Introduction to Foundations of Security Protocols

R. Ramanujam
The Institute of Mathematical Sciences
Chennai
jam@imsc.res.in

S. P. Suresh
Chennai Mathematical Institute
Chennai
spsuresh@cmi.ac.in

1 Background

Computer security has come to occupy an increasingly central place in our lives over the past twenty years. This has been a direct result of the enormous increase in the development and use of networked and distributed systems over this period. Financial transactions on the Internet is gaining currency now. Distributed financial transactions — even if they are in the simple form of withdrawing money from an ATM — have become part of many peoples' lives today. Even more pervasive is the routine use of electronic mail (which is sometimes even used to share confidential information). The consequences of a misuse of such systems are potentially disastrous. This places a high premium on ensuring that such systems are not misused.

Security can basically be considered as a study of what the potential misuses of such systems are and how they can be averted. A system may be said to be secure if the properties of *confidentiality*, *integrity*, *availability*, *authenticity*, etc. of the various system entities are maintained. Broadly speaking, a system maintains confidentiality if no information can be *accessed* except by those entities which are authorised to access it. Similarly, a system maintains integrity if no information can be *altered* except by those entities which are authorised to alter it. Availability simply means that the desired information (or resource) is available when desired. An entity is said to be authentic if its apparent identity is genuine, i.e., the entity in question does not masquerade as some other entity.

The main challenge in security is to maintain some (or all) of the above attributes in the presence of malicious users, accidental misuse or under some kinds of system failures.

Historically, many different traditions have contributed to developments in computer security. Developments in operating systems, military security, and cryptography have all driven advances in security. From its early days, research in security has focused on *formal methods* for proving systems correct. This is easily understandable, since the consequences of a security-related error in a system could be disastrous, and thus the utmost care is required in ensuring the security of systems. Formal methods are a useful aid in the design and analysis of such systems.

Research on formal methods related to security has grown so much over the years that it is no longer possible to consider it as a unified whole. Based on the differences in the focus of research and the techniques and tools used, we have several subdisciplines. Our contributions lie in the area

of security protocols, which we look at in detail in the following sections. Meanwhile, we briefly look at some of the other disciplines below.

Program security: This is a classic area of study in security. The fundamental focus of research in this area is to devise methods which ensure that no program learns information that it is not authorised to know. Examples of programs which learn information in such an unauthorised manner are *viruses* and *Trojan horses*. For high-security systems like those used in the military, it is highly important to check all the programs to see if they have *secure information flow*. Formal methods are of immense help here. The fundamental theoretical problem studied here is whether a given problem has secure information flow ([12], [26]). A simple definition of a program having secure information flow is as follows: if the variables used in the program are partitioned into high-security and low-security variables, observations of the low-security variables do not reveal any information about the initial values of the high-security variables. Closely related is the problem of detecting *covert flows* [43], where information is leaked indirectly, through variations in program behaviour. The research in this area has focussed on syntactic mechanisms (like typing, see [74] for instance) and semantic methods (see [45], for example), to ensure secure information flows in programs and to detect information leaks.

Security policy: This is another widely studied area in security, which has its origins in the access control model for confidentiality used in operating systems (see [44], for instance). The central problem here is somewhat similar to that in program security, but is more general. The focus is on ensuring that there is no unauthorised access to information. Most of the solutions depend on restricting the behaviour of the system to achieve security. A classic example is *multilevel security*. Let us assume for simplicity that there are two user levels: *high* and *low*. Let us also assume that there are two security levels for objects: *confidential* and *public*. The typical restrictions on such a system might include *no read-up*: a low user cannot read a confidential file, and *no write-down*: a high user cannot write to a public file. Note that these are restrictions on the *run-time* behaviour of the systems. The fundamental theoretical challenge is to come up with good *security policy models*, which are formal specifications of the desired security-related behaviour of systems. [12] and [38] are two early papers dealing with security models. They propose models for confidentiality which are directly based on access control models for operating systems. The model proposed in [12] has features for access control as well as *multilevel security*. The current trend of research in this area is to use more abstract models based on the so called *interface models*, which derive from [34]. See [49] for a good survey of security models.

Database security: The main focus in this line of research is the same as that of the above two — to ensure that every piece of *information* in a database is learnt only by users authorized to know it. This implies much more than protecting *data*, which can be implemented by some kind of access control mechanism. A simple example to illustrate this point involves a salary database where salaries above a certain threshold have to be kept secret. It is easy enough to prevent queries from directly accessing the records which have salary above the given threshold. But there are other kinds of information which could be learned, like the average or sum of the salaries above the threshold. In such cases, it is possible that information about individual records can be inferred by cleverly asking many queries. For instance, if S is a set of employees and $S' = S \cup \{a\}$, then by learning the sum of the salaries of the employees in

S, and the same for the employees in S', a's salary can be learned. In some cases, even the fact that there exists a record of a particular kind is vital information, even if the exact data cannot be accessed. In most of these cases, the operation of aggregation introduces much complexity in the system, by introducing many potential means to learn information. Much of the research has focussed on statistical techniques to prevent the inference of information. A brief introduction to the field (as also a general insight into computer security) can be had from [35].

1.1 Security protocols

Security protocols are specifications of communication patterns which are intended to let agents share secrets over a public network. They are required to perform correctly even in the presence of malicious intruders who listen to the message exchanges that happen over the network and also manipulate the system (by blocking or forging messages, for instance). Obvious correctness requirements include secrecy: an intruder cannot read the contents of a message intended for others, and authenticity: if B receives a message that appears to be from agent A and intended for B, then A indeed sent the same message intended for B in the recent past.

