Security Protocol Verification: Symbolic and Computational Models

Bruno Blanchet

Security Protocol Verification: Symbolic and Computational Models

Bruno Blanchet

INRIA, École Normale Supérieure, CNRS, Paris
blanchet@di.ens.fr

Abstract. Security protocol verification has been a very active research area since the 1990s. This paper surveys various approaches in this area, considering the verification in the symbolic model, as well as the more recent approaches that rely on the computational model or that verify protocol implementations rather than specifications. Additionally, we briefly describe our symbolic security protocol verifier ProVerif and situate it among these approaches.

1 Security Protocols

Security protocols are programs that aim at securing communications on insecure networks, such as Internet, by relying on cryptographic primitives. Security protocols are ubiquitous: they are used, for instance, for e-commerce (e.g., the protocol TLS [109], used for `https://` URLs), bank transactions, mobile phone and WiFi networks, RFID tags, and e-voting. However, the design of security protocols is particularly error-prone. This can be illustrated for instance by the very famous Needham-Schroeder public-key protocol [160], in which a flaw was found by Lowe [147] 17 years after its publication. Even though much progress has been made since then, many flaws are still found in current security protocols (see, e.g., `http://www.openssl.org/news/` and `http://www.openssh.org/security.html`). Security errors can have serious consequences, resulting in loss of money or loss of confidence of users in the system. Moreover, security errors cannot be detected by functional software testing because they appear only in the presence of a malicious adversary. Automatic tools can therefore be very helpful in order to obtain actual guarantees that security protocols are correct. This is a reason why the verification of security protocols has been a very active research area since the 1990s, and is still very active. This survey aims at summarizing the results obtained in this area. Due to the large number of papers on security protocol verification and the limited space, we had to omit many of them; we believe that we still present representatives of the main approaches. Additionally, Sect. 2.2 briefly describes our symbolic verification tool ProVerif.

1.1 An Example of Protocol

We illustrate the notion of security protocol with the following example, a simplified version of the Denning-Sacco public-key key distribution protocol [108].

$$\begin{array}{l}\text{Message 1. } A \to B : \{\{k\}_{sk_A}\}_{pk_B} \quad k \text{ fresh}\\ \text{Message 2. } B \to A : \{s\}_k\end{array}$$

As usual, $A \to B : M$ means that A sends to B the message M; $\{M\}_{sk}$ denotes the signature of M with the secret key sk (which can be verified with the public key pk); $\{M\}_{pk}$ denotes the public-key encryption of message M under the public key pk (which can be decrypted with the corresponding secret key sk); $\{M\}_k$ denotes the shared-key encryption of message M under key k (which can be decrypted with the same key k). In this protocol, the principal A chooses a fresh key k at each run of the protocol. She signs this key with her signing key sk_A, encrypts the obtained message with the public key of her interlocutor B, and sends him the message. When B receives it, he decrypts it (with his secret key sk_B), verifies the signature of A, and obtains the key k. Having verified this signature, B is convinced that the key was chosen by A, and encryption under pk_B guarantees that only B could decrypt the message, so k should be shared between A and B. Then, B encrypts a secret s under the shared key k. Only A should be able to decrypt the message and obtain the secret s.

In general, in the literature, as in the example above, the protocols are described informally by giving the list of messages that should be exchanged between the principals. Nevertheless, one must be careful that these descriptions are only informal: they indicate what happens in the absence of an adversary. However, an adversary can capture messages and send his own messages, so the source and the target of a message may not be the expected one. Moreover, these descriptions leave implicit the verifications done by the principals when they receive messages. Since the adversary may send messages different from the expected ones, and exploit the obtained reply, these verifications are very important: they determine which messages will be accepted or rejected, and may therefore protect or not against attacks. Formal models of protocols, such as [5, 7, 72, 117] make all this precise.

Although the explanation above may seem to justify its security informally, this protocol is subject to an attack:

$$\begin{array}{ll}\text{Message 1. } A \to C : & \{\{k\}_{sk_A}\}_{pk_C}\\ \text{Message 1'. } C(A) \to B : & \{\{k\}_{sk_A}\}_{pk_B}\\ \text{Message 2. } B \to C(A) : & \{s\}_k\end{array}$$

In this attack, A runs the protocol with a dishonest principal C. This principal gets the first message of the protocol $\{\{k\}_{sk_A}\}_{pk_C}$, decrypts it and re-encrypts it under the public key of B. The obtained message $\{\{k\}_{sk_A}\}_{pk_B}$ corresponds exactly to the first message of a session between A and B. Then, C sends this message to B impersonating A; above, we denote by $C(A)$ the dishonest participant C impersonating A. B replies with the secret s, intended for A, encrypted under k. C, having obtained the key k by the first message, can decrypt this message and obtain the secret s.

The protocol can easily be fixed, by adding the identity of B to the signed message, which yields the following protocol:

Message 1. $A \to B : \{\{B, k\}_{sk_A}\}_{pk_B}$ k fresh
Message 2. $B \to A : \{s\}_k$

When he receives the first message, B verifies that his own identity appears as first component. After this change, in a session between A and C, the adversary C receives $\{\{C, k\}_{sk_A}\}_{pk_C}$. It cannot transform this message into $\{\{B, k\}_{sk_A}\}_{pk_B}$, because it cannot transform the signature that contains C into a signature that contains B instead. Therefore, the previous attack is impossible. However, this point does not prove that the protocol is correct: there may be other attacks, so a security proof is needed.

1.2 Models of Protocols

In order to obtain proofs that security protocols are correct, one first needs to model them mathematically. Two models of protocols have been considered:

- In the *symbolic model*, due to Needham and Schroeder [160] and Dolev and Yao [110] and often called Dolev-Yao model, the cryptographic primitives are represented by function symbols considered as black-boxes, the messages are terms on these primitives, and the adversary is restricted to compute only using these primitives. This model assumes perfect cryptography. For instance, shared-key encryption is basically modeled by two function symbols, enc and dec, where $\mathsf{enc}(x, y)$ stands for the encryption of x under key y and $\mathsf{dec}(x, y)$ for the decryption of x with key y, with the equality:
$$\mathsf{dec}(\mathsf{enc}(x, y), y) = x. \tag{1}$$
Hence, one can decrypt $\mathsf{enc}(x, y)$ only when one has the key y. More generally, one can add equations to model algebraic properties of the cryptographic primitives, but one always makes the assumption that the only equalities that hold are those explicitly given by these equations.
- In the *computational model*, developed at the beginning of the 1980s by Goldwasser, Micali, Rivest, Yao, and others (see for instance [123, 124, 180]), the messages are bitstrings, the cryptographic primitives are functions from bitstrings to bitstrings, and the adversary is any probabilistic Turing machine. This is the model generally used by cryptographers.
In this model, the length of keys is determined by a value named security parameter, and the runtime of the adversary is supposed to be polynomial in the security parameter. A security property is considered to hold when the probability that it does not hold is negligible in the security parameter. (A function is said to be negligible when it is asymptotically smaller than the inverse of any polynomial.) This probability can also be bound explicitly as a function of the runtime of the adversary and of the probability of breaking each cryptographic primitive; this is called exact security.
For instance, shared-key encryption can be modeled by two functions enc and dec with the same equality (1) as above, but the security of encryption is expressed (informally) by saying that the adversary has a negligible probability of distinguishing encryptions of two messages of the same length [39]. Equalities other than (1) may exist, even if they are not made explicit.

The computational model is much more realistic, but complicates the proofs, and until recently these proofs were only manual. The symbolic model, however, is suitable for automation, essentially by computing the set of all messages the adversary can know. Starting in the 1990s, the proof of protocols in the symbolic model has been an important application field for formal verification methods.

We emphasize that even the computational model is just a model, which ignores many important aspects of reality. In particular, it ignores physical attacks against the devices: side-channel attacks exploit power consumption, timing, noise, ... and fault attacks introduce faults in the system in order to break its security. As protocols are better studied and verified formally, physical attacks become increasingly important and are an area of active research, with some workshops, such as FTDC (Fault Diagnosis and Tolerance in Cryptography) and CHES (Cryptographic Hardware and Embedded Systems), focusing mainly on this area. This survey will not deal with physical attacks.

1.3 Security Properties

Security protocols can aim at a wide variety of security goals. The main security properties can be classified into two categories, *trace properties* and *equivalence properties*. We define these categories and mention two particularly important examples: secrecy and authentication. These are two basic properties required by most security protocols. Some protocols, such as e-voting protocols [104], require more complex and specific security properties, which will not be discussed here.

Trace and Equivalence Properties. Trace properties are properties that can be defined on each execution trace (each run) of the protocol. The protocol satisfies such a property when it holds for all traces in the symbolic model, except for a set of traces of negligible probability in the computational model. For example, the fact that some states are unreachable is a trace property.

Equivalence or indistinguishability properties mean that the adversary cannot distinguish two processes. For instance, one of these processes can be the protocol under study, and the other one can be its specification. Then, the equivalence means that the protocol satisfies its specification. Equivalences can be therefore be used to model many subtle security properties. In the symbolic model, this notion is called process equivalence, with several variants (observational equivalence, testing equivalence, trace equivalence) [5–7], while in the computational model, one rather talks about indistinguishability. Equivalences provide compositional proofs: if a protocol P is equivalent to P', P can be replaced with P' in a more complex protocol. In the computational model, this is at the basis of the idea of universal composability [70]. However, in the symbolic model, their proof is more difficult to automate than the proof of trace properties: they cannot be expressed on a single trace, they require relations between traces (or processes). So most equivalence proofs are still manual, even if tools begin to appear as we shall see in Sect. 2.1.

