Static Analysis of Authentication ⋆

Riccardo Focardi

Dipartimento di Informatica
Università Ca'Foscari di Venezia
e-mail: focardi@dsi.unive.it

Abstract. Authentication protocols are very simple distributed algorithms whose purpose is to enable two entities to achieve mutual and reliable agreement on some piece of information, typically the identity of the other party, its presence, the origin of a message, its intended destination. Achieving the intended agreement guarantees is subtle because they typically are the result of the encryption/decryption of messages composed of different parts, with each part providing a "piece" of the authentication guarantee. This tutorial paper presents the basics of authentication protocols and illustrates a specific technique for statically analysing protocol specifications. The technique allows us to validate protocols in the presence of both malicious outsiders and compromised insiders, with no limitation on the number of parallel sessions.

This paper covers the course "Static Analysis of Authentication" given by the author at the FOSAD'04 school. The static analysis technique described here is a joint work with Michele Bugliesi and Matteo Maffei (Università di Venezia) [8, 12].

1 Introduction

Security protocols are designed to provide diverse security guarantees in possibly hostile environments: typical guarantees include the secrecy of a message exchange between two trusted entities, the freshness and authenticity of a message, the authenticity of a claimed identity, ... and more. The presence of hostile entities makes protocol design complex and often error prone, as shown by many attacks to long standing protocols reported in the literature (see, e.g., [13, 14, 22, 27, 28]). In most cases, such attacks dwell on flaws in the protocols' logic, rather than on breaches in the underlying cryptosystem. Indeed, even when cryptography is assumed as a fully reliable building-block, an intruder can engage a number of potentially dangerous actions, notably, intercepting/replaying/forging messages, to break the intended protocol invariants. Formal methods have proved very successful as tools for protocol design and validations. On the one hand, failures to model-check protocols against formal specifications have lead to the

⋆ Work partially supported by EU Contract IST-2001-32617 'MyThS' and by MIUR Project 'Abstract Interpretation: Design and Applications' (AIDA).

discovery of several attacks (see, e.g., [26, 28]). On the other hand, static techniques, based on type systems and control-flow analyses have proved effective in providing static guarantees of correctness [1, 2, 5, 6, 10, 16, 17].

Informally, *authentication* protocols enable two entities to achieve mutual and reliable agreement on some piece of information, typically the identity of the other party, its presence, the origin of a message, its intended destination. Even if there are many formalizations of authentication and many formal tools to verify it, not much has been done on the side of static analysis. Static analysis techniques like, e.g., type systems and control flow, aim at proving a property of a program, or a protocol, by only inspecting the source code. They are appealing because (*i*) only the code is inspected and there is no need of generating and exploring all the possible execution sequences; (*ii*) by exploiting compositionality, it becomes feasible to (even automatically) prove correctness of protocols with unbounded number of sessions and participants. These advantages are payed by approximating the property of interest thus loosing precision: in fact, static analyses are typically sound, i.e., an incorrect protocol is never validated, but not complete. As a consequence, there might be false-negatives, i.e., correct protocols that cannot be validated.

This tutorial paper introduces the basics of authentication protocols in a smooth and incremental way (Section 2), discusses how authentication is formalized in terms of correspondence assertions (Section 3), illustrates a type-based technique for statically analysing protocol specifications (Section 4) and briefly discusses related work (Section 5). This paper covers the course "Static Analysis of Authentication" given by the author at the FOSAD'04 school. The static analysis technique described in Section 4 is a joint work with Michele Bugliesi and Matteo Maffei (Università di Venezia). Here, we only focus on the basic ideas and concepts of such analysis and more detail can be found in [8, 12].

2 Authentication Protocols

As mentioned above, *authentication* protocols enable two entities to achieve mutual and reliable agreement on some piece of information, typically the identity of the other party, its presence, the origin of a message, its intended destination. Authentication is thus related to the crucial problem of reliably agreeing on information that is spread on a (possibly untrusted) network like, e.g., the Internet. In this section, we present the basic techniques to achieve this property, mainly focussing on its simplest form called *entity authentication* or *identification*, which can be informally stated as follows:

> **Entity authentication** enable one entity, the *claimant*, to prove its claimed identity to another entity, the *verifier*.

Entity authentication basically amounts to reliably agreeing on the claimant identity. The application fields are extremely numerous, since identification is a basic requirement for many applications. For example, we need entity authentication for personal accounting on a system, physical access to restricted areas, payment for the use of resources or services, e-commerce, and so on.

A first important classification of authentication techniques can be done depending on *what* they are based on [25]:

1. *Something known*: the claimant proves its identity by demonstrating the knowledge of a secret like, e.g., a password, a Personal Identification Number (PIN), a cryptographic key, ...
2. *Something possessed*: the claimant proves its identity by demonstrating the possession of a physical object like, e.g., a magnetic card, a smart card, a palmtop or Personal Digital Assistant (PDA), ...
3. *Something inherent*: the claimant proves its identity by showing a physical characteristic or involuntary action like, e.g., a signature, a fingerprint, e retinal pattern, the writing rate on a keyboard, ...

It is important to notice that these techniques are often combined together in order to simplify their use and complement their respective benefits. A very common example is the cell-phone smart card (i.e., *something possessed*) which contains a cryptographic key (i.e., *something known*) used to authenticate with the service provider. The user does not need to remember the fairly long cryptographic key contained in the smart card: she is just required to enter a short PIN (i.e., *something known*) which enables the smart card to authenticate with the provider. This "two-phase" authentication has the aim of simplifying the user task without decreasing the level of security. The use of cryptographic keys to authenticate will be treated in Section 2.2.

2.1 Password-based authentication

The simpler, and undoubtedly most used, authentication scheme is the *password-based* one. It is a *something known* scheme in which the claimant and the verifier share the knowledge of a secret password. The password is shown by the claimant in order to authenticate herself to the verifier, as illustrated below:

$$
\begin{array}{ccc}
A & & B \\
 & \xleftarrow{\quad\text{login?}\quad} & \\
 & \xrightarrow{\quad\text{Alice}\quad} & \\
 & \xleftarrow{\quad\text{password?}\quad} & \\
 & \xrightarrow{\quad\text{``wonderland''}\quad} &
\end{array}
$$

Notice that this scheme is perfectly safe if Alice and Bob communication is protected, but it becomes completely vulnerable whenever the password can be intercepted by a malicious third party. As a matter of fact, once password secrecy is lost, anyone knowing it can easily impersonate Alice by just re-running the protocol above with Bob. When this happens, it is violated the basic property every authentication protocol should provide:

Non-impersonation S should never be able to impersonate A with B, even after observing previous protocol runs between A and B.

This ideal property requiring the impossibility of impersonating the claimant is typically achieved in a computational form:

Non-impersonation (computational) The probability that S impersonates A with B is negligible, even after observing previous protocol runs between A and B.

where negligible means "is so small that it is not of practical significance". For example, in this case, it should be small enough to guarantee that once S has broken the protocol, A has already changed her password. This computational version of non-impersonation is of much more practical interest, since in cryptography every key can be broken given enough time either by "brute force", i.e., by trying all the possible keys, or using some more sophisticated cryptanalysis technique. Thus, it is important to guarantee that this happens with a probability that is small enough to make the attack useless.

Even if the password-based scheme is very vulnerable on an untrusted network, like the Internet, it has been used quite widely in the past and is still used by many web-sites. The telnet protocol is a striking example: in order to connect to a remote host, authentication was implemented by following the above scheme, thus allowing an intruder to easily intercept the password sent in clear on the network. Many web-sites are still not protected by cryptography (through, e.g., the Secure Sockets Layer, SSL, protocol), and, similarly to what happens with telnet, authentication is achieved by a password or a PIN which is just sent in clear to the web server.

To develop stronger authentication protocols it is first necessary to ask ourselves where the weakness of the password-based scheme comes from. The answer is quite easy to give: proving the knowledge of a secret by just revealing it is very unsafe, especially if we want to reuse such secret for further authentications! In the next section we give the basics of *challenge-response* protocols, whose aim is exactly to show the knowledge of a secret without revealing it, thus circumventing the weakness discussed above. We will see that, even if those protocols are more solid that password-based ones, some subtle attacks can be yet easily mounted if the protocols are not carefully developed.

2.2 Challenge-response protocols

The idea of challenge-response protocols is to prove the knowledge of a secret without revealing it: this is done by replying to a claimant's challenge with a response that depends on the secret without revealing much of it. More precisely, knowing the secret it must be easy to check if the response depends on it, but it must be (computationally) unfeasible to derive the secret from the response.