The presence of intruders necessitates the use of encrypted communication. Thus developments in the field of cryptography provide the foundation for the design of security protocols. Research in cryptography has a long and glorious history. The field has come into its own in the past century, with more and more sophisticated mathematical techniques used to develop more and more sophisticated cryptographic schemes. As a result, a wide variety of cryptographic tools are available to the security protocol designer: conventional (shared-key) cryptography, public-key cryptography, digital signature schemes, etc.

The operation of encryption typically involves transforming a given *plaintext* to a *ciphertext* with the use a *key*, such that given the key it is easy to compute the ciphertext from the plaintext and vice versa, and without the key it is hard to compute the plaintext from the ciphertext. The inverse operation of computing the plaintext given the ciphertext and the key, is called *decryption*. The ciphertext is intended to be communicated over a possibly insecure network. Conventional cryptography uses the same key for both encryption and decryption. Public-key cryptography systems ([28], [65]) use a pair of keys for each user of the system (the user's public and private keys), where messages are encrypted using the receiver's public key and decrypted using the receiver's private key. A comprehensive introduction to cryptography can be had from [69].

Research in cryptography primarily aims at developing new cryptosystems with improved mathematical guarantees. But the focus of research in security protocols is different. It has been widely acknowledged that even the use of the most perfect cryptographic tools does not always ensure the desired security goals. (See [11] for an illuminating account.) This situation arises primarily because of logical flaws in the design of protocols.

Quite often, protocols are designed with features like ease of use, efficiency etc. in mind, in addition to some notion of security. For instance, if every message of a protocol were signed in the sender's name and then encrypted with the receiver's public key, it appears as if a lot of the known security flaws do not occur. But it is not usual for every message of a protocol to be signed. This could either be for reasons of efficiency or because frequent use of certain long-term keys might increase the chance of their being broken using cryptanalysis. Great care needs to be exercised in such situations. The following example protocol highlights some of the important issues nicely. It is based on a protocol designed by Needham and Schroeder ([57]) and is aimed at allowing two

agents A and B to exchange two independent, secret numbers. It uses public-key encryption but does not require agents to sign their messages.

$$\begin{array}{lrcl} \text{Msg 1.} & A & \rightarrow & B \;: \{x, A\}_{pubk_B} \\ \text{Msg 2.} & B & \rightarrow & A \;: \{x, y\}_{pubk_A} \\ \text{Msg 3.} & A & \rightarrow & B \;: \{y\}_{pubk_B} \end{array}$$

Here $pubk_A$ and $pubk_B$ are the public keys of A and B, respectively, and $\{x\}_k$ is the notation used to denote x encrypted using key k. In the protocol, x and y are assumed to be newly generated, unguessable (with high probability, of course!), previously unused numbers, also called nonces (nonce stands for "*n*umber *once* used"). In message 2, B includes A's nonce. On seeing it A is assured that B has received message 1, since only B can decrypt the first message and use x in a later message. Similarly on receipt of the third message, B is assured of A's receipt of y.

At the end of a session of the protocol, both A and B share the secrets x and y and both also know that the other agent knows x and y. But it has been shown ([46]) that x and y are not necessarily known *only to A and B*. (Such a property needs to be satisfied if we want to use a combination of x and y as a key shared between A and B, for example.) The attack (called Lowe's attack) is given below:

$$\begin{array}{lrcl} \text{Msg } \alpha.1. & A & \rightarrow & I \;: \{x, A\}_{pubk_I} \\ \text{Msg } \beta.1. & (I)A & \rightarrow & B \;: \{x, A\}_{pubk_B} \\ \text{Msg } \beta.2. & B & \rightarrow & (I)A \;: \{x, y\}_{pubk_A} \\ \text{Msg } \alpha.2. & I & \rightarrow & A \;: \{x, y\}_{pubk_A} \\ \text{Msg } \alpha.3. & A & \rightarrow & I \;: \{y\}_{pubk_I} \\ \text{Msg } \beta.3. & (I)A & \rightarrow & B \;: \{y\}_{pubk_B} \end{array}$$

In the above attack, $(I)A \rightarrow B : x$ means that the intruder is sending message x to B in A's name, whereas $A \rightarrow (I)B : x$ means that the intruder is blocking a message sent by A intended for B. The above attack consists of two parallel sessions of the protocol, one (whose messages are labelled with α) involving A as the initiator and I as responder, and the other (whose messages are labelled with β) involving I (in A's name) as the initiator and B as the responder. (This shows that the names A, B, x and y mentioned in the protocol specification are just placeholders or abstract names, which can be concretely instantiated in different ways when the protocol is run. So according to A and B, they have just had a normal protocol session with I and A, respectively. But I knows better!) After the fifth message above, the intruder gets to know y which is the secret generated by B in a session with someone whom B believes to be A. This shows that the protocol does not satisfy the following property: *whenever an agent B engages in a session of the protocol as a responder and B believes that the initiator is A, then the secret generated by B is known only to A and B.* The seriousness of this flaw depends on the kinds of use the protocol is put to. It is worth noting that this attack does not depend on weaknesses of the underlying encryption mechanism (nor even on some keys being guessed by chance). It is also worth noting that this attack on the (simple enough) Needham-Schroeder protocol was discovered seventeen years after the original protocol was proposed. [46] also suggests a fix for the protocol:

$$\begin{array}{lrcl} \text{Msg 1.} & A & \rightarrow & B \;: \{x, A\}_{pubk_B} \\ \text{Msg 2.} & B & \rightarrow & A \;: \{x, y, B\}_{pubk_A} \\ \text{Msg 3.} & A & \rightarrow & B \;: \{y\}_{pubk_B} \end{array}$$

It is easy to see that the above attack does not happen anymore, but that still doesn't prove that the protocol does not have any vulnerabilities.