Secrecy. Secrecy, or confidentiality, means that the adversary cannot obtain some information on data manipulated by the protocol. In the symbolic model, secrecy can be formalized in two ways:

- Most often, secrecy means that the adversary cannot compute exactly the considered piece of data. In case of ambiguity, this notion will be called syntactic secrecy. For instance, in the protocol of Sect. 1.1, we may want to prove that the adversary cannot obtain s nor the key k shared between A and B. These properties hold only for the fixed protocol of Sect. 1.1.
- Sometimes, one uses a stronger notion, strong secrecy, which means that the adversary cannot detect a change in the value of the secret [1, 48]. In other words, the adversary has no information at all on the value of the secret. In the fixed protocol of Sect. 1.1, we could also show strong secrecy of s.

The difference between syntactic secrecy and strong secrecy can be illustrated by a simple example: consider a piece of data for which the adversary knows half of the bits but not the other half. This piece of data is syntactically secret since the adversary cannot compute it entirely, but not strongly secret, since the adversary can see if one of the bits it knows changes. Syntactic secrecy cannot be used to express secrecy of data chosen among known constants. For instance, talking about syntactic secrecy of a bit 0 or 1 does not make sense, because the adversary knows the constants 0 and 1 from the start. In this case, one has to use strong secrecy: the adversary must not be able to distinguish a protocol using the value 0 from the same protocol using the value 1. These two notions are often equivalent [91], both for atomic data (which are never split into several pieces, such as nonces, which are random numbers chosen independently at each run of the protocol) and for probabilistic cryptographic primitives.

Strong secrecy is intuitively closer to the notion of secrecy used in the computational model, which means that a probabilistic polynomial-time adversary has a negligible probability of distinguishing the secret from a random number [9].

Syntactic secrecy is a trace property, while strong secrecy and computational secrecy are equivalence properties.

Authentication. Authentication means that, if a participant A runs the protocol apparently with a participant B, then B runs the protocol apparently with A, and conversely. In general, one also requires that A and B share the same values of the parameters of the protocol.

In the symbolic model, this is generally formalized by correspondence properties [148, 179], of the form: if A executes a certain event e_1 (for instance, A terminates the protocol with B), then B has executed a certain event e_2 (for instance, B started a session of the protocol with A). There exist several variants of these properties. For instance, one may require that each execution of e_1 corresponds to a distinct execution of e_2 (injective correspondence) or, on the contrary, that if e_1 has been executed, then e_2 has been executed at least once (non-injective correspondence). The events e_1 and e_2 may also include more or fewer parameters depending on the desired property. These properties are trace properties.

For example, in the fixed protocol of Sect. 1.1, we could show that, if B terminates the protocol with A and a key k, then A started the protocol with B and the same k. The injective variant does not hold, because the adversary can replay the first message of the protocol.

The formalization is fairly similar in the computational model, with the notion of *matching conversations* [41] and more recent formalizations based on session identifiers [9, 40], which basically require that the exchanged messages seen by A and by B are the same, up to negligible probability. This is also a trace property.

2 Verifying Protocols in the Symbolic Model

A very large number of techniques and tools exist for verifying protocols in the symbolic model. We first present a survey of these techniques, then provide additional details on the tool that we have developed, ProVerif.

2.1 Verification Techniques

The automatic verification of protocols in the symbolic model is certainly easier than in the computational model, but it still presents significant challenges. Essentially, the state space to explore is infinite, for two reasons: the message size is not bounded in the presence an active adversary; the number of sessions (runs) of the protocol is not bounded. However, we can easily bound the number of participants to the protocol without forgetting attacks [84]: for protocols that do not make difference tests, one honest participant is enough for secrecy if the same participant is allowed to play all roles of the protocol, two honest participants are enough for authentication.

A simple solution to this problem is to explore only part of the state space, by limiting arbitrarily both the message size and the number of sessions of the protocol. One can then apply standard model-checking techniques, using systems such as FDR [147] (which was used to discover the attack against the Needham-Schroeder public-key protocol), Murϕ [158], Maude [106], or SATMC (SAT-based Model-Checker) [16]. These techniques allow one to find attacks against protocols, but not to prove the absence of attacks, since attacks may appear in an unexplored part of the state space. (One can indeed construct a family of protocols such that the n-th protocol is secure for $n-1$ sessions but has an attack with n parallel sessions [153]. More generally, an arbitrary number of sessions may be needed [84].)

If only the number of sessions is bounded, the verification of protocols remains decidable: protocol insecurity (existence of an attack) is NP-complete with reasonable assumptions on the cryptographic primitives [170]. When cryptographic primitives have algebraic relations, the verification is much more difficult, but the complexity class does not necessarily increase. For instance, exclusive or is handled in the case of a bounded number of sessions in [78, 79, 86] and the Diffie-Hellman key agreement in [77], still with an NP-complexity. Practical algorithms

have been implemented to verify protocols with a bounded number of sessions, by constraint solving, such as [154] and CL-AtSe (Constraint-Logic-based Attack Searcher) [80], or by extensions of model-checking such as OFMC (On-the-Fly Model-Checker) [35].

The previous results only deal with trace properties. The verification of equivalence properties is much more complex. First, decision procedures were designed for a fixed set of basic primitives and without else branches [112, 134], but their complexity was too large for practical implementations. Recently, more practical algorithms were designed for processes with else branches and non-determinism [75, 76] or for a wide variety of primitives with the restriction that processes are determinate, that is, their execution is entirely determined by the adversary inputs [81, 87]. Diff-equivalence, a strong equivalence between processes that have the same structure but differ by the terms they contain, is also decidable [36]; this result applies in particular to the detection of off-line guessing attacks against password-based protocols and to the proof of strong secrecy. These techniques rely on symbolic semantics: in a symbolic semantics, such as [64, 103, 146], the messages that come from the adversary are represented by variables, to avoid an unbounded case distinction on these messages.

For an unbounded number of sessions, the problem is undecidable [114] for a reasonable model of protocols. Despite this undecidability, many techniques have been designed to verify protocols with an unbounded number of sessions, by restricting oneself to subclasses of protocols, by requiring user interaction, by tolerating non-termination, or with incomplete systems (which may answer "I don't know"). Most of these techniques deal with trace properties; only the type system of [1] and ProVerif [54] deal with equivalence properties. Next, we present a selection of these techniques.

- Logics have been designed to reason about protocols. Belief logics, such as the BAN logic, by Burrows, Abadi, and Needham [69], reason about what participants to the protocol believe. The BAN logic is one of the first formalisms designed to reason about protocols. However, the main drawback of these logics is that they do not rely directly on the operational semantics of the protocol.
  Another logic, PCL (Protocol Composition Logic) [100, 115] makes it possible to prove that a formula holds after some participant has run certain actions, by relying on the semantics of the protocol. It allows systematic and rigorous reasoning on protocols, but has not been automated yet.
- Theorem proving was used for proving security properties of protocols [164]. Proofs in an interactive theorem prover typically require much human interaction, but allow one to prove any mathematically correct result.
- Typing was also used for proving protocols. Abadi [1] proved strong secrecy for protocols with shared-key encryption. Abadi and Blanchet [2] designed a type system for proving secrecy, which supports a wide variety of cryptographic primitives. Gordon and Jeffrey [125–127] designed the system Cryptyc for verifying authentication by typing. They handle shared-key and public-key cryptography.

In all these type systems, the types express information on the security level of data, such as "secret" for secret data and "public" for public data. Typing is better suited for at least partly manual usage than for fully automatic verification: type inference is often difficult, so type annotations are necessary. Type checking can often be automated, as in the case of Cryptyc. Types provide constraints that can help protocol designers guaranteeing the desired security properties, but existing protocols may not satisfy these constraints even if they are correct.

- Strand spaces [117] are a formalism that allows to reason about protocols. This formalism comes with an induction proof technique based on a partial order that models a causal precedence relation between messages. It was used both for manual proofs and in the automatic tool Athena [174] which combines model checking and theorem proving, and uses strand spaces to reduce the state space. Scyther [99] uses an extension of Athena's method with trace patterns to analyze a group of traces simultaneously. These tools sometimes limit the number of sessions to guarantee termination.
- Broadfoot, Lowe, and Roscoe [66, 67, 169] extended the model-checking approach to an unbounded number of sessions. They recycle nonces, to use a finite number of nonces for an infinite number of executions.
- One of the very first approaches for protocol verification is the Interrogator [155, 156]. In this system, written in Prolog, the reachability of a state after a sequence of messages is represented by a predicate, and the program runs a backward search to determine whether a state is reachable or not. The main problem of this approach is non-termination. It is partly solved by making the program interactive, so that the user can guide the search. The NRL Protocol Analyzer (NPA, which evolved into Maude-NPA) [116, 150] considerably improves this technique by using narrowing in rewrite systems. It does not make any abstraction, so it is sound and complete but may not terminate.
- Decidability results can be obtained for an unbounded number of sessions, for subclasses of protocols. For example, Ramanujan and Suresh [168] showed that secrecy is decidable for a class of tagged protocols. Tagged protocols are protocols in which each message is distinguished from others by a distinct constant, named tag. Their tagging scheme prevents blind copies, that is, situations in which a message is copied by a participant of the protocol without verifying its contents. Extensions of this decidability result include [14, 82]. In general, these decidability results are very restrictive in practice.
- Several methods rely on abstractions [98]: they overestimate the attack possibilities, most often by computing a superset of the knowledge of the adversary. They yield fully automatic but incomplete systems.
  - Bolignano [63] was a precursor of abstraction methods for security protocols. He merges key, nonces, ... so that a finite set remains. He can then apply a decision procedure.
  - Monniaux [159] introduced a verification method based on an abstract representation of the knowledge of the adversary by tree automata. This method was extended by Goubault-Larrecq [128]. Genet and Klay [122]

combine tree automata with rewriting. This method lead to the implementation of the verifier TA4SP (Tree-Automata-based Automatic Approximations for the Analysis of Security Protocols) [62].
This approach abstracts away relational information on terms: when a variable appears several times in a message, one forgets that it has the same value at all its occurrences in the message, which limits the precision of the analysis. However, thanks to this approximation, this method always terminates.