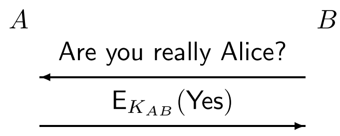
The generation of these particular responses is based on cryptographic techniques. We thus briefly review the basic notions of cryptography [29]. A cryptosystem $\mathcal{S}$ is a quintuple $(\mathcal{P}, \mathcal{C}, \mathcal{K}, E, D)$ where $\mathcal{P}$ is the set of *plaintext*, $\mathcal{C}$ is the set of *ciphertexts*, $\mathcal{K}$ is the set of *keys*, and $E : \mathcal{P} \times \mathcal{K} \to \mathcal{C}$ and $D : \mathcal{C} \times \mathcal{K} \to \mathcal{P}$ are the *encryption* and *decryption* functions, respectively. Function E, given a plaintext and an encryption key k, returns the corresponding ciphertext, while function D, given a ciphertext and the corresponding decryption key k^{-1}, returns the corresponding plaintext. We write $E_k(x)$ and $D_{k^{-1}}(y)$ in place of $E(x, k)$ and $D(y, k^{-1})$, respectively. Formally, we have that $D_{k^{-1}}(E_k(x)) = x$, meaning that decrypting an encrypted message, using the appropriate key, gives back the original message. When $k = k^{-1}$ the cryptosystem is symmetric, otherwise it is asymmetric. Every "good" cryptosystem should fulfill at least the following properties:

Property 1. the probability of encrypting and decrypting without knowing the encryption and decryption key is negligible.

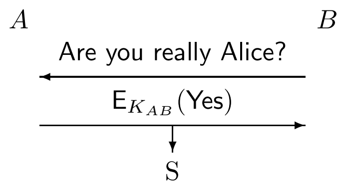
Property 2. the probability of finding the encryption and decryption keys by only observing a ciphertext is negligible.

In practice, there are no cryptosystems with an absolute proof of security in terms of the properties above. What can be proven are statements like "a given cryptosystem is secure if it is unfeasible to factor a large number", i.e., the security of cryptosystems is given in term of some well-known problem which is difficult to solve. We are not interested in giving more detail about cryptography. From now on we will assume the two properties above hold, and we invite the interested reader to refer to, e.g., [29, 25] for more detail about cryptography.

Replay Attacks Let us now do a first attempt to implement challenge-response in a very simple, unfortunately flawed, way. Let K_{AB} be a key shared between Alice and Bob and let E be a symmetric encryption function:



When Bob receives the response, he decrypts it through K_{AB} and checks the reply. Since only Alice and Bob know the secret key, Bob might deduce that this reply really comes from Alice. As a matter of fact, by property 1, the probability for an intruder to forge the right response is negligible. Moreover, property 2 guarantees that the secret key is not compromised by divulging encrypted messages. There is however a tremendous flaw in the protocol due to the fact that the challenge is always the same and so is the expected response: the intruder S, as done in the password scheme, just needs to intercept one Alice's reply and use it for later authentication.



With $S(A)$ we denote S impersonating A:

$S(A)$ $\qquad\qquad$ B

$\xleftarrow{\text{Are you really Alice?}}$

$\xrightarrow{\mathsf{E}_{K_{AB}}(\mathsf{Yes})}$

Notice that S is not breaking cryptography, but is just *replaying* a message previously sent by one of the honest parties. For this reason this kind of attacks are typically called *replay-attacks*.

So far, we have seen that having a response just depending on the secret is not of big help if it is always the same: intercepting it is equivalent to intercepting the secret and naturally leads to replay attacks. In order to make the response non-reusable we force it to be different at every protocol run. This is achieved by adding a so called *time-variant* parameter inside the response. Typical time-variant parameters are nonces, sequence numbers and time-stamps [25]. Here, for the sake of simplicity, we only consider nonces, i.e., numbers used only once that the verifier checks for avoiding replays of old messages. A simple way for implementing nonces is to use large enough random numbers that are used for just one session. This idea is better illustrated through an example:

A $\qquad\qquad$ B

generates a random N

$\xleftarrow{\text{Alice, please encrypt N}}$

$\xrightarrow{\mathsf{E}_{K_{AB}}(\mathsf{N})}$

decrypts and checks
that N is the same he sent

B decrypts the received message using K_{AB} and accepts only if the result is equal to nonce N. Let us see how this prevents replay attacks:

A $\qquad\qquad$ B

$\xleftarrow{\text{Alice, please encrypt N}}$

$\xrightarrow{\mathsf{E}_{K_{AB}}(\mathsf{N})}$

S

...

$S(A)$ $\qquad\qquad$ B

$\xleftarrow{\text{Alice, please encrypt N'}}$

$\xrightarrow{\mathsf{E}_{K_{AB}}(\mathsf{N})}$

B refuses since N $\neq$ N'

Since the response must contain the randomly generated nonce, S cannot reuse previously intercepted responses. To summarize, we always need that

– Challenge is time-variant;
– Response depends on both the challenge and the secret.

Reflection Attacks Even with time-variant challenges and responses, there is still a source of possible flaws due to the ambiguity of encrypted messages. Let us assume that the protocol above can indifferently be run by Alice and Bob to authenticate with Bob and Alice, respectively. In other words, we assume that the protocol can also be run with Alice as verifier and Bob as claimant, as illustrated below:

$$\begin{array}{ccc} A & & B \\ & \xrightarrow{\textsf{Bob, please encrypt N}} & \\ & \xleftarrow{\mathsf{E}_{K_{AB}}(\mathsf{N})} & \end{array}$$

In this (quite realistic) setting, it might happen that the same message is interpreted in two different ways, leading to a security breach. Let us see a concrete example of the so called *reflection-attack*:

$$\begin{array}{ccc} S(A) & & B \\ & \left.\begin{array}{c} \xleftarrow{\textsf{Alice, please encrypt N}} \\ \left.\begin{array}{c} \xrightarrow{\textsf{Bob, please encrypt N}} \\ \xleftarrow{\mathsf{E}_{K_{AB}}(\mathsf{N})} \end{array}\right\} \textsf{Run 2} \\ \xrightarrow{\mathsf{E}_{K_{AB}}(\mathsf{N})} \end{array}\right\} \textsf{Run 1} & \end{array}$$

The first and the last message are part of $\mathsf{Run\ 1}$ where A should authenticate with B. S succeeds impersonating A by starting a second run where he asks B to authenticate himself apparently with A. To do so S "reflects" in the second message the nonce N to B and obtains as a response the encryption $\mathsf{E}_{K_{AB}}(\mathsf{N})$. This can be reflected to B to conclude the first run where B asked A to encrypt N. There are thus two parallel sessions of the same protocol with exchanged roles. Notice that A might be even off-line, since all the encryptions are performed by B himself.

The flaw is based on the ambiguity of message $\mathsf{E}_{K_{AB}}(\mathsf{N})$ that does not clarify whether it is a response from Alice to Bob or from Bob to Alice. The use of symmetric key encryption contributes to this ambiguity since both Alice and Bob could have encrypted such a message.

The easiest way to make explicit an information is just to include it inside the encryption, e.g., indicating the claimant of the session:

$$\begin{array}{ccc} A & & B \\ & \xleftarrow{\textsf{Alice, please encrypt N}} & \\ & \xrightarrow{\mathsf{E}_{K_{AB}}(\mathsf{A,N})} & \end{array}$$

Bob now checks both the nonce and the identifier of Alice. If we try to mount the reflection attack we obtain the following execution:

$S(A)$ $\qquad\qquad\qquad\qquad$ B

$\xleftarrow{\textsf{Alice, please encrypt N}}$ (Run 1)

$\xrightarrow{\textsf{Bob, please encrypt N}}$ (Run 2)

$\xleftarrow{\mathsf{E}_{K_{AB}}(\mathsf{B,N})}$ (Run 2)

$\xrightarrow{\mathsf{E}_{K_{AB}}(\mathsf{B,N})}$ (Run 1)

B refuses since he expects A as claimant!

Thus, together with a time-variant challenge and a response that depends both on the challenge and on the secret, we also need that ciphertexts are explicit enough about who is the claimant and who is the verifier. In many cases, it is sufficient to specify one of the two, since the other one is deduced by the cryptographic key. For example, in the protocol above we have only specified the claimant since the verifier is implicitly the other entity sharing the key with the claimant (we are assuming that the key is shared between only two participants). Based on this notions it is possible to develop many protocol variants depending on which entity is specified (A or B) and, more interestingly, depending on which messages are encrypted, as discussed below.