The following example illustrates a freshness attack (or replay attack), and also highlights the use of nonces. Consider the following protocol (which is inspired by the Denning-Sacco protocol [27]) which uses symmetric (shared-key) encryption, where A is *Aandal*, B is a *bank*, and S is a *key server*. We assume that every agent C shares a key k_{CS} with the server, which only C and S know.

$$\begin{array}{lllll} \text{Msg 1.} & A & \rightarrow & S & : A, B \\ \text{Msg 2.} & S & \rightarrow & A & : \{B, k, \{A, k\}_{k_{BS}}\}_{k_{AS}} \\ \text{Msg 3.} & A & \rightarrow & B & : \{A, k\}_{k_{BS}} \end{array}$$

In message 1, A requests from the server S a key to communicate with B. S generates k and creates message 2. Only A can decrypt this message successfully and learn k, since she alone possesses k_{AS}. She then passes on the component $\{A, k\}_{k_{BS}}$ to B. Now B also learns k. Now A can enter into a session with B using the key k. Since only A and B know k, there is no danger of any information being leaked out, as long as the key k is safe. But unfortunately, there is the following attack:

$$\begin{array}{lllll} \text{Msg } \alpha.1. & A & \rightarrow & S & : A, B \\ \text{Msg } \alpha.2. & S & \rightarrow & A & : \{B, k, \{A, k\}_{k_{BS}}\}_{k_{AS}} \\ \text{Msg } \alpha.3. & A & \rightarrow & B & : \{A, k\}_{k_{BS}} \\ \text{Msg } \beta.3. & (I)A & \rightarrow & B & : \{A, k\}_{k_{BS}} \end{array}$$

The attack is quite simple. Sufficiently long after the session α has happened, the intruder masquerades as A and enters into a session with B with the same old key k. This is possible because all the intruder has to do is to replay message 3 from the old session. There might be a question as to what this achieves, since the intruder cannot continue the session meaningfully unless k is leaked. But this is not a scenario which can be ignored. It might be the case that the key k has actually been compromised by long hours of cryptanalysis, much after the original session was played out. The above attack then gives the intruder a chance for putting this key into use. Or it might be the case that in the original session α, after setting up the key k, A sends the following message:

$$\text{Msg } \alpha.4. \quad A \rightarrow B : \{\text{Deposit Rs. 10000 from my account into } I\text{'s}\}_k$$

(This might well be money which is legitimately owed to I by A.) The intruder, who watches all the communication over the network, infers from the effect of the above message (Rs. 10000 deposited into I's own account) the content of message $\alpha.4$, and just replays it as part of session β.

$$\text{Msg } \beta.4. \quad (I)A \rightarrow B : \{\text{Deposit Rs. 10000 from my account into } I\text{'s}\}_k$$

Since the bank thinks that the request is coming from A, I ends up richer by Rs. 10000.

A simple solution to the problem is for A and B to generate fresh nonces at the start of each session, then obtain the key from S and check the timeliness of the key received from S as follows:

$$\begin{array}{lllll} \text{Msg 1.} & A & \rightarrow & B & : A, B \\ \text{Msg 2.} & B & \rightarrow & A & : y \\ \text{Msg 3.} & A & \rightarrow & S & : A, B, x, y \\ \text{Msg 2.} & S & \rightarrow & A & : \{x, B, k, \{y, A, k\}_{k_{BS}}\}_{k_{AS}} \\ \text{Msg 4.} & A & \rightarrow & B & : \{y, A, k\}_{k_{BS}} \end{array}$$

The use of the fresh nonces prevents the intruder from replaying old messages as new. Of course, it is imperative that for each session a unique, unguessable, random number is chosen as a nonce, since otherwise replay attacks cannot be prevented.

A different kind of problem exists with type-flaw attacks. This is illustrated by the following simple example (see [25] for more examples of interesting type-flaw attacks), where A sends a fresh, random secret x to B and also gets an assurance that B has received it.

$$\begin{array}{lllll} \text{Msg 1.} & A & \rightarrow & B & : \{(A, \{x\}_{pubk_B})\}_{pubk_B} \\ \text{Msg 2.} & B & \rightarrow & A & : \{x\}_{pubk_A} \end{array}$$

The intruder can use the structure of message 1 and get the secret generated in place of x leaked, as the following attack shows:

$$\begin{array}{lllll} \text{Msg } \alpha.1. & A & \rightarrow & (I)B & : \{(A, \{m\}_{pubk_B})\}_{pubk_B} \\ \text{Msg } \beta.1. & I & \rightarrow & B & : \{(I, \{(A, \{m\}_{pubk_B})\}_{pubk_B})\}_{pubk_B} \\ \text{Msg } \beta.2. & B & \rightarrow & I & : \{(A, \{m\}_{pubk_B})\}_{pubk_I} \\ \text{Msg } \gamma.1. & I & \rightarrow & B & : \{(I, \{m\}_{pubk_B})\}_{pubk_B} \\ \text{Msg } \gamma.2. & B & \rightarrow & I & : \{m\}_{pubk_I} \\ \text{Msg } \alpha.2. & (I)B & \rightarrow & A & : \{m\}_{pubk_A} \end{array}$$

