treated the messages components having the same name as equivalent, since they do provide conceptually the same abstraction. For the “signed” version of the SET purchase protocol,
Goal                                      | unsigned SET | signed SET |
---                                         | O | R | O | R |
\(pan(C)\) secret between \(C,A\)          |   |   | - | + |
\(C \rightarrow A : (\text{pan}(C))\)     | + | + | + | + |
\(A\) authenticates \(C\) on \(\text{pan}(C)\) | - | - | + | + |
\(\text{OrderDesc} \) secret between \(C,M\) | + | + | + | + |
\(\text{PurchAmt} \) secret between \(C,M,A\) | - | - | + | + |
\(\text{AuthCode} \) secret between \(C,M,A\) | - | - | - | + |
\(M\) authenticates \(A\) on \(\text{AuthCode}\) | + | + | + | + |
\(C\) authenticates \(A\) on \(\text{AuthCode}\) | - | + | - | + |
\(C\) authenticates \(M\) on \(\text{AuthCode}\) | +\* | - | +\* | - |
\(\text{TID} \) secret between \(C,M,A\)    | - | - | - | + |
\(\text{HOD} \) secret between \(C,M,A\)    | - | - | - | + |
\(A\) authenticates \(C\) on \(\text{contract}\) | - | - | w | + |
\(M\) authenticates \(C\) on \(\text{contract}\) | - | - | + | + |
\(A\) authenticates \(M\) on \(\text{contract}\) | w | + | w | + |
\(C\) authenticates \(M\) on \(\text{contract}\) | + | + | + | + |
\(C\) authenticates \(A\) on \(\text{contract}, \text{AuthCode}\) | - | + | - | + |
\(M\) authenticates \(A\) on \(\text{contract}, \text{AuthCode}\) | + | + | + | + |

* goal satisfied only fixing step 5 as in [10]

\(w \equiv \text{only weak authentication}\)

\[
\text{revised SET } \Rightarrow \text{ contract} = \text{PriceAmt, LIDM, XID, } [\text{PIData}:A], [\text{OIData}:M]
\]

\[
\text{original SET } \Rightarrow \text{ contract} = \text{PriceAmt, LIDM, XID, hash(PIData), hash(OIData)}
\]

Table 16: Security goals satisfied by Original and Revised SET purchase protocol

the \textsf{AnBx} code of the revised and the original versions is shown in the Appendix (Tables 19 and 20).

In general, our version of the \textit{SET} purchase protocols outperforms the original one, i.e. it satisfies more and stronger security goals, both for the signed and the unsigned versions (Table 16). Similarly to what was already explained for \textit{iKP} (Section 6) we benefit of the declarative nature of \textsf{AnBx} message primitives, which are defined as channel abstractions. They offer strong encapsulation mechanisms on the messages they exchange, and, when authentication is required, they implement automatically the Abadi and Needham’s [[4]] explicitness principle.

It is interesting to note that two known flaws affecting the \textit{SET} original specification do not compromise our revised version.

The \textbf{first flaw} [[10]] involves the 5-th step of the protocol: often in \textit{SET} the signed messages lack of explicitness and therefore in step 5 it is not possible to link univocally the identity of the Acquirer and the Merchant with the particular transaction and authentication code. The original instruction

\[
A \rightarrow M : \{S_A(LIDM,XID,\text{PurchAmt},\text{AuthCode})\}_{PK_M}
\]

should then be amended to

\[
A \rightarrow M : \{S_A(M,LIDM,XID,\text{PurchAmt},\text{AuthCode})\}_{PK_M}
\]

otherwise the goal \textit{C authenticates Me on AuthCode} cannot be satisfied.

In our version the message at step 5 include both the identity of the Merchant and of the
Customer. This happens because the payload is automatically compiled to a message that includes the following sub-message:

\[ S_A (M, C, LIDM, XID, PurchAmt, AuthCode) \]

Interestingly the same implementation prevents also the second flaw presented in [14]. In this paper the specification of the protocol is more detailed than in [10] and it introduces an additional field \( AuthRRTags \), which includes the identity of the Merchant. The attack is against the purchase phase and exploits a lack of verification in the payment authorization process. It may allow a dishonest Customer to cheat on the Merchant. This flaw is subtle: unlike the attack in [10], requiring the existence of a corrupted Acquirer, this attack requires only a collusion between a dishonest Merchant and a Customer.

The attack is based on the fact that neither \( LIDM \) nor \( XID \) can be considered unique and they cannot be used to identify the Merchant. Therefore the customer can start a parallel purchase with an accomplice (playing the role of another merchant). Both merchants generate their authorization requests and send them to the Acquirer, but the intruder intercepts the good merchant’s request and destroys it. The Acquirer proceeds with the intruder’s message where in one field \( (AuthRRTags) \) the intruder has changed its identity with the identity of the good merchant.

