

# A Magic Approach to the Analysis of Security Protocols

A. Bracciali, A. Brogi, G. Ferrari, E. Tuosto

*Department of Computer Science*

*University of Pisa, Italy*

{braccia, brogi, giangi, etuosto}@di.unipi.it

## Abstract

The paper proposes a novel, semi-automatic approach to the analysis of security protocols. The verification of security protocols is reduced to the verification of correctness properties in component-based software development.

We use  $\pi$ -calculus for describing the interactive behaviour of components. Protocol principals are then described as interactive components that employ cryptographic primitives. Integrity and secrecy properties of protocols are expressed and verified in terms of the possibility that a malicious intruder successfully joins a context of components. The process is semi-automated in the sense that the verification phase requires the explicit specification of an instance of the protocol that “magically” respects the desired properties.

## 1 Introduction

Networked heterogeneous applications are becoming the primary part of modern software environments nowadays. These applications (e.g., as web-browsers for cellular phones) require stationary servers and mobile devices to cooperate and exchange services while the application is running. The dynamic integration of network services is a crucial aspect in this scenario.

Modern programming techniques adopt the abstraction of components to encapsulate services inside a black-box which has a standard interface which reflects service behavioral features through suitable interaction protocols. The services offered by a component are described by means of interface description languages. Recent developments have focussed on extending these languages so as to allow the specification of the interaction protocols of components. Service integration is obtained by means of composition languages (also called coordination languages) that describe the way in which components are glued together.

Networked applications strongly emphasize the problem of security in software composition. Indeed components may be malicious and may have been designed to steal and/or corrupt information on the hosts they will run. The

problem of specifying and enforcing security policies has been extensively studied. Many approaches aim at providing secure communication channels by means of protocols that exploit shared-key or public-key encryption to guarantee secrecy, authentication and integrity of messages. Different approaches have been proposed to support the formal reasoning on security protocols and proving their correctness. The “logic of protocols” [4], inductive methods based on higher order logic [17], finite state analysis and model checking [15, 11], and process algebraic techniques [1, 10] are some examples.

In this paper we propose a novel, semi-automatic approach to the analysis of security protocols. The verification of security protocols is reduced to the verification of correctness properties in component-based software development.

Our starting point is the notion of *interaction pattern*, originally introduced in [3] to describe the interactive behaviour of a software component. The interaction pattern of a component is expressed with a subset of the  $\pi$ -calculus [16], which provides a suitable abstraction for describing interactive behaviours. Differently from the standard  $\pi$ -calculus, interaction patterns permit us to express only finite fragments of the full behaviour of components. This simplification aims at (locally) reasoning about properties of both static and dynamic component composition. A set of interaction patterns forms a context which may evolve either because of interactions occurring within the context, or because a new component joins the context. The main interest of the overall setting described in [3] is that it supports the efficient verification — both statically and dynamically — of several properties of open interacting systems.

In this paper, we extend the notion of interaction pattern of [3] with the aim of developing a unifying methodology for the verification of both security protocols and secure software compositions. Our approach can be summarized as follows:

- We extend the notion of interaction pattern by introducing cryptographic primitives in the style of the spi-calculus of Abadi and Gordon [1]. These are used to provide authentication, to distribute keys, and more generally to ensure various forms of secrecy. Protocol principals are then described as interactive components that employ cryptographic primitives.
- Integrity and secrecy properties of protocols are expressed and verified in terms of the possibility that a malicious intruder successfully joins a context of components.

The main contribution of this paper is the development of a proof technique to verify correctness of security protocols, and, therefore correctness of the secure composition of components. Indeed, in this paper we focus only on the specification of the security requirements of composition. Nonetheless, it is worth underlining that the whole calculus obtained permits us to uniformly treat both composition and security issues by means of the same framework.

The proof technique is based on the idea of *magic instance* originally introduced in [1]. The verification procedure consists of two steps:

1. Protocol correctness is reduced to the problem of ensuring the integrity of certain critical data exchanged among principals. In other words, the chosen data fully characterize the properties of the protocol we want to verify. The protocol where such data are fixed is called the *magic instance*.
2. Construction of an intruder (i.e. a principal), if it exists, which violates the integrity of the chosen data. The relative magic instance plays a crucial role in detecting such violation.

Given the specifications of a protocol  $\mathcal{P}$  and of the security properties to be verified (“ $\mathcal{P}$ ”), the verification process reduces to determining whether there may exist an intruder component capable of violating the security of the protocol. More precisely, we will show that a protocol  $\mathcal{P}$  is not secure w.r.t. the properties specified by “ $\mathcal{P}$ ” if there exists an intruder that can drive  $\mathcal{P}$  to successful termination while failing to do the same with “ $\mathcal{P}$ ”. In other words, we will reduce the verification process to proving the termination of a context of components.

Most importantly, we will present an effective algorithm capable of verifying whether a protocol  $\mathcal{P}$  satisfies a set of security properties “ $\mathcal{P}$ ”. The algorithm (nondeterministically) tries to construct an intruder violating the security of the protocol, and returns it if and only if one such intruder exists. The proof of the correctness of the algorithm is included.

The rest of the paper is organized as follows. Section 2 introduces syntax and semantics of the calculus for security protocols that we will use. Section 3 describes how security properties can be reduced to verifying the termination of “magic” contexts of components. Section 4 is devoted to present the algorithm for verifying the security property of a protocol and to prove its correctness. Section 5 contains a discussion of related work, while Section 6 contains some concluding remarks.

## 2 A calculus for security protocols

In this section we introduce the calculus we use for studying secure composition of components. We present a fragment of a more general calculus that has been designed to reason about the construction of (open) distributed systems and which belongs to the  $\pi$ -calculus family, [3].

The calculus focuses on the features which are necessary for expressing security protocols and on formal mechanisms for reducing protocol correctness to coordination and termination of components.

### 2.1 Syntax

Components, i.e. the principals of protocols, communicate along the only public channel *net*, which belongs to the name space of each principal. Every data sent over *net* can be received by any of the principals. We consider only deterministic non-recursive protocols, and each principal is represented by a sequence

of synchronous communication actions, called *behavioural expression*. The data that can be communicated between principals consist of:

- Names ( $N$ ): The distinguished element  $net$  and infinitely many others, that are indicated by  $n, m, o, \dots$ . In the full calculus names represent (private) communication channels, here, they only play the role of generic basic data.
- Keys ( $K$ ): Infinitely many denoted by  $k_1, k_2, \dots$ . They permit the encryption and decryption of data. Even if it would not be difficult to deal with both symmetric and asymmetric keys, for simplicity, we consider only symmetric keys: Two principals must know the appropriate key to communicate each other safely.
- Principals ( $P$ ): Infinitely many denoted by  $A, B, S, I, \dots$
- Variables ( $X$ ): Infinitely many denoted by  $w, x, y, z, \dots$

**Definition 1 (Principals)** *A principal is represented by a closed<sup>1</sup> expression of the form:  $[E]$ , where  $E$ , the behavioural expression of the principal, is defined by the following grammar:*

$$\begin{aligned}
 E & ::= 0 \mid \alpha.E \mid E \parallel E \mid E + E \\
 \alpha & ::= in(net, D) \mid out(net, D) \mid \tau \\
 D & ::= N \mid K \mid P \mid X \mid \{D\}_D \mid (D, D).
 \end{aligned}$$

As a matter of notation, to explicitly name a principal, we write  $A \triangleq [E]$ .