2.3 Three classes of challenge-response protocols

Here, we extend the discussion above to asymmetric encryption and we write SK_A and PK_A to denote the private and the public key of entity A. We assume that only A knows her private key while every entity knows everyone else's public key; moreover all messages encrypted with SK_A and PK_A can be decrypted using PK_A and SK_A, respectively. As a consequence, everyone can generate ciphertext $\mathsf{E}_{\mathsf{PK}_A}(M)$ but only Alice can decrypt it; on the other hand, only Alice can generate ciphertext $\mathsf{E}_{\mathsf{SK}_A}(M)$, representing a digital signature, but every other entity can decrypt it.

We classify nonce-based protocols into three categories depending on what is encrypted and what is sent in clear [11, 8, 12].

Plain-Cipher (PC) B sends out the nonce in clear and receives it back encrypted together with a message which is authenticated. A proves her identity to B by showing the knowledge of the encryption key. An example is the protocol discussed so far, that we extend by adding a message M that Alice wants to authenticate with Bob:

A $\qquad\qquad\qquad\qquad$ B

$\xleftarrow{\mathsf{N}}$

$\xrightarrow{\mathsf{E}_{K_{AB}}(\mathsf{A,N,M})}$

This protocol authenticates A sending message M to B, since only A may have generated the ciphertext. The same effect can be achieved by using B in place of A in the ciphertext, or, in an asymmetric cryptosystem, using private Alice's key SK_A in place of K_{AB}. In this latter case it is necessary to specify B in place of A in the ciphertext, since the private key only indicates the claimant A.

Cipher-Plain (CP) B sends out the nonce encrypted and receives it back in clear. A proves her identity to B by showing the knowledge of the decryption key. For example:

$$A \xleftarrow{\mathsf{E}_{K_{AB}}(\mathsf{B},\mathsf{N},\mathsf{M})} B$$
$$A \xrightarrow{\quad\mathsf{N}\quad} B$$

This protocol authenticates A receiving message M from B, since only A may have decrypted the ciphertext. A similar effect is achieved by using A in place of B in the ciphertext, or, with asymmetric keys, using PK_A in place of the symmetric key K_{AB}.

Cipher-Cipher (CC) The nonce is sent out and received back encrypted. This is useful if both entities want to exchange messages. A proves her identity to B by showing the knowledge of either the encryption key (as in PC) or the decryption key (as in CP). For example:

$$A \xleftarrow{\mathsf{E}_{\mathsf{PK}_A}(\mathsf{B},\mathsf{M}_B,\mathsf{N})} B$$
$$A \xrightarrow{\mathsf{E}_{\mathsf{PK}_B}(\mathsf{A},\mathsf{M}_A,\mathsf{N})} B$$

This protocol authenticates A sending message M_A to B and receiving message M_B from B, as only A may have decrypted the first ciphertext. Again, the same effect may be achieved using either the private keys SK_B and SK_A, or a symmetric key K_{AB} in place of the two public keys.

Notice that the above mentioned categories are a generalization of POSH (Public Out Secret Home), SOPH (Secret Out Public Home) and SOSH (Secret Out Secret Home), introduced in [18]. We actually relax "Secret" into "Cipher" as handshakes may be composed of signed messages, guaranteeing integrity rather than secrecy. For example, as already noticed, PC includes protocols with a cleartext challenge and a signed response, which would not "fit well" into the POSH category since the nonce is not sent back as a secret. Notice also that PC and CP are called incoming and outgoing tests in [19].

2.4 A well known example: the Needham-Schroeder protocol

We illustrate a very well known case study that was discovered to be flawed by Gavin Lowe in [23]. Using the classification above it is rather easy to see how the protocol works and where is actually flawed. The original protocol is as follows:

$$
\begin{array}{ccc}
A & & B \\
 & \xrightarrow{\mathsf{E}_{\mathsf{PK}_B}(\mathsf{A},\mathsf{N}_A)} & \\
 & \xleftarrow{\mathsf{E}_{\mathsf{PK}_A}(\mathsf{N}_A,\mathsf{N}_B)} & \\
 & \xrightarrow{\mathsf{E}_{\mathsf{PK}_B}(\mathsf{N}_B)} &
\end{array}
$$

The aim is to achieve mutual authentication between A and B. To understand the protocol logic it is useful to decompose it into two (chained) challenge-response subprotocols, each achieving unilateral authentication. The first two messages are indeed a CC challenge-response in which Bob is authenticated to Alice:

$$
\begin{array}{ccc}
A & & B \\
 & \xrightarrow{\mathsf{E}_{\mathsf{PK}_B}(\mathsf{A},\mathsf{N}_A)} & \\
 & \xleftarrow{\mathsf{E}_{\mathsf{PK}_A}(\mathsf{N}_A,\mathsf{N}_B)} &
\end{array}
$$

Similarly, the second and the third message should authenticate Alice to Bob:

$$
\begin{array}{ccc}
A & & B \\
 & \xleftarrow{\mathsf{E}_{\mathsf{PK}_A}(\mathsf{N}_A,\mathsf{N}_B)} & \\
 & \xrightarrow{\mathsf{E}_{\mathsf{PK}_B}(\mathsf{N}_B)} &
\end{array}
$$

However, it is quite evident that this subprotocol does not specify B as verifier as done for A in the first subprotocol. This lack is indeed the source of the man-in-the-middle attack discovered in [23] and illustrated below:

$$
\begin{array}{ccccc}
A & & S(A) & & B \\
 & \xrightarrow{\mathsf{E}_{\mathsf{PK}_E}(\mathsf{A},\mathsf{N}_A)} & & & \\
 & & & \xrightarrow{\mathsf{E}_{\mathsf{PK}_B}(\mathsf{A},\mathsf{N}_A)} & \\
 & & & \xleftarrow{\mathsf{E}_{\mathsf{PK}_A}(\mathsf{N}_A,\mathsf{N}_B)} & \\
 & \xleftarrow{\mathsf{E}_{\mathsf{PK}_A}(\mathsf{N}_A,\mathsf{N}_B)} & & & \\
 & \xrightarrow{\mathsf{E}_{\mathsf{PK}_E}(\mathsf{N}_B)} & & & \\
 & & & \xrightarrow{\mathsf{E}_{\mathsf{PK}_B}(\mathsf{N}_B)} &
\end{array}
$$

We have a correct session between A and S which is exploited by S to impersonate A with B, on the right-hand side. The trick is that the response sent by B does not contain his name and can be thus forwarded by S to A in order to be decrypted. In the end, B is convinced that A has decrypted N_B for him but instead Alice has decrypted such a nonce for S. So, Bob authenticates Alice but Alice has started the protocol with the enemy S. How to patch the protocol should be clear from the discussion above, and we leave it as an exercise.

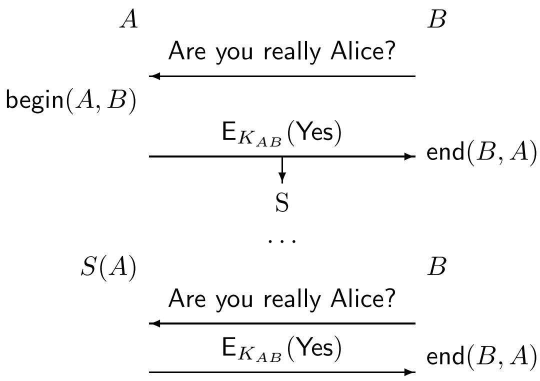
3 Authentication as Correspondence

In this section, we discuss how to formalize authentication as a property of protocol behaviour. Differently from, e.g., secrecy, authentication is not easily formulated as a property of the execution state. For example, for secrecy we can require that we never reach a state where secret data have been learnt by the intruder S. What about authentication? How can we formalize that S has impersonated Alice? A very powerful idea was proposed by Woo and Lam in [31]. Let us assume that the start and the termination of a protocol session generate two observable events, say $\mathsf{begin}(A, B)$ and $\mathsf{end}(B, A)$.

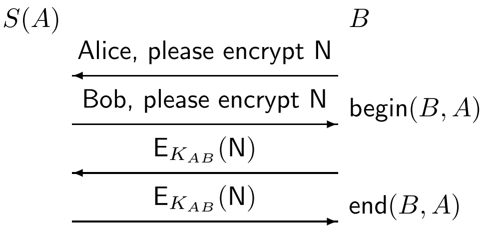
Definition 1 (Correspondence). *A protocol guarantees authentication of A with B iff every* $\mathsf{end}(B, A)$ *is matched by a previous distinguished* $\mathsf{begin}(A, B)$.

Intuitively, every time Bob authenticates Alice, then Alice should have started a session with Bob. We see how this formalization works by reconsidering the previously illustrated flaws.