The important point about this attack is that in session β, the intruder is using the term $\{(A, \{m\}_{pubk_I}$ in place of x. In the absence of any mechanism to indicate the type of data being received, B believes that he has received a nonce. By cleverly using the structure of the protocol over two sessions, the intruder learns the secret m at the end of message 2 of session γ. This example also shows that the length of messages occurring in runs of a protocol can be much more than that of the messages occurring in the protocol specifications. Of course, this attack can be simply thwarted by modifying the protocol as follows:

$$\begin{array}{lllll} \text{Msg 1.} & A & \rightarrow & B & : \{(A, x)\}_{pubk_B} \\ \text{Msg 2.} & B & \rightarrow & A & : \{x\}_{pubk_A} \end{array}$$

The above examples illustrate the kinds of attacks which typically happen. Much more details on authentication protocols, attacks on them, and the techniques used to tackle them can be found in the excellent survey article [21].

The above discussion illustrates the pitfalls in security protocol design, and also highlights the need for a systematic approach to protocol design and analysis. There are two possible approaches:

- Development of a design methodology following which we can always generate provably correct protocols. Much work in the protocol design community focuses on this approach. [5] gives a flavour of the kinds of useful heuristics which improve protocol design. But there has not been much theoretical development towards formally justifying these design guidelines.
- Development of systematic means of analysing protocols for possible design flaws. The bulk of the work in formal methods for security protocols focuses on this approach. Here again, there are two possibilities:
  - Development of methods for proving the correctness of certain aspects of protocols.
  - Development of systematic methods for finding flaws of those protocols which are actually flawed.

Our main contributions in this area lie in the field of formal analysis methods for security protocols. We now briefly look at some of the approaches which have been advocated in the literature for proving properties of protocols and detecting flaws in them.

An important stream of work relating to proving protocols right is *automated theorem proving*. The typical approach in this style of work is as follows: a formal protocol model is defined based on an expressive logic like first-order logic or higher-order logic. To every protocol, a theory in the logic is associated. Properties of protocols are also specified using the same logic. A property holds of a protocol if it can be derived from the theory of the protocol using the rules of the logic. Established proof techniques and tools in the logic can now be used to efficiently prove properties of protocols. Examples of this approach include [59] and [16]. The advantage of this approach is that the highly expressive logics in the framework can code up any protocol, and formally prove most of the desired properties. Some possible disadvantages are that it requires expert knowledge to code up a protocol into a theory, and that the theorem proving process is not fully automatic. Expert intervention is needed to guide the proof search. The complexity involved in defining the theory of a protocol introduces further chances for error. Another possible drawback is that the formal proofs are not intuitive, and thus hard for humans to understand and base further developments on them.

An alternative approach is to use belief logics to prove properties of protocols. The pioneering work in this line is [18], in which a modal logic (called the *BAN logic*) was introduced as a tool to specify and reason about properties of protocols. It is based on modalities which seek to formalise the epistemic reasoning of the agents involved in the protocol. This logic has many attractive features, chief among them being that it produces simple and abstract proofs, but there are also some drawbacks. To use the logic, the authors propose a systematic idealisation step, which converts each message of the given protocol into a formula which represents the potential knowledge gained after receipt of the message. This feature introduces a chance for error, since there is a possibility that a wrong idealisation might be used to prove properties of the protocol. [17], [36], and [58] are some papers which contain a discussion of this feature and suggest further improvements to the BAN logic. [7], [15], and [73] are some papers which attempt to improve the original logic with either new modalities or through new semantic features. While they address some weaknesses of BAN logic, the simplicity of the original logic is lost. More recently, there have been attempts to connect BAN style logics with other formal models for security protocols ([8] and [72], for example). There have also been attempts at automated reasoning about protocols using BAN-style logics ([42], for instance). [72] provides a comprehensive survey of BAN-style logics for authentication protocols. The modalities which these logics concentrate on are fairly abstract, like *belief*, *trust*, *control* etc. While it may not be difficult to formalise these modalities, it is not clear whether they are fundamental to reasoning about security. The iteration of these modalities also brings a lot of complexity in its wake, complicating many of the technical questions regarding these logics. Thus it is worthwhile to look at logics with simpler modalities.

Much of the literature is devoted to methods for detecting flaws in protocols using the so-called *model checking* approach. The main idea is to consider a finite state version (preferably with a small number of states) of the given protocol (by imposing bounds on the set of nonces and keys used) and prove that all states of the finite state system satisfies the desired property. This does not necessarily mean that the protocol itself satisfies the desired property, since use of unboundedly many data might possibly introduce more attacks. But if a violation of the desired property is discovered using the small system, it usually means that the protocol is also flawed. The focus of

research in this area is to devise methods which will guarantee that a finite state version of the protocol has most of the errors that the big system has, and to devise techniques for efficiently verifying the small system.