A behavioural expression may consist of the no action 0 or of an action  $\alpha$  prefixed (“.”) to a behavioural expression, or it may be made out of the parallel composition ( $\parallel$ ) or the non-deterministic choice (+) composition of two sub-expressions. Prefix, parallel composition and non-deterministic choice are the usual composition operators present in process algebras. Two general remarks are worth being made about the features of this calculus and its usage:

- Principals can express only behaviours consisting of a finite number of actions, there are not recursive constructs. This choice simplifies the problem of reasoning about properties of protocols.
- All the protocols we intend to analyse are finite sequential and deterministic, i.e. the operators  $\parallel$  and  $+$  are not present in their expressions. This choice covers anyway a wide class of interesting protocols and allows for reasoning about them in a more effective way than considering the class of all the possible protocols.

---

<sup>1</sup>As more precisely explained in the following of this section, free and bound variables and closed expressions correspond to the expected notions.

Communication actions send or receive data  $d$  over the channel  $net$ . The choice of making explicit the only channel along which communication occurs, viz.  $net$ , is motivated by the compatibility with the full calculus, where principals (i.e. components) can communicate over different (private) channels.

Communication actions encapsulate the encryption and decryption mechanisms: As it will be made clear by the semantical rule for communication, the attempt to receive an encrypted data by means of a given key,  $in(net, \{x\}_k)$ , will succeed only if the key results the correct one, otherwise a deadlock occurs. On the other hand, it is also possible to receive an encrypted data as it is,  $in(net, x)$ , without attempting to decrypt it. This is at the basis of the protocol verification technique proposed, which reduces “protocol security” to “correct termination”.

As usual, the action  $\tau$  represents an internal step of computation.<sup>2</sup> Data, in turn, can be names, keys, variables, encrypted data and pair of data. The syntax of the language permits us to write “wrong” terms, like for example  $in(net, \{k\}_{(m,n)})$  which would represent the attempt to receive from  $net$  a key encrypted by means of a pair of names. These are cases of wrong coding and resolve in run-time errors since no type discipline is provided: The computation will deadlock since no rule applies to such configuration.

We say that a variable  $x$  occurs in *object position* in a datum  $d$ ,  $x \in obj(d)$ , if  $x$  is never used as encryption key in  $d$ . For instance, in  $\{x\}_y$ ,  $x$  is in object position, while  $y$  is not. A variable  $x$  occurs *bound* in a behavioural expression  $E$  if it is under the scope of an action  $in(net, d)$  and  $x$  is in object position in  $d$ . A variable  $x$  is free in  $E$  if there is in  $E$  an occurrence of  $x$  that is not bound. We denote with  $obj(in(net, d))$  the set of variables occurring in object position in  $d$ . With  $fv(E)$  we denote the set of free variables of  $E$ . Given an output or silent action  $\alpha$ ,  $obj(\alpha)$  is void. The expression  $E$  of a principal is closed in the sense that it does not contain free variables. From the fact that a behavioural expression of a principal is closed and from the fact that  $in(net, \{y\}_x)$  does not bind  $x$ , it follows that in every attempt of receiving (and decrypting) an encrypted data, the key ( $x$  in this case) must be given (instantiated). Substitution, sometime written  $[v/x]$ , works as expected.

Different principals can communicate with each other when inserted in the same context. In this sense, a protocol is represented by a context containing all the principals involved in the protocol.

**Definition 2 (Context)** *A context  $\mathcal{C}$  is a multi-set of principals sharing the channel name  $net$ .*

**Example 1** *We now show how a simple protocol can be formalised using our calculus. The protocol is the Wide Mouthed Frog [4] protocol. We consider the simplified version as described in [1].*

<sup>2</sup>Using the calculus in its full generality, the silent action is used to model, together with the choice and prefix operators, a local choice: While  $E + F$  can behave either like  $E$  or  $F$  depending on its environment,  $\tau.E + \tau.F$  autonomously decides which one of the two possibility to execute.

Suppose that two principals  $A$  and  $B$  share private keys  $k_{as}$  and  $k_{bs}$  with a (trusted) server  $S$ . When  $A$  wants to communicate (securely) a name  $n$  to  $B$  on the public (i.e. non-secure) channel net, she creates a new key  $k$ , sends it encrypted under  $k_{as}$  to  $S$  and then  $S$  forwards  $k$  to  $B$  encrypted under  $k_{bs}$ . The protocol may be informally described as:

- (1)  $A \rightarrow S : \{k\}_{k_{as}}$
- (2)  $S \rightarrow B : \{k\}_{k_{bs}}$
- (3)  $A \rightarrow B : \{n\}_k$

The protocol is essentially formed by steps (1) and (2). Step (3) is not needed and is reported only to show a possible use of the Wide Mouthed Frog protocol, namely,  $A$  authenticates herself to  $B$  via  $S$ , then  $B$  accepts to communicate with  $A$ . In our case, the protocol is coded as follows:

$$\begin{aligned}
A &\triangleq [\text{out}(\text{net}, \{k\}_{k_{as}}).\text{out}(\text{net}, \{n\}_k).0] \\
S &\triangleq [\text{in}(\text{net}, \{x\}_{k_{as}}).\text{out}(\text{net}, \{x\}_{k_{bs}}).0] \\
B &\triangleq [\text{in}(\text{net}, \{y\}_{k_{bs}}).\text{in}(\text{net}, \{w\}_y).0] \\
\text{WMF} &\triangleq \{A, S, B\}
\end{aligned}$$

We can see that  $A$  generates a new key  $k$ , sends it to  $S$  then sends the ciphertext  $\{n\}_k$  to  $B$ .  $B$  waits for a message that must be formed by data encrypted with the key  $k_{bs}$ . Then  $B$  waits for a second message cryptogram whose encryption key is the clear text obtained in the first input. As it will be clear from the semantics, at this point  $B$  can only receive a message encrypted by the key  $k$ , otherwise he is stuck. Notice that in the instantiation of the protocol  $\text{WMF}$ , the keys  $k_{as}$  and  $k_{bs}$  are not known outside the intended principals.