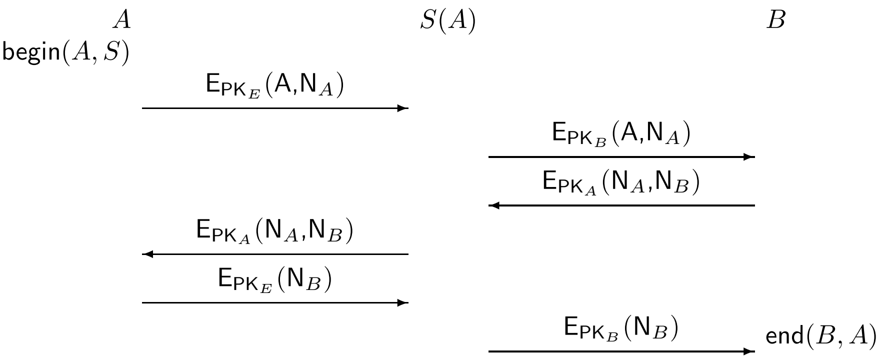
A reply attack is characterized by many end's with just one begin. In fact, messages from one session are replayed in order to impersonate Alice:



A reflection attack is instead captured since we have a $\mathsf{end}(B, A)$ with no matching $\mathsf{begin}(A, B)$, but with $\mathsf{begin}(B, A)$ instead. As a matter of fact, A might be even off-line during the attack, and a parallel run of B is exploited to mount the attack:



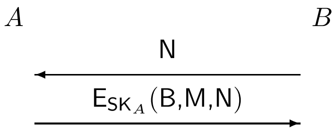
The man-in-the-middle attack is captured since end(B, A) is not matched by the required begin(A, B), but by begin(A, S) instead. Alice is running with S and this session is used to impersonate her with Bob:



In the next section we show how this dynamic correspondence property may be proved statically, i.e., by only inspecting the protocol code.

4 Static Analysis

Developing static analyses for authentication protocols is particularly difficult since authentication is typically provided by encrypting/decrypting messages which are composed of different parts, each of them providing a "piece" of the authentication guarantee. As an example, consider the following PC protocol:



in which Bob is sending to Alice a random nonce N which is signed by Alice together with a message M and Bob identifier B. Let us recall how each message component of $\mathsf{E}_{\mathsf{SK}_A}(\mathsf{B},\mathsf{M},\mathsf{N})$ contributes in achieving authentication: N is needed to guarantee *freshness*, i.e., $\mathsf{E}_{\mathsf{SK}_A}(\mathsf{B},\mathsf{M},\mathsf{N})$ cannot be a replay of an old protocol session since N is used only once; B specifies the intended receiver of M and the signature guarantees that A is the sender.

Our approach, first introduced in [10], is based on considering a "minimal" set of authentication *patterns* that allow parties to authenticate. Through suitable tags, we make explicit in the encrypted message which pattern it correspond to. For example message $\mathsf{E}_{\mathsf{SK}_A}(\mathsf{B},\mathsf{M},\mathsf{N})$ is tagged as $\mathsf{E}_{\mathsf{SK}_A}(\mathsf{Id}(\mathsf{B}),\mathsf{Auth}(\mathsf{M}),\mathsf{Verif}_{\mathsf{PC}}(\mathsf{N}))$ to denote the fact that the nonce N is authenticating M to the verifier B in a PC nonce handshake. When a message is decrypted, tags are recognized by the party that may exploit them to achieve/provide authentication guarantees.

The advantages of this tag-based approach may be summarized as follows: (*i*) it is extremely compositional: tags constitute a common trusted "interface" that distributed parties may use to authenticate each other; the correct use of tags is enforced locally (through suitable typing rules) and correctness is preserved when communicating parties are concurrently executed; (*ii*) this strong form of compositionality allows to safely mix different protocols once their sequential components are type-checked; thus, our tagging discipline naturally scales to multi-protocol settings; (*iii*) the fact that tags correspond to a small set of a-priori selected patterns, makes the type system quite simple and easy to use; in [15] we have implemented a tool that is able to automatically infer tags and types needed to validate protocols, thus obtaining a completely automated validation technique for protocol with unbounded number of sessions and participants; (*iv*) even if the set of authentication patterns we have selected is small, it is expressive enough to capture many of the protocols in literature; this gives also new insights on which are the basic mechanisms for guaranteeing authentication.

In this section we give an overview of the technique. In particular, Section 4.1 presents the ρ-spi calculus, a simple language for specifying cryptographic protocols, Section 4.2 illustrates the types-and-effect system and Section 4.3 discusses the main safety results. More detail can be found in [8, 12].

4.1 A simple calculus for protocols

The ρ-spi calculus derives from the spi calculus [2], and inherits many of the features of *Lysa* [5], a version of the spi calculus proposed for the analysis of authentication protocols. ρ-spi differs from both calculi in several respects: it incorporates the notion of tagged message exchange from [9], it provides new authentication-specific constructs, and offers primitives for declaring process identities and keys.

The syntax is reported in Table 1 and described below. Patterns, denoted by $\mathcal{M}, \mathcal{K}$, are recursively defined over names, variables, public and private keys, tagged patterns and pairs We presuppose two countable sets: $\mathcal{N}$ of names and $\mathcal{V}$ of variables. We reserve a, b, k, n for names and x, y, z for variables, with m ranging over both names and variables. Identities $\mathcal{ID}$ are a subset of names and are ranged over by I and J. Identities are further partitioned into *trusted principals* $\mathcal{ID}_{\mathcal{P}}$, ranged over A and B, and *enemies* $\mathcal{ID}_{\mathcal{E}}$, ranged over by E. The pair composed by a public key and the corresponding private one is noted by $\mathsf{Pub}(m)$, $\mathsf{Priv}(m)$, similarly to [2]. In the rest of the paper, we will use the

Notation: $TAG \in \{\mathsf{Id}, \mathsf{Auth}, \mathsf{Verif}_H, \mathsf{Claim}_H | H = \mathsf{PC}, \mathsf{CP}, \mathsf{CC?}, \mathsf{CC!}\}$
$m \in \mathcal{N} \cup \mathcal{V}$

$\mathcal{M}, \mathcal{K}$ ::=	*Patterns*	P, Q ::=	*Processes*	
a, b, k, n	names	$I \triangleright S$		(principal)
x, y, z	variables	$I \triangleright !S$		(replication)
$\mathsf{Pub}(m)$	public key	$P \mid Q$		(composition)
$\mathsf{Priv}(m)$	private key	$\mathsf{let}\ k = \mathsf{sym\text{-}key}(I_1, I_2).P$		(symmetric-key)
$TAG(\mathcal{M})$	tagged pattern	$\mathsf{let}\ k = \mathsf{asym\text{-}key}(I).P$		(asymmetric-key)
$(\mathcal{M}_1, \mathcal{M}_2)$	pair			

S ::= *Sequential Processes*	
$\mathbf{0}$	(nil)
$\mathsf{new}(n).S$	(restriction)
$\mathsf{in}(\mathcal{M}).S$	(input)
$\mathsf{out}(\mathcal{M}).S$	(output)
$\mathsf{encrypt}\{\mathcal{M}\}_{\mathcal{K}}\ \mathsf{as}\ x.S$	(symmetric encryption)
$\mathsf{encrypt}\{\|\mathcal{M}\|\}_{\mathcal{K}}\ \mathsf{as}\ x.S$	(asymmetric encryption)
$\mathsf{decrypt}\ x\ \mathsf{as}\ \{\mathcal{M}\}_{\mathcal{K}}.S$	(symmetric decryption)
$\mathsf{decrypt}\ x\ \mathsf{as}\ \{\|\mathcal{M}\|\}_{\mathcal{K}}.S$	(asymmetric decryption)
$\mathsf{begin}(I_1, I_2, \mathcal{M}_1; \mathcal{M}_2).S$	(begin)
$\mathsf{end}(I_1, I_2, \mathcal{M}_1; \mathcal{M}_2).S$	(end)

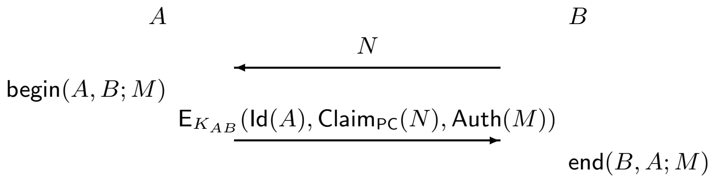
Table 1. The syntax of ρ-spi calculus.

following notation convention: $\overline{\mathsf{Pub}} = \mathsf{Priv}$ and vice-versa. Tags, denoted by TAG, are a special category of names. They specify the role of each message component. Specifically: all identifiers relevant to authentication are tagged by Id; messages that should be authenticated are tagged by Auth; finally, nonces are tagged by Verif_H or Claim_H, with $H \in \{\mathsf{PC}, \mathsf{CP}, \mathsf{CC!}, \mathsf{CC?}\}$. Such tags specify the role played by the entity tagged by Id (verifier or claimant) and the kind of nonce hand-shake. Notice that in CC nonce handshakes we distinguish challenge from response ciphertexts, denoted by $\mathsf{CC?}$ and $\mathsf{CC!}$, respectively.