As we will see later, when we model security protocols formally, we get infinite state systems. Thus there is no given finite state system which one can verify. The finite model should be constructed from the protocol specification by using appropriate abstractions. The different subdivisions of research in this line basically reflect the different techniques using which the finite state system can be defined, and the different techniques that can be used to verify it. For example, [46], [48], [55], [67], and [68] advocate an approach based on process algebra, in which important security properties are defined using some form of process equivalence. [50], [51], [52] advocate an approach based on logic programming, where the protocol is modelled by a set of rules which tell us how each action of the protocol changes the state of the system, and several specialized proof techniques are used to prove that a bad state can never be reached by a protocol. [16] uses standard techniques based on abstract interpretation to define a finite-state system from a protocol. Techniques based on tree automata ([56], [37], [22], [23]) have been proposed to efficiently represent and manipulate the intruder's state. Typically the intruder's state is the cause of the infinite state nature of protocols, and hence methods of finitely representing the intruder state can help construct a finite state system from a protocol.

The model checking approach has enjoyed great success in unearthing bugs in many protocols, long after they had been put into use. [21] is a good reference for the many attacks which have been uncovered by formal verification tools. But the main drawback in this approach is that the use of a finite state system is not always justified. In fact, the general verification problem for security protocols is undecidable, and therefore there exist protocols which are not "equivalent" to any system with bounded number of states. In this context, [47] proves that for a certain syntactic subclass of protocols and for some particular kinds of properties, checking whether the protocol satisfies those properties amounts to checking whether a particular small system satisfies them. This provides a justification for verification algorithms, most of which define a small system of the above kind from a given protocol, and verify the small system. Our own decidability results are in the same spirit as the results of [47].

2 Formal modelling of Security Protocols

The formal modelling of security protocols is a nontrivial problem in itself. For example, consider the Needham-Schroeder protocol presented in Section 1.1.

- The protocol is specified in terms of two agents A and B and two secrets x and y. But as evidenced in Lowe's attack, these are just abstract names which act as placeholders and can be concretely instantiated with different values to create many different sessions of the protocol.
- It is also evident from Lowe's attack that runs typically contain many parallel sessions.
- Further there could be infinitely many sessions of a given protocol and it is possible that a run consists of unboundedly many sessions.
- A further complication is that the abstract terms in the protocol can be instantiated with arbitrary messages (not just atomic messages) to carry out certain attacks. This was illustrated

by the second example of Section 1.1.

So we see that while protocol specifications are finite (usually quite small), the system which generates the set of runs of the protocol needs to remember an unbounded amount of information, and is thus an infinite state system. Thus a formal model for security protocols involves many details which need to be got right. The large gap in complexity between a protocol specification and the system which generates the runs of the protocol makes the task of formally modelling protocols nontrivial.

Further, at every step of defining a model, the modeller is presented with choices which have to be resolved one way or the other. Some of the possible questions that she might face are:

- what should be the structure of the messages?
- how are protocols to be presented?
- what should be the assumptions on intruders?
- how do agents construct new messages from old?
- what is the underlying model of communication?

As always, the manner in which the choices are resolved is driven by the application in hand. Thus it is not surprising that a consensus has still not been reached, and that the literature abounds with many different models for security protocols.

Before a description of our model, we briefly look at some of the other popular styles of modelling security protocols.

Process algebra models Examples of these kinds of models include the CSP-based models of [46], [48], and [67], and the the spi calculus model of [4]. We look at the spi-calculus model to provide a flavour of these kinds of models. It is an extension of the pi calculus [54] with cryptographic primitives. The basic idea is that every protocol is represented by a spi calculus process (which gives the operational semantics of the protocol, in the sense that the process displays exactly the same run-time behaviour as the protocol). The process for a protocol is typically a parallel composition of (possibly many different instantiations of) a process for each role of the protocol. The other process algebra models also model the behaviour of the intruder as an *intruder process*, and the process corresponding to a protocol is defined as a parallel composition of the processes for the roles and the intruder process. But the spi calculus differs from them in that it does not fix an intruder process. We will see a little later how intruder behaviour is modelled in the spi calculus. Security properties of protocols can now be translated to properties of the process representing the protocol. These are typically various kinds of observational equivalences between processes, which basically say that no observer interacting with the two processes can distinguish between the two.

For instance, let us say that a protocol which uses an abstract term x is represented by a process $P(x)$. (The notation signifies that the definition of P is parametrized by x.) Let us say that the protocol involves sending x from A to B securely. For every concrete term m, we define $P_{spec}(m)$ to be a process which is "obviously correct" in its behaviour with respect to m. (For instance, it might say that irrespective of what happens after A sends the message m, at some future point of time B (either normally or magically) receives the same message m.)

Now a possible definition of secrecy is that for any two distinct messages m and m', $P(m)$ is *observationally equivalent* to $P(m')$. If the secret is not revealed, then no external observer can see any difference between a run of the protocol which uses secret m and one which uses secret m'. A possible definition of authentication is that for all m, $P(m)$ is observationally equivalent to $P_{spec}(m)$. This says that if the A sends the message m, then if at all the receiver receives a message which purports to be from A, the message has to be m.

Since the notion of observational equivalence used in the spi calculus refers to *all processes*, there is no need to explicitly define an intruder process. If there is an attack on a protocol, it will definitely manifest in the form of the two relevant processes being distinguishable by a process coding up the intruder behaviour in the attack.

The main focus of research in spi calculus is to develop generic proof techniques that work for classes of protocols ([3], [1], [2]). It is also possible to use existing tools for the process algebra models and apply them to security. An example is the FDR model checker for CSP, which has been successfully used in discovering attacks on protocols (see [46], for example).

The inductive approach This approach was pioneered by [59], which advocates a theorem - proving approach to verifying cryptographic protocols. The theorem prover used in [59] is Isabelle/HOL, which works with higher-order logic.