## 2.2 Semantics

The semantics of the calculus is given in terms of a labeled transition system,  $(\rightarrow)$ , defined up to structural congruence<sup>3</sup> (Definition 3). The labeled transition system models the “stand alone” behaviour of a principal, i.e. how it can evolve according to its behavioural expression. Moreover, a reduction semantics,  $(\mapsto)$ , models communication of principals inside a context. Note how parallel components of the same principal can not communicate with each other.

**Definition 3**  $\rightarrow$

$$\begin{array}{c}
\frac{}{\alpha.E \xrightarrow{\alpha} E} \text{ (in)} \qquad \frac{E \xrightarrow{\alpha} E'}{E + F \xrightarrow{\alpha} E'} \text{ (+)} \\
\\
\frac{E \xrightarrow{\alpha} E'}{E \parallel F \xrightarrow{\alpha} E' \parallel F} \text{ (||)} \text{ where } \text{obj}(\alpha) \cap \text{fv}(F) = \emptyset
\end{array}$$

<sup>3</sup>Structural congruence,  $\equiv$ , is obtained by extending  $\alpha$ -conversion with  $\parallel$  and  $+$  as monoidal operators, whose neutral element is 0.

We define now how principals can communicate inside a context. The joining of a principal in a context will be subject to the respect of some conditions of correctness. Intuitively speaking, a principal can access a context only if it does not spoil the (possible) termination of the context itself.<sup>4</sup>

**Definition 4**  $\hookrightarrow$

$$\frac{E \xrightarrow{in(net,d')} E' \quad F \xrightarrow{out(net,d)} F' \quad d'\sigma = d}{\{[E], [F]\} \cup \mathcal{C} \hookrightarrow \{[E'\sigma], [F']\} \cup \mathcal{C}} \text{ (comm)}$$

$$\frac{E \xrightarrow{\tau} E'}{\{[E]\} \cup \mathcal{C} \hookrightarrow \{[E']\} \cup \mathcal{C}} (\tau)$$

where  $\sigma$  is a ground substitution. Rule *(comm)* says that two principals can communicate inside a context if they can perform complementary input/output actions. The input data, appropriately instantiated, must equal - via a ground substitution - the output data. This mechanism encompasses decryption. As an example, consider the actions  $in(net, \{x\}_k)$  and  $in(net, x)$ , both of them can match the action  $out(net, \{m\}_k)$ . The first one expects to receive data encrypted with the key  $k$  (which, as said, can not be an uninstantiated variable), and hence  $\sigma = [m/x]$ , and the variable  $x$  assumes the decrypted value  $m$ . The second one expects a generic message,  $\sigma = [\{m\}_k/x]$ , but the message has not been decrypted. As a last example, consider the case of  $in(net, \{x\}_{k'})$ , with  $k \neq k'$ : Communication can not happen with the previous action  $out(net, \{m\}_k)$ , since, for no  $\sigma$ ,  $\{x\}_{k'}\sigma = \{m\}_k$  holds.

The rule  $(\tau)$  models the internal action of a principal.

**Definition 5 (Successful context)** A context  $\mathcal{C} = \{[E_1], \dots, [E_n]\}$ , is successful if and only if  $\forall i \in [1, \dots, n] : E_i \equiv 0$ .

A context is *may-correct* if it is either successful or if exists at least one possible evolution which leads to a successful context.

**Definition 6 (May-correct context)** A context  $\mathcal{C}$  is may-correct if and only if either:

- (a)  $\mathcal{C}$  is successful, or
- (b)  $\exists \mathcal{C}' : \mathcal{C} \hookrightarrow \mathcal{C}'$ ,  $\mathcal{C}'$  is may-correct.

**Definition 7 (Compatibility)** Given a context  $\mathcal{C}$ , a principal  $[E]$  is compatible with  $\mathcal{C}$ , or  $\mathcal{C}$ -compatible iff

$$\mathcal{C} \text{ is may - correct} \Rightarrow \mathcal{C}' = \{[E]\} \cup \mathcal{C} \text{ is may - correct}$$

---

<sup>4</sup>Note that, differently from the calculus presented here, where the channel *net* is always available to all the principals in a context, in the full calculus with private names the access will be subject also to some notion of sharing of the appropriate private channels.

At this point we may like to extend the  $\hookrightarrow$  relation: A context can evolve also because of a new principal joining it.

**Definition 8**  $\hookrightarrow$

$$\frac{[E] \text{ is } \mathcal{C} \text{ - compatible} \quad \wedge \quad \mathcal{C}' = \{[E]\} \cup \mathcal{C}}{\mathcal{C} \hookrightarrow \mathcal{C}'} \text{ (join)}$$

This operation, permits to uniformly model both the construction of a context by joining into it all the principals involved, and the dynamics of an open system, where components can join in it at run time.

### 3 Security properties and “magic” contexts

We now illustrate a methodology for verifying security properties of protocols. The methodology is based on the notion of *magic instance*, originally introduced in [1].

Given a protocol  $\mathcal{P}$ , a *magic instance* of  $\mathcal{P}$ , written “ $\mathcal{P}$ ”, is an instance in which some variables have been fixed to ground values. Informally speaking, all possible successful executions of a secure protocol  $\mathcal{P}$  in an untrusted environment should assign the fixed values to the selected data, in spite of any possible interference of an intruder.

We then reduce the problem of verifying the security properties of a protocol to verify the compatibility of intruders with both the protocol and its magic version. Namely:

$\mathcal{P}$  is not secure w.r.t. the properties defined by “ $\mathcal{P}$ ”.

$$\Leftrightarrow \tag{1}$$

$\exists I : I \text{ is } \mathcal{P}\text{-compatible and } I \text{ is not “}\mathcal{P}\text{”-compatible}$

In other words, if the protocol is unsecure, an intruder  $I$  may succeed in cheating a principal of  $\mathcal{P}$  by forging a different value from the expected one, still preserving the termination of  $\mathcal{P} \cup I$ . We require that any intruder can not crash the protocol: If an intruder joins a may-correct context where the protocol is executed, the resulting context must remain may-correct. On the other hand,  $I$  cannot cheat the magic version of principals that (magically) knows the correct values and hence are not able to receive a different data (“ $\mathcal{P}$ ”  $\cup I$  deadlocks).

The methodology consists of two parts:

1. *Expressing security properties in terms of expected values*

The idea is that a “secure” protocol guarantees the integrity of some of the data exchanged during its execution: A secure protocol does not permit anybody to access in any way such data, and in particular it does not permit anybody to modify them. In this sense, the security of a protocol is

reduced to the identity of some data in the communication of the protocol. Determining the proper set of data and their values in a way that their integrity implies the security property desired is, therefore, the crucial task. This task is not formalised in our approach and it is demanded to the expertise of who designs and tests the security protocols.