- Weidenbach [178] introduced an automatic method for proving protocols based on resolution on Horn clauses. This method is at the heart of the verifier ProVerif and will be detailed in Sect. 2.2. It is incomplete since it ignores the number of repetitions of each action of the protocol. Termination is not guaranteed in general, but it is guaranteed on certain subclasses of protocols, and it can be obtained in all cases by an additional approximation, which loses relational information by transforming Horn clauses into clauses of the decidable subclass $\mathcal{H}_1$ [129]. This method can be seen as a generalization of the tree automata verification method. (Tree automata can be encoded as Horn clauses.) With Martín Abadi [2], we showed that this method is equivalent to the most precise instance of a generic type system for security protocols.
- Other abstraction-based techniques for security protocol verification include control-flow analysis [59–61], Feret's abstract-interpretation-based relational analysis [118], Heather and Schneider's rank functions verifier [133], Backes et al.'s causal graph technique [19], and the Hermès protocol verifier [65]. While most verifiers compute the knowledge of the adversary, Hermès computes forms of messages, such as encryption under certain keys, that guarantee preservation of secrecy.

Platforms that group several verification techniques have also been implemented:

- CAPSL (Common Authentication Protocol Specification Language) [107] provides a protocol description language, which is translated into an intermediate language, CIL (CAPSL Intermediate Language), based on multiset rewriting (or equivalently on Horn clauses with existentials in linear logic) [72]. This intermediate language can be translated into the input languages of Maude, NPA, Athena, and of the constraint solving verifier of [154].
- AVISPA (Automated Validation of Internet Security Protocols and Applications) [17] provides, like CAPSL, a protocol description language HLPSL (High-Level Protocol Specification Language), which is translated into an intermediate language based on multiset rewriting. Four verifiers take as input this intermediate language: SATMC for a bounded state space, CL-AtSe and OFMC for a bounded number of sessions, TA4SP for an unbounded number of sessions.

Even if it is rather long, this survey of protocol verification techniques in the symbolic model is certainly not exhaustive. It still shows the wide variety of techniques that have been applied to protocol verification, and the interest generated by this problem in the formal method community.

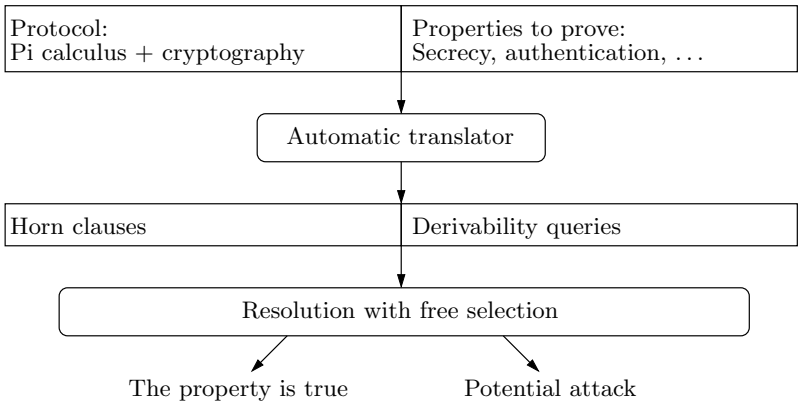

Fig. 1. The verification method of ProVerif

2.2 ProVerif

As mentioned above, the protocol verifier ProVerif is based on an abstract representation of the protocol by a set of Horn clauses, and on a resolution algorithm on these clauses. This tool has the following features:

- It is fully automatic. The user gives only the specification of the protocol and the properties to verify.
- It can handle a wide variety of cryptographic primitives, defined by rewrite rules or by certain equations.
- In contrast to finite state techniques, it can verify protocols without arbitrarily bounding the number of executed sessions (even in parallel) of the protocol or the size of messages. This makes it possible to obtain actual proofs of the security properties.
- It can verify secrecy, correspondence, and some equivalence properties.

Of course, there is a price to pay for these advantages: ProVerif does not always terminate and it is not complete (it may find false attacks). It is still precise and efficient in practice, as demonstrated by case studies, such as [3, 4, 52, 55].

The verification method is summarized in Fig. 1. The Horn clause verification technique is not specific to any formalism for representing the protocol. Among the many existing formalisms, we focused on extensions of the pi calculus with cryptographic primitives. The pi calculus itself [157] is a minimal programming language that models systems communicating on channels. Its cryptographic extensions are particularly well-suited for specifying cryptographic protocols. This line of research was pioneered by the spi calculus [7], which adds encryption, signatures, and hash functions to the pi calculus. It was considerably extended by the applied pi calculus [5], which provides a generic treatment of cryptographic primitives, defined by an equational theory. In our work, we first focused on a simpler case in which cryptographic primitives are defined by rewrite rules. This case can still represent many cryptographic primitives. We distinguish two kinds

of primitives: constructors and destructors. Constructors, such as encryption enc, build new terms, while destructors, such as decryption dec, compute on terms. Destructors are defined by rewrite rules. For example, shared-key decryption can be defined by the rewrite rule: $\mathsf{dec}(\mathsf{enc}(x, y), y) \rightarrow x$. Decrypting a ciphertext $\mathsf{enc}(x, y)$ with the encryption key y yields the cleartext x.

We then extended the tool to support some primitives defined by equations, by translating these equations into rewrite rules automatically [54]. Hence resolution, on which ProVerif relies, can still use ordinary syntactic unification (instead of unification modulo the equational theory), and thus remains efficient. In particular, this technique supports block ciphers, for which decryption never fails (it may return junk), and a simple model of Diffie-Hellman key agreements. It still has limitations; in particular, it cannot handle associativity, so it does not support XOR (exclusive or). Extensions have been proposed for supporting XOR [139] and for improving the treatment of Diffie-Hellman key agreements [140]. Support for associative-commutative symbols can be offered by unification modulo the equational theory, as in Maude-NPA [116].

The protocol represented in this calculus is automatically translated into a set of Horn clauses (a logic program). This translation is defined in [2]. The main idea of the Horn clause representation is to use a predicate $\mathsf{attacker}$, such that $\mathsf{attacker}(M)$ means "the attacker may have the message M". For example, the fact that the attacker can encrypt, resp. decrypt, when it has the key is represented by the following two clauses:

$$\mathsf{attacker}(x) \wedge \mathsf{attacker}(y) \Rightarrow \mathsf{attacker}(\mathsf{enc}(x, y))$$
$$\mathsf{attacker}(\mathsf{enc}(x, y)) \wedge \mathsf{attacker}(y) \Rightarrow \mathsf{attacker}(x)$$

When the attacker has the cleartext x and the key y, it can built the ciphertext $\mathsf{enc}(x, y)$, and when the attacker has the ciphertext and the key, it can obtain the cleartext. The messages exchanged by the honest participants of the protocol can also be represented by similar clauses. The participants are considered as oracles that the attacker can call to increase its knowledge. When a participant A sends a message M after receiving messages $M_1, \ldots, M_n$, we have a clause:

$$\mathsf{attacker}(M_1) \wedge \ldots \wedge \mathsf{attacker}(M_n) \Rightarrow \mathsf{attacker}(M)$$

Indeed, when the attacker has $M_1, \ldots, M_n$, it can send them to A; A replies with M, which the attacker can intercept. For instance, in the original protocol of Sect. 1.1, B receives a message of the form $\{\{y\}_{sk_A}\}_{pk_B}$, modeled by the term $\mathsf{penc}(\mathsf{sign}(y, sk_A), \mathsf{pk}(sk_B))$, where penc represents the public-key encryption, sign the signature, and pk computes the public key from the corresponding secret key. Then, B replies with the secret s encrypted under the key y, $\{s\}_y$, modeled by the term $\mathsf{enc}(s, y)$. Hence, we obtain the clause:

$$\mathsf{attacker}(\mathsf{penc}(\mathsf{sign}(y, sk_A), \mathsf{pk}(sk_B))) \Rightarrow \mathsf{attacker}(\mathsf{enc}(s, y))$$

More details on this representation as well as the complete coding of the protocol of Sect. 1.1 can be found in [53].

This representation of protocols is approximate in that the application of Horn clauses can be repeated any number of times, while the real protocol repeats each step only once per session. So, the state of the participants is only partly modeled. A model that does not make such an approximation can be obtained by using clauses in linear logic instead of classical logic, to control the number of repetitions of each step [113]. The Horn clause model can be seen as a sound abstraction, in the abstract interpretation sense [98], of the linear logic model, obtained by ignoring the number of repetitions of each action [49]. Hence, our technique is sound (when it says that a security property is true, then it is actually so), but not complete (false attacks can be found). However, in our tests, false attacks rarely occur. In fact, false attacks occur typically for protocols that first need to keep data secret, then publish them later in the protocol. In that situation, the Horn clause model considers that the attacker can re-inject the secret in the early part of the run, which is not possible in reality (V. Cortier, personal communication). Ignoring the number of repetitions of each action is a key to verify protocols without bounding the number of sessions.