Processes (or *protocols*), ranged over by P, Q, are the parallel composition of principals. Each principal is a sequential process associated with an identity I, noted $I \triangleright S$. The replicated form $I \triangleright !S$ indicates an arbitrary number of copies of $I \triangleright S$. In order to allow the sharing of keys among principals, we provide ρ-spi with let-bindings: $\mathsf{let}\ k = \mathsf{sym\text{-}key}(I_1, I_2).P$ declares (and binds) the long-term key k shared between I_1 and I_2 in the scope P. Similarly, $\mathsf{let}\ k = \mathsf{asym\text{-}key}(I).P$ declares, and binds in the scope P, the key pair $\mathsf{Pub}(k)$, $\mathsf{Priv}(k)$ associated to I.

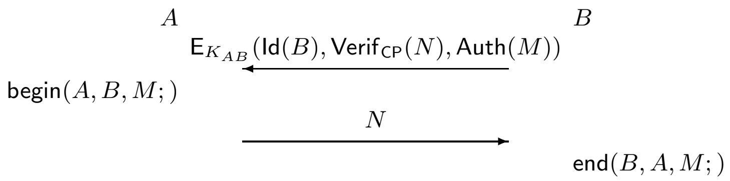
Sequential processes may never fork into parallel components: this assumption helps assign unique identities to (sequential) processes, and involves no significant loss of expressive power as protocol principals are typically specified as sequential processes, possibly sharing some long-term keys. The sequential process $\mathbf{0}$ is the null process that does nothing, as usual. Process $\mathsf{new}(n).S$ generates a fresh name n local to S. We presuppose a unique (anonymous) public channel, the network, from/to which all principals, including intruders, read/send messages. Similarly to *Lysa*, our input primitive may (atomically) test part of the message read, by pattern-matching. If the input message matches the pattern, then the variables occurring in the pattern are bound to the remaining subpart of the message; otherwise the message is not read at all. For example, process '$\mathsf{in}(\mathsf{Claim}(x)).P$' may only read messages of the form $\mathsf{Claim}(M)$, binding x to M in P. Encryption just binds x to the encrypted message, while decryption checks if the message contained in x matches the form $\{\mathcal{M}\}_{\mathcal{K}}$ (or $\{|\mathcal{M}|\}_{\mathcal{K}}$), i.e., the payload matches $\mathcal{M}$ and is encrypted with the appropriate key. Only in this case x is decrypted and the variables in the pattern $\mathcal{M}$ get bound to the decrypted messages. Similarly to the input primitive, decryption may also test part of the decrypted messages by pattern-matching mechanism. Finally, the $\mathsf{begin}(I_1, I_2, \mathcal{M}_1; \mathcal{M}_2).S$ and $\mathsf{end}(I_2, I_1, \mathcal{M}_1; \mathcal{M}_2).S$ primitives are used to check the *correspondence assertions* [31]. The former primitive declares that I_1 is beginning a protocol session with I_2 for confirming the reception of message $\mathcal{M}_1$ and authenticating message $\mathcal{M}_2$; the latter one indicates that I_2 is ending a protocol session with I_1 getting confirmation from I_1 of the reception of message $\mathcal{M}_1$ and authenticating message $\mathcal{M}_2$.

Example 1. To illustrate the use of tags and correspondence assertions, let us consider the protocols presented so far. The first protocol of Section 2.3 can be decorated with tags and correspondence assertions as follows:



The tagged structure specifies that that the nonce N is used by the verifier B for authenticating M with A in a PC nonce hand-shake. According to the correspondence assertions, at the end of the protocol B authenticates A *sending* message M.

The second protocol of Section 2.3 can be decorated as follows:



Indeed, the nonce N is used by the verifier B for authenticating M with A in a CP nonce handshake. According to the correspondence assertions, at the end of the protocol B authenticates A *receiving* message M.

$$
\begin{array}{rl}
\mathit{Protocol} \triangleq & \mathsf{let}\ k_A = \mathsf{asym\text{-}key}(A)\ .\ (B \triangleright !\mathit{Initiator} \mid A \triangleright !\mathit{Responder}) \\
\mathit{Initiator} \triangleq & \mathsf{new}(n).\mathsf{out}(n).\mathsf{in}(z). \\
 & \qquad \mathsf{decrypt}\ z\ \mathsf{as}\ \{|\mathsf{Id}(B), \mathsf{Auth}(x), \mathsf{Verif}_{\mathsf{PC}}(n)|\}_{\mathsf{Pub}(k_A)}.\mathsf{end}(B, A; x).\mathbf{0} \\
\mathit{Responder} \triangleq & \mathsf{in}(x).\mathsf{new}(m).\mathsf{begin}(A, B; m). \\
 & \qquad \mathsf{encrypt}\ \{|\mathsf{Id}(B), \mathsf{Auth}(m), \mathsf{Verif}_{\mathsf{PC}}(x)|\}_{\mathsf{Priv}(k_A)}\ \mathsf{as}\ z.\mathsf{out}(z).\mathbf{0}
\end{array}
$$

Table 2. PC Protocol in ρ-spi calculus

Finally, the third protocol of Section 2.3 can be decorated as follows:

$$
\begin{array}{lcr}
A & & B \\
 & \xleftarrow{\quad \mathsf{E}_{\mathsf{PK}(A)}(\mathsf{Id}(B), \mathsf{Auth}(M_B), \mathsf{Verif}_{\mathsf{CC?}}(N)) \quad} & \\
\mathsf{begin}(A, B, M_B; M_A) & & \\
 & \xrightarrow{\quad \mathsf{E}_{\mathsf{PK}(B)}(\mathsf{Id}(A), \mathsf{Auth}(M_A), \mathsf{Claim}_{\mathsf{CC!}}(N)) \quad} & \\
 & & \mathsf{end}(B, A, M_B; M_A)
\end{array}
$$

Notice that the use of ? and ! in the nonce tag disambiguates whether the ciphertext is used in a CC handshake as challenge or response. By an inspection of the correspondence assertions, at the end of the protocol B authenticates A receiving message M_B and sending message M_A. □

Example 2. To illustrate the ρ-spi calculus syntax, let us consider the following public key variant of the first protocol in Example 1.

$$
\begin{array}{lcr}
A & & B \\
 & \xleftarrow{\qquad\qquad \mathsf{N} \qquad\qquad} & \\
 & \xrightarrow{\qquad \mathsf{E}_{\mathsf{SK}_A}(\mathsf{B,M,N}) \qquad} &
\end{array}
$$

The difference is that A signs the response instead of using symmetric cryptography. The ρ-spi calculus specification is in Table 2. After declaring the key pair for A, an unbounded number of instances of B as initiator and an unbounded number of instances of A as responder are run in parallel. The *Initiator* B generates a fresh nonce and sends it in clear on the network. Then it reads a message from the network and tries to decrypt it with the public key of the responder A and checks that its own identifier tagged by Id is the first component of the message payload and the nonce tagged by $\mathsf{Verif}_{\mathsf{PC}}$ is fresh, i.e., it is the one just generated. If this is the case, B authenticates A sending message x, namely the message tagged by Auth. The *Responder* A receives a nonce from the network, generates a new message m, declares the start of the session with the initiator B for authenticating message m, signs m together with the responder identifier and the nonce (all message components are tagged) and sends the obtained ciphertext on the network. □

Operational Semantics. We define the operational semantics of ρ-spi in terms of *traces*, after [7]. A trace is a possible sequence of *actions* performed by a process. Each process primitive has an associated action and we denote with *Act* the set of all possible actions. The dynamics of the calculus is formalized by means of a transition relation between *configurations*, i.e., pairs $\langle s, P\rangle$, where $s \in Act^*$ is a trace, P is a (closed) process. Each transition $\langle s, P\rangle \to \langle s :: \alpha, P'\rangle$ simulates one computation step in P and records the corresponding action in the trace.

Principals do not directly synchronize with each other. Instead, they may receive from the unique channel an arbitrary message M known by the environment, which models the Dolev-Yao intruder: the environment knows all the identity labels, the messages sent on the network, the content of ciphertexts whose decryption key is known, ciphertexts created by its knowledge and all the keys declared as owned by E together with all the public keys. Finally, it may create fresh names not appearing in the trace. The transition relation is given in detail in [8, 12].