To give a flavor of this step of the methodology, let us consider the simple protocol represented by the following principals:

$$\begin{aligned} A &\triangleq [\text{out}(\text{net}, k).\text{out}(\text{net}, \{n\}_k).0] \\ B &\triangleq [\text{in}(\text{net}, x).\text{in}(\text{net}, \{x\}_y).0] \end{aligned}$$

and suppose we want to check if the secrecy of  $k$  is preserved (that is evidently false). In this case, requiring that the value of the variable  $x$  in the first input action of  $B$  is exactly  $k$ , independently from any possible interference in the execution of the protocol, would guarantee, for example, that nobody would be able to forge a different key for  $B$ . The magic instance is:

$$\begin{aligned} A &\triangleq [\text{out}(\text{net}, k).\text{out}(\text{net}, \{n\}_k).0] \\ \text{"B"} &\triangleq [\text{in}(\text{net}, k).\text{in}(\text{net}, \{x\}_k).0] \end{aligned}$$

## 2. Constructing an intruder, if any, which violates the expected values

Once the security of the protocol is expressed in terms of the integrity of some data, we can verify whether there may exist an intruder which can violate the integrity of the data by (maliciously) interfering with the protocol. We propose a proof technique to verify the existence of an intruder. We assume the perfect encryption hypothesis, meaning that the cryptographic system underlying the protocol is assumed to be “non-attackable”. In particular, this implies that keys cannot be guessed or deduced (even if a large number of messages is collected by an intruder). Moreover, the Dolev-Yao model [7] of intruder is assumed: An intruder can potentially interfere with all the communications happening over the untrusted channel  $\text{net}$ . An example of an intruder for the previous protocol is:

$$I \triangleq [\text{in}(\text{net}, u).\text{in}(\text{net}, \{v\}_u).\text{out}(\text{net}, k').\text{out}(\text{net}, \{m\}'_k).0]$$

It is easy to note that  $\{I, A, B\}$  is a may-correct context, while the magic instance  $\{I, A, \text{"B"}\}$  deadlocks after the first two inputs of  $I$ , since the semantic rule (*comm*) guarantees, by the pattern matching mechanism, that the only output action that matches  $\text{in}(\text{net}, k)$  is  $\text{out}(\text{net}, k)$ .

Having illustrated the ideas on which our methodology is based, we can now formally define the notion of *magic instance*.

**Definition 9 (Magic-instance)** Let  $\mathcal{P}$  be a given context. “ $\mathcal{P}$ ” is a magic-instance of  $\mathcal{P}$  if it is obtained from  $\mathcal{P}$  by completely instantiating one or more of the variables in  $\mathcal{P}$ . A variable  $x$  is completely instantiated to a value  $v$  if the substitution  $[v/x]$  has been applied to all the occurrences of the variable bound by the same input action, included the occurrences in the input action itself.

We conclude this section with a brief discussion about the expressiveness of our approach. An important problem is to understand which is the class of security properties that our methodology can deal with. In general, all security protocols rely on exchanged data. For instance, a principal may deduce identity of partners by considering the data received when executing an authentication protocol. Hence, our methodology is suitable for stating and verifying *integrity* and *secrecy properties*. We can also deal with secrecy when it relies on integrity of particular data.

As far as *authentication* is concerned, we plan to extend the approach. Indeed, authentication properties are usually based on freshness of nonces or session keys. Moreover, many authentication protocols adopt a public key cryptographic infrastructure. We intend to extend our calculus also to public key cryptography. We conjecture that the full calculus may be suitably used for this task. Indeed, the *join* construct may describe the fact that the intruder “convince”  $A$  to start a protocol session with him. More precisely, in a public key protocol, a principal potentially may interact with anyone; the join operation may link together  $A$  and an incoming intruder by modeling the acquisition of the intruder’s public key by joining an *open key* of  $A$  with an *open key* of the intruder.

## 4 From the magic to the algorithm

This section is devoted to present an effective algorithm capable of deciding whether a protocol  $\mathcal{P}$  is secure w.r.t. the properties defined by its magic version “ $\mathcal{P}$ ”. Given a protocol  $\mathcal{P}$  and its magic version “ $\mathcal{P}$ ”, the algorithm

- either returns an intruder that is compatible with  $\mathcal{P}$  and not compatible with “ $\mathcal{P}$ ”,
- or it returns **none** if an intruder does not exist.

According to (1), in the first case the protocol  $\mathcal{P}$  is not secure, while in the second case  $\mathcal{P}$  is secure.

We will present a non-deterministic version of the algorithm. The algorithm tries to incrementally construct an intruder that is compatible with  $\mathcal{P}$  and not compatible with “ $\mathcal{P}$ ”. The (possibly) generated intruder will consist of a sequence of input and output communication operations. Such a sequence is incrementally built by matching each time one communication action that can be performed by one of the principals of protocol  $\mathcal{P}$ . Intuitively speaking, the matching communications are constructed so as to let the intruder interact with some principal of  $\mathcal{P}$  but not with the corresponding principal in “ $\mathcal{P}$ ”. This

is achieved by enforcing the values to be communicated to be different from the values expected/sent by “ $\mathcal{P}$ ”, whenever we are considering one action whose magic instance contains a value that has been fixed.

The algorithm tries to non-deterministically construct a malicious intruder. The non-determinism of the algorithm originates from the choice of the communication action of  $\mathcal{P}$  to be matched, and from the choice of the value to be communicated by the intruder. To reduce the non-deterministic branching, the algorithm only analyses intruders which consist of a sequence of communication actions. Moreover, the considered intruders keep on matching all output operations performed by the principals until there is no output communication to match. Only then, the intruders output some value to let the interaction proceed. At the end, the algorithm either returns an intruder that is compatible with  $\mathcal{P}$  and not compatible with “ $\mathcal{P}$ ”, or it returns `none` if it non-deterministically fails to find it. (The correctness of the algorithm will be discussed later.)