Using this representation, secrecy can be inferred from non-derivability: if $\mathsf{attacker}(M)$ is not derivable, then the attacker cannot have M, that is, M is secret. Even if derivability is undecidable in general, several techniques can be used to determine whether a fact is derivable from a set of clauses. However, the simplest techniques, such as SLD-resolution used in Prolog, would never terminate. (For example, the clause for decryption given above immediately leads to a loop.) More elaborate resolution techniques succeed in this task:

- Ordered resolution with selection has been used in [178] and is implemented in the theorem prover SPASS (`http://www.spass-prover.org/`).
- Ordered resolution with factorization and splitting terminates on protocols that blindly copy at most one message at each step [83]. (This class of protocols results in clauses with at most one variable.)
- ProVerif uses resolution with free selection (without ordering) [18]. This strategy terminates on *tagged protocols* [57]: in these protocols, each application of a cryptographic primitive is distinguished from others by a constant (the tag). For example, we use $\mathsf{enc}((c_0, m), k)$ for encrypting m under k, instead of $\mathsf{enc}(m, k)$. It is easy to add tags, and it is also a good design practice: it can make protocols more secure, in particular by avoiding type flaw attacks [132]. When we verify a tagged protocol, the implemented protocol should of course also be tagged, since the security proof for the tagged protocol does not imply the security of a non-tagged version. A key to obtain termination is to avoid resolving on facts of the form $\mathsf{attacker}(x)$. Indeed, these facts resolve with all facts of the form $\mathsf{attacker}(M)$, which leads to non-termination in almost all examples coming from protocols.

These three techniques terminate on numerous practical examples, even outside the decision classes mentioned above.

In case $\mathsf{attacker}(M)$ is derivable from the representation of the protocol, ProVerif cannot prove secrecy. In this case, ProVerif uses the derivation of

$\mathsf{attacker}(M)$ to reconstruct an attack automatically [13]. (Such a reconstruction fails if a false attack has been found.)

We have extended this technique to more complex security properties:

- ProVerif can verify complex non-injective and injective correspondence properties [52], which can in particular model authentication.
- It can also verify a limited class of process equivalences: it verifies a strong equivalence between processes that have the same structure, but differ only by the terms they contain [54] (named diff-equivalence in [36]). This equivalence is useful, for instance to prove strong secrecy [48] and to detect guessing attacks against password-based protocols. ProVerif is so far the only tool that can prove process equivalences for an unbounded number of sessions.

Using our tool, we verified numerous protocols from the literature, finding known attacks or proving the correctness of the protocols. Most examples were verified in less than 0.1 s [52]. We also used ProVerif for verifying a certified email protocol [3], the protocol JFK (a proposed replacement for the key exchange protocol of IPsec) [4], and the cryptographic filesystem Plutus [55]. ProVerif was also used by other authors, for instance for verifying Web services, by translating XML protocols to ProVerif using the tool TulaFale [47, 149], e-voting protocols [21, 104, 138], zero-knowledge protocols [23], RFID protocols [68], and the TPM (Trusted Platform Module) [74, 105]. An extension was proposed for supporting protocols with mutable global state [15]. ProVerif can be downloaded at `http://www.proverif.ens.fr/`.

3 Verifying Protocols in the Computational Model

Proving protocols automatically in the computational model is much more difficult than in the symbolic model. Still, much research tackled this task. This section presents these approaches.

3.1 Computational Soundness

An attack in the symbolic model directly leads to an attack in the computational model. However, the converse is not true in general: a protocol may be proved secure in symbolic model and still be subject to attacks in the computational model. Following the seminal work by Abadi and Rogaway [8], many computational soundness results have been proved. These results show that, modulo additional assumptions, if a protocol is secure in the symbolic model, then it is also secure in the computational model. They provide a way of obtaining automatic proofs of protocols in the computational model, by first proving them in the symbolic model, then applying a computational soundness theorem. Some work following this line follows.

- Abadi and Rogaway [8] showed that, if two messages are indistinguishable in the symbolic sense, then they are also indistinguishable in the computational sense, if the only primitive is shared-key encryption, assuming a few additional technical restrictions.

- This initial result was followed by considerable extensions. In particular, Micciancio and Warinschi [151] showed that states and traces in the computational model match (up to negligible probability) states and traces in the symbolic model, for public-key encryption in the presence of an active adversary. Therefore, authentication in the symbolic model implies authentication in the computational model. This result was further extended to signatures [92, 135], hash functions [89, 136], non-malleable commitment [121], and zero-knowledge proofs [29]. Cortier and Warinschi [92] also showed that syntactic secrecy in the symbolic model implies secrecy in the computational model for nonces. A tool [88] was built based on [92] to obtain computational proofs using the symbolic verifier AVISPA, for protocols that use public-key encryption and signatures.
  While the previous results dealt with traces, Comon and Cortier showed a computational soundness result for observational equivalence, for protocols that use authenticated shared-key encryption [85].
  These results consider a fixed protocol language and a few primitives at a time, limiting the scope of the results. Frameworks were designed to make computational soundness proofs modular, by encoding many input languages into one [20, 24] and by allowing to compose proofs obtained independently for several primitives [93].
- Backes, Pfitzmann, and Waidner [25–27] developed an abstract cryptographic library including authenticated shared-encryption, public-key encryption, message authentication codes, signatures, and nonces, and have shown its soundness with respect to computational primitives, under arbitrary active attacks. This work relates the computational model to a non-standard version of the Dolev-Yao model, in which the length of messages is present. It has been used for a proof of the Needham-Schroeder protocol fixed by Lowe [147] verified in a proof assistant [175].
- Canetti and Herzog [71] showed how a symbolic analysis in the style of the Dolev-Yao model can be used to prove security properties of protocols in the framework of universal composability [70] for a restricted class of protocols that use only public-key encryption. They then use ProVerif [48] to verify protocols in this framework.

We refer the reader to [90] for a more detailed survey of computational soundness results. This approach enjoyed important successes, but also has limitations: additional hypotheses are necessary, since the two models do not match exactly. The cryptographic primitives need to satisfy strong security properties so that they match the symbolic primitives. For instance, encryption has to hide the length of messages, or the symbolic model must be modified to take into that length. These results often assume that all keys (even those of the adversary) are generated by the correct key generation algorithm. Moreover, the protocols need to satisfy certain restrictions. Indeed, for shared-key encryption, there must be no key cycle (in which a key is encrypted directly or indirectly under itself, as in $\{k\}_k$ or $\{k\}_{k'}, \{k'\}_k$) or a specific definition of security of encryption is necessary [10, 28]. (The existence of key cycles for a bounded number of sessions

is a NP-complete problem [94].) These limitations have lead to the idea of directly automating proofs in the computational model.

3.2 Adapting Techniques from the Symbolic Model

Another way of proving protocols in the computational model is to adapt techniques previously designed for the symbolic model.

For instance, the logic PCL [100, 115], first designed for proving protocols in the Dolev-Yao model, was adapted to the computational model [101, 102]. Other computationally sound logics include CIL (Computational Indistinguishability Logic) [30] and a specialized Hoare logic designed for proving asymmetric encryption schemes in the random oracle model [95, 96].

Similarly, type systems [97, 143, 145, 172] can provide computational security guarantees. For instance, [143] handles shared-key and public-key encryption, with an unbounded number of sessions. This system relies on the Backes-Pfitzmann-Waidner library. A type inference algorithm is given in [22].

3.3 Direct Computational Proofs

Finally, the direct approach to computational proofs consists in mechanizing proofs in the computational model, without relying at all on the symbolic model. Computational proofs made by cryptographers are typically presented as sequences of games [42, 171]: the initial game represents the protocol to prove; the goal is to show that the probability of breaking a certain security property is negligible in this game. Intermediate games are obtained each from the previous one by transformations such that the difference of probability between consecutive games is negligible. The final game is such that the desired probability is obviously negligible from the form of the game. The desired probability is then negligible in the initial game. Halevi [131] suggested to use tools for mechanizing these proofs, and several techniques have been used for reaching this goal.

CryptoVerif [50, 51, 56, 58], which we have designed, is the first such tool. It generates proofs by sequences of games automatically or with little user interaction. The games are formalized in a probabilistic process calculus. CryptoVerif provides a generic method for specifying security properties of many cryptographic primitives. It proves secrecy and authentication properties. It also provides a bound on the probability of success of an attack. It considerably extends early work by Laud [141, 142] which was limited either to passive adversaries or to a single session of the protocol. More recently, Tšahhirov and Laud [144, 177] developed a tool similar to CryptoVerif but that represents games by dependency graphs; it handles public-key and shared-key encryption and proves secrecy properties.

The CertiCrypt framework [31, 32, 34, 37, 38] enables the machine-checked construction and verification of cryptographic proofs by sequences of games. It relies on the general-purpose proof assistant Coq, which is widely believed to be correct. EasyCrypt [33] generates CertiCrypt proofs from proof sketches that formally represent the sequence of games and hints, which makes the tool easier

to use. Nowak et al. [11, 161, 162] follow a similar idea by providing Coq proofs for several basic cryptographic primitives.

4 Verifying Protocol Implementations

The approaches mentioned so far verify specifications of protocols in models such as the applied pi calculus or its variants. However, errors may be introduced when the protocol is implemented. It is therefore important to prove security properties on the implementation of the protocol itself. Two approaches to reach this goal can be distinguished.

A simple approach consists in translating the model into an implementation by a suitable compiler which has been proved correct. This approach was used in tools such as [152, 165, 167, 173]. A limitation of this approach is that the protocol modeling language offers less flexibility in the implementation of the protocol than a standard programming language.