Definition 2 (Traces). *The set $T(P)$ of traces of process P is the set of all the traces generated by a finite sequence of transitions from the configuration $\langle \epsilon, P\rangle$: $T(P) = \{s \mid \exists P' \ s.t. \ \ \langle \epsilon, P\rangle \to^* \langle s, P'\rangle\}$*

The notion of safety extends the standard *correspondence* property of [24, 31] by distinguishing between received and sent messages.

Definition 3 (Safety). A trace s is safe if and only if whenever $s = s_1 :: end(B, A, M_1; M_2) :: s_2$, then $s_1 = s_1' :: begin(A, B, M_1; M_2) :: s_1''$, and $s_1' :: s_1'' :: s_2$ is safe. A process P is safe if, $\forall s \in T(P)$, s is safe.

A trace is *safe* if every $end(B, A, M_1; M_2)$ action is preceded by a distinct $begin(A, B, M_1; M_2)$. Intuitively, this guarantees that whenever B authenticates A receiving M_1 and sending M_2, then A has received M_1 and has sent M_2 in a protocol session with B.

Example 3. To illustrate the semantics of the calculus and the notion of safety, let us consider a (flawed) simplification of the protocol of Example 2, obtained by eliminating the nonce.

$$A \xrightarrow{\quad \mathsf{E}_{\mathsf{SK}_A}(\mathsf{B,M}) \quad} B$$

The ρ-spi calculus specification may be easily obtained from the one of the original protocol in Table 2:

$$\begin{array}{ll} \mathit{Initiator}_f & \triangleq \mathsf{in}(z).\mathsf{decrypt}\ z\ \mathsf{as}\ \{|B, x|\}_{\mathsf{Pub}(k_A)}.\mathsf{end}(B, A; x).\mathbf{0} \\ \mathit{Responder}_f & \triangleq \mathsf{new}(m).\mathsf{begin}(A, B; m).\mathsf{encrypt}\ \{|B, m|\}_{\mathsf{Priv}(k_A)}\ \mathsf{as}\ z.\mathsf{out}(z).\mathbf{0} \end{array}$$

This protocol suffers of the standard replay attack where S impersonates A by just replaying a previously intercepted message, which can be observed through the following execution trace:

$$\begin{array}{l} asym-key(k_A, A) :: new(m) :: \mathbf{begin}(A, B; m) :: \\ encrypt\{|B, m|\}_{\mathsf{Priv}(k_A)} :: out(\{|B, m|\}_{\mathsf{Priv}(k_A)}) :: \\ in(\{|B, m|\}_{\mathsf{Priv}(k_A)}) :: decrypt\{|B, m|\}_{\mathsf{Priv}(k_A)} :: \mathbf{end}(B, A; m) :: \\ in(\{|B, m|\}_{\mathsf{Priv}(k_A)}) :: decrypt\{|B, m|\}_{\mathsf{Priv}(k_A)} :: \mathbf{end}(B, A; m) \end{array}$$

Notice that the same message $\{B, m\}_{\mathsf{Priv}(k_A)}$ is read twice by two different instances of the Responder. This causes two ends with just one begin, thus making this trace unsafe. The presence of the nonce repairs the protocol avoiding this replay attack. □

4.2 ρ-spi calculus types and effects: an overview

Types in ρ-spi calculus regulate the use of terms in the authentication task and are reported below.

$$\begin{array}{lll} T ::= & \mathsf{SharedKey}(I, J) & \text{symmetric key} \\ & \mathsf{Key}(I) & \text{asymmetric seed} \\ & \mathsf{PublicKey}(I) & \text{public key} \\ & \mathsf{PrivateKey}(I) & \text{private key} \\ & \mathsf{Un} & \text{untrusted} \\ & \mathsf{Nonce}^{\ell}(I, J) & \text{nonce} \\ & \mathsf{Enc}(e; f) & \text{ciphertext} \end{array}$$

Key types aim at preserving key secrecy and regulating the correct use of keys: a long-term key shared between I and J has type $\mathsf{SharedKey}(I, J)$ and it can only be used by I and J; a name used for creating a key pair owned by I has type $\mathsf{Key}(I)$; the public and private keys generated by that name have type $\mathsf{PublicKey}(I)$ and $\mathsf{PrivateKey}(I)$, respectively. While private keys can only be used by their owners, public keys are available to every principal. Every untagged term potentially known by the enemy has type Un.

Nonce and ciphertext types are more specifically related to authentication: A nonce used by I and J has type $\mathsf{Nonce}^{\ell}(I, J)$, where $\ell \in \{Un, Ciph, Priv, Int\}$ specifies secrecy and integrity properties of the nonce: it is Un when the nonce is sent in clear on the network (in PC challenges and CP responses), $Ciph$ when the nonce is sent encrypted but it is supposed to be sent back in clear (CP challenges), $Priv$ when the nonce secrecy is preserved by the hand-shake (CC hand-shakes using public or symmetric keys) and Int when the integrity of the nonce is guaranteed by signatures (CC hand-shakes using private keys). Ciphertexts encrypted by trusted principals have type $\mathsf{Enc}(e; f)$, where e and f are effects conveyed or required but such a ciphertext when it is decrypted and encrypted. Effects have the following syntax:

$$\begin{array}{lll} t ::= & \mathsf{fresh}^{\ell}(n) & \text{nonce freshness} \\ & [?|!]\mathsf{Chall}^{\ell}_{N}(I, J, M) & \text{challenge} \\ & [?|!]\mathsf{Resp}^{\ell}_{N}(I, J, M) & \text{response} \end{array}$$

If the ciphertext is a challenge sent by I to J containing the nonce N and the message M, then it will have type $\mathsf{Enc}(\mathsf{Chall}^{\ell}_{N}(I,J,M),\_)$, Similarly, if the ciphertext is a response sent by I to J containing the nonce N and the message M, then the type will be $\mathsf{Enc}(\_,\mathsf{Resp}^{\ell}_{N}(I,J,M))$. So, in $\mathsf{Enc}(e,f)$, e is used for challenge effects and f for response effects. The crucial point is that these ciphertext types can be reconstructed by the tags, i.e., tagged messages convey enough information to decide whether they are challenges or responses, who are the parties involved, which is the nonce, which are the authenticated messages and which kind of challenge-response mechanism is used (label $\ell \in \{\mathit{Un}, \mathit{Ciph}, \mathit{Priv}, \mathit{Int}\}$).

The intuitive meaning of the remaining effects is given below:

- The atomic effect $\mathsf{fresh}^{\ell}(n)$ tracks the freshness of the nonce n. A nonce is fresh if it is new and it has not yet been used for justifying an end event.
- The atomic effect $?\mathsf{Chall}^{\ell}_{N}(I,J,M)$ tracks the decryption of a ciphertext representing a challenge, containing the nonce N and the message M, sent by I to J. Similarly for the atomic effect $?\mathsf{Resp}^{\ell}_{N}(I,J,M)$.
- The atomic effect $!\mathsf{Chall}^{\ell}_{N}(I,J,M)$ tracks the generation of a challenge ciphertext, containing the nonce N and the message M, sent by I to J.
- The atomic effect $!\mathsf{Resp}^{\ell}_{N}(I,J,M)$, instead, *enables* the generation of a ciphertext representing a response, containing the nonce N and the message M, sent by I to J.

We illustrate, through a simple example, the use of types and effects. Let us consider a CC handshake based on symmetric cryptography, allowing the initiator B to authenticate the responder A receiving message M_C and sending message M_R. The narration of the protocol, decorated with correspondence assertions, types and effects, is depicted in Table 3. Protocol code is written close to $\bullet$'s and message exchanges (left/right arrows), the nonce type is declared inside the new command, and the effects are enclosed in frame boxes placed aside down-arrows. The exchanged messages are not completely specified, since we want to give an idea of the general scheme adopted by the static analysis.

The initiator B generates a fresh nonce n of type $\mathsf{Nonce}^{\ell}(B,A)$: the atomic effect $\mathsf{fresh}^{\ell}(n)$ tracks the freshness of the nonce. The corresponding rule is

$$\frac{B;\Gamma, n:\mathsf{Nonce}^{\ell}(B,A) \vdash S : e}{B;\Gamma \vdash \mathsf{new}(n:\mathsf{Nonce}^{\ell}(B,A)).S : e - [\mathsf{fresh}^{\ell}(n)]}$$

Notice that judgements have the form $I;\Gamma \vdash S : e$ meaning that "the sequential process S can be typed according to the typing environment Γ and the effect e when executed by I". So in the example above, $\mathsf{new}(n:\mathsf{Nonce}^{\ell}(B,A)).S$ is typed under $B;\Gamma$ if the continuation process S is typed under the same environment enriched with nonce type $n:\mathsf{Nonce}^{\ell}(B,A)$ and with the effect e possibly containing $\mathsf{fresh}^{\ell}(n)$. (Notice that writing $e - [\mathsf{fresh}^{\ell}(n)]$ in the thesis of the rule allows effect $\mathsf{fresh}^{\ell}(n)$ to be in e without forcing its presence, thus allowing to type-check protocols that declare nonces without checking them.)