#### 4.1 The algorithm

We assume that all bound variables are distinct. The first step of the algorithm consists of transforming the magic version “ $\mathcal{P}$ ” of the protocol into a constraint. Namely the algorithm determines a constraint  $\chi$  of the form:

$$(x_1 \neq v_1) \vee (x_2 \neq v_2) \vee \dots \vee (x_n \neq v_n).$$

Each constraint  $(x_i \neq v_i)$  in  $\chi$  refers to a variable  $x_i$  of  $\mathcal{P}$  which is completely instantiated to the value  $v_i$  in “ $\mathcal{P}$ ”. Intuitively speaking, these constraints will be used to let the intruder output values which are different from the values expected from the magic version “ $\mathcal{P}$ ”. We call  $Extract(\mathcal{P}, \text{“}\mathcal{P}\text{”})$  the function that, given a protocol  $\mathcal{P}$  and a magic instance “ $\mathcal{P}$ ”, returns the set of constraints  $\chi$ . More precisely, the construction of the constraint  $\chi$  is performed as follows:

1. Let  $W$  be an initially empty set of constraints.
2. For each variable  $x$  in  $\mathcal{P}$  which is completely instantiated to a value  $v$  in “ $\mathcal{P}$ ”, add the constraint  $(x = v)$  to  $W$ .
3.  $\chi$  is the constraint obtained by negating the conjunction of the constraints in  $W$ , that is:

$$\chi = \neg\left(\bigwedge_{w \in W} w\right)$$

After converting the magic version “ $\mathcal{P}$ ” of the protocol into the constraint  $\chi$ , the algorithm starts to try to (non-deterministically) construct an intruder that is compatible with  $\mathcal{P}$  and not compatible with “ $\mathcal{P}$ ”.

The pseudo-code of the algorithm is reported in Table 1. The construct `choice()` is used to express the non-deterministic choice of a value from a set. The effect of executing `choice(S)` is the creation of one parallel instance of the

algorithm for each value of  $S$ . The command **return**( $R$ ) is supposed to stop the program and to return the result  $R$ . The function **nvar**() returns a new variable. The algorithm either returns an intruder or it returns **none** if all its non-deterministic instances return **none**.

For the sake of generality, the algorithm is parameterized w.r.t. the initial knowledge  $\kappa$  that malicious intruders may have before starting to interact with the protocol. Such knowledge represents data —such as keys, nonces, or cryptograms— that are not used within the protocol and that intruders may possess. We will denote by  $\partial(\kappa)$  the minimal set of data that the intruder can deduce from a knowledge  $\kappa$  so that:

$$\begin{aligned} k, \{d\}_k \in \partial(\kappa) &\implies d \in \partial(\kappa) \\ d, k \in \partial(\kappa) &\implies \{d\}_k \in \partial(\kappa) \\ d, d' \in \partial(\kappa) &\implies (d, d') \in \partial(\kappa) \\ (d, d') \in \partial(\kappa) &\implies d, d' \in \partial(\kappa) \end{aligned}$$

We now outline the steps of the algorithm. At each iteration of the main loop, the outputs offered by principals of  $\mathcal{P}$  are collected (line 1) in the set  $Out$ .

- Line 1 constructs  $\chi$ .
- If  $Out$  is not empty, one of its outputs is non-deterministically selected and removed from  $\mathcal{P}$  (lines 4-5); the corresponding input actions are added to  $I$  and  $\kappa$  is augmented by the datum of the output action (lines 6-7).
- If  $Out$  is void, the inputs are collected in the set  $In$  (line 8) and, if no input is possible, then the algorithm ends returning **none** (line 10). If  $In$  contains some input, one of them is chosen non-deterministically (line 11), the set  $\Delta$  of data derivable from  $\kappa$  that satisfy  $\chi$  and match the value in the chosen input is determined (line 12). If  $\Delta$  is void then **none** is returned (line 14), otherwise a value is selected (lines 15). The matched input is removed from  $\mathcal{P}$  —as a synchronization had taken place— and its object variables are instantiated in its continuation (line 16). The intruder is augmented by the output action determined (line 17).

The main loop is normally (i.e. no previous **return** had occurred) exited when the protocol contains no actions and the intruder is returned (line 19).

Let us now illustrate the functioning of the algorithm by applying it to two examples. The first shows how the algorithm finds an intruder for a protocol that does not respect a given property. The second example shows that the algorithm returns **none** when the protocol respects the required property.

**Example 2** Consider the following protocol:

$$\begin{aligned} A &\triangleq [\text{out}(\text{net}, \{m\}_k).\text{out}(\text{net}, k).0] \\ B &\triangleq [\text{in}(\text{net}, \{x\}_k).\text{in}(\text{net}, u).0] \\ \mathcal{P} &\triangleq \{A, B\}. \end{aligned}$$

```

1.  $\chi = \text{Extract}(\mathcal{P}, \text{"P"});$ 
   do {
2.    $\text{Out} := \{\text{out}(\text{net}, d) : \exists[E] \in \mathcal{P} : E \xrightarrow{\text{out}(\text{net}, d)} F\}$ 
3.   if  $\text{Out} \neq \emptyset$  then {
4.      $\text{out}(\text{net}, d) = \text{choice}(\text{Out});$ 
5.      $\mathcal{P} = (\mathcal{P} \setminus \{E\}) \cup \{F\};$  //  $E \in \mathcal{P}, E \xrightarrow{\text{out}(\text{net}, d)} F$ 
6.      $I := I.\text{in}(\text{net}, \text{nvar}());$ 
7.      $\kappa := \kappa \cup \{d\}$ 
   }
   else {
8.      $\text{In} := \{\text{in}(\text{net}, d) : \exists[E] \in \mathcal{P} : E \xrightarrow{\text{in}(\text{net}, d)} F\};$ 
9.     if  $\text{In} = \emptyset$ 
10.    then return(none)
   else {
11.     $\text{in}(\text{net}, d') := \text{choice}(\text{In});$ 
12.     $\Delta := \{d \in \partial(\kappa) : d \text{ satisfied } \chi \wedge \exists \sigma \text{ ground substitution} : d'\sigma = d\};$ 
13.    if  $\Delta = \emptyset$ 
14.    then return(none)
   else {
15.     $d := \text{choice}(\Delta);$ 
16.     $\mathcal{P} = (\mathcal{P} \setminus \{E\}) \cup \{F\sigma\};$  //  $E \in \mathcal{P}, E \xrightarrow{\text{in}(\text{net}, d')} F, d'\sigma = d$ 
17.     $I := I.\text{out}(\text{net}, d);$ 
   }
   }
}
18. until  $\mathcal{P}$  contains no actions;
19. return( $[I.0]$ )

```

Table 1: The algorithm

*This protocol does not respect the secrecy of key  $k$  and, therefore, integrity of  $m$  is not respected too. Indeed, principal  $A$  sends a datum  $m$  ciphered by  $k$  and then sends  $k$  on the public channel net. However, an intruder cannot violate the integrity of  $m$  until the second output of  $A$  is performed.*

*Consider the following magic version of  $\mathcal{P}$ :*

$$\begin{aligned}
\text{"B"} &\triangleq [\text{in}(\text{net}, \{m\}_k).\text{in}(\text{net}, u).0] \\
\text{"P"} &\triangleq \{A, \text{"B"}\}.
\end{aligned}$$

*The magic instance of  $B$  intends to formalize the integrity on the parameter  $m$ : If the protocol respects the integrity of  $m$ , only the cryptogram  $\{m\}_k$  can be sent to  $B$ . Note that  $B$  can retrieve any message on its second input.*

- The magic version "P" is converted into the constraint  $\chi = (x \neq m)$ . Let us assume that the initial knowledge  $\kappa$  of the intruders includes only a name  $n$  distinct from  $m$ .