A protocol is formalised as a set of *traces*, where each trace is a sequence of *events*. Example of events include Says A B X and Notes A X. Says A B X means that A says X to B, it does not imply that B heard what A says. Notes A X means that A learns the message X. The important point is that the set of traces of the protocol is defined inductively, starting with the empty trace, adding "proper" actions for the honest principals, and any "admissible" action for the intruder. "Proper" actions are those which follow the protocol. For instance the fifth message of a role can be sent only after the fourth message. "Admissible" means that the message that is being communicated in the event can be constructed by the agent from the information already learnt by him. The operators synth and analz formalize the way in which new messages are constructed from old.

A protocol is said to satisfy a property if all its traces satisfy the property. This can be verified by letting a theorem-prover inductively check that all traces of the protocol satisfy the said property. If a property does not hold of a protocol, then the failed attempts at a proof lead one to an attack scenario.

The inductive approach has been used as a basis for proving the correctness of some very complicated protocols [13].

Strand spaces This is a model introduced in [33]. In this model, a protocol is assumed to be presented by set of *(parametrized) strands*, which are sequences of send or receive actions. A *node* of a protocol is a pair consisting of an instantiation s of a parametrized strand and an index i which is at most the length of s. A strand space corresponding to a protocol is a graph whose nodes consist of all the nodes of the protocol and whose edges reflect the local and communication dependency between events. A very important component of the model is the formalisation of the intruder behaviour in terms of *penetrator strands*. Each penetrator strand describes an atomic behaviour of the intruder. Examples of such behaviour include receiving a message, creating a copy of a message that has been received, splitting a message of the form (t, t') to get t, encrypting t using a key k to obtain $\{t\}_k$, and so on. The

penetrator strands of this model, the intruder process in the process algebra models, and the intruder theory in the multi-set rewriting model (to be described below) roughly correspond to one another. A *bundle* of a protocol (which basically stands for a run of the protocol) is a finite partially ordered subgraph of the strand space of the protocol, with the condition that for every event in the bundle, its causal past is also included in the bundle. The significant feature of this model is that runs of a protocol are formalised as partially ordered objects.

Significant properties of protocols can now be expressed in terms of the model. An example of an authentication property is the requirement that whenever node n_1 occurs in a bundle, node n_2 should also occur. Secrecy properties are formalised by saying that some kinds of nodes do not occur in any bundle of the protocol. (These are typically nodes which reveal some secret to the intruder). A significant amount of the research here is devoted to developing techniques for proving general bounds on the intruder's abilities in any run of a protocol (or a class of protocols). There have also been attempts at automatic analysis of protocols based on the strand spaces model (see [70], for example). There have also been attempts to provide a semantics for BAN logic in terms of the strand space model ([72], for example).

Multi-set rewriting Like the spi calculus and the inductive model, this is also a general-purpose model in which we can embed security protocols. [32] is an introduction to the model, whereas [31] and [19] present technical results about the framework.

The basic idea here is that a security protocol is given by a *theory* which is a finite set of rules, where each rule is of the form $P_1(\cdots), \ldots, P_k(\cdots) \longrightarrow \vec{\exists}.\, Q_1(\cdots), \ldots, Q_l(\cdots)$. The P's and Q's are atomic formulas (of the predicate calculus). The theory of a protocol is got by composing a theory for each role with a standard intruder theory. A *state* is a finite multiset of atomic sentences. Rules are allowed to have free variables, but ground instantiations of rules are applied to states to yield new states. A rule application on a state s yields another state s' iff:

- all the preconditions of the rule all belong to s,
- the preconditions which are not postconditions do not belong to s',
- for every copy of a postcondition which is not a precondition, a copy of it is added to s',
- the rest of s is copied into s', and
- each existentially quantified variable is instantiated by a new constant not occurring in s.

In fact, the semantics of rules has close connections with the proof theory of linear logic.

Properties of security protocols can be easily formalised in this framework. For instance, the *secrecy problem* is essentially a state reachability problem (the input for the problem is a theory, an initial state and an atomic sentence). The problem is to determine whether there is a reachable state in which the said atomic sentence holds.

We now describe our model informally. While it does not differ drastically from any of the models described above, still there are differences in emphasis. Our focus is on retaining enough distinctions at the level of protocol specification so that it is easy to define certain syntactic subclasses, for which we later prove the decidability of verifying secrecy.

Protocol specifications: Security protocols are typically specified as a (finite) set of roles (typically with names like challenger, responder and so on). These are abstract patterns of communication which specify what messages are sent when, and how to respond to the receipt of any message. The content of these messages is (usually) not relevant, but the structure is; hence abstract variables suffice to describe the protocol. For example, the Needham-Schroeder can be viewed as consisting of two roles, an initiator role given by

$$A!B{:}\{x, A\}_{pubk_B}; \quad A?B{:}\{x, y\}_{pubk_A}; \quad A!B{:}\{y\}_{pubk_B}$$

and a responder role given by

$$B?A{:}\{x, A\}_{pubk_B}; \quad B!A{:}\{x, y\}_{pubk_A}; \quad B?A{:}\{y\}_{pubk_B}.$$

Roles are typically sequences of actions, which can either be a send action of the form $A!B{:}t$ (which stands for A sending t over the network intended for B) or a receive action of the form $A?B{:}t$ (which stands for A receiving t over the network with some indication that the sender is B).