Due to a faulty verification process (as specified by the \( SET \) documentation, and reported in [14]), the Acquirer does not compare the identity in \( AuthRRTags \) with the other identities present in the message. The authorization message sent to the intruder at step 5 has the same structure of the message that the good merchant expects, so the Merchant can be convinced that he has been paid. Therefore the Merchant can then deliver the goods to the dishonest customer. The result is that the Acquirer has authorized the payment in favor of the intruder playing the role of the dishonest merchant. Full details about this attack can be found in [14]. As the authors suggest, the fix is to bind under the Acquirer signature the merchant’s and the client’s identities, and this is what our version does in step 5 and 6.

To check this solution, we tested the versions of \( SET \) presented in [14] and [10] with OFMC and verified that they need to be fixed also at the sixth (and final) step. This issue has already been outlined in [35]. The identity of the Customer must be included in the message, otherwise the Customer cannot authenticate the Merchant on \( AuthCode \). This issue also lead us to more interesting considerations about how to prove the authorization of the transaction.

**Proving authorization of the transaction**

It should be noticed that \( AuthCode \) is a term generated by the Acquirer, but in the original \( SET \) the Customer does not have an evidence of the origin, rather she relies only on information provided by the Merchant. This is a symptom that something is going wrong. As we have seen (Section 5), in general, the non-repudiable proofs of transaction authorization can be achieved by means of six authentication goals involving pairs of principals, however \( SET \) fails on some of them and this is a major limitation of \( SET \).

In our general e-payment protocol (Section 5), \( contract = price, tid, date, [can:A], [desc:M] \) defines the terms of the transaction on which all participants should ideally agree. In the revised \( SET \) the equivalent term is \( PriceAmt, TID, [OIData:M], [PIData:A] \). There is no loss of generality if we ignore the field \( data \) which contains only complementary information. Expanding the definitions of \( TID, OIData \) and \( PIData \) we obtain

\[ contract = PriceAmt, LIDM, XID, [OrderDesc:M], [pan(C):A]. \]

In analogy in the original \( SET \) we can define

\[ contract = PriceAmt, LIDM, XID, hash(PIData), hash(OIData). \]

Our analysis showed that some of desired six goals are not met by the original “signed” \( SET \) (Table 16). The same result, using a different analysis technique, was already described in [35], but we can show that amending our revised version of \( SET \), it is possible to achieve all six goals.
For example giving to \( C \) a proof that \( A \) authorized the payment requires substantial modification of the 6-th step of the protocol. In fact, instead of letting the Merchant signing a message for the customer, we exploit the \( \text{AnBx} \) forward mode to bring to the customer the authorization of the payment signed directly by the Acquirer. It is interesting to notice that employing the \textit{forward fresh mode} in the sixth step, we can achieve the strong authenticity goal on \text{contract, AuthCode} even though the transaction identifier is not unique (as in the second flaw).

The resulting protocol is very similar to the general e-payment protocol, introduced in Section 5. This should not surprise, since it confirms the results outlined in [35] showing that while \( \text{iKP} \) meet all the non-repudiation goals, \( \text{SET} \) does not.

In summary we showed that revisiting \( \text{SET} \) in \( \text{AnBx} \) not only offers a simpler and clearer design, but also stronger guarantees compared to the original specification. The most interesting points are that designing \( \text{SET} \) by means of \( \text{AnBx} \) primitives, produces a protocol that is immune to the known flaws present in the original \( \text{SET} \), and it is possible to fully prove authorization of transaction.

8 Related Work and Conclusions

Our approach shares motivations and ideas with various papers on design and implementation of high-level process calculi for distributed protocols and systems [2, 3, 1, 5, 16]. 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. Some research more closely related to ours has been carried out in [22, 23, 31]. Guttmann [22] has proposed a protocol design process centered on the authentication tests, a method for protocol verification based on the strand space theory. Guttmann, Herzog, Ramsdell, and Sniffle [23] 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 makes use of cryptographic primitives while we avoid any reference to explicit cryptography.

Mödersheim and Viganò [31] 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 [7, 31] to verify the \( \text{AnBx} \) 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) cannot be specified only by means of the bulleted channels.

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 that the experience with \( \text{AnBx} \) we have reported in this paper provides further evidence of the advantages of the approach: as we have illustrated, using adequate abstractions results not only in simpler design, but also in a more robust code and stronger security. In fact, being defined as channel-level abstractions, the \( \text{AnBx} \) primitives convey protection on all message components, yielding stronger encapsulation mechanisms, and consequently, stronger and more scalable security guarantees.