- During the first iteration, the algorithm looks for output actions to match. The only possible choice is the output action of A. The algorithm<sup>5</sup> then starts constructing the intruder with:

$$\begin{aligned}
I &= in(net, z_1) \\
A &= [out(net, k).0] \\
B &= [in(net, \{x\}_k).in(net, u).0] \\
\kappa &= \{n, \{m\}_k, k, m\}
\end{aligned}$$

we remark that  $m \notin \kappa$  because  $k \notin \kappa$  and, therefore,  $\{m\}_k$  cannot be decrypted.

- The second iteration consumes the second output of A.

$$\begin{aligned}
I &\triangleq [in(net, z_1).in(net, z_2)] \\
\kappa &\triangleq \{n, \{m\}_k, k, m\} \\
A &\triangleq [0] \\
B &\triangleq [in(net, \{x\}_k).in(net, u).0]
\end{aligned}$$

Notice that  $m \in \kappa$  because, after  $out(net, k)$  has been grabbed by the intruder, he can decrypt  $\{m\}_k$  using  $k$ .

- The third iteration calculates  $Out$ ; this time  $Out = \emptyset$ . The algorithm examines the input offered by the protocol, non-deterministically selects one of them to which he can send a message derived from its actual knowledge that satisfies the constraints. In this case, the intruder is able to forge a cryptogram using  $k$  as key and a message  $n \neq m$  as required by the constraint on  $x$ . Therefore, we have

$$\begin{aligned}
I &= in(net, z_1).in(net, z_2).out(net, \{n\}_k) \\
\kappa &= \{n, \{m\}_k, k, m\} \\
A &= [0] \\
B &= [in(net, u).0].
\end{aligned}$$

- The last iteration simply has to provide a value for the last input action of B. Since no constraints are stated on  $u$ , any data derivable from  $\kappa$  may be used.

$$\begin{aligned}
I &= in(net, z_1).in(net, z_2).out(net, \{n\}_k).out(net, m) \\
\kappa &= \{n, \{m\}_k, k\} \\
A &= [0] \\
B &= [0]
\end{aligned}$$

The result of the algorithm is  $[I.0]$  and it is easy to see that it is compatible with the protocol and not with its magic version.

---

<sup>5</sup>The algorithm prescribes that all the bound variables appearing in the protocol must be different and the above version satisfies this requirement.

**Example 3** *This example shows a case in which the algorithm returns none. We consider the WMF protocol and we check if it respects the secrecy of  $k$ . Intuitively, the protocol is correct (with respect to secrecy) because the intruder cannot “deduce” the key  $k$  without knowing  $k_{as}$  or  $k_{bs}$ . The protocol is:*

$$\begin{aligned}
A &\triangleq [\text{out}(\text{net}, \{k\}_{k_{as}}).\text{out}(\text{net}, \{n\}_k).0] \\
S &\triangleq (\text{net}, \{x\}_{k_{as}}).\text{out}(\text{net}, \{x\}_{k_{bs}}).0] \\
B &\triangleq [\text{in}(\text{net}, \{v\}_{k_{bs}}).\text{in}(\text{net}, \{w\}_v).0] \\
\text{WMF} &\triangleq \{A, S, B\}
\end{aligned}$$

*Note that this protocol has only one terminating trace.*

*The property we want to prove is that  $k$  is not leaked. This is equivalent to say that  $v$  is always instantiated to  $k$  in  $B$ . The magic instance is*

$$\begin{aligned}
\text{“}B\text{”} &\triangleq [\text{in}(\text{net}, \{k\}_{k_{bs}}).\text{in}(\text{net}, \{w\}_k).0] \\
\text{“}WMF\text{”} &\triangleq \{A, S, \text{“}B\text{”}\}
\end{aligned}$$

*and the associated constraint that invalidates the property is  $v \neq k$ .*

*Initially, the knowledge of  $I$  is void.*

- *The first two iterations of the algorithm are devoted to collect the outputs offered by  $A$ . At the end of this phase, the situation is:*

$$\begin{aligned}
I &= \text{in}(\text{net}, z_1).\text{in}(\text{net}, z_2) \\
\kappa &= \{\{k\}_{k_{as}}, \{n\}_k\} \\
A(p) &= [0] \\
S &= [\text{in}(\text{net}, \{x\}_{k_{as}}).\text{out}(\text{net}, \{x\}_{k_{bs}}).0] \\
B &= [\text{in}(\text{net}, \{v\}_{k_{bs}}).\text{in}(\text{net}, \{w\}_v).0]
\end{aligned}$$

- *The second iteration considers the two inputs offered by WMF, but the only message that satisfies  $\chi$  that can be derived by  $\kappa$  is  $\{k\}_{k_{as}}$ .*

$$\begin{aligned}
I &= \text{in}(\text{net}, z_1).\text{in}(\text{net}, z_2).\text{out}(\text{net}, \{k\}_{k_{as}}) \\
\kappa &= \{\{k\}_{k_{as}}, \{n\}_k\} \\
A &= [0] \\
S &= [\text{out}(\text{net}, \{k\}_{k_{bs}}).0] \\
B &= [\text{in}(\text{net}, \{v\}_{k_{bs}}).\text{in}(\text{net}, \{w\}_v).0]
\end{aligned}$$

- *Third iteration reconsiders outputs and the output of the server is stolen by  $I$ , obtaining*

$$\begin{aligned}
I &= \text{in}(\text{net}, z_1).\text{in}(\text{net}, z_2).\text{out}(\text{net}, \{k\}_{k_{as}}).\text{in}(\text{net}, z_3) \\
\kappa &= \{\{k\}_{k_{as}}, \{n\}_k, \{k\}_{k_{bs}}\} \\
A &= [0] \\
S &= [0] \\
B &= [\text{in}(\text{net}, \{v\}_{k_{bs}}).\text{in}(\text{net}, \{w\}_v).F[w]]
\end{aligned}$$

- In the fourth iteration the algorithm has to provide a value for the first input of  $B$ , in  $(net, \{v\}_{k_{bs}})$  and such a value must respect the constraints  $v \neq k$ . The only cryptogram that may be derived from  $\kappa$  that has  $k_{bs}$  as encryption key is  $\{k\}_{k_{bs}}$  that does not respect  $\chi$ . Hence the algorithm returns **none**.

Note that in this example only one trace is possible, hence the non deterministic choices are not “visible”.