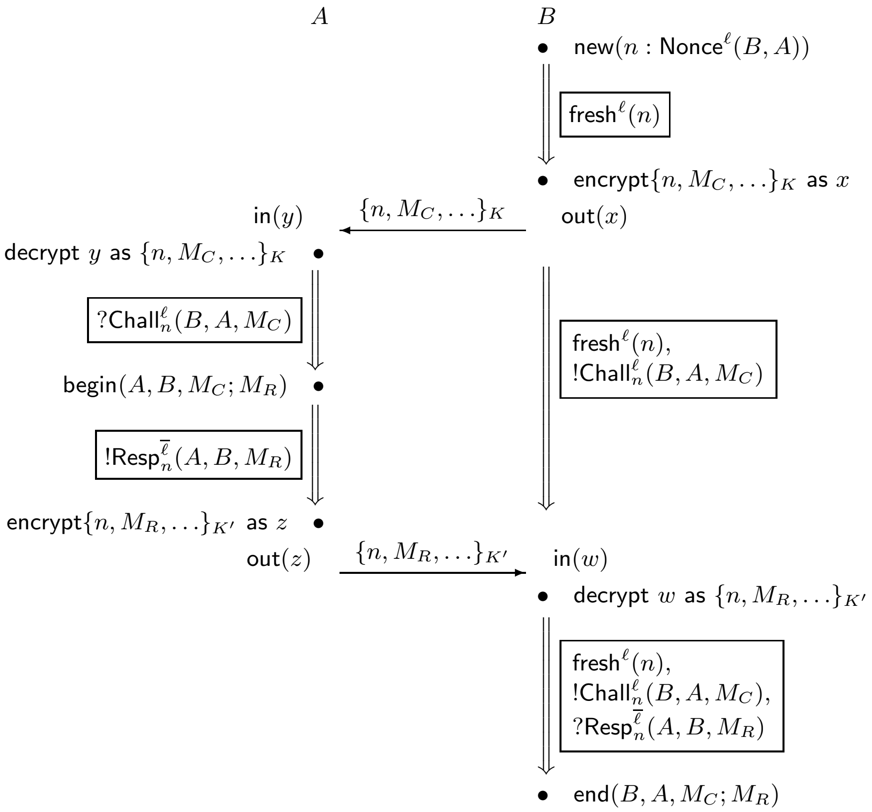


Table 3. The use of types and effects.

The nonce is encrypted together with message M_C in a challenge sent by B to A (effect $!\mathsf{Chall}^\ell_n(B, A, M_C)$). This atomic effect is derived according to the tagged structure attached to the ciphertext, and omitted here for simplicity. We just assume that tags are such that the message can be recognized to be a challenge from B to A for authenticating message M_C, i.e., the type of the message is $\mathsf{Enc}(\mathsf{Chall}^\ell_n(B, A, M_C), \_)$. The effect $\mathsf{Chall}^\ell_n(B, A, M_C)$ conveyed by this type is "transferred" to the continuation process in the form $!\mathsf{Chall}^\ell_n(B, A, M_C)$, via the following rule:

$$\frac{B; \Gamma \vdash M : \mathsf{Enc}(e_C; e_R) \qquad B; \Gamma, z : \mathsf{Un} \vdash S : e + !e_C}{B; \Gamma \vdash \mathsf{encrypt}\ M\ \mathsf{as}\ z.S : e + !e_R}$$

The effect $!\mathsf{Chall}^\ell_n(B, A, M_C)$ records the fact the challenge has been generated. Similarly, the ciphertext is received and decrypted by the responder A producing an effect $?\mathsf{Chall}^\ell_n(B, A, M_C)$.

$$\begin{array}{rl} \mathit{Protocol} \triangleq & \mathsf{let}\ k_A = \mathsf{asym\text{-}key}(A)\ .\ (B \triangleright !\mathit{Initiator} \mid A \triangleright !\mathit{Responder}) \\ \mathit{Initiator} \triangleq & \mathsf{new}(n : \mathsf{Un}).\mathsf{out}(n).\mathsf{in}(z). \\ & \quad \mathsf{decrypt}\ z\ \mathsf{as}\ \{|\mathsf{Id}(B), \mathsf{Auth}(x), \mathsf{Verif}_{\mathsf{PC}}(n)|\}_{\mathsf{Pub}(k_A)}.\mathsf{end}(B, A; x).\mathbf{0} \\ \mathit{Responder} \triangleq & \mathsf{in}(x).\mathsf{new}(m : \mathsf{Un}).\mathsf{begin}(A, B; m). \\ & \quad \mathsf{encrypt}\ \{|\mathsf{Id}(B), \mathsf{Auth}(m), \mathsf{Verif}_{\mathsf{PC}}(x)|\}_{\mathsf{Priv}(k_A)}\ \mathsf{as}\ z.\mathsf{out}(z).\mathbf{0} \end{array}$$

Table 4. Typed PC Protocol in ρ-spi calculus

The following $\mathsf{begin}(A, B, M_C; M_R)$ assertion requires a challenge containing the message M_C (effect $?\mathsf{Chall}_n^{\ell}(B, A, M_C)$) and justifies a response for authenticating M_R (effect $!\mathsf{Resp}_n^{\overline{\ell}}(A, B, M_R)$), as formalized by the corresponding typing rule:

$$\frac{A; \Gamma \vdash S : e + [!\mathsf{Resp}_N^{\overline{\ell}}(A, I, M_2)] \qquad \Gamma \vdash N : \mathsf{Nonce}^{\ell}(I, A) \qquad \Gamma \vdash M_2 : T}{A; \Gamma \vdash \mathsf{begin}(A, I, M_1; M_2).S : e + [?\mathsf{Chall}_N^{\ell}(I, A, M_1)]}$$

The relation between ℓ and $\overline{\ell}$ determines how the nonce may be sent out and received back by the originator: the nonce may be sent out in clear and received back encrypted in PC handshakes, thus $\overline{\mathit{Un}} = \mathit{Ciph}$, or vice-versa in CP handshakes, formalized as $\overline{\mathit{Ciph}} = \mathit{Un}$. Finally, in CC handshakes the nonce may be either sent out and received back encrypted ($\overline{\mathit{Priv}} = \mathit{Priv}$) or signed ($\overline{\mathit{Int}} = \mathit{Int}$).

The atomic effect $!\mathsf{Resp}_n^{\overline{\ell}}(A, B, M_R)$ enables A to generate a ciphertext with type $\mathsf{Enc}(\_; [\mathsf{Resp}_n^{\overline{\ell}}(A, B, M_R)])$. Hence the ciphertext is received and decrypted by B: such a decryption is tracked by $?\mathsf{Resp}_n^{\overline{\ell}}(A, B, M_R)$, as done before.

Notice that the effects on B's side describe the generation of a fresh challenge ($!\mathsf{Chall}_n^{\ell}(B, A, M_C)$ and $\mathsf{fresh}^{\ell}(n)$) and the reception of a corresponding response based on the same nonce ($?\mathsf{Resp}_n^{\overline{\ell}}(A, B, M_R)$). This is all we need to correctly conclude the protocol through $\mathsf{end}(B, A, M_C; M_R)$. This is formalized by the typing rule for end:

$$\frac{B; \Gamma \vdash S : e \qquad \Gamma \vdash n : \mathsf{Nonce}^{\ell}(B, A)}{B; \Gamma \vdash \mathsf{end}(B, A, M_1; M_2).S : e + [!\mathsf{Chall}_n^{\ell}(B, A, M_1), ?\mathsf{Resp}_n^{\overline{\ell}}(A, B, M_2), \mathsf{fresh}^{\ell}(n)]}$$

This rule "consumes" the relative effects thus guaranteeing the injectivity of authentication: every end is matched by a distinct begin.

Notice that the only human effort when typing a protocol is to provide the correct tags and the nonce types. For example, the simple protocol of Table 2 can be type-checked by just adding types declaration inside the primitive new, as shown in Table 4. The nonce type for this example is simply Un, representing the fact that the challenge is sent as cleartext. This fits the general scheme explained above because of the type equality $\mathsf{Nonce}^{\mathsf{Un}}(I, J) = \mathsf{Un}$, i.e., an untrusted value is also an untrusted nonce between every possible parties I and J.