In our model, we pay close attention to protocol specifications. In fact, our technical results show that the manner in which protocols are specified has a major bearing on problems like verifying secrecy of a given protocol. In fact, the negative results we have point out that the above style of presenting protocols admits too many complicated protocols, which are not representative of the protocols which arise in practice ([21]). So, for our positive results we focus on the more manageable class of protocols which are presented as sequence of communications of the form $A \rightarrow B : t$. This is also the informal style of presenting protocols which is popular in the literature. There are also some admissibility conditions here that are assumed implicitly in the literature. We make them explicit and point out their crucial role in the analysis of protocols. The class of protocols which satisfy these conditions are called well-formed protocols.

Starting from such descriptions of a protocol, we formally define the semantics of each protocol. This is slightly different from the style current in the literature. For instance, in the inductive model, a protocol is formally a set of rules (in higher-order logic) which specify the conditions under which runs of the protocol can be extended by adding an event. In the spi calculus model, a protocol is formally a spi calculus process (which can generate the set of all runs of the protocol). The passage from an informal protocol specification (as a sequence of communications) to the formal object is not given much attention (as that is usually trivially achieved). But formally any finite set of rules (or any process) can be a protocol. The advantage of such an approach is the high expressive power of the model. Any protocol can be coded up as a formal object of the model. A possible disadvantage is that it is sometimes difficult to isolate a certain (syntactic or semantic) class of protocols that we wish to concentrate on. Further, it is sometimes difficult to judge whether a technical result (like undecidability of verification, for instance) holds because of something inherent to protocols or because it is a general result which holds of the model itself.

Messages: A protocol as specified above is run by a set of agents, who are of two kinds: the malicious intruder and the rest, who are honest. They perform message exchanges as prescribed in the protocol. Following the lead of Dolev and Yao ([30]), we will assume that the terms

which are communicated in message exchanges come from a free algebra of terms with tupling and encryption operators. This means that we are operating on a space of symbolic terms, abstracting away from the fact that in the underlying system all messages are bit strings.

We work with a simple syntax of messages which allows only atomic keys. We disallow constructed keys, using which one can form messages of the form $\{x\}_{\{k\}_{k'}}$. While this choice certainly limits the applicability of our model and the results, we want to consider key technical questions like the decidability of the secrecy problem in this important setting, before moving on to more complex settings. On the other hand we feel that some of the other extensions to the message syntax, like hashing, can be easily handled and almost all our results will go through with minor modifications.

Cryptographic assumptions: Following the lead of Dolev and Yao ([30]) we make the perfect encryption assumption. This means that a message encrypted with key k can be decrypted only by an agent who has the corresponding inverse $\overline{k}$. We thus abstract away cryptographic concerns and treat encryption and decryption as symbolic operators. There is a different tradition to studying security protocols, called the "computational approach". In this approach, protocols are shown correct by reducing the protocol to the underlying cryptography, i.e., it is shown that if there exists an adversary with a significant chance of attacking the protocol, there exists another adversary with a significant chance of breaking the underlying cryptographic scheme itself. The work [14] is an example of this approach. We have chosen the more abstract framework which is preferred by most researchers in formal methods for cryptographic protocols. Recently, there has been some important work in reconciling the two approaches to cryptography. (See [6], [40], [41], for examples of such work.)

We also abstract away the real-life phenomenon in which some honest agents lose their long-term keys. This is modelled in [59], for example, by the notion of an Oops event. This reflects the probabilistic nature of the underlying cryptography, all the current schemes being not absolutely secure but only unbreakable with a very high probability. While we can model more attacks this way, we opt for a more restricted model in which decidability questions are easier to handle. Further our focus is mainly on logical flaws in protocols which exist even under the assumption that cryptography is absolutely unbreakable.

Intruder capabilities: We assume an all-powerful intruder, who can copy every communication in the system, can block any message and can pretend to be any agent. In addition he also has the message building capabilities available to every agent. It is assumed that the intruder has unlimited computational resources and can keep a record of every public system event and utilize it at an arbitrarily later time. However, we assume that the intruder cannot break encryption. These assumptions keep the intruder model technically simple. They are also followed widely in the literature.

The different models in the literature have tended to agree on most aspects of the intruder modelling. Such an intruder is called a Dolev-Yao intruder. Some variations to the above model have been tried but it has been shown that they do not significantly alter the intruder's powers. For example, we might consider a group of colluding intruders rather than a single intruder. But such a collusion cannot cause more attacks than a single intruder acting alone, as has been proved in [20].

Events and runs of a protocol: An event of a protocol is an action of some role of the protocol

with a substitution which supplies concrete terms for the abstract placeholders mentioned in the roles. As observed earlier, arbitrary terms can be substituted in place of nonces. An important class of events we will consider are the class of well-typed events which are obtained by substitutions which replace nonces only by nonces. It is clear that there are potentially infinitely many events of a protocol. If the set of nonces and keys is assumed to be infinite, it is possible that even the set of well-typed events is infinite.

A run of a protocol can informally be thought of as a sequence of events which respects certain admissibility conditions, which will be detailed below. Thus it is seen that we do not place any bounds on the number of plays occurring in a run, or on the number of plays which are active simultaneously (parallel sessions, as we called them earlier). It is to be noted that in [53] and [66], certain decidability results are obtained by essentially placing bounds on the number of plays that can occur in any run of the protocol. We follow an alternative approach by retaining the more general model and proving the corresponding decidability results for syntactic subclasses of protocols.

We consider sequential runs, like most of the other models in the literature, and unlike the strand spaces model. We choose sequential runs over partially ordered runs since we find it is easier to present the decidability arguments in that setting.