Acknowledgments

This work was partially supported by MIUR Projects SOFT “Security Oriented Formal Techniques” and IPODS “Interacting Processes in Open-ended Distributed Systems”.

29
References


Appendix

- Tables 17 and 18: revised and original iKP specification
- Tables 19 and 20: revised and original SET specification

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: 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 17: AnBx specification of the revised 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 18: Portion of the AnBx specification of the original 3KP
Protocol: SET_Revised

# Signed Purchase Request

Types:
Agent C, Me, a;
Certified C, Me, a;
Number LIDM, XID, PurchAmt, OrderDesc, AuthCode;

Function pan

Definitions:
TID: LIDM, XID;
OIdata: OrderDesc;
PIdata: pan(C);
HOD: dig(OIdata, Me), dig(PIdata, a)

Knowledge:
C: C, Me, a, pan(C);
Me: C, Me, a;
a: C, Me, a

Actions:

# 0. Setup/Initial Knowledge
C *->* Me: PurchAmt, OrderDesc
Me -> C: empty

# 1. Purchase Initialization Request
C -> Me, (@C, Me): LIDM

# 2. Purchase Initialization Response
Me -> C, (@Me, C): XID

# 3. Purchase Request
C -> Me, (@C, Me): TID, PurchAmt, HOD
Me -> C: empty
C -> Me, (@C, a): TID, PurchAmt, PIdata, HOD

# 4. Authorization Request
Me -> a, (@C, a): TID, PurchAmt, PIdata, HOD
a -> Me: empty
Me -> a, (@Me, a): TID, PurchAmt, HOD

# 5. Authorization Response
a -> Me, (@a, Me): C, TID, AuthCode

# 6. Purchase Response
Me -> C, (@a, C): C, TID, AuthCode

Goals:
pan(C) secret between C, a
a authenticates C on pan(C)
OrderDesc secret between C, Me
PurchAmt secret between C, Me, a
AuthCode secret between C, Me, a
Me authenticates a on AuthCode
C authenticates a on AuthCode
C authenticates Me on AuthCode
TID secret between C, Me, a
HOD secret between C, Me, a

Table 19: AnBx specification of the revised SET
Protocol: SET_Original

# Signed Purchase Request
# G. Bella, F. Massacci and L.C. Paulson "Verifying the SET Purchase Protocols"

Types:
- Agent C, Me, a;
- Certified C, Me, a;
- Number PurchAmt, XID, OrderDesc, LIDM, ChallC, ChallM, AuthCode;
- Function pan

# pan(C) abstract CardSecret, PAN, PANSecret see AVISPA SM (AI)

Definitions:
- HOD: hash(OrderDesc, PurchAmt);
- PIHead: LIDM, XID, HDD, PurchAmt, Me, hash(XID, pan(C));
- OIData: LIDM, XID, ChallC, HDD, ChallM;
- PANData: pan(C);
- PIData: PIHead, PANData;
- PIDualSign: {hash(PIData), hash(OIData)} inv(sk(C)), {PIHead, hash(OIData), PANData} pk(a);
- OIDualSign: OIData, hash(PIData)

# CompCode: PurchAmt, AuthCode, Status

Knowledge:
- C: C, Me, a, pan(C);
- Me: C, Me, a;
- a: C, Me, a

Actions:

# 0. Setup/Initial Knowledge
C *->* Me: PurchAmt, OrderDesc
Me -> C: empty

# 1. Purchase Initialization Request
# The Cardholder sends the Merchant a freshness challenge (ChallC)
# and a local transaction identifier (LIDM).
C -> Me: LIDM, ChallC
Me -> C: {LIDM, XID, ChallC} inv(sk(Me))

# 2. Purchase Initialization Response
# The Merchant replies with a signed message that includes a freshness
# challenge (ChallM) and generates a nonce that serves as the globally
# unique transaction identifier XID
Me -> C: {LIDM, XID, ChallC, ChallM} inv(sk(Me))

# 3. Purchase Request
# Payment Instructions PIData and the Order Information OIData
C -> Me: PIDualSign, OIDualSign

# 4. Authorization Request
Me -> a: {{LIDM, XID, hash(OIData), HDD, PIDualSign} inv(sk(Me))} pk(a)

# 5. Authorization Response
a -> Me: {{Me, LIDM, XID, PurchAmt, AuthCode} inv(sk(a))} pk(Me)

# 6. Purchase Response
Me -> C: {LIDM, XID, ChallC, hash(PurchAmt), AuthCode} inv(sk(Me))

Goals:
[
]

Table 20: Portion of the AnBx specification of the original SET