A more flexible, but more difficult, approach consists in analyzing the implementation of the protocol. Results in this approach differ by the input language they consider. Analyzing C code is obviously more difficult than analyzing languages such as F# and Java, in particular due to pointers and memory safety. However, it allows one to verify practical implementations, which are generally written in C. We can also distinguish two ways of analyzing implementations:

- One can extract a protocol specification from the implementation, and verify it using existing protocol verification tools. For instance, the tools FS2PV [46] and FS2CV [120] translate protocols written in a subset of the functional language F# into the input language of ProVerif and CryptoVerif, respectively, so that protocol can be proved in the symbolic model and in the computational model. These techniques were applied to an important case study: the protocol TLS [44]. They analyze reference implementations written in F# in order to facilitate verification; one verifies that these implementations interoperate with other implementations, which provides some assurance that they match practical implementations; however, it is very difficult to analyze the code of implementations written without verification in mind.
  Similarly, Elijah [163] translates Java programs into LySa protocol specifications, which can be verified by the LySatool [59].
  Aizatulin et al. [12] use symbolic execution in order to extract ProVerif models from pre-existing protocol implementations in C. This technique currently analyzes a single execution path of the protocol, so it is limited to protocols without branching. Furthermore, computational security guarantees are obtained by applying a computational soundness result.
- One can also adapt protocol verification methods to the verification of implementations or design new methods for verifying implementations. The tool CSur [130] analyzes protocols written in C by translating them into Horn clauses, yielding a model fairly similar to the one used in ProVerif. These clauses are given as input to the $\mathcal{H}_1$ prover [129] to prove properties

of the protocol. Similarly, JavaSec [137] translates Java programs into first-order logic formulas, which are then given as input to the first-order theorem prover e-SETHEO.

The tools F7 and F* [43, 45, 176] use a dependent type system in order to prove security properties of protocols implemented in F#, therefore extending to implementations the approach of Cryptyc [125–127] for models. This approach scales well to large implementations but requires type annotations, which facilitate automatic verification. This approach is also being extended to the computational model [119]: one uses a type system to verify the conditions needed in order to apply a game transformation. Then, the game transformation is applied, and the obtained game is typed again, with a different typing judgment, to justify the next game transformation, and transformations can continue in this way until security can be proved directly by inspecting the game.

Poll and Schubert [166] verified an implementation of SSH in Java using ESC/Java2: ESC/Java2 verifies that the implementation does not raise exceptions, and follows a specification of SSH by a finite automaton, but does not prove security properties.

ASPIER [73] uses software model-checking, with predicate abstraction and counter-example guided abstraction refinement, in order to verify C implementations of protocols, assuming the size of messages and the number of sessions are bounded. In particular, this tool has been used to verify the main loop of OpenSSL 3. Dupressoir et al. [111] use the general-purpose C verifier VCC in order to prove both memory safety and security properties of protocols, in the symbolic model. They use "ghost state" in order to relate C variables and symbolic terms.

5 Conclusion and Future Challenges

This survey shows that research in the field of security protocol verification has been very active, and has enjoyed unquestionable successes. Progress has been made in all directions: verification both in the symbolic model and in the computational model, as well as verification of implementations. We believe that the verification of protocols in the symbolic model has reached a fairly mature state, even though some aspects still need further research, for instance the proof of process equivalences or the treatment of some complex equational theories. However, there is still much work to do regarding the verification of protocols in the computational model and the verification of implementations. We are still far from having a push button tool that would take as input a practical implementation of the protocol and would prove it secure in the computational model. Even if this goal may be out of reach, more progress is undoubtedly possible in this direction. Taking into account physical attacks is a challenging area in which formal methods just start to be used, and in which much research will certainly be done in the future.

Acknowledgments. We thank Pierpaolo Degano for helpful comments on a draft of this paper. This work was partly supported by the ANR project ProSe (decision ANR-2010-VERS-004-01).

References

1. Abadi, M.: Secrecy by typing in security protocols. Journal of the ACM 46(5), 749–786 (1999)
2. Abadi, M., Blanchet, B.: Analyzing security protocols with secrecy types and logic programs. Journal of the ACM 52(1), 102–146 (2005)
3. Abadi, M., Blanchet, B.: Computer-assisted verification of a protocol for certified email. Science of Computer Programming 58(1–2), 3–27 (2005)
4. Abadi, M., Blanchet, B., Fournet, C.: Just Fast Keying in the pi calculus. ACM TISSEC 10(3), 1–59 (2007)
5. Abadi, M., Fournet, C.: Mobile values, new names, and secure communication. In: POPL'01. pp. 104–115. ACM, New York (2001)
6. Abadi, M., Gordon, A.D.: A bisimulation method for cryptographic protocols. Nordic Journal of Computing 5(4), 267–303 (1998)
7. Abadi, M., Gordon, A.D.: A calculus for cryptographic protocols: The spi calculus. Information and Computation 148(1), 1–70 (1999)
8. Abadi, M., Rogaway, P.: Reconciling two views of cryptography (the computational soundness of formal encryption). Journal of Cryptology 15(2), 103–127 (2002)
9. Abdalla, M., Fouque, P.A., Pointcheval, D.: Password-based authenticated key exchange in the three-party setting. IEE Proceedings Information Security 153(1), 27–39 (2006)
10. Adão, P., Bana, G., Herzog, J., Scedrov, A.: Soundness of formal encryption in the presence of key-cycles. In: de Capitani di Vimercati, S., Syverson, P., Gollmann, D. (eds.) ESORICS'05. LNCS, vol. 3679, pp. 374–396. Springer, Heidelberg (2005)
11. Affeldt, R., Nowak, D., Yamada, K.: Certifying assembly with formal cryptographic proofs: the case of BBS. In: AVoCS'09. Electronic Communications of the EASST, vol. 23. EASST (2009)
12. Aizatulin, M., Gordon, A.D., Jürjens, J.: Extracting and verifying cryptographic models from C protocol code by symbolic execution. In: CCS'11. pp. 331–340. ACM, New York (2011)
13. Allamigeon, X., Blanchet, B.: Reconstruction of attacks against cryptographic protocols. In: CSFW'05. pp. 140–154. IEEE, Los Alamitos (2005)
14. Arapinis, M., Duflot, M.: Bounding messages for free in security protocols. In: Arvind, V., Prasad, S. (eds.) FSTTCS'07. LNCS, vol. 4855, pp. 376–387. Springer, Heidelberg (2007)
15. Arapinis, M., Ritter, E., Ryan, M.D.: StatVerif: Verification of stateful processes. In: CSF'11. pp. 33–47. IEEE, Los Alamitos (2011)
16. Armando, A., Compagna, L., Ganty, P.: SAT-based model-checking of security protocols using planning graph analysis. In: Araki, K., Gnesi, S., Mandrioli, D. (eds.) FME'03. LNCS, vol. 2805, pp. 875–893. Springer, Heidelberg (2003)
17. Armando, A., et al.: The AVISPA tool for automated validation of Internet security protocols and applications. In: Etessami, K., Rajamani, S.K. (eds.) CAV'05. LNCS, vol. 3576, pp. 281–285. Springer, Heidelberg (2005)

18. Bachmair, L., Ganzinger, H.: Resolution theorem proving. In: Handbook of Automated Reasoning, vol. 1, chap. 2, pp. 19–100. North Holland (2001)
19. Backes, M., Cortesi, A., Maffei, M.: Causality-based abstraction of multiplicity in security protocols. In: CSF'07. pp. 355–369. IEEE, Los Alamitos (2007)
20. Backes, M., Hofheinz, D., Unruh, D.: CoSP: A general framework for computational soundness proofs. In: CCS'09. pp. 66–78. ACM, New York (2009)
21. Backes, M., Hritcu, C., Maffei, M.: Automated verification of remote electronic voting protocols in the applied pi-calculus. In: CSF'08. pp. 195–209. IEEE, Los Alamitos (2008)
22. Backes, M., Laud, P.: Computationally sound secrecy proofs by mechanized flow analysis. In: CCS'06. pp. 370–379. ACM, New York (2006)
23. Backes, M., Maffei, M., Unruh, D.: Zero-knowledge in the applied pi-calculus and automated verification of the direct anonymous attestation protocol. In: IEEE Symposium on Security and Privacy. pp. 202–215. IEEE, Los Alamitos (2008)
24. Backes, M., Maffei, M., Unruh, D.: Computationally sound verification of source code. In: CCS'10. pp. 387–398. ACM, New York (2010)
25. Backes, M., Pfitzmann, B.: Symmetric encryption in a simulatable Dolev-Yao style cryptographic library. In: CSFW'04. pp. 204–218. IEEE, Los Alamitos (2004)
26. Backes, M., Pfitzmann, B.: Relating symbolic and cryptographic secrecy. IEEE Transactions on Dependable and Secure Computing 2(2), 109–123 (2005)
27. Backes, M., Pfitzmann, B., Waidner, M.: A composable cryptographic library with nested operations. In: CCS'03. pp. 220–230. ACM, New York (2003)
28. Backes, M., Pfiztmann, B., Scedrov, A.: Key-dependent message security under active attacks—BRSIM/UC soundness of symbolic encryption with key cycles. In: CSF'07. pp. 112–124. IEEE, Los Alamitos (2007)
29. Backes, M., Unruh, D.: Computational soundness of symbolic zero-knowledge proofs against active attackers. In: CSF'08. pp. 255–269. IEEE, Los Alamitos (2008)
30. Barthe, G., Daubignard, M., Kapron, B., Lakhnech, Y.: Computational indistinguishability logic. In: CCS'10. pp. 375–386. ACM, New York (2010)
31. Barthe, G., Grégoire, B., Béguelin, S.Z., Lakhnech, Y.: Beyond provable security. Verifiable IND-CCA security of OAEP. In: Kiayias, A. (ed.) CT-RSA'11. LNCS, vol. 6558, pp. 180–196. Springer, Heidelberg (2011)
32. Barthe, G., Grégoire, B., Heraud, S., Béguelin, S.Z.: Formal certification of ElGamal encryption. A gentle introduction to CertiCrypt. In: Degano, P., Guttman, J., Martinelli, F. (eds.) FAST'08. LNCS, vol. 5491, pp. 1–19. Springer, Heidelberg (2009)
33. Barthe, G., Grégoire, B., Heraud, S., Béguelin, S.Z.: Computer-aided security proofs for the working cryptographer. In: Rogaway, P. (ed.) CRYPTO'11. LNCS, vol. 6841, pp. 71–90. Springer, Heidelberg (2011)
34. Barthe, G., Grégoire, B., Zanella, S.: Formal certification of code-based cryptographic proofs. In: POPL'09. pp. 90–101. ACM, New York (2009)
35. Basin, D., Mödersheim, S., Viganò, L.: An on-the-fly model-checker for security protocol analysis. In: Snekkenes, E., Gollman, D. (eds.) ESORICS'03. LNCS, vol. 2808, pp. 253–270. Springer, Heidelberg (2003)
36. Baudet, M.: Sécurité des protocoles cryptographiques: aspects logiques et calculatoires. Ph.D. thesis, Ecole Normale Supérieure de Cachan (2007)
37. Béguelin, S.Z., Barthe, G., Heraud, S., Grégoire, B., Hedin, D.: A machine-checked formalization of sigma-protocols. In: CSF'10. pp. 246–260. IEEE, Los Alamitos (2010)