## 4.2 Correctness

We outline the proof of the correctness of the algorithm. As stated before, we consider deterministic, non recursive protocols; this is equivalent to say that protocols are obtained by parallel compositions of *sequential processes*, namely processes that are sequences of actions.

If our algorithm, applied to initial knowledge  $\kappa$  and to property  $\chi$  (expressed as conjunction of constraints), returns a process  $I$ , then  $I$  is an intruder of the protocol that falsifies the security property  $\chi$ . Moreover,  $I$  is a sequential process. In this section we want to prove that, in the case the algorithm returns **none**, an attacker that falsifies  $\chi$  for the protocol does not exist.

The proof is made of three steps:

1. We prove that if a context has a compatible process then we can derive a sequential compatible process (Proposition 1 and its proof).
2. We prove that given a protocol, if an attacker exists then there exists a sequential attacker as well (Lemma 1);
3. If an attacker exists, the algorithm does not return **none** (Proposition 2).

**Proposition 1** *Let  $E$  be a behaviour expression and  $C$  a context. If  $E$  is compatible with  $C$ , then there exists  $F$  sequential and  $C$ -compatible.*

**Proof.** If  $E$  is  $C$ -compatible, then there exists a trace of the context  $C \cup E$  that terminates. If we sequentially compose the actions of  $E$  that occur in this trace, we obtain a sequential process that is obviously  $C$ -compatible.  $\square$

**Lemma 1** *Let  $\mathcal{P}$  be a protocol and “ $\mathcal{P}$ ” be a magic instance of  $\mathcal{P}$ . If an intruder  $I$  for  $\mathcal{P}$  with respect to “ $\mathcal{P}$ ” exists then a sequential intruder exists too.*

**Proof.** By hypothesis,  $I$  is compatible with  $\mathcal{P}$  then, by Proposition 1 there exists a sequential process  $\bar{I}$ , obtained as in the proof of Proposition 1, that is  $\mathcal{P}$ -compatible. The process  $\bar{I}$  is an intruder because it is not possible that “ $\mathcal{P}$ ”  $\cup \bar{I}$  has a terminating trace otherwise also  $I$  should be “ $\mathcal{P}$ ”-compatible, that, by hypothesis, is false.  $\square$

**Proposition 2** *If  $\mathcal{P}$  is a protocol and  $I$  is an intruder for  $\mathcal{P}$  with initial knowledge  $\kappa$  that makes the protocol non correct with respect to  $\chi$ , then the algorithm with parameter  $\mathcal{P}$ ,  $\kappa$  and  $\chi$ , does not return **none**.*

**Proof outline.** The context  $\mathcal{P} \cup I$  is may correct. By Lemma 1 there exists a sequential intruder  $\bar{I}$  such that  $\bar{I}$  is  $\mathcal{P}$ -compatible. Let us assume that the algorithm returns `none`.

- If the algorithm returns `none` in line 10, then both  $In$  and  $Out$  are void at the first iteration. This means that both  $\mathcal{P}$  and “ $\mathcal{P}$ ” are empty. Hence, a  $\mathcal{P}$ -compatible intruder  $I$  can only perform  $\tau$  actions, but in this case it will also be “ $\mathcal{P}$ ”-compatible contradicting the hypothesis that  $I$  is an intruder: A protocol that does nothing is not attackable.
- Otherwise, the algorithm returns `none` if the protocol principals offer only input actions and the knowledge of the intruder cannot forge messages that does not violate the constraints (line 14). However, the knowledge of  $\bar{I}$  is at most great as the knowledge accumulated by algorithm intruder because it grabs all the messages sent by the principals. Therefore, if  $\bar{I}$  is able to forge a message that respects the constraints, then also the intruder of the algorithm can forge it. This contradicts the hypothesis that the algorithm returns `none`, since there exists a trace that make the protocol to terminate.  $\square$

## 5 Related Works

Process algebras have been intensively used in protocol analysis [8, 9, 13, 14, 19, 20, 18]. Our calculus is inspired by the spi calculus [1] that, among process algebras for cryptographic protocols has received great attention. Spi uses behavioural equivalence for analysing protocols.

The idea of ”magic” instance of a protocol appears in [1]. There it is used to express integrity properties of protocols in terms of testing equivalence ( $\cong$ ) [6] between the protocol and its magic instance. In [1] the instantiation of a protocol parameter takes place in the ”continuation” of the protocol. The ”real” continuation and the magical one are checked for testing equivalence. An hypothesis is that the continuation ”must” depend on the value of instantiated variables. However, attention must be paid in considering the behaviour of the continuations. Indeed, let  $x$  be a protocol parameter ranging over natural numbers and let  $F[x]$  be the continuation that depends on  $x$ . If we suppose that  $F[n] \cong F[m]$ , for all even numbers  $n$  and  $m$ , then we are not sure that integrity of  $x$  is respected by simply checking the equivalence between the normal and magic continuations: An intruder might succeed in breaking the protocol by changing the even value of  $x$  with a different even number and maintaining the equivalence. Our approach avoids this problem. It is not necessary to consider the use of the parameter after it has been exchanged: We just request that an intruder should not be able to modify the value within protocol communications and does not rely on continuations. We can associate a constraint to each magic variable. Once the variables are selected (depending on the property one wants to check), constraints are deduced directly from protocols. Basically, constraints

test the ability of intruders to forge data different from data prescribed by the protocol (that may be accepted by the principals as plausible data).

Strictly related to our approach is the calculus proposed in [2], where a process algebra that simplifies spi is proposed. In [2], all processes interact each other by means of a unique public name. Once a message has been sent over this channel it increases the “environment” knowledge. The semantics is given by a symbolic semantic relying on unification that restricts the number of trace of the processes. Protocol analysis is obtained through a search for a (symbolic) trace that respects some properties. There are some similarities with our approach. First, the uniqueness of the public channel is a key feature of both the process algebras. Second, the knowledge of the intruder (in a give instant) is represented by all the messages that have been sent along the public channel. Our verification procedure analyses protocol traces, but the analysis is driven by constraints.

The protocols we consider in this work have the same characteristics of the protocols that may be modeled using *strand spaces* [24, 23]. In the case of strand spaces, a principal in a protocol is represented as a finite strand of nodes representing input or output actions; each input node may receive its value from a single output node. A principal cannot have non-deterministic behaviour. Differently from strand spaces, we do not consider message multicasting. A similar idea has been followed in [5].

Secure composition of components has been studied in [22, 21]. The problem addressed by the authors is the gluing of software components, that may be (partially) untrusted, into a system. They enclose components into *wrapper programs* to encapsulate and enforce security policies. Wrappers are expressed in terms of *box- $\pi$*  calculus, a small variation of  $\pi$ -calculus. Information flow analysis on the resulting systems is done by a type system which uses causal information. The main difference between our work and the above approach lays in the fact that *box- $\pi$*  the emphasis is on security properties of “closed” systems, i.e. systems that may decide to introduce a new (off-the-shelf) component and have the control over computation and interactions. Here, we deal with open systems that have no control on the presence of new components that may arrive in any moment and, possibly, without the other components of the system get aware of their presence.

## 6 Conclusions and future work

Thanks to an original interpretation of the notion of magic instance, an approach to the verification of security protocols has been presented. A methodology has been provided together with an algorithm that implements the automatic part of the methodology. The correctness of the algorithm has been discussed. The non-automatic part of the methodology relies on the ability of who performs the protocol analysis: he or she must express security properties in terms of values that one would observe in the communications of the protocol if it was secure. Several examples illustrate various cases of the applicability of the methodology.

The task of “off-line” setting what to observe revealed as crucial. Our methodology offers, to whom may want to check the security of a protocol, a formal setting with effective procedures within which developing the desired analysis. Moreover, the freedom of analysis is augmented by the possibility of verifying properties under the assumption that a malicious intruder has been provided with some supplementary knowledge.

Several issues need to be explored in future work:

- The importance of having a common framework to represent both composition and security issues suggests to deepen the relations among the calculus of this paper and the full interaction pattern language. The latter deals also with names of (private) channels shared among the components and with formal mechanisms to allow the (dynamic) access of components in a running system. This suggests the possibility of testing protocols in more general scenarios, like, e.g., hierarchies of contexts where components (and principals) can enter at run-time and possibly register their asymmetric keys in order to be visible inside the context.
- We plan to analyse the relationships between our methodology and the approach based on non-interference, [10].
- As already pointed out in Section 3, it would be important to more clearly define the properties that can be expressed by the usage of the magic instance. The examples indicate that secrecy and integrity can be easily modeled, but still lacks a clear way to extend the methodology to, e.g., authenticity and authentication. As well as regarding the relations among the properties one wants to prove and the appropriate magic instance: which is the best way to reduce properties to sets of variables/values ?
- The last direction for future investigations regards the definition of a hierarchy of protocols. Starting from the possibility provided by our algorithm of studying the effects of enhancing an hypothetical intruder with a given amount of knowledge, it could be possible both to understand which is the (minimal) knowledge requested to broken a protocol, and classify the kind of data to which security protocols can be more sensitive. The first analysis could provide a hierarchy of protocols in terms of their robustness, the second one could characterise the data which should be better protected in a protocol.  
On the other hand, also the dual problem of ordering, given a protocol, the environments in which the protocol can be executed, depending on their endangering power, appears worth of further investigations, along an idea originally presented in [12], where general properties like bisimulation are recasted with respect to a given context.

## References

- [1] Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1–70, 10 January 1999.
- [2] Michele Boreale. Symbolic Analysis of Cryptographic Protocols in the Spi-Calculus. In *Proc. of LICS'00*, 2000. to appear.
- [3] Andrea Bracciali, Antonio Brogi, and Franco Turini. Coordinating interaction patterns. In *Proceedings of the ACM Symposium on Applied Computing, Las Vegas, USA*. ACM, 2001.
- [4] Michael Burrows, Martín Abadi, and Roger Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, February 1990.
- [5] Iliano Cervesato, Nancy A. Durgin, Patrick D. Lincoln, John C. Mitchell, and Andre Scedrov. Relating strands and multiset rewriting for security protocol analysis. In P. Syverson, editor, *13th IEEE Computer Security Foundations Workshop — CSFW'00*, pages 35–51, Cambridge, UK, 3–5 July 2000. IEEE Computer Society Press.
- [6] Rocco De Nicola and Matthew C. B. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.
- [7] Danny Dolev and Andrew C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.
- [8] Riccardo Focardi and Roberto Gorrieri. A classification of security properties. *Journal of Computer Security*, 3(1), 1995.
- [9] Riccardo Focardi and Roberto Gorrieri. The Compositional Security Checker: A tool for the verification of information flow security properties. *IEEE*, 23(9):550–571, September 1997.
- [10] Riccardo Focardi, Roberto Gorrieri, and Fabio Martinelli. Non-interference for the analysis of cryptographic protocols. In *Proc. ICALP'2000, LNCS*, 2000.
- [11] U. Ster J.C. Mitchell, M. Mitchell. Automated analysis of cryptographic protocols using  $\text{mur}\phi$ . In *10th IEEE Computer Security Foundations Workshop, IEEE Press*, pages 141–151, 1997.
- [12] Kim G. Larsen. A context dependent equivalence between processes. *Theoretical Computer Science*, 49:185–215, 1987.
- [13] Gavin Lowe. A hierarchy of authentication specifications. In *PCSFW: Proceedings of The 10th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1997.

- [14] Gavin Lowe. Defining information flow. Technical report, Department of Mathematics and Computer Science, University of Leicester, 1999.
- [15] W. Marrero, E.M. Clarke, , and S. Jha. Model checking for security protocols. In *Formal Verification of Security Protocols*, 1997.
- [16] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, I and II. *Information and Computation*, 100(1):1–40,41–77, September 1992.
- [17] Paulson. Proving properties of security protocols by induction. In *PCSFW: Proceedings of The 10th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1997.
- [18] Ryan and Schneider. Process algebra and non-interference. In *PCSFW: Proceedings of The 12th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1999.
- [19] Steve Schneider. Modelling security properties with CSP. Technical Report CSD-TR-96-04, Department of Computer Science, Royal Holloway, University of London, Department of Computer Science, Egham, Surrey TW20 0EX, England, February 1996.
- [20] Steve Schneider. Verifying authentication protocols in CSP. *IEEE Transactions on Software Engineering*, 24(9):741–758, September 1998.
- [21] Sewell and Vitek. Secure composition of untrusted code: Wrappers and causality types. In *PCSFW: Proceedings of The 13th Computer Security Foundations Workshop*. IEEE Computer Society Press, 2000.
- [22] Peter Sewell and Jan Vitek. Secure composition of insecure components. In *Proceedings of the Computer Security Foundations Workshop, CSFW-12*, 1999.
- [23] F. Javier Thayer Fabrega, Amy L. Herzog, and Joshua D. Guttman. Honest ideals on strand spaces. In *Proceedings of the 11th IEEE Computer Security Foundations Workshop (CSFW '98)*, pages 66–78, Washington - Brussels - Tokyo, June 1998. IEEE.
- [24] F. Javier Thayer Fabrega, Amy L. Herzog, and Joshua D. Guttman. Strand spaces: Why is a security protocol correct? In *RSP: 19th IEEE Computer Society Symposium on Research in Security and Privacy*, 1998.