4.3 Safety and Compositionality

Our main result states that if a process can be typed with empty effect and by only assigning type Un to all the identities, then every trace generated by that process is safe.

Theorem 1 (Safety [8, 12]). *Let P be a process. If $\overline{I} : \mathsf{Un} \vdash P : []$, where $\overline{I}$ are the identities in P, then P is safe.*

Interestingly, our analysis is strongly compositional, as stated by the following theorem. Let $\mathbf{keys}(k_1, \ldots, k_n)$ denote a sequence of key declarations.

Theorem 2 (Strong Compositionality [8, 12]). *Let P be the protocol of the form $\mathbf{keys}(k_1, \ldots, k_n).(I_1 \triangleright !S_1 | \ldots | I_m \triangleright !S_m)$ and $I_1, \ldots, I_m$ be the identities in P. Then*

$$I_1, \ldots, I_m : \mathsf{Un} \vdash P : []$$

$$\textit{iff}$$

$$I_1, \ldots, I_m : \mathsf{Un} \vdash \mathbf{keys}(k_1, \ldots, k_n).I_i \triangleright !S_i : [] \qquad \forall i \in [1, m]$$

Intuitively, a protocol is safe if so are all the protocol participants. In addition, judging a participant safe only requires knowledge of the long-term keys it shares with other participants. This is a fairly mild assumption as the information conveyed by the keys is relative to identities of the parties sharing them, not to the protocol they are running. This flexibility has a price, however, in that our result relies critically on the run-time checks on the message tags provided by pattern-matching.

5 Related Work

Tagging is not a new idea and it is proposed and used for verification purposes in [3, 4, 16, 17, 22]. Typically, tagging amounts to add a different label to each encrypted protocol message, so that ciphertexts cannot be confused. Our tagging is less demanding, as we do not require that every message is unambiguously tagged since we tag only certain components. In particular, for protocols implemented with stronger tagging techniques, our tags can be safely removed without compromising the protocols' safety.

The Strand Spaces formalism [19–21, 30] is an interesting framework for studying authentication. There are interesting similarities between our analysis and the way the three kinds of nonce-handshakes are checked in Strand Spaces. It would be interesting to explore how our type system could be applied in such a framework, in order to provide mechanical proofs of safety.

The recent work by Bodei *et al.* on a control-flow analysis for message authentication in *Lysa* [5, 6] is also strongly related to our present approach. The motivations and goals, however, are different, since message authentication concerns the origin of a message while agreement provides guarantees about the presence in the current session of the claimant and its willingness to authenticate with the verifier.

Finally, [11] compares our type and effect system with the one by Gordon and Jeffrey, drawing on a translation of tagged protocols, validated by our system, into protocols that type check with Gordon and Jeffrey's system. The paper shows that tags discussed in this paper can be compiled even in the static types of [18]. This allows the tag inference procedure of [15] to be exploited for inferring such types.

References

1. M. Abadi and B. Blanchet. Secrecy types for asymmetric communication. *Theor. Comput. Sci.*, 298(3):387–415, 2003.
2. M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1–70, 1999.
3. M. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, 1996.
4. B. Blanchet and A. Podelski. Verification of cryptographic protocols: Tagging enforces termination. In *Proceedings of Foundations of Software Science and Computation Structures*, pages 136–152, 2003.
5. C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. Riis Nielson. Automatic validation of protocol narration. In *Proceedings of the 16th IEEE Computer Security Foundations Workshop (CSFW'03)*, pages 126–140. IEEE Computer Society Press, June 2003.
6. C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. Riis Nielson. Control flow analysis can find new flaws too. In *Proceedings of the Workshop on Issues on the Theory of Security (WITS'04)*, ENTCS. Elsevier, 2004.
7. M. Boreale. Symbolic trace analysis of cryptographic protocols. In *Proceedings of ICALP 01*, volume 2076, pages 667–681. LNCS 2076, Springer Verlag, 2001.
8. M. Bugliesi, R. Focardi, and M. Maffei. Dynamic types for authentication (full version). Submitted for Publication.
9. M. Bugliesi, R. Focardi, and M. Maffei. Principles for entity authentication. In *Proceedings of 5th International Conference Perspectives of System Informatics (PSI 2003)*, volume 2890 of *Lecture Notes in Computer Science*, pages 294–307. Springer-Verlag, July 2003.
10. M. Bugliesi, R. Focardi, and M. Maffei. Compositional analysis of authentication protocols. In *Proceedings of European Symposium on Programming (ESOP 2004)*, volume 2986 of *Lecture Notes in Computer Science*, pages 140–154. Springer-Verlag, 2004.
11. M. Bugliesi, R. Focardi, and M. Maffei. Analysis of typed-based analyses of authentication protocols. In *Proceedings of 18th IEEE Computer Security Foundations Workshop (CSFW 2005)*. IEEE Press, 2005. To appear.
12. M. Bugliesi, R.Focardi, and M.Maffei. Authenticity by tagging and typing. In *2nd ACM Workshop on Formal Methods in Security Engineering: From Specifications to Code (FMSE 2004), ISBN 1-58113-971-3*, pages 1–12. ACM press, October 2004.
13. J. Clark and J. Jacob. A survey of authentication protocol literature: Version 1.0. http://www.cs.york.ac.uk/~jac/papers/drareview.ps.gz, November 1997.

14. R. Focardi, R. Gorrieri, and F. Martinelli. Non interference for the analysis of cryptographic protocols. In *Proceedings of ICALP'00*, pages 354–372. Springer LNCS 1853, July 2000.
15. R. Focardi, M. Maffei, and F. Placella. Inferring authentication tags. In *Proceedings of IFIP WG 1.7, ACM SIGPLAN and GI FoMSESS Workshop on Issues on the Theory of Security (WITS 2005)*. ACM Digital Library, January 2005.
16. A. Gordon and A. Jeffrey. Authenticity by typing for security protocols. In *Proceedings of 14th IEEE Computer Security Foundations Workshop (CSFW'01)*, pages 145–159. IEEE Computer Society Press, June 2001.
17. A. Gordon and A. Jeffrey. Types and effects for asymmetric cryptographic protocols. In *Proceedings of 15th IEEE Computer Security Foundations Workshop (CSFW'02)*, pages 77–91. IEEE Computer Society Press, 24-26 June 2002.
18. A. Gordon and A. Jeffrey. Types and effects for asymmetric cryptographic protocols. *Journal of Computer Security*, 12(3/4):435–484, 2004.
19. J. D. Guttman and F. J. Thayer Fàbrega. Authentication tests and the structure of bundles. *Theoretical Computer Science*, 283(2):333–380, 2002.
20. J. D. Guttman, F. J. Thayer, J. A. Carlson, J. C. Herzog, J. D. Ramsdell, and B. T. Sniffen. Trust management in strand spaces: a rely-guarantee method. In *Proceedings of European Symposium on Programming (ESOP 2004)*, volume 2986 of *Lecture Notes in Computer Science*, pages 325–339. Springer-Verlag, 2004.
21. Joshua D. Guttman and F. Javier Thayer. Protocol independence through disjoint encryption. In *Proceedings of 13th IEEE Computer Security Foundations Workshop (CSFW'00)*, pages 24–34. IEEE Computer Society Press, July 2000.
22. J. Heather, G. Lowe, and S. Schneider. How to prevent type flaw attacks on security protocols. In *Proceedings of 13th IEEE Computer Security Foundations Workshop (CSFW'00)*, pages 255–268. IEEE Computer Society Press, July 2000.
23. G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer-Verlag, 1996.
24. G. Lowe. "A Hierarchy of Authentication Specification". In *Proceedings of the 10th Computer Security Foundation Workshop (CSFW'97)*, pages 31–44. IEEE Computer Society Press, 1997.
25. A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.
26. J. C. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using murϕ. In *Proceedings of the 1997 IEEE Symposium on Research in Security and Privacy*, pages 141–153. IEEE Computer Society Press, 1997.
27. R. M. Needham and M. D. Schroeder. Authentication revisited. *ACM SIGOPS Operating Systems Review*, 21(1):7–7, 1987.
28. L. C. Paulson. Relations between secrets: Two formal analyses of the yahalom protocol. *Journal of Computer Security*, 9(3):197–216, 2001.
29. Douglas R. Stinson. *Cryptography, Theory and Practice.* CRC Press, 1995.
30. J. Thayer, J. Herzog, and J. Guttman. Strand spaces: Proving security protocols correct. Journal of Computer Security, 1999. 7(2/3).
31. T.Y.C. Woo and S.S. Lam. "A Semantic Model for Authentication Protocols". In *Proceedings of 1993 IEEE Symposium on Security and Privacy*, pages 178–194, 1993.