38. Béguelin, S.Z., Grégoire, B., Barthe, G., Olmedo, F.: Formally certifying the security of digital signature schemes. In: IEEE Symposium on Security and Privacy. pp. 237–250. IEEE, Los Alamitos (2009)
39. Bellare, M., Desai, A., Jokipii, E., Rogaway, P.: A concrete security treatment of symmetric encryption. In: FOCS'97. pp. 394–403. IEEE, Los Alamitos (1997)
40. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated key exchange secure against dictionary attacks. In: Preneel, B. (ed.) EUROCRYPT'00. LNCS, vol. 1807, pp. 139–155. Springer, Heidelberg (2000)
41. Bellare, M., Rogaway, P.: Entity authentication and key distribution. In: Stinson, D.R. (ed.) CRYPTO'93. LNCS, vol. 773, pp. 232–249. Springer, Heidelberg (1993)
42. Bellare, M., Rogaway, P.: The security of triple encryption and a framework for code-based game-playing proofs. In: Vaudenay, S. (ed.) EUROCRYPT'06. LNCS, vol. 4004, pp. 409–426. Springer, Heidelberg (2006)
43. Bengtson, J., Bhargavan, K., Fournet, C., Gordon, A., Maffeis, S.: Refinement types for secure implementations. ACM TOPLAS 33(2) (2011)
44. Bhargavan, K., Corin, R., Fournet, C., Zălinescu, E.: Cryptographically verified implementations for TLS. In: CCS'08. pp. 459–468. ACM, New York (2008)
45. Bhargavan, K., Fournet, C., Gordon, A.: Modular verification of security protocol code by typing. In: POPL'10. pp. 445–456. ACM, New York (2010)
46. Bhargavan, K., Fournet, C., Gordon, A., Tse, S.: Verified interoperable implementations of security protocols. ACM TOPLAS 31(1) (2008)
47. Bhargavan, K., Fournet, C., Gordon, A.D., Pucella, R.: TulaFale: A security tool for web services. In: de Boer, F.S., Bonsangue, M.M., Graf, S., de Roever, W.P. (eds.) FMCO'03. LNCS, vol. 3188, pp. 197–222. Springer, Heidelberg (2003)
48. Blanchet, B.: Automatic proof of strong secrecy for security protocols. In: IEEE Symposium on Security and Privacy. pp. 86–100. IEEE, Los Alamitos (2004)
49. Blanchet, B.: Security protocols: From linear to classical logic by abstract interpretation. Information Processing Letters 95(5), 473–479 (2005)
50. Blanchet, B.: Computationally sound mechanized proofs of correspondence assertions. In: CSF'07. pp. 97–111. IEEE, Los Alamitos (2007)
51. Blanchet, B.: A computationally sound mechanized prover for security protocols. IEEE Transactions on Dependable and Secure Computing 5(4), 193–207 (2008)
52. Blanchet, B.: Automatic verification of correspondences for security protocols. Journal of Computer Security 17(4), 363–434 (2009)
53. Blanchet, B.: Using Horn clauses for analyzing security protocols. In: Cortier, V., Kremer, S. (eds.) Formal Models and Techniques for Analyzing Security Protocols, Cryptology and Information Security Series, vol. 5, pp. 86–111. IOS Press, Amsterdam (2011)
54. Blanchet, B., Abadi, M., Fournet, C.: Automated verification of selected equivalences for security protocols. Journal of Logic and Algebraic Programming 75(1), 3–51 (2008)
55. Blanchet, B., Chaudhuri, A.: Automated formal analysis of a protocol for secure file sharing on untrusted storage. In: IEEE Symposium on Security and Privacy. pp. 417–431. IEEE, Los Alamitos (2008)
56. Blanchet, B., Jaggard, A.D., Scedrov, A., Tsay, J.K.: Computationally sound mechanized proofs for basic and public-key Kerberos. In: ASIACCS'08. pp. 87–99. ACM, New York (2008)
57. Blanchet, B., Podelski, A.: Verification of cryptographic protocols: Tagging enforces termination. Theoretical Computer Science 333(1-2), 67–90 (2005)

58. Blanchet, B., Pointcheval, D.: Automated security proofs with sequences of games. In: Dwork, C. (ed.) CRYPTO'06. LNCS, vol. 4117, pp. 537–554. Springer, Heidelberg (2006)
59. Bodei, C., Buchholtz, M., Degano, P., Nielson, F., Nielson, H.R.: Automatic validation of protocol narration. In: CSFW'03. pp. 126–140. IEEE, Los Alamitos (2003)
60. Bodei, C., Buchholtz, M., Degano, P., Nielson, F., Nielson, H.R.: Static validation of security protocols. Journal of Computer Security 13(3), 347–390 (2005)
61. Bodei, C., Degano, P., Nielson, F., Nielson, H.R.: Flow logic for Dolev-Yao secrecy in cryptographic processes. Future Generation Comp. Syst. 18(6), 747–756 (2002)
62. Boichut, Y., Kosmatov, N., Vigneron, L.: Validation of Prouvé protocols using the automatic tool TA4SP. In: TFIT'06. pp. 467–480 (2006)
63. Bolignano, D.: Towards a mechanization of cryptographic protocol verification. In: Grumberg, O. (ed.) CAV'97. LNCS, vol. 1254, pp. 131–142. Springer, Heidelberg (1997)
64. Borgström, J., Briais, S., Nestmann, U.: Symbolic bisimulation in the spi calculus. In: Gardner, P., Yoshida, N. (eds.) CONCUR'04. LNCS, vol. 3170, pp. 161–176. Springer, Heidelberg (2004)
65. Bozga, L., Lakhnech, Y., Périn, M.: Pattern-based abstraction for verifying secrecy in protocols. International Journal on Software Tools for Technology Transfer (STTT) 8(1), 57–76 (2006)
66. Broadfoot, P.J., Roscoe, A.W.: Embedding agents within the intruder to detect parallel attacks. Journal of Computer Security 12(3/4), 379–408 (2004)
67. Broadfoot, P., Lowe, G., Roscoe, B.: Automating data independence. In: ESORICS'00. LNCS, vol. 1895, pp. 175–190. Springer, Heidelberg (2000)
68. Brusó, M., Chatzikokolakis, K., den Hartog, J.: Formal verification of privacy for RFID systems. In: CSF'10. pp. 75–88. IEEE, Los Alamitos (2010)
69. Burrows, M., Abadi, M., Needham, R.: A logic of authentication. Proceedings of the Royal Society of London A 426(1871), 233–271 (1989)
70. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: FOCS'01. pp. 136–145. IEEE, Los Alamitos (2001)
71. Canetti, R., Herzog, J.: Universally composable symbolic analysis of mutual authentication and key exchange protocols. In: Halevi, S., Rabin, T. (eds.) TCC'06. LNCS, vol. 3876, pp. 380–403. Springer, Heidelberg (2006)
72. Cervesato, I., Durgin, N., Lincoln, P., Mitchell, J., Scedrov, A.: A meta-notation for protocol analysis. In: CSFW'99. pp. 55–69. IEEE, Los Alamitos (1999)
73. Chaki, S., Datta, A.: ASPIER: An automated framework for verifying security protocol implementations. In: CSF'09. pp. 172–185. IEEE, Los Alamitos (2009)
74. Chen, L., Ryan, M.: Attack, solution and verification for shared authorisation data in TCG TPM. In: Degano, P., Guttman, J.D. (eds.) FAST'09. LNCS, vol. 5983, pp. 201–216. Springer, Heidelberg (2010)
75. Cheval, V., Comon-Lundh, H., Delaune, S.: Automating security analysis: symbolic equivalence of constraint systems. In: Giesl, J., Haehnle, R. (eds.) IJCAR'10. LNAI, vol. 6173, pp. 412–426. Springer, Heidelberg (2010)
76. Cheval, V., Comon-Lundh, H., Delaune, S.: Trace equivalence decision: Negative tests and non-determinism. In: CCS'11. pp. 321–330. ACM, New York (2011)
77. Chevalier, Y., Küsters, R., Rusinowitch, M., Turuani, M.: Deciding the security of protocols with Diffie-Hellman exponentiation and products in exponents. In: Pandya, P.K., Radhakrishnan, J. (eds.) FSTTCS'03. LNCS, vol. 2914, pp. 124–135. Springer, Heidelberg (2003)

78. Chevalier, Y., Küsters, R., Rusinowitch, M., Turuani, M.: An NP decision procedure for protocol insecurity with XOR. In: LICS'03. pp. 261–270. IEEE, Los Alamitos (2003)
79. Chevalier, Y., Küsters, R., Rusinowitch, M., Turuani, M.: An NP decision procedure for protocol insecurity with XOR. Theoretical Computer Science 338(1–3), 247–274 (2005)
80. Chevalier, Y., Vigneron, L.: A tool for lazy verification of security protocols. In: ASE'01. pp. 373–376. IEEE, Los Alamitos (2001)
81. Ştefan Ciobâcă: Automated Verification of Security Protocols with Appplications to Electronic Voting. Ph.D. thesis, ENS Cachan (2011)
82. Comon, H., Cortier, V.: Tree automata with one memory, set constraints and cryptographic protocols. Theoretical Computer Science 331(1), 143–214 (2005)
83. Comon-Lundh, H., Cortier, V.: New decidability results for fragments of first-order logic and application to cryptographic protocols. In: Nieuwenhuis, R. (ed.) RTA'03. LNCS, vol. 2706, pp. 148–164. Springer, Heidelberg (2003)
84. Comon-Lundh, H., Cortier, V.: Security properties: two agents are sufficient. Science of Computer Programming 50(1–3), 51–71 (2004)
85. Comon-Lundh, H., Cortier, V.: Computational soundness of observational equivalence. In: CCS'08. pp. 109–118. ACM, New York (2008)
86. Comon-Lundh, H., Shmatikov, V.: Intruder deductions, constraint solving and insecurity decision in presence of exclusive or. In: LICS'03. pp. 271–280. IEEE, Los Alamitos (2003)
87. Cortier, V., Delaune, S.: A method for proving observational equivalence. In: CSF'09. pp. 266–276. IEEE, Los Alamitos (2009)
88. Cortier, V., Hördegen, H., Warinschi, B.: Explicit randomness is not necessary when modeling probabilistic encryption. In: Dima, C., Minea, M., Tiplea, F. (eds.) ICS'06. ENTCS, vol. 186, pp. 49–65. Elsevier, Amsterdam (2006)
89. Cortier, V., Kremer, S., Küsters, R., Warinschi, B.: Computationally sound symbolic secrecy in the presence of hash functions. In: Garg, N., Arun-Kumar, S. (eds.) FSTTCS'06. LNCS, vol. 4246, pp. 176–187. Springer, Heidelberg (2006)
90. Cortier, V., Kremer, S., Warinschi, B.: A survey of symbolic methods in computational analysis of cryptographic systems. Journal of Automated Reasoning 46(3-4), 225–259 (2011)
91. Cortier, V., Rusinowitch, M., Zălinescu, E.: Relating two standard notions of secrecy. Logical Methods in Computer Science 3(3) (2007)
92. Cortier, V., Warinschi, B.: Computationally sound, automated proofs for security protocols. In: Sagiv, M. (ed.) ESOP'05. LNCS, vol. 3444, pp. 157–171. Springer, Heidelberg (2005)
93. Cortier, V., Warinschi, B.: A composable computational soundness notion. In: CCS'11. pp. 63–74. ACM, New York (2011)
94. Cortier, V., Zălinescu, E.: Deciding key cycles for security protocols. In: Hermann, M., Voronkov, A. (eds.) LPAR'06. LNCS, vol. 4246, pp. 317–331. Springer, Heidelberg (2006)
95. Courant, J., Daubignard, M., Ene, C., Lafourcade, P., Lakhnech, Y.: Towards automated proofs for asymmetric encryption schemes in the random oracle model. In: CCS'08. pp. 371–380. ACM, New York (2008)
96. Courant, J., Daubignard, M., Ene, C., Lafourcade, P., Lakhnech, Y.: Automated proofs for asymmetric encryption. In: Dams, D., Hannemann, U., Steffen, M. (eds.) Concurrency, Compositionality, and Correctness. LNCS, vol. 5930, pp. 300–321. Springer, Heidelberg (2010)

97. Courant, J., Ene, C., Lakhnech, Y.: Computationally sound typing for noninterference: The case of deterministic encryption. In: Arvind, V., Prasad, S. (eds.) FSTTCS'07. LNCS, vol. 4855, pp. 364–375. Springer, Heidelberg (2007)
98. Cousot, P., Cousot, R.: Systematic design of program analysis frameworks. In: POPL'79. pp. 269–282. ACM, New York (1979)
99. Cremers, C.J.F.: Scyther - Semantics and Verification of Security Protocols. Ph.D. thesis, Eindhoven University of Technology (2006)
100. Datta, A., Derek, A., Mitchell, J.C., Pavlovic, D.: A derivation system and compositional logic for security protocols. Journal of Computer Security 13(3), 423–482 (2005)
101. Datta, A., Derek, A., Mitchell, J.C., Shmatikov, V., Turuani, M.: Probabilistic polynomial-time semantics for a protocol security logic. In: Caires, L., Monteiro, L. (eds.) ICALP'05. LNCS, vol. 3580, pp. 16–29. Springer, Heidelberg (2005)
102. Datta, A., Derek, A., Mitchell, J.C., Warinschi, B.: Computationally sound compositional logic for key exchange protocols. In: CSFW'06. pp. 321–334. IEEE, Los Alamitos (2006)
103. Delaune, S., Kremer, S., Ryan, M.D.: Symbolic bisimulation for the applied picalculus. In: Arvind, V., Prasad, S. (eds.) FSTTCS'07. LNCS, vol. 4855, pp. 133–145. Springer, Heidelberg (2007)
104. Delaune, S., Kremer, S., Ryan, M.D.: Verifying privacy-type properties of electronic voting protocols. Journal of Computer Security 17(4), 435–487 (2009)
105. Delaune, S., Kremer, S., Ryan, M.D., Steel, G.: Formal analysis of protocols based on TPM state registers. In: CSF'11. pp. 66–82. IEEE, Los Alamitos (2011)
106. Denker, G., Meseguer, J., Talcott, C.: Protocol specification and analysis in Maude. In: FMSP'98 (1998)
107. Denker, G., Millen, J.: CAPSL integrated protocol environment. In: DISCEX'00. pp. 207–221. IEEE, Los Alamitos (2000)
108. Denning, D.E., Sacco, G.M.: Timestamps in key distribution protocols. Commun. ACM 24(8), 533–536 (1981)
109. Dierks, T., Rescorla, E.: RFC 4346: The Transport Layer Security (TLS) protocol, version 1.1 (2006), `http://tools.ietf.org/html/rfc4346`
110. Dolev, D., Yao, A.C.: On the security of public key protocols. IEEE Transactions on Information Theory IT-29(12), 198–208 (1983)
111. Dupressoir, F., Gordon, A.D., Jürjens, J., Naumann, D.A.: Guiding a general-purpose C verifier to prove cryptographic protocols. In: CSF'11. pp. 3–17. IEEE, Los Alamitos (2011)
112. Durante, L., Sisto, R., Valenzano, A.: Automatic testing equivalence verification of spi calculus specifications. ACM TOSEM 12(2), 222–284 (2003)
113. Durgin, N.A., Lincoln, P.D., Mitchell, J.C., Scedrov, A.: Undecidability of bounded security protocols. In: FMSP'99 (1999)
114. Durgin, N., Lincoln, P., Mitchell, J.C., Scedrov, A.: Multiset rewriting and the complexity of bounded security protocols. Journal of Computer Security 12(2), 247–311 (2004)
115. Durgin, N., Mitchell, J.C., Pavlovic, D.: A compositional logic for proving security properties of protocols. Journal of Computer Security 11(4), 677–721 (2003)
116. Escobar, S., Meadows, C., Meseguer, J.: A rewriting-based inference system for the NRL protocol analyzer and its meta-logical properties. Theoretical Computer Science 367(1-2), 162–202 (2006)
117. Fábrega, F.J.T., Herzog, J.C., Guttman, J.D.: Strand spaces: Proving security protocols correct. Journal of Computer Security 7(2/3), 191–230 (1999)

118. Feret, J.: Analysis of mobile systems by abstract interpretation. Ph.D. thesis, École Polytechnique (2005)
119. Fournet, C., Kohlweiss, M.: Modular cryptographic verification by typing. In: FCC'11 (2011)
120. http://msr-inria.inria.fr/projects/sec/fs2cv/
121. Galindo, D., Garcia, F.D., van Rossum, P.: Computational soundness of non-malleable commitments. In: Chen, L., Mu, Y., Susilo, W. (eds.) ISPEC'08. LNCS, vol. 4991, pp. 361–376. Springer, Heidelberg (2008)
122. Genet, T., Klay, F.: Rewriting for cryptographic protocol verification. In: McAllester, D. (ed.) CADE'00. LNCS, vol. 1831, pp. 271–290. Springer, Heidelberg (2000)
123. Goldwasser, S., Micali, S.: Probabilistic encryption. Journal of Computer and System Sciences 28, 270–299 (1984)
124. Goldwasser, S., Micali, S., Rivest, R.: A digital signature scheme secure against adaptative chosen-message attacks. SIAM Journal of Computing 17(2), 281–308 (1988)
125. Gordon, A., Jeffrey, A.: Typing one-to-one and one-to-many correspondences in security protocols. In: Okada, M., Pierce, B., Scedriv, A., Tokuda, H., Yonezawa, A. (eds.) ISSS'02. LNCS, vol. 2609, pp. 263–282. Springer, Heidelberg (2002)
126. Gordon, A., Jeffrey, A.: Authenticity by typing for security protocols. Journal of Computer Security 11(4), 451–521 (2003)
127. Gordon, A., Jeffrey, A.: Types and effects for asymmetric cryptographic protocols. Journal of Computer Security 12(3/4), 435–484 (2004)
128. Goubault-Larrecq, J.: A method for automatic cryptographic protocol verification (extended abstract). In: Rolim, J., et al. (eds.) FMPPTA'2000. LNCS, vol. 1800, pp. 977–984. Springer, Heidelberg (2000)
129. Goubault-Larrecq, J.: Deciding $\mathcal{H}_1$ by resolution. Information Processing Letters 95(3), 401–408 (2005)
130. Goubault-Larrecq, J., Parrennes, F.: Cryptographic protocol analysis on real C code. In: Cousot, R. (ed.) VMCAI'05. LNCS, vol. 3385, pp. 363–379. Springer, Heidelberg (2005)
131. Halevi, S.: A plausible approach to computer-aided cryptographic proofs. Cryptology ePrint Archive, Report 2005/181 (2005), available at http://eprint.iacr.org/2005/181
132. Heather, J., Lowe, G., Schneider, S.: How to prevent type flaw attacks on security protocols. In: CSFW'00. pp. 255–268. IEEE, Los Alamitos (2000)
133. Heather, J., Schneider, S.: A decision procedure for the existence of a rank function. Journal of Computer Security 13(2), 317–344 (2005)
134. Hüttel, H.: Deciding framed bisimilarity. In: INFINITY'02. pp. 1–20 (2002)
135. Janvier, R., Lakhnech, Y., Mazaré, L.: Completing the picture: Soundness of formal encryption in the presence of active adversaries. In: Sagiv, M. (ed.) ESOP'05. LNCS, vol. 3444, pp. 172–185. Springer, Heidelberg (2005)
136. Janvier, R., Lakhnech, Y., Mazaré, L.: Relating the symbolic and computational models of security protocols using hashes. In: Degano, P., Küsters, R., Viganò, L., Zdancewic, S. (eds.) FCS-ARSPA'06. pp. 67–89 (2006)
137. Jürjens, J.: Security analysis of crypto-based Java programs using automated theorem provers. In: ASE'06. pp. 167–176. IEEE, Los Alamitos (2006)
138. Kremer, S., Ryan, M.D.: Analysis of an electronic voting protocol in the applied pi calculus. In: Sagiv, M. (ed.) ESOP'05. LNCS, vol. 3444, pp. 186–200. Springer, Heidelberg (2005)