Admissibility: Arbitrary interleavings of plays of a protocol are not counted as runs. They have to be realisable, in the sense that for every action a occurring in the run, if t is the term communicated in a and if agent A is the communicator, t can be constructed from the information which is presented to A in the initial state along with the information learnt by her from the message exchanges preceding a. Another important requirement is that certain secrets which are used as instantiations of *new nonces* (i.e., abstract secret names which are specified as "fresh" by the protocol) should satisfy the property of freshness, i.e. these secrets have not been used before in the run. Thus a record of the secrets used so far in the run has to be necessarily kept. These considerations lead us to the notions of information state of an agent and message construction rules. The agents are supposed to have learnt all the messages which have been communicated to them. Further they can construct new messages from old by tupling, detupling, encryption and decryption using known keys, and by generating new unguessable nonces which have not been previously used by anyone. The formal counterparts of the message generation rules are the operators synth and analz which are at the heart of many of our technical results.

It is to be noted that our definition of runs is quite close to that given in [59]. At the level of defining runs, the admissibility conditions are quite standard in the literature. The key element in our model is that we consider incorporating some of these conditions in the protocol specification itself as a formalisation of a notion of a "well-behaved protocol".

Initial knowledge: This is another feature of security protocol modelling in which the different existing models have tended to display slight differences. One typical approach is to let this be part of the specification of protocols. For instance, we might say that every agent shares a key with the server in the initial state, while the server has (or can generate) all the other keys, which the agents can request and obtain. Or we might say that every agent shares a key with every other agent in the initial state. We follow the technically simple approach of fixing a set of keys known to each of the agents in the initial state, independent of the

protocol. This looks restrictive, but the model can be easily adapted to include such protocol specifications. We only need to add a few consistency conditions (for instance, at every state, if a key is available to some agent, then its inverse is also available to some (not necessarily the same) agent) for some of the technical results to go through.

Closely related to this is the issue of constant terms of a protocol. Typical names occurring in a protocol specification (like the names A, B, x, etc. of the Needham-Schroeder protocol) are placeholders which can be substituted with any other term to generate runs. But some protocols might refer to some agents like a key server, whose role can be played only by some designated processes. Thus we do not allow the meanings of these names to change during the course of a protocol run. While we usually do not distinguish between the rest of the honest agents either in terms of their initial knowledge or in terms of their computational power, designated agents like the key server might have some extra information in the initial state, and some added computational power as well.

3 Our contributions

Given a security protocol, *the secrecy problem* asks if there is a run of the protocol which leaks a secret or not. Our main contribution is to identify subclasses of protocols for which it is possible to automatically verify this property.

It turns out that when we model security protocols precisely, we get infinite state systems. There are many sources of unboundedness in the model which contribute to this. The first type of unboundedness occurs because there is no *a priori* bound on the number of sessions occurring in a run, and thus there is no bound on the length of the runs of a protocol as well. Further, requirements such as freshness might necessitate the use of a fresh nonce or key for each session. Since the number of sessions in a run is unbounded, it follows that there is no *a priori* bound on the number of distinct nonces and keys used in a run of a protocol. Further, as evidenced in the type-flaw attack which was shown earlier, messages occurring in runs of a protocol can be longer than those occurring in the protocol specification. Thus there is no *a priori* bound on the length of the messages which are part of the runs as well.

As such, it is to be expected that it is not possible to verify even simple reachability properties, and thus security properties like secrecy as well, of such systems. It has been formally proved in ([31], [39], [10]) that in fact, such simple problems are undecidable for these systems. Of the factors which lead to unboundedness of these systems, the number of nonces and the message length are of special importance. It is proved in [31] that even when the message length is restricted to be bounded, allowing an unbounded number of nonces to occur in runs of a protocol leads to undecidability. Dually, in [39] and [10], it is proved that even if the nonces and keys come from a fixed finite set, allowing arbitrarily long messages to occur in protocol runs leads to undecidabilty. In [62] we provide simple and uniform proofs for the above two undecidability results.

The literature consists of many proposals to cope with the undecidability results. If there is a bound on the number of nonces as well as the message length, then every run can be shown to be equivalent to a run of bounded length, in terms of the security-relevant information learnt by the various parties at the end of the run. This has been used to prove decidability in [31]. Another common approach is to place bounds on the number of plays of any run of the protocol, effectively yielding a finite state system. [10], [53] and [66] contain examples of this approach. There are also approaches which impose restrictions on the way messages can be constructed. Examples of this

include [29] and [10] where restrictions are imposed on the way messages are concatenated with one another to form new messages. The work in [23] uses techniques from tree automata to show decidability for a subclass of protocols in which every agent copies at most one piece of any message it receives into any message it sends. The survey article [24] gives a nice overview of the various approaches to decidability of security protocol verification, and also the various undecidability results. [10] also provides a nice perspective on the various factors which affect decidability of security protocol verification.

The literature also consists of work where decidability is obtained without placing such 'external' bounds. For example, the work [71] seeks to identify some simple semantic properties which lead to decidability and argue that these properties are satisfied by a large class of protocols found in the literature. [9] introduces checkable syntactic conditions which entail the equivalence of the given protocol to a finite-state system, and then gives methods of checking the finite-state systems for security breaches. A significant work in this line is [47], where decidability is proved for a *syntactic* subclass of protocols, under the assumption that message length is bounded but without any assumptions on the number of nonces. Our work [64] is in this spirit. Assuming that message length is bounded and the set of nonces is not, we prove decidability of the secrