Formal Support to Security Protocol Development: A Survey
Soporte Formal para el Desarrollo de Protocolos de Seguridad: una Visión General

Juan Carlos López Pimentel and Raúl Monroy
Computer Science Department
Tecnológico de Monterrey, Campus Estado de México
Carretera al lago de Guadalupe, Km 3.5, Atizapán de Zaragoza, 52926, México
juan.pimentel@itesm.mx, raulm@itesm.mx

Article received on April 16, 2008; accepted on June 20, 2008

Abstract
Security protocols aim to allow two or more principals to establish a secure communication over a hostile network, such as the Internet. The design of security protocols is particularly error-prone, because it is difficult to anticipate what an intruder may achieve interacting through a number of protocol runs, claiming to be an honest participant. Thus, the verification of security protocols has attracted a lot of interest in the formal methods community and as a result lots of verification techniques/tools, as well as good practices for protocol design, have appeared in the two last decades. In this paper, we describe the state of the art in automated tools that support security protocol development. This mainly involves tools for protocol verification and, to a lesser extent, for protocol synthesis and protocol diagnosis and repair. Also, we give an overview of the most significant principles for the design of security protocols and of the major problems that still need to be addressed in order to ease the development of security protocols.
Keywords: Formal methods, security protocols, protocol synthesis, protocol diagnosis and repair.

Resumen
Los Protocolos de Seguridad tienen como objetivo permitir que dos o más agentes puedan establecer una comunicación de manera segura en una red a pesar de ambientes hostiles, tales como Internet. El diseño de estos protocolos es particularmente propenso a errores, por eso, es difícil anticipar lo que un intruso puede lograr cuando, pretendiendo ser un participante honesto, interactúa con una cantidad considerable de corridas del protocolo. Así, la verificación de protocolos de seguridad ha atraído un gran interés en la comunidad de los métodos formales, dando como resultado la aparición, en las dos últimas décadas, de una gran cantidad de técnicas/herramientas, además de buenas prácticas para mejorar su diseño. En este artículo, describimos el estado del arte de las herramientas automatizadas que soportan el desarrollo de protocolos de seguridad. Principalmente, incluímos herramientas para su verificación, y en menor grado, trabajos sobre su síntesis; además de métodos en el diagnóstico y reparación de protocolos incorrectos. También, damos un resumen de los principios más importantes para mejorar el diseño de esta clase de protocolos y los principales problemas que todavía necesitan ser resueltos para facilitar su desarrollo.
Palabras claves: Métodos formales, protocolos de seguridad, síntesis de protocolos, diagnóstico y reparación de protocolos

1 Introduction

Computer security is presently a strong requirement. Crimes such as user impersonation or unauthorized access to information have already resulted in countless losses. The goal of security protocols is to allow the exchange of sensitive data over insecure networks, such as the Internet. A *security protocol* is a protocol that aims to achieve a security related goals, such as key-agreement, entity-authentication, information integrity, confidentiality and non-repudiation.

Cryptographic primitives are the basic building-blocks of security protocols. They include one-way hash functions and encryption functions. Mainly hash functions aim to provide integrity or non-repudiation and encryption, whether symmetric or not, confidentiality or authentication. Though very reliable, cryptographic primitives cannot address the security requirements of a security protocol. This is both, because they need to be applied in combination rather than in isolation and because the design of a security protocol is especially error-prone.

Indeed, protocol flaws can be subtle and hard to find. The well-known Needham and Schroeder Public Key (NSPK) protocol was found to be flawed 17 years after its publication [Lowe 1996].

Not surprisingly, there is a number of methods/tools that support the formal development of security protocols, particularly in the verification phase. Verification methods/tools belong to one of three approaches: belief logics, e.g. BAN [Burrows, et al., 1989] and BGNY [Brackin 1996]; state-exploration, e.g. NRL [Lowe 1996], AVISPA [AVISPA 2005], Athena [Song, et al., 2001]; and theorem-proving, e.g. Paulson's inductive approach [Paulson 1998] and Coral [Steel et al. 2003]. Of all existing tools, AVISPA and Athena are the most powerful. They are both capable of automatically determining whether a protocol fails to satisfy a security requirement, in which case they output a counterexample. A *counterexample* is an interleaving of one or more protocol runs violating a given security requirement. We call it an *attack* for short.

More often than not, research reports on attacks to protocols that were not known to be flawed. This has led the research community both to postulate good practices for protocol design and to explore techniques complementary to protocol verification. The most prominent example of a set of good practices for protocol design are Abadi and Needham's [Abadi and Needham 1996]. These principles capture common features among protocols that Abadi and Needham found hard to analyze. If these features are avoided, they concluded, protocols tend to become more readable and, more importantly, correct. The most prominent examples of application of formal methods to aspects other than protocol verification are synthesis [Perrig and Song 2000], diagnosis and repair [López-Pimentel, et al., 2007b].

This paper aims to describe state-of-the-art tools that support the development of security protocols. This involves mainly tools for protocol verification and, to a lesser extent, for protocol synthesis and protocol diagnosis and repair. Also we give an overview of the most significant principles for the design of security protocols and of the major problems that still need to be addressed in order to ease the development of security protocols.

The remainder of the paper is organized as follows. First, Section 2, we give an overview of the most important security protocol libraries. The number of faulty protocols in these libraries evidences the need for further support in protocol security development. Then, Section 3, we describe state-of-the-art tools for verifying security protocols. Next, Section 4, we summarize seminal guidelines for protocol design. Then, Section 5, we briefly describe significant existing approaches to protocol synthesis and computer-aided protocol design. We also give an outline of existing approaches to diagnosis and repair of faulty security protocols. Next, Section 6, we briefly describe ongoing research, aimed at strengthening and enhancing security protocol verification. Finally, conclusions drawn from this survey are discussed.

2 Security Protocol Analysis

Software development is a structure which describes how to approach the tasks involved in software production. The waterfall model is the best-known model for software development and consists of the following phases: requirements analysis, specification, design, construction, integration, testing or verification, and maintenance.

When software is safety critical, the use of rigorous, mathematical models to software development becomes recommendable but sometimes mandatory. By using mathematical models, software problems are identified at earlier stages, prior to construction. There exists two approaches to the verification of security protocols, one is based on the complexity and the probability of breaking the cryptographic primitives of a protocol, and the other on what can be learned from interacting with several principals engaged in an arbitrary number of protocol runs. These two approaches differ mainly in how they model cryptographic primitives.

The computational complexity approach claims that cryptographic primitives are simply functions on strings of bits. For example, as explained in [Abadi and Rogaway 2002], a symmetric encryption scheme might be defined as a triple of algorithms $\Pi = (K, E, D)$. Algorithm K is a key generator which after making random choices generates a string k (the key). E is an encryption algorithm which flips random coins r to map strings k (the key) and m (the plain-text) into a string $E_k(m, r)$ (the cypher-text). Algorithm D is a decryption algorithm that maps strings k (the key) and c (the cypher-text) into a string $D_k(c)$, from which it recuperates m, after presenting *r*. It is expected that $D_k(E_k(m,r)) = m$ for appropriate k, m and r.

The adversary is essentially modeled as a Turing machine which has access to an oracle. The oracle aims to find a string k' that is able to decrypt a given cypher-text $E_k(m,r)$. To do this task, the oracle usually has some clues that facilitate its task. Clues involve knowledge of some components of m, knowledge of other messages encrypted under the same key, etc. Roughly, a protocol is considered good if the oracle cannot find k', or while consuming the computational power at hand the probability of finding k' is slow-growing under a determined threshold. Although providing strong security guarantees, proofs under this approach are in general harder and more difficult to automate.

The formal methods approach adopts the *perfect cryptography assumption*. This assumption states that the only way to decrypt a cypher-text is by applying the appropriate key. Crucially, there is no way to recover the message M or the key K from just the message that results from the encryption of M under K, denoted by $\{|M|\}_K$. This is in contrast to the computational complexity approach.

The adversary is modeled as an omnipresent agent that controls the network but cannot make cryptanalysis. In general, the adversary can intercept messages, analyze them, and decrypt them if he possesses the corresponding decryption key. He can also inject new messages to the network and send them under any agent name. Dolev and Yao, [Dolev and Yao 1983], formalized a model of the adversary, so-called the *spy*, with these abilities. In addition to the Dolev and Yao model, some authors assume that the spy knows all public keys, his own shared key and private key, and all shared and private keys of a collection of compromised agents. Another authors also consider that the spy knows a set of lost *session keys*.[1]

Connections between the formal view and the computational view are currently subject of investigation. In their seminal paper, Abadi and Rogaway [Abadi and Rogaway 2002] have established a relation between syntactic equivalence of message terms, where encryption is treated as a formal operator, and secure encryption, where encryption is defined in terms of computational indistinguishability. The result of Abadi and Rogaway is applicable for symmetric encryption in the presence of a passive attacker. Later on Baudet *et al.* [Baudet, et al., 2005] introduced a reasoning framework for proving soundness of implementations of equational theories, which are used to specify cryptographic primitives. More recently, Kremer and Mazare [Kremer and Mazaré 2007] have extended Baudet *et al.*'s work to consider an adaptive user, rather than a purely passive one.

Our research lies in the formal methods approach. So, in the next section, we survey the most significant verification tools. Before going further, though, we briefly describe the formal approach to software verification and give an example faulty security protocol.

2.1 Formal Software Verification

In the application of formal methods to software verification, both the system and its specification are first expressed as formulas of some (but not necessarily the same) logic. Then, mathematical reasoning is used to prove that the system and the specification are related somehow, for example by logical implication. A state-of-the-art verification tool is capable of yielding either of two outputs: i) OK, indicating that the system is error-free, at least with respect to the coverage analysis of the corresponding tool; and ii) a counterexample, indicating how a system execution violates the specification.

In the context of security protocol verification, the system is the security protocol under analysis, the specification a protocol security requirement and the counterexample actually is an attack. Authentication and secrecy are the most common examples of protocol security requirements. These properties have no universal interpretation and are formalized according to the context. Roughly, *user authentication* amounts to attempting to verify the identity of a protocol participant and *secrecy* to ensuring that certain message parts sent over the network remain readable only to their intended recipients.

[1]A session key is a key used for encrypting one message or a group of messages in a communication session.

Most security protocols are documented in two main libraries: Clark and Jacob's[2] and AVISPA's[3]. The Clark and Jacob library (Table 1) comprehends 50 protocols, 26 of which are known to be faulty and the rest of them are not. The AVISPA library (Tables 2 and 3) comprehends 70 protocols, 17 of which are faulty and the rest of them are not. For every protocol that is faulty, these libraries include an extended version of it for which no attack is known. Each row displays the security services pursued by each protocol (Authentication, **Auth**; Secrecy, **Sec**; Key distribution, **Kdis**; and Key establishment, **Kest**), whether or not the protocol includes a Server, **S**, and the main cryptographic primitives (shared key, **SK**; or/and public key, **PK**) used by the protocol..

2.2 A Faulty Security Protocol

To give the reader a flavor as to the difficulties of designing sound security protocols, let us consider the NSPK Protocol, which aims to provide strong authentication of the initiator and the responder. Given in the so-called Alice and Bob notation, the specification of NPSK is as follows:

$$
\begin{aligned}
&1.\ A \rightarrow B : \{|N_a, A|\}_{K_B^+}\\
&2.\ B \rightarrow A : \{|N_a, N_b|\}_{K_A^+}\\
&3.\ A \rightarrow B : \{|N_b|\}_{K_B^+}
\end{aligned}
$$

where $A \rightarrow B : M$ means agent A sends message M to agent B, which B is meant to receive; "," denotes message concatenation; K_P^+ the public key of principal P; and $\{|M|\}_K$ denotes the message M encrypted under the key K.

Roughly speaking, A initiates a session with B by sending it him a *nonce* (a random number, newly generated) N_a, indicating the start of a new session. After decrypting the message received at step 1, called message 1 for short, B sends A back both the nonce N_a that A apparently sent it, and a new nonce B has generated, N_b. Upon decryption of message 2, A checks that nonce N_a corresponds to the nonce it generated to identify this run of the protocol and then sends B back the nonce N_b (c.f. message 3).

[2]The Clark and Jacob library is available at http://www.lsv.ens-cachan.fr/spore/index.html
[3]The AVISPA library is available at http://www.avispa-project.org/

Table 1. The Clark and Jacob library

Name	SK	PK	S	Auth	Sec	Kdis	Kest	Protocol's steps
Andrew Secure RPC	*			*	*	*		4
BAN modified Andrew Secure RPC	*			*	*	*		4
BAN concrete Andrew Secure RPC	*			*	*	*		4
Lowe modified BAN concrete Andrew S.RPC	*			*	*	*		4
Bull's Authentication Protocol	*		*	*	*	*		6
CAM				*				1
CCITT X.509 (1)		*		*				1
CCITT X.509 (1c)		*		*				1
CCITT X.509 (3)		*		*				3
BAN modified CCITT X.509 (3)		*		*				3
Denning-Sacco SK	*		*	*	*	*		3
Lowe modified Denning -Sacco shared key	*		*	*	*	*		5
Denning-Sacco ASK		*	*	*	*	*		5
Diffie Helman				*	*	*		4
GJM			*	*				6
Gong			*	*	*	*		5
Kao Chow Auth-V.1	*		*	*	*	*		4
Kao Chow Auth-V.2	*		*	*	*	*		4
Kao Chow Auth-V.3	*		*	*	*	*		4
Kerberos V5	*		*	*	*	*		6
KSL	*		*	*	*	*		8
Lowe modified KSL	*		*	*	*	*		8
Neumann Stubblebine	*		*	*	*	*		7
Neumann Stub.modified	*		*	*	*	*		7
NSPK		*		*				7
Lowe's fixed NSPK		*		*				7
Needham Schroeder SK	*		*	*	*	*		5
Amended Needham S.SK	*		*	*	*	*		7
Otway Rees	*		*	*	*	*		4
O-Rees BAN version	*		*	*	*	*		3
Schnorr's Protocol		*		*				3
Shamir-Rivest-Adleman	*			*				3
SK3	*		*	*	*	*		11
SmartRight view-only	*			*				3
SPLICE/AS		*		*	*	*		6
SPLICE/AS Modified		*		*	*	*		6
C-J modified SPLICE/AS		*		*	*	*		6
TMN	*	*	*	*	*	*		4
WMF protocol	*		*	*	*	*		2
Lowe modified WMF	*		*	*	*	*		4
Woo and Lam Mutual	*		*	*	*	*		7
Woo and Lam Pi	*		*	*				5
Woo and Lam Pi 1	*		*	*				5
Woo and Lam Pi 2	*		*	*				5
Woo and Lam Pi 3	*		*	*				5
Woo and Lam Pi f	*		*	*				5
Yahalom	*		*	*	*	*		4
BAN modified Yahalom	*		*	*	*	*		4
Lowe's modified Yahalom	*		*	*	*	*		5
Paulson's strengthened version of Yahalom	*		*	*	*	*		4

Table 2. The AVISPA library

Name	SK	PK	S	Auth	Sec	Kdis	Kest	Protocol's steps
AAA Mobile IP	*		*	*	*			9
CTP-non_predictive-fix	*			*	*			6
SIP			*	*				16
H.530	*			*				6
H.530 fix	*			*				6
QoS-NSLP	*		*	*				4
Geopriv	*	*	*	*				5
Geopriv-self-signatures		*	*	*	*			6
Geopriv-two_pseudonyms	*	*	*	*	*			5
Geopriv-pervasive	*	*	*	*				5
Geopriv-password	*	*	*	*				6
SIMPLE	*		*	*				4
LIPKEY-SPKM-known-initiator		*	*	*				3
LIPKEY-SPKM-unknown-initiator		*	*	*				3
CHAPv2	*			*				4
APOP	*		*	*				3
CRAM-MD5			*	*				3
DHCP-delayed-auth			*	*				2
TSIG	*		*	*				2
EAP_AKA	*		*	*	*		*	5
EAP_Archie	*		*	*	*		*	6
EAP_IKEv2		*	*	*	*		*	7
EAP_SIM	*		*	*	*		*	7
EAP_TLS		*	*	*	*		*	7
EAP_TTLS_CHAP		*	*	*	*		*	7
PEAP	*	*	*	*	*		*	8
S/KEY			*	*	*			4
EKE	*			*	*	*		5
EKE2	*			*	*	*		3
SPEKE	*			*				5
SRP	*			*				4
IKEv2-DS		*		*	*	*		4
IKEv2-DSx		*		*	*	*		6
IKEv2-MAC	*			*	*	*		4
IKEv2-MACx	*			*	*	*		6
IKEv2-CHILD	*			*	*	*		2
IKEv2-EAP-Archie	*	*		*	*	*		8
RADIUS-RFC2865	*		*	*				5
8021x_Radius	*		*	*		*		10
HIP		*		*				4
PBK		*		*				4
PBK-fix		*		*				4
PBK-fix-weak-auth		*		*				4
Kerbers-basic	*		*	*	*	*		6
Kerbers-Ticket-Cache	*		*	*	*	*		6
Kerberos-Cross-Realm	*		*	*	*	*		8
Kerberos-Forwardable	*		*	*	*	*		8
Kerberos-PKINIT	*	*	*	*	*	*		6
Kerberos-preauth	*		*	*	*	*		6
TESLA		*		*				2

Table 3. The AVISPA library

Name	SK	PK	S	Auth	Sec	Kdis	Kest	Protocol's steps
SSH-transport			*	*	*		*	6
TSP		*		*				2
TLS		*		*	*			8
ASW		*		*				4
FairZG		*	*	*				5
SET-purchase		*		*	*			6
SET-purchase-hpg		*		*	*			6
UMTS_AKA	*		*	*	*		*	2
ISO1		*		*				1
ISO2		*	*	*				2
ISO3		*		*				2
ISO4		*		*				3
2pRSA		*		*				3
LPD-MSR		*	*	*			*	3
LPD-IMSR		*		*	*		*	3
SHARE		*		*	*	*		4
NSPK		*		*	*			3
NSPK fix		*		*	*			3
NSPK-KS-fix		*		*	*			3
NSPKxor		*		*	*			3
NSPK-KS		*		*	*			3

The NSPK protocol seems right at first glance: A seems to know that it is running the protocol apparently with B, because only B could have decrypted message 1 and then have sent nonce N_a encrypted under A's key in message 2. However, this protocol is faulty. Lowe found that an intruder could impersonate one agent by concurrently holding a session with another agent [Lowe 1995]. Lowe's attack is shown in Figure 1.

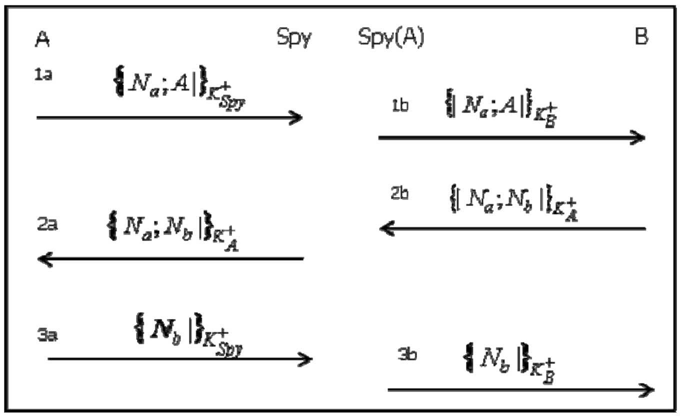
A
Spy
Spy(A)
B
1a $\{|N_a;A|\}_{K^+_{Spy}}$
1b $\{|N_a;A|\}_{K^+_B}$
2b $\{|N_a;N_b|\}_{K^+_A}$
2a $\{|N_a;N_b|\}_{K^+_A}$
3a $\{|N_b|\}_{K^+_{Spy}}$
3b $\{|N_b|\}_{K^+_B}$

Fig. 1. An attack on the NSPK protocol

From Figure 1, one can see that *one* instance of message 2 is being used in *two* independent runs. We infer that while B knows that A has recently participated in a run of the protocol; it cannot tell whether A is running it apparently with it.

Notice that identifying the protocol flaw; that is, what makes it possible to elaborate the attack, is not trivial, never mind patching the protocol. To patch NSPK, Lowe suggested adding the responder name (in this case B) to message 2 so that the initiator (in this case A) could know the identity of that message's originator, yielding:

$$\begin{aligned} &1.\ A \rightarrow B : \{|Na, A|\}_{K_B^+} \\ &2.\ B \rightarrow A : \{|B, Na, Nb|\}_{K_A^+} \\ &3.\ A \rightarrow B : \{|Nb|\}_{K_B^+} \end{aligned}$$

This new version of NSPK has been proved to satisfy authentication. Thus, the design of security protocols is error-prone, requiring the use of rigorous methods for software development.

In the following section, we shall describe the most important tools/methods to formal security protocol verification. For each method, we highlight coverage and most significant contributions. We use this information to come out with a comparison.

3 Security Protocol Verification

There are three main verification approaches: belief logic, state exploration, and theorem proving. We look at each approach, one on each of the following sections.

3.1 Belief logics

Belief logic was the first attempt to automate the verification of security protocols. BAN [Burrows, et al., 1989] is the most significant and prominent belief logic. BAN aims to formalize what an agent may infer from the messages it has received. BAN allows short, abstract proofs but it is not able to identify a wide variety protocol flaws. This is because BAN does not consider an intruder on the network. Its main characteristic is its simplicity; its main disadvantage is that it assumes that agents are all honest. When the search for a BAN proof fails, it is not obvious how to use the deduction tree to generate a counterexample. To address its weaknesses, BAN has been extended resulting in new logics, e.g. BGNY [Brackin 1996]. These new methods though have proven to be very limited to deal with the general problem of protocol verification and are far beyond from BAN's simplicity.

3.2 State Exploration

In the state exploration approach, a protocol is characterized as the set of all its possible traces. Given an input security protocol, the verification method explores as many execution paths of the protocol as possible, checking at each state reached if some conditions hold. The search space generated from analyzing a cryptographic protocol may be infinite, because principals may engage in a number of protocol runs, simultaneously play different roles (initiator, responder, etc.) and because the adversary is capable of building an infinite number of messages. Yet, most state exploration tools use theoretical results to avoid exploring the entire state space. An example result is Syverson *et al.*'s [Syverson, et al., 2000], who proved that whenever there is an attack, there is an attack in which compromised principals do not send arbitrary messages, only messages in valid protocol form. Another example result is Rusinowitch and Turuani's [Rusinowitch and Turuani 2001], who showed that whenever there is an attack, there is an attack of polynomial size (for a bounded number of sessions). State exploration methods are powerful and automatic. When a property does not hold, they can easily exploit the traces that have been generated in order to build a counterexample.

In what follows, we survey the most important state exploration tools. We start with a brief discussion of the work of Dolve and Yao. Then we carry on sudying more advanced tools based on model checking and strand spaces.

3.2.1 Early Attempts at Protocol Verification

Dolev and Yao's method The first attempt at using state exploration for verifying security protocols was due to Dolev and Yao [Dolev and Yao 1983], in 1983. Dolev and Yao's approach is however extremely limited, because it considers only the verification of secrecy and accounts for a few cryptographic primitives. Dolev and Yao's main contribution is a formal model of the intruder, which has been used largely albeit after suffering minor extensions. After Dolev and Yao’s work, the verification of security protocols was by large ignored. It was not until 1996, more than a decade later, that Meadows, Lowe and others revived the problem.

3.2.2 Model checking

Lowe's method Gavin Lowe, the man who broke the NSPK protocol [Lowe 1995], has approached the verification of security protocols using Hoare's calculus of Communicating Sequential Processes (CSP). To verify a security protocol, Lowe first writes a CSP model of every entity in the (communicating) system: the agents taking part in the protocol, the communicating network and the Dolev-Yao spy. Then, he uses the standard CSP theory of traces to conduct the protocol analysis.

In 1996, Lowe showed that a modified version of the NSPK protocol is secure at least for a small number of participants [Lowe 1996]. In his experiments, he used Failures Divergences Refinement (FDR), a model checker to verify CSP programs. The task of producing a CSP description of the protocol to verify is very time consuming and requires a high level of skill. To get around this situation, Lowe developed Casper [Lowe 1998]. Casper translates the Alice and Bob description of a security protocol into a CSP model, suitable for FDR. While Lowe’s results were encouraging, it was clear then that special-purpose tools were required for the verification of large security protocols to be possible.

Basin's method In 1999, Basin [Basin 1999] presented an approach to protocol analysis that combines complementary aspects of model checking and Paulson's formalism (see Section 3.3). In an attempt to deal with the state explosion problem, Basin’s approach used lazy data types to model the (possibly infinite) state-space of the protocol. Crucially, lazy constructors build data types without evaluating their arguments, thus allowing for the representation and computation of infinite data.

Motivated by, and closely following to, the work of Paulson, Basin used a trace-based interleaving semantics for modeling security protocols. Protocols are then formalized as infinite trees. The branches of each of these trees are traces. Adding a tree child node corresponds to a trace extension which captures the execution of a step of the protocol or an action taken by a Dolev-Yao spy. Hence, the run of a protocol defines an infinite tree and a security property is a property of nodes in the tree. Violations of security properties are found by a kind of infinite-state model checking, which is performed by searching the tree in a lazy fashion. Notice that if there is an attack, it will be present in a trace located at some tree node. However, finding this node may not be easy: not only may the tree be infinitely deep, but it may also yield an exponentially large branching factor.

With a large branching factor, standard search algorithms would hardly succeed in finding attacks that lie at shallow depths. To get around this problem, Basin used two simple heuristics, one for pruning the search tree and the other for reordering the way in which the tree is searched. The first heuristic consists of pruning traces that contain spurious events, such as the reception of a message that does not obey the rules of the protocol. The second heuristic consists of assigning the highest priority to events involving the start of another execution of the protocol or the execution of an action by the spy.

Basin *et al.*'s OFMC Later on, in 2003, Basin *et al*. introduced the On-The-Fly Model-Checker (OFMC) [Basin, et al., 2003], a tool that combines lazy data types with a set of symbolic techniques, that, in a demand-driven way, model the actions of a Dolev-Yao intruder. OFMC’s methods significantly reduce the search space without excluding attacks. OFMC was validated on a large number of experiments. It rediscovered the attacks documented in the Clark and Jacob library and found new attacks. Iwas also able to discover attacks to a wider variety of protocols. OFMC’s results are reported in the AVISPA library [Basin, et al., 2004]; they show it is capable of determining whether or not a protocol is valid (usually in a few milliseconds) and, in the case of unsatisfiability, of producing a counter-example. One downside of OFMC is that it cannot verify group security protocols.

The formal model that OFMC tool uses for protocol analysis is based on two specification languages: one is a high-level protocol specification language (HLPSL) and the other a low-level one (the Intermediate Format IF). These languages were developed in the context of the AVISPA project.[4] AVISPA is a toolbox, which includes several backends, the most prominent of which are OFMC and CL-Atse.

Y. Chevalier and L. Vigneron. Chevalier e*t al.* developed the Constraint-Logic-Based Attack Searcher (CL-Atse), 2002 [Chevalier and Vigneron 2002]. It is an OCaml-based implementation of the deduction rules developped in the AVISPA European project. CL-Atse combines rewrite-based first order theorem proving with constrain logic in order to handle properties such as associability/commutativiy of operators for representing sets of messages. A protocol is written in the Avispa's itermediate format and it is executed over a finite (user decided) number of iterations, or entirely if no loop is involved. CL-Atse performs two kind of optimisations : First, the protocol specification is simplified to accelerate the search for attacks. Second, various optimisations in the deduction rules try to reduce, and often eliminate, redundancies and uselessness in the protocol execution.

Bozga *et al.* In 2003, Bozga *et al.* introduced HERMES [Bozga 2003], a tool specialized to verify secrecy properties. Roughly, HERMES takes a protocol and two sets, one contains messages which are to remain secret, called *S*, and the other hypotheses about secret keys, called *H*. Then, HERMES calculates all reachable states for a set of secrets *S'* (e.g. nonces and session keys) and a set of protected messages *H'* (e.g. cypher-texts). The protocol guarantees secrecy for secrets *S*, if all messages in *S'* are protected under *H'* and both, *S* is a subset of *S'* and *H'* is a subset of *H*. HERMES has a frontend (EVA) where users can specify protocols in a friendly way. It has been successfully tested on the Clark and Jacob library. One downside of HERMES is that it cannot prove security properties other than secrecy.

3.2.3 Strand Spaces approach

Fabrega et al.'s method Model checking tools aim at identifying if there exists a state in the execution of a protocol that violates a given security requirement. They do not aim to establish that a protocol really is correct. Motivated by this limitation, Fabrega, Herzog and Guttman invented the strand space model, in 1998 [Thayer-Fabrega, et al., 1998]. A *strand* is a sequence of events that an agent may execute in a protocol. A *strand space* is a set of strands reflecting the activity of principals, whether honest or not, involved in te run of a protocol

The strand space model is graph-based. An event is represented by a node in the graph. If the event corresponds to the sending of a message *M*, the node is labeled +*M*. Otherwise it corresponds to *M*'s reception and then is labeled -*M*. Every node belongs to a unique strand. A strand contains one or more nodes; it specifies the behavior of one agent. An inter-strand edge, denoted by $n_1 \to n_2$, captures agent communication, meaning: at n_1 (node 1) a message is sent, which is then received at n_2 (node 2).

A *bundle* is a finite acyclic subgraph, consisting of a number of strands all hooked together. It is such that for every node, n_2, annotated with a $-M$, there is a unique node, n_1, in a different strand such that $n_1 \to n_2$. Protocol correctness depends essentially on the freshness of a few data items such as nonces or session keys. These elements must not be in the possession of the spy. Typically, for a protocol to be correct, there must be one such a bundle consisting of one strand for each of the legitimate parties, all agreeing on the participants, nonces and session keys. Notice that a spy strand may also be entangled in a bundle, even if the protocol is correct, but that should not prevent the legitimate parties from agreeing on the data values or from maintaining the secrecy of the chosen values.

For proving authentication of type non-injective agreement with B acting as a responder and A as an initiator (according to Lowe's hierarchy of authentication [Lowe 1997]), it is necessary to establish that whenever a bundle

[4]The AVISPA project (Automated Validation of Internet Security Protocols and Applications) supports the simple integration of four different back-end search engines: the On the Fly Model Checker (OFMC), [Basin, et al., 2003]; the Constraint-Logic-Based Attack Searcher (CL-AtSe), [Chevalier and Vigneron 2002]; the SAT-based Model-Checker (SATMC), [Armando and Compagna 2004]; and the Tree Automata based on Automatic Approximations for the Analysis of Security Protocols (TA4SP), [Heam, et al., 2004].

contains a responder strand using certain data items, x, then such a bundle must also contain an initiator strand using x. Now, proving authentication of type agreement amounts to showing that such a bundle contains an unique initiator strand using x.

The strand space model was not designed to report a counterexample. Nor was it automated, at least initially. However, the strand space provided a simple, graphic way of expressing protocol information flow. The model has been found appealing because protocol runs can be expressed in parallel.

Dawn Song's method In 2001, Song, Berezin and Perrig used techniques from both model checking and theorem proving in order to extend the strand space model so that it is both automatic and able to prove correctness of many security protocols with an arbitrary number of concurrent runs. The implemention of these methods yielded Athena, a checker that efficiently analyzes infinite sets of protocol runs. Athena includes optimizations to improve efficiency, including the use of a pruning procedure based on novel theorems and an automatic evaluator of well-formed formulas.

Athena comes with a dedicated logic for describing security protocols and their intended properties. This logic is based on strands and bundles. To verify the protocol, Athena transforms the security properties into an initial sequent, capturing an initial state. It then applies a small set of inference rules with certain decision procedures to the states, building a proof tree, until it either completes the proof or refutes the sequent. In the latter case Athena reports that the protocol is incorrect, and the state of the refuted sequent represents a counterexample.

Athena has been tested on the Clark and Jacob library. An important downside of Athena is that the verification procedure is not guaranteed to terminate. Termination can be forced by limiting the number of concurrent protocol runs or the size of the message terms.

3.3 Theorem proving based approaches

Theorem proving based approaches attempt to produce a formal proof, given a description of the protocol, a set of logical axioms, and a set of inference rules. These approaches have two branches: one is based on Higher-Order Logic (HOL) and the other based on First-Order Logic (FOL). Although, the former approach can inductively simulate an infinite number of agents running in an infinite number of sessions, it requires considerable interaction with the theorem proving framework. The latter approach is more automatic and more viable for generating counterexamples.

3.3.1 Methods Based on Higher-Order Logic

Paulson's method The inductive approach to the verification of security protocols was invented by Paulson in 1998 [Paulson 1998]. This method combines some characteristics from state exploration and belief logics. From the first approach, it borrows a concrete notion of events; from the second one, the idea of deriving guarantees from each message. It has been widely used for verifying protocols, but it requires a high level of skill to use. A user must guide the proof process, selecting the logical rules or tactics to be applied. To compound the problem, the inductive approach involves cumbersome and onerous proofs. The analysis is supported by the generic, interactive higher-order logic theorem prover Isabelle [Paulson 1994].

A protocol is inductively defined as a set of (all possible) traces. A *trace* is a list of (communication) events. An *event* refers to either sending a message, or receiving one. Protocol descriptions include attacks and accidental losses of critical information, so-called an *Oops* event. The model involves four sorts of theories: *message*, *event*, *shared* and *public*. The message theory involves three main operators. Each operator is defined on possibly infinite sets of messages and is used to model properties of a protocol and reason about its runs. It also includes sorts of agents and sorts of messages. The event theory models the sending and receiving communication events, the knowledge of agents and a *freshness* operator. The theories shared and public formalize shared-key and public-key cryptography respectively. A protocol is defined by a collection of inference rules, each of which has zero or more hypotheses and one conclusion. The inference rules state the various forms in which a protocol trace can be possibly extended with new events. Proofs proceed by induction on the inference rules defining the protocol, taking into account the capabilities of a Dolev-Yao intruder.

Isabelle has been used to verify a number of protocols, including VISA's SET protocol. It was (as far as we know) the first to prove a group (recursive) protocol, namely Otway-Bull. This proof however was later on argued to be faulty, as it ignores a protocol failure[5] in the use of XOR [Ryan and Schneider 1998]. One downside of Paulson's approach is that there is no automated support for finding attacks on faulty protocols. Another downside is that verifying protocols requires interactive theorem proving, which demands considerable effort.

3.3.2 Methods Based on First-Order Logic

Meadows's method One of the first attempts at using automated theorem proving to security protocol analysis was due to Meadows. In 1996, she presented the NRL Protocol Analyzer (NPA) [Meadows 1996], a special-purpose tool, written in Prolog. Notably, NPA could be used to verify if a protocol was up to satisfy principal authentication or key distribution. Meadows's work extended Dolev and Yao's in two main respects. First, the spy attempts more than just finding out a secret. The spy may try, for example, to convince an agent that a message has certain properties that it does not. Second, NPA's method was a general, term-rewriting proof procedure, instead of an ad-hoc one (c.f. Dolev and Yao's approach).

NPA aims to prove that a collection of user-specified protocol states are unreachable. Using this strategy, it found two previously unknown attacks, one on Simmons's Selective Broadcast Protocol and the other on Burns-Mitchell's Resource Sharing Protocol. NPA also identified hidden assumptions in two protocols: the Neuman-Stubblebine re-authentication protocol and the Aziz-Diffie wireless communication protocol. NPA however is highly interactive, much in the same way a theorem prover is, relying on the user to guide the search for a proof. It is not guaranteed to terminate and has no means of converting a protocol description into a set of term-rewriting rules.

Blanchet's method Blanchet (2001), [Blanchet 2001], developed also a Prolog-based tool for verifying secrecy properties of security protocols. Blanchet uses Prolog rules to represent both the actions of the attacker and the rules of the protocol, which are modeled as messages known to the attacker. Messages and channels are represented as Prolog terms. Proving the secrecy of a message *M* amounts to disproving the fact *attacker(M)*, denoting that *M* is in the Spy possession. This verification task is beyond the Prolog solving strategy and so Blanchet provided a special-purpose solving algorithm that performs well in practice. Blanchet performs two abstractions to avoid bounding the number of protocol runs. The first abstraction ignores the number of times a message has appeared, recording only that it has been used. The second abstraction distinguishes when it is safe to use a unique term for the nonce associated with a pair of participants, regardless the number of runs in which they are involved. Blanchet's tool has been able to prove secrecy of a number of protocols, including Skeme. However, the tool representation of a protocol is approximate. On the one hand, the protocol representation is safe, in the sense that if there is an attack on the protocol that violates secrecy, the tool will find it. Yet, on the other hand it makes the tool yield false attacks; indeed, Blanchet's tool could not prove that Lowe's version of the NSPK is secure. While it does not bound the number of parallel protocol runs, the tool assumes a bounded number of participants.

Weidenbach's method Weidenbach's approach [Weidenbach 1999] aims to combine the benefits of the inductive method and finite state analysis. To this end, Weidenbach uses a fragment of first-order logic that is expressive enough to have infinite, inductive models, but that is still subject to automated theorem proving. Weidenbach's approach is as follows. First, the security protocol and its requirement are both translated into first-order monadic Horn formulae.[6] Then, these formulae are input to SPASS, a first-order logic automated theorem prover based on saturation. Finally, if SPASS terminates, the protocol is free of flaws. Otherwise, it is faulty. In the latter case of proof failure, SPASS's output can often be used to build an attack, though this process has not been automated.

Using SPASS, Weidenbach analyzed the Neumann-Stubblebine protocol (1993) and found new flaws in it, suggesting possible solutions. Weidenbachs' approach has four main downsides. First, the spy is weaker than Dolev-

[5] A *protocol failure* is a protocol flaw where the designer overlooks crucial properties of a crypt-algorithm and as a result the protocol security function is contrary to what is expected.

[6]A clause is Horn if it contains at most one positive literal. A *monadic Horn theory* is a set of Horn clauses where all occurring predicates are monadic (that is, they take only one argument).

Yao’s. Second, the approach considers that there are only two agents and, thus, a verification proof requires proving meta-theorems that relax this limitation. Third, it is not as expressive as Paulson's. Fourth, it does not automatically generate counter-examples.

Ernie Cohen's method In 2002, Cohen [Cohen 2000] introduced a first-order logiic method, capable of finding verification proofs similar to those proven in Isabelle, only that faster and almost without user guidance. In Cohen’s method, a protocol is modeled as a transition system. The state of a protocol run is then given by the set of transitions that have been executed together with the set of associate messages that have been exchanged. Transitions may generate a fresh value (used as either a nonce or a key). Provided that some preconditions hold, a transition fires resulting in the sending of a new message. Transitions are also used to model the actions of the spy. The state of a protocol run can be further restricted by user-supplied axioms.

Cohen implemented his method on TAPS[7] and then used it to analyze more than 70 protocols [Cohen 2003], most of which were borrowed from the Clark and Jacob Library. For a verification proof to go through, TAPS uses the protocol description to generate a number of protocol invariants, which are then used to establish security properties, such as secrecy or authentication. Often, TAPS is able to generate one such protocol invariant automatically, but sometimes the user has to provide them. Cohen's approach surpasses Paulson's in that it requires considerably less user guidance. One main downside of Cohen’s method is that it does not generate counterexamples because it searches for proofs, not attacks.

Graham Steel's method So far, we have described verification tools geared towards analyzing security protocols with a fixed number of participants. We now turn our attention into Coral [Steel et al. 2003], a theorem-prover based tool that aims to analyze group protocols, where an arbitrary number of parties may be involved in a single run [Meadows 2000].

Coral, developed by Steel *et al*., is built on top of SPASS, a first-order theorem prover. Crucially, Coral uses proof by consistency, a technique designed for automating inductive proofs in first-order logic. Proof by consistency, also known both as inductionless induction and as implicit induction, has the property of being refutation complete [Comon and Nieuwenhuis 1998]. This completeness result guarantees that Coral will eventually find an attack, if any [Steel et al. 2002]

Coral is closely related to Paulson's method, as it uses a first-order version of trhe inductive approach for protocol verification. A protocol is specified as an inductive datatype, which is used to capture the set of all possible protocol traces. Axioms specify the ways in which a trace may be extended, either by the sending of a message by an agent (following the protocol), or by the sending of a fake message by the spy. Steelk *et* al.’s model allows an indeterminate number of agents to get involved in one or more protocol runs, play any role and issue an arbitrary number of fresh nonces or keys.

The protocol description, together with the actions of the Dolev-Yao spy, is specified in Horn Clauses. To discover an attack on a protocol, the security property to be verified, the goal, is first negated and then SPASS is made attempt to disprove it. Thus, in Coral one attempts to find a counterexample that breaks the security property, rather than proving a protocol correct.

CORAL has found two new attacks on the Asokan-Ginzboorg protocol for establishing a group key in an ad-hoc network of Bluetooth devices. One downside of Coral is that it may not terminate. It does not therefore compete against state exploration tools for attack discovery in two- or three-party protocols, since these latter tools are faster.

4 Informal Design Guidelines in the Literature

In an attempt to prevent designers from incurring in known, published mistakes, many authors have suggested principles to guide the design of security protocols. These principles are just a collection of good practices, recommendations that are neither necessary nor sufficient for protocol correctness. Most principles have been

[7]The Telcordia Authentication Protocol System.

welcome by the community, only a few have received great criticism as they are thought to unnecessarily ask for a number of requirements.

A noteworthy work is that of Abadi and Needham's [Abadi and Needham 1996], who suggested eleven design principles for strengthening the design of security protocols. Abadi and Needham addressed two big issues: i) the messages involved in a protocol, together with their content (principles 1, 3-10); and ii) the trust relations upon which the participants of the protocol depend (principles 2 and 11). Principles 3-9 recommend good practices in the meaning of a message such as agent names (principle 3), nonces (principles 6 and 7), timestamps (principle 8), session keys (principle 9) and cyphered messages (principles 4 and 5). Principle 10 emphasizes that the messages of a protocol should be distinguishable one another. Anderson and Needham [Anderson and Needham 1995] extended Abadi and Needham's work, suggesting principles to avoid protocol failures, when applying in particular digital signatures and public key cryptography (c.f. Abadi and Needham's principles 4 and 5).

After Abadi and Needham's work, it was identified that the crux to security is to protect messages (specially cypher-texts) from being vulnerable to a replay attack. A *replay attack* is a form of attack where a data transmission is repeated or delayed out of the current run of the protocol. To avoid replay attacks, Carlsen [Carlsen 1994] gave a full list of information that should be attached to cypher-texts (he called this list *full information*): protocol identifier, step identifier, message subcomponents identifier, primitive type of data items and protocol run identifier. Protocol designers, however, find cumbersome including all this full information.

In a similar vein, Aura [Aura 1997] postulated four recommendations to avoid replay attacks: i) use a different cryptographic algorithm for each cyphered message in a protocol; ii) attach a hash function (including full information similar to Carlsen) to authenticator protocol messages; iii) produce unique session keys using hash functions; and iv) do not use any trust assumption about principals. While using a hash function does not affect the performance of a protocol, the use of a different cryptographic algorithm for each cyphered message is uncommon in the literature and it is not a design task.

Malladi *et al*. [Malladi, et al., 2002] have also suggested a recommendation to avoid replay attacks. The recommendation is similar to Aura's: associate with every protocol run a session-id generated by all protocol participants. They took a step further, proving that this condition guarantees that no replay attack can be elaborated, but did not provide a mechanism for how the participants are to agree upon the session-id. This is actually one of the main difficult problems in key-agreement protocols is to establish a secret between the participants.

5 Additional Support in the Development of Security Protocols

So far, we have described tools that support formal verification and collections of guidelines for security protocol design. In this section we present complementary approaches for protocol development, which include protocol repair and protocol synthesis.

5.1 Protocol repair

Lopez *et al*.'s tool for protocol repair Motivated by the number of faulty protocols repoted in the literature, Lopez *et al*. proposed methods for patching faulty security protocols. In [López-Pimentel, et al., 2007a] they proposed a method for patching security protocols vulnerable to a replay attack, where the flaw is to do with the omission of agent names. Later on, they developed Shrimp, [López-Pimentel, et al., 2007b] a tool which relies on existing state-of-the-art tools, capable of repairing a wider variety of protocols. Shrimp analyzes the protocol and an attack to this protocol in order to pinpoint faulty steps in the protocol and then suggests appropriate changes to fix them. This yields an improved version of the protocol that should be analyzed and possibly repaired again until no further flaws can be detected.

To approach protocol diagnostic and repair, Lopez *et al.* introduced a collection of rules, they called patch methods, each of which is able to deal with a class of faults. To identify and patch a protocol flaw, each rule makes use of Abadi and Needham's design principles, which Lopez *et al*. have translated into formal requirements. Shrimp deals with the full class of replay attacks, with respect to the taxonomy proposed by Syverson [Syverson 1994] (the only exception being the *type flaw* subclass, see Section 6.3.) Shrimp has been successfully tested on a number of faulty protocols, obtaining a repair rate of 90%.

R. Choo' tool on protocol repair R. Choo [Choo 2006] has also looked at the problem of automated protocol repair. He uses a model checker to perform state-space analysis on an indistinguishability-based model of a protocol, that uses asynchronous product automata. If the protocol is faulty, Choo's approach automatically repairs it. A fair comparison between Choo's approach and Shrimp is not possible since Choo's method is not elaborated within an archival publication.

5.2 Design and Protocol's Synthesis

Gong and Syverson's tool on fail-stop protocol Gong and Syverson [Gong and Syverson 1995] have developed a methodology to facilitate the design and analysis of security protocols. They proposed to restrict protocol design to well-defined practices. In particular, Gong and Syverson suggest to start with a fail-stop protocol, which should then be verified. A protocol is fail-stop if: i) the content of each message contains full information, as introduced by Carlsen; ii) each message contains the identies of the sender and the receiver; iii) each message is encrypted under the key shared between the sender and the intended recipient; iv) every honest agent follows the rules of the protocol; and v) an agent halts on any protocol run in which a message does not arrive within a specified time period. Gong and Syverson have come out with methods that produce protocols that are fail-stop and whenever a protocol is not, the method suggests changes to turn it into one that is. They have argued that, if fail-stop, a protocol is guaranteed to satisfy secrecy. To validate this claim, they have used BAN in the verification of fail-stop protocols. In this respect, Gong and Syverson's methodology suffers from the same limitations as the BAN logic.

Perrig and Song on synthesis of Protocols Perrig and Song [Perrig and Song 2000] have developed a system, called APG, for the synthesis of security protocols. The synthesis process, though automated, is generate and test: APG generates (extends) a protocol, step by step, taking into account a collection of security requirements. After generating a new collection of extended protocols, APG applies Athena to analyze the protocols, discarding those that do not satisfy such requirements. If no protocol satisfies the requirements, APG adds a new step to the protocols and then the resulting protocols are analyzed again. This cycle is repeated as many times as necessary until the requirements are met, if ever. APG is limited to generate only 3-party protocols (two principals and one server). The main downside of this approach is the combinatorial explosion: the search space is of order 10^{12} according to the authors.

6 Ongoing research

Ongoing research is mainly concerned with the so-called intruder deduction problem and with extending protocol analysis without limiting the number of participants or the number of protocol runs. The *intruder deduction problem* is stated as follows: given a state of one or more protocol runs, determine whether the intruder is able either to construct a message of the form that some honest agent is expecting to receive or to obtain a message that is intended to be secret, e.g. a key shared by two honest agents [Basin, et al., 2005]. In what follows, we summarize recent results on these topics.

6.1 Equational Message Theories and Message Typing

The intruder deduction problem is at the core of formally analyzing security protocols. Currently, research is concerned with extending the intruder deduction problem to consider algebraic properties of message operators and its relation to detecting type flaw attacks.

Algebraic Theories At first, most verification tools work under the following assumptions: i) the Spy is of type Dolev-Yao, ii) crypt-algorithms are perfect, and iii) terms are freely generated. Under the latter assumption, so-called *free algebra assumption*, two terms are equal if and only if they are syntactically equal. This assumption is inadequate in general as it abstracts away real features of concrete messages. For example, message concatenation, at the bit level, is associative; cryptographic primitives such as Diffie-Hellman exponentiation and bitwise XOR, are both associative and commutative. Considering equational properties of message operators in the intruder deduction problem is commonly referred to as to algebraic intruder theories. We summarize some interesting results below.

Common-Lundh and Shmatikov [Comon and Shmatikov2003] discovered decision results for faulty protocols in the presence of equational theories, like the bitwise XOR. Chevalier and Rusinowitch [Chevalier and Rusinowitch 2005] invented a general decision procedure for deciding security of protocols in the presence of disjoint sets of operators, provided that solvability of special ordered intruder constraints can be decided in each theory; this procedure, however, has not yet been mechanized. Basin *et al.* [Basin, et al., 2005] developed a framework which handles algebraic properties of cryptographic operators, both by using modular rewriting, and by introducing two bounding parameters, one for the depth of message terms and the other for the number of operations that the intruder can use to carry out message analysis; this framework is expected to be part of the OFMC distribution. Lafourcade *et al*. [Lafourcade, et al., 2007] have proved decidability of the intruder deduction problem for commutative group operators, such as XOR; they have developed a polynomial time decision procedure for the restricted case where the number of parallel protocol sessions is bounded.

Type Flaw Attacks A *type flaw attack* is an attack where a principal accepts a message component of one type as a message of another. It relies on a principal's inability to separate two messages of different steps [Meadows 2003]. Type flaw attacks are closely related to the intruder deduction problem because the intruder must be able to construct a message of the form that some honest agent is expecting to receive.

Heather et al. [Heather, et al., 2003] have shown that tagging every protocol message, and each element thereof, with a string indicating its intended type prevents all type flaw attacks. However, as noticed by Malladi and Alves-Foss [Malladi and Alves-Foss 2003], message tagging makes it easier to elaborate a password guessing attack. This is because each tag provides the adversary a means for identifying a hit, since the tag, which is a meaningful string, might become readable after a decryption attempt. Later on, Meadows [Meadows 2003] argued that even when messages are tagged type flaw attacks can still arise, since there is always a chance that two messages can be confused. She developed a method that enables the intruder to identify a strategy, if any, to raise the probability of two messages are confused above a preset threshold.

6.2 Group Protocols

Ongoing research is also concerned with enabling tools to perform verification of group protocols, in a way that the number of participants is not fixed in advance. Most tools were designed to verify two- or three-party security protocols, precluding the general modeling of group protocols. Moreover, they were designed to verify protocols with limited requirements. By contrast, a group protocol is designed to address several requirements, including: efficiency (e.g., the crypt-algorithm used, the number of messages exchanged); a wider variety of security guarantees (e.g., implicit key agreement, exchange fairness); the explicit type of channel (private, public); a specified logical group topology (e.g., ring); an explicit communication technology (e.g., wired, wireless); the scheme used in key generation (centralised, distributed); the level of involvement of the server, if any (e.g., online or offline); the communication paradigm (e.g., RPC, connection-oriented); and trust hierarchy. Thus, a protocol may be correct in some scenarios but not in others. Similarly, an attack may be valid in some scenarios but not in others. This makes it very difficult to attempt to extend current technology on protocol verification in a straightforward manner, prompting the need for starting from scratch. We will have to wait for a general setting to verify these kinds of protocols.

7 Conclusions

The design of security protocols is error-prone; flaws in security protocols have gone unnoticed for several years. Because of that, the formal methods community has put special interest in the rigorous verification of security protocols. We have described a number of approaches for the analysis of cryptographic protocols. We started with belief logics, where an honest environment is assumed. Then, we analyzed state-exploration approaches, which include a model of an antagonist intruder. We have seen that state-exploration tools are fast and automatic, but due to the state explosion problem they make strong assumtpions and thus some protocol flaws can be overlooked. Next, we studied theorem-proving based frameworks, of which the inductive approach is the most powerful, as it accommodates arbitrary numbers of agents running in a number of parallel sessions. However, the inductive approach has not yet been fully automated.

In an attempt to benefit from finite state exploration and the inductive approach, Steel *et al*. [Steel et al. 2003], Cohen [Cohen 2000], Basin [Basin 1999] and Weidenbach [Weidenbach 1999], among others have suggested approaches which are as expressive as the inductive approach but are still amenable to automation. Athena and AVISPA are in general considered the state of the art. Athena uses the strand space model, which is widely used to study what guarantees can be obtained through message dependencies. AVISPA is fast and has a number of switches that enable it to discover even type flaw attacks. Coral deserves more attention since it can find attacks on group security protocols automatically; this feature goes well beyond standard tools for protocol analysis.

When a protocol has been found to be faulty, the designer of the security protocol must fix the protocol manually. This task is onerous and difficult because the designer has to find the specific flaw causing the problem in the protocol and patch it considering several possibilities. In addition, even experimented designers must take care because they could introduce more vulnerabilities while patching a protocol. This has motivated the need for automated protocol repair. Shrimp is an automated tool that given the specification of a protocol and one of its counterexamples can detect the root of the protocol flaw and then suggest candidate patches. Protocol synthesis is another avenue for the development of sound security protocols. The most prominent tool for protocol synthesis is APG.

Ongoing research in the literature is about extending the verification of security protocols in order to capture cryptographic primitives, like Diffie-Hellman, XOR, exponentiation, etc. These extensions are very important because new protocols such as key agreement protocols, group protocols, etc., could be verified automatically. Ongoing research is also about dealing with the verification of security protocols vulnerable to a wider variety of attacks, including type flaws.

References

1. **Abadi, M. and Needham, R.,** Prudent Engineering Practice for Cryptographic Protocols. *IEEE Transactions on Software Engineering*, 22(1):6-15, 1996.
2. **Abadi, M. and Rogaway, P.,** Reconciling Two Views of Cryptography (The Computational Soundness of Formal Encryption). *Journal of Cryptology*, 15(2):103-127, 2002.
3. **Anderson, R.-J. and Needham, R.-M.,** Robustness Principles for Public Key Protocols. In *Proceedings of the 15th Annual International Cryptology Conference on Advances in Cryptology, CRYPTO '95*, edited by Don Coppersmith, LNCS Vol. 963, pp. 236-247, London, UK, 1995. Springer-Verlag.
4. **Armando, A. and Compagna, L.,** SATMC: A SAT-based model checker for security protocols. In *Proceedings of the 9th European Conference in Logics in Artificial Intelligence, JELIA'04*, edited by Alferes, J.-J. and Leite, J.-A., LNCS Vol. 3229, pp. 730-733. Springer, 2004.
5. **Asokan, N. and Ginzboorg, P.,** Key-Agreement in Ad-hoc Networks. *Computer Communications*, 23(17):1627-1637, 2000.
6. **Aura, Tuomas.,** Strategies against Replay Attacks. In *Proceedings of the 10th Computer Security Foundations Workshop (CSFW '97)*, page 59, Washington, DC, USA, 1997. IEEE Computer Society.
7. **AVISPA Team,** *AVISPA v1.0 User Manual*. v1.0 edition, 2005.
8. **Basin, D. and Mödersheim, S. and Viganò, L.,** An On-the-Fly Model-Checker for Security Protocol Analysis. In *Proceedings of the 8th European Symposium on Research in Computer Security, ESORICS'03*, edited by Gollmann, D. and Snekkenes, E., LNCS Vol. 2808, pp. 253-270, Gjøvik, Norway, 2003. Springer-Verlag.
9. **Basin, D. and Mödersheim, S. and Viganò, L.,** Algebraic Intruder Deductions. In Geoff Sutcliffe and Andrei Voronkov, editors, *Proceedings of Logic for Programming Artificial Intelligence and Reasoning, LPAR'05*, edited by Sutcliffe, G. and Voronkov, A., LNCS Vol. 3835, pp. 549-564, 2005. Springer-Verlag.

10. **Basin, D. and Mödersheim, S. and Viganò, L.,** OFMC: A Symbolic Model-Checker for Security Protocols. Technical report, 450, ETH Zürich, Computer Science, 2004.
11. **Basin, David.** Lazy Infinite-State Analysis of Security Protocols. In Baumgart, Rainer, editors, *Proceedings of the International Exhibition and Congress on Secure Networking*, *CQRE'99*, edited by Baumgart, R., LNCS Vol. 1740, pp. 30-42, London, UK, 1999. Springer-Verlag.
12. **Baudet, M. and Cortier, V. and Kremer, S.,** Computationally sound implementations of equational theories against passive adversaries. In *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming, ICALP'05,* edited by Caires, L. and Italiano, G.-F. and Monteiro, L. and Palamidessi, C. and Yung, M., LNCS Vol. 3580, pp. 652-663, 2005. Springer.
13. **Blanchet**, **Bruno**, An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *Proceedings of the 14th IEEE Computer Security Foundations Workshop, CSFW'01*, pp. 82-96, IEEE Computer Society, 2001.
14. **Bozga, L. and Lacknech, Y. and Périn, M**. HERMES: An Automatic Tool for Verification of Secrecy in Security Protocols. In *Proceedings of the 15th International Conference in Computer Aided Verification CAV'03*, LNCS Vol. 2725, pp. 219-222, Boulder, CO, USA, 2003. Springer.
15. **Brackin, S.-H.,** A HOL Extension of GNY for Automatically Analyzing Cryptographic Protocols. In *Proceedings of The 9th Computer Security Foundations Workshop*, *CSFW'96*, page 62, Washington, DC, USA, 1996. IEEE Computer Society Press.
16. **Burrows, M. and Abadi, M. and Needham, R.-M.,** A Logic of Authentication. *Proceedings of the Royal Society of London*, 426(1):233-71, 1989.
17. **Carlsen, Ulf**, Cryptographic Protocols Flaws. In *Proceedings IEEE Computer Security Foundations Workshop, CSFW'94*, pp. 192-200, 1994. IEEE Computer Society Press.
18. **Chevalier and Vigneron 2002]Chevalier, Y. and Vigneron, L.,** Automated unbounded verification of security protocols. *In Proceedings of the 14th International Conference on Computer Aided Verification*, *CAV '02,* edited by Brinksma, E. and Larsen, K.-G., LNCS Vol. 2404, pp. 324-337, London, UK, 2002. Springer-Verlag**.**
19. **Chevalier, Y. and Rusinowitch, M.,** Combining Intruder Theories. In *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming, ICALP`05,* edited by Caires, L. and Italiano, G.-F. and Monteiro, L. and Palamidessi, C. and Yung, M., LNCS Vol. 3580, pp. 639-651, 2005. Springer Berlin / Heidelberg.
20. **Choo, K.-K. Raymond.**, An Integrative Framework to Protocol Analysis and Repair: Indistinguishability Based Model + Planning + Model Checker. In *Proceedings of Five-minute Talks at CSFW'06,* 2006.
21. **Cohen, Ernie.,** First-order verification of cryptographic protocols. *Journal of Computer Securirity*, 11(2):189-216, 2003.
22. **Cohen, Ernie.,** TAPS: A First-Order Verifier for Cryptographic Protocols. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop, CSFW '00*, pp. 144, Washington, DC, USA, 2000. IEEE Computer Society.
23. **Comon, H. and Nieuwenhuis, R.,** Induction = I-Axiomatization + First-Order Consistency. Technical report, LSV-98-9, Laboratoire Spécification et Vérification, ENS Cachan, France, Cachan, France, 1998.
24. **Comon-Lundh, H. and Shmatikov, V.,** Intruder Deductions, Constraint Solving and Insecurity Decision in Presence of Exclusive Or. In *Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science*, *LICS '03*, pp. 271, Washington, DC, USA, 2003. IEEE Computer Society.
25. **Dolev, D. and Yao, A.-C.,** On the security of public key protocols. Technical report, 2, Stanford University, Stanford, CA, USA, 1983.
26. **Gong, L. and Syverson P.,** Fail-stop protocols: A new approach to designing secure protocols. *In Proceedings of the 5th International Working Conference on Dependable Computing for Critical Applications*, pp. 44-55, 1995.
27. **Heam, P.-C. and Boichut, Y. and Kouchnarenko, O. and Oehl, F.,** Improvements on the genet and klay technique to automatically verify security protocols. In *Proceedings of the International WS on Automated Verification of Infinite-State Systems, AVIS'2004,* joint to ETAPS'04, pp. 1–11, Barcelona, Spain, 2004.
28. **Heather, J. and Lowe, G. and Schneider, S.,** How to prevent type flaw attacks on security protocols. *Journal of Computer Security*, 11(2):217-244, 2003.

29. **Kremer, S. and Mazaré, L.**, Adaptive Soundness of Static Equivalence. In *Proceedings of the 12th European Symposium on Research in Computer Security, ESORICS'07,* edited by Biskup, J. and Lopez, J., LNCS Vol. 4734, pp. 610-625, 2007. Springer.
30. **Lafourcade, P. and Lugiez, D. and Treinen, R.,** Intruder deduction for the equational theory of Abelian groups with distributive encryption. *Information and Compututation*, 205(4):581-623, 2007.
31. **López-Pimentel, J.-C. and Monroy, R. and Hutter, D.,** A Method for Patching Interleaving-Replay Attacks in Faulty Security Protocols. *Electronic Notes in Theoretical Computer Science*, 174:117-130, 2007. Also available from the Proceedings of the 1st FLoC Workshop on Verification and Debugging.
32. **López-Pimentel, J.-C. and Monroy, R. and Hutter, D.,** On the Automated Correction of Faulty Security Protocols Susceptible to a Replay Attack. In *Proceedings of the 12th European Symposium Research Computer Security, ESORICS'07*, edited by Biskup, J. and Lopez, J., LNCS Vol.4734, pp. 594-609, 2007. Springer.
33. **Lowe, Gavin.,** An Attack on the Needham-Schroeder Public-Key Authentication Protocol. *Information Processing Letters*, 56(3):131-133, 1995.
34. **Lowe, Gavin.,** A Hierarchy of Authentication Specifications. In *Proceedings of the 10th Computer Security Foundations Workshop, CSFW* '97, pp. 31, Rockport, Massachusetts, USA, 1997. IEEE Computer Society.
35. **Lowe, Gavin.**, Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR. In *Proceedings of the Second International Workshop on Tools and Algorithms for Construction and Analysis of Systems, TACAS'96*, edited by Margaria, T. and Steffen, B., LNCS Vol. 1055, pp. 147-166, London, UK, 1996. Springer-Verlag.
36. **Lowe, Gavin.**, Casper: a compiler for the Analysis of Security Protocols. In *Proceedings of the 10th Computer Security Foundations Workshop, CSFW'97,* pp. 53-84, Journal in Computer Security, Vol. 6, IEEE Computer Society, Washington, DC, USA, 1998.
37. **Malladi and Alves-Foss 2003]Malladi, S. and Alves-Foss, J.,** How to prevent type-flaw guessing attacks on password protocols. In *Proceedings of the 2003 Workshop on Foundations of Computer Security (FCS03),* pp. 1-12, 2003. Technical Report of University of Ottawa.
38. **Malladi, S. and Alves-Foss, J. and Heckendorn, R.,** On Preventing Replay Attacks on Security Protocols. In *Proceedings International Conference on Security and Management, ICSM'02*, pp. 77-83, 2002.
39. **Meadows, Catherine.,** The NRL Protocol Analyzer: An Overview. *Journal of Logic Programming*, 26(2):113-131, 1996.
40. **Meadows, Catherine.**, Extending Formal cryptographic protocol analysis techniques for group protocols and low-level cryptographic primitives. In *Proceedings of the First Workshop on Issues in the Theory of Security, WITS'00*, edited by Degano, P., pp. 87-92, Geneva, Switzerland, July, 2000.
41. **Meadows, Catherine,** A Procedure for Verifying Security Against Type Confusion Attacks. In *Proceedings of the 16th IEEE Computer Security Foundations Workshop, CSFW'03*, pp. 62, Pacific Grove, CA, USA, 2003. IEEE Computer Society.
42. **Paulson, L.-C.** , *Isabelle: a Generic Theorem Prover*. Springer-Verlag, 1994.
43. **Paulson, L.-C.,** The Inductive Approach to Verifying Cryptographic Protocols. *Journal in Computer Security*, 6(1-2):85-128, 1998.
44. **Pereira, O. and Quisquater, J.-J.,** Some attacks upon authenticated group key agreement protocols. *Journal in Computer Security*, 11(4):555-580, 2003.
45. **Perrig, A. and Song D.,** Looking for Diamonds in the Desert --- Extending Automatic Protocol Generation to Three-Party Authentication and Key Agreement Protocols. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop, CSFW'00,* pp. 64-76, 2000. IEEE Computer Society Press.
46. **Rusinowitch, Michaël and Turuani, Mathieu**. Protocol Insecurity with Finite Number of Sessions is NP-Complete. *In Proceedings of the 2001 Computer Security Foundations Workshop, CSFW 2001*, pp. 174-190, Computer Science Press, 2001.
47. **Ryan, P.Y.-A. and Schneider, S.-A**. An attack on a recursive authentication protocol; a cautionary tale. *Information Processing Letters,* **65**(1):7-10 (1998).
48. **Song, X. D. and Berezin, S. and Perrig, A.,** Athena: A Novel Approach to Efficient Automatic Security Protocol Analysis. *Journal of Computer Security*, 9(1-2):47-74, 2001.

49. **Steel, G. and Bundy, A. and Denney, E.,** Finding Counterexamples to Inductive Conjectures and Discovering Security Protocol Attacks. *Proceedings of the Foundations of Computer Security Workshop, (FCS'02)*, pp. 81-90, 2002. Also appeared in *Proceedings of The Verify'02 Workshop*. Also available as Informatics Research Report EDI-INF-RR-0141.
50. **Steel, G. and Bundy, A. and Maidl, M.,** Attacking the Asokan-Ginzboorg Protocol for Key Distribution in an Ad-Hoc Bluetooth Network Using CORAL. In *Proceedings of 23rd IFIP International Conference on Formal Techniques for Networked and Distributed Systems, IFIP TC6 /WG 6.1, FORTE'03*, edited by König, H. and Heiner, M. and Wolisz, A., pp. 1-10, 2003.
51. **Syverson, Paul.**, A taxonomy of replay attacks. In *Proceedings of the Seventh Computer Security Foundations Workshop*, CSFW'94, pp. 187-191, Franconia, New Hampshire, USA, 1994. IEEE Computer Society Press.
52. **Syverson, P. and Meadows, C. and Cervesato, I**. Dolev-Yao is no better than Machiavelli. In *Proceedings of the First Workshop on Issues in the Theory of Security, WITS'00*, 2000.
53. **Thayer-Fabrega, F.-J. and Herzog, J.-C. and Guttman, J.-D.,** Strand spaces: Why is a security protocol correct? In *Proceedings of the 1998 Symposium on Security and Privacy*, pp. 160-171, Oakland, CA, USA, 1998. IEEE computer Society.
54. **Weidenbach, Christoph.**, Towards an Automatic Analysis of Security Protocols in First-Order Logic. In *Proceedings of the 16th International Conference on Automated Deduction, CADE-16*, edited by Harald Ganzinger, LNCS Vol. 1632, pp. 314-328, London, UK, 1999. Springer-Verlag.

Juan Carlos López Pimentel *graduated with honours a BSc on Engineering in Computer Systems at ITTG, December of 2001. Later in December 2003, he concluded his MSc in Computer Science. He has just finished his PhD in Computer Science, (May of 2008) at Tecnológico de Monterrey-Campus Estado de México under the supervision of Dr. Raúl Monroy. Mr. Lopez's research focused on patching automatically faulty security protocols. Nowadays, he is Professor in Computing at Tecnológico de Monterrey, Campus Chiapas. He is interested in Information Technology, Networking and Computer Security*

Raúl Monroy *obtained a PhD in Artificial Intelligence in 1998 from Edinburgh University, under the supervision of Prof. Alan Bundy. He is Associate Professor in Computing at Tecnológico de Monterrey, Campus Estado de México. Since 1998 he is a member of CONACyT's National Research System. Dr. Monroy's research is concerned with the discovery and application of general search control strategies for uncovering and correcting errors in either a system or its specification; for detecting anomalies endangering information security; and for planning robot motion.*