139. Küsters, R., Truderung, T.: Reducing protocol analysis with XOR to the XOR-free case in the Horn theory based approach. In: CCS'08. pp. 129–138. ACM, New York (2008)
140. Küsters, R., Truderung, T.: Using ProVerif to analyze protocols with Diffie-Hellman exponentiation. In: CSF'09. pp. 157–171. IEEE, Los Alamitos (2009)
141. Laud, P.: Handling encryption in an analysis for secure information flow. In: Degano, P. (ed.) ESOP'03. LNCS, vol. 2618, pp. 159–173. Springer, Heidelberg (2003)
142. Laud, P.: Symmetric encryption in automatic analyses for confidentiality against active adversaries. In: IEEE Symposium on Security and Privacy. pp. 71–85. IEEE, Los Alamitos (2004)
143. Laud, P.: Secrecy types for a simulatable cryptographic library. In: CCS'05. pp. 26–35. ACM, New York (2005)
144. Laud, P., Tšahhirov, I.: A user interface for a game-based protocol verification tool. In: Degano, P., Guttman, J. (eds.) FAST'09. LNCS, vol. 5983, pp. 263–278. Springer, Heidelberg (2009)
145. Laud, P., Vene, V.: A type system for computationally secure information flow. In: Liśkiewicz, M., Reischuk, R. (eds.) FCT'05. LNCS, vol. 3623, pp. 365–377. Springer, Heidelberg (2005)
146. Liu, J., Lin, H.: A complete symbolic bisimulation for full applied pi calculus. In: van Leeuwen, J., Muscholl, A., Peleg, D., Pokorný, J., Rumpe, B. (eds.) SOFSEM'10. LNCS, vol. 5901, pp. 552–563. Springer, Heidelberg (2010)
147. Lowe, G.: Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In: TACAS'96. LNCS, vol. 1055, pp. 147–166. Springer, Heidelberg (1996)
148. Lowe, G.: A hierarchy of authentication specifications. In: CSFW '97. pp. 31–43. IEEE, Los Alamitos (1997)
149. Lux, K.D., May, M.J., Bhattad, N.L., Gunter, C.A.: WSEmail: Secure internet messaging based on web services. In: ICWS'05. pp. 75–82. IEEE, Los Alamitos (2005)
150. Meadows, C.A.: The NRL protocol analyzer: An overview. Journal of Logic Programming 26(2), 113–131 (1996)
151. Micciancio, D., Warinschi, B.: Soundness of formal encryption in the presence of active adversaries. In: Naor, M. (ed.) TCC'04. LNCS, vol. 2951, pp. 133–151. Springer, Heidelberg (2004)
152. Milicia, G.: χ-spaces: Programming security protocols. In: NWPT'02 (2002)
153. Millen, J.: A necessarily parallel attack. In: FMSP'99 (1999)
154. Millen, J., Shmatikov, V.: Constraint solving for bounded-process cryptographic protocol analysis. In: CCS'01. pp. 166–175. ACM, New York (2001)
155. Millen, J.K.: The Interrogator model. In: IEEE Symposium on Security and Privacy. pp. 251–260. IEEE, Los Alamitos (1995)
156. Millen, J.K., Clark, S.C., Freedman, S.B.: The Interrogator: Protocol security analysis. IEEE Transactions on Software Engineering SE-13(2), 274–288 (1987)
157. Milner, R.: Communicating and mobile systems : the π-calculus. Cambridge University Press (1999)
158. Mitchell, J.C., Mitchell, M., Stern, U.: Automated analysis of cryptographic protocols using Murφ. In: IEEE Symposium on Security and Privacy. pp. 141–151. IEEE, Los Alamitos (1997)
159. Monniaux, D.: Abstracting cryptographic protocols with tree automata. Science of Computer Programming 47(2–3), 177–202 (2003)
160. Needham, R.M., Schroeder, M.D.: Using encryption for authentication in large networks of computers. Commun. ACM 21(12), 993–999 (1978)

161. Nowak, D.: A framework for game-based security proofs. In: Qing, S., Imai, H., Wang, G. (eds.) ICICS'07. LNCS, vol. 4861, pp. 319–333. Springer, Heidelberg (2007)
162. Nowak, D.: On formal verification of arithmetic-based cryptographic primitives. In: Lee, P.J., Cheon, J.H. (eds.) ICISC'08. LNCS, vol. 5461, pp. 368–382. Springer, Heidelberg (2008)
163. O'Shea, N.: Using Elyjah to analyse Java implementations of cryptographic protocols. In: FCS-ARSPA-WITS'08 (2008)
164. Paulson, L.C.: The inductive approach to verifying cryptographic protocols. Journal of Computer Security 6(1–2), 85–128 (1998)
165. Pironti, A., Sisto, R.: Provably correct Java implementations of spi calculus security protocols specifications. Computers and Security 29(3), 302–314 (2010)
166. Poll, E., Schubert, A.: Verifying an implementation of SSH. In: WITS'07 (2007)
167. Pozza, D., Sisto, R., Durante, L.: Spi2Java: Automatic cryptographic protocol Java code generation from spi calculus. In: AINA'04. vol. 1, pp. 400–405. IEEE, Los Alamitos (2004)
168. Ramanujam, R., Suresh, S.: Tagging makes secrecy decidable with unbounded nonces as well. In: Pandya, P., Radhakrishnan, J. (eds.) FSTTCS'03. LNCS, vol. 2914, pp. 363–374. Springer, Heidelberg (2003)
169. Roscoe, A.W., Broadfoot, P.J.: Proving security protocols with model checkers by data independence techniques. Journal of Computer Security 7(2, 3), 147–190 (1999)
170. Rusinowitch, M., Turuani, M.: Protocol insecurity with finite number of sessions is NP-complete. Theoretical Computer Science 299(1–3), 451–475 (2003)
171. Shoup, V.: Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332 (2004), available at `http://eprint.iacr.org/2004/332`
172. Smith, G., Alpízar, R.: Secure information flow with random assignment and encryption. In: FMSE'06. pp. 33–43 (2006)
173. Song, D., Perrig, A., Phan, D.: AGVI—Automatic Generation, Verification, and Implementation of security protocols. In: Berry, G., Comon, H., Finkel, A. (eds.) CAV'01. LNCS, vol. 2102, pp. 241–245. Springer, Heidelberg (2001)
174. Song, D.X., Berezin, S., Perrig, A.: Athena: a novel approach to efficient automatic security protocol analysis. Journal of Computer Security 9(1/2), 47–74 (2001)
175. Sprenger, C., Backes, M., Basin, D., Pfitzmann, B., Waidner, M.: Cryptographically sound theorem proving. In: CSFW'06. pp. 153–166. IEEE, Los Alamitos (2006)
176. Swamy, N., Chen, J., Fournet, C., Strub, P.Y., Bharagavan, K., Yang, J.: Secure distributed programming with value-dependent types. In: Chakravarty, M.M.T., Hu, Z., Danvy, O. (eds.) ICFP'11. pp. 266–278. ACM, New York (2011)
177. Tšahhirov, I., Laud, P.: Application of dependency graphs to security protocol analysis. In: Barthe, G., Fournet, C. (eds.) TGC'07. LNCS, vol. 4912, pp. 294–311. Springer, Heidelberg (2007)
178. Weidenbach, C.: Towards an automatic analysis of security protocols in first-order logic. In: Ganzinger, H. (ed.) CADE'99. LNAI, vol. 1632, pp. 314–328. Springer, Heidelberg (1999)
179. Woo, T.Y.C., Lam, S.S.: A semantic model for authentication protocols. In: IEEE Symposium on Security and Privacy. pp. 178–194. IEEE, Los Alamitos (1993)
180. Yao, A.C.: Theory and applications of trapdoor functions. In: FOCS'82. pp. 80–91. IEEE, Los Alamitos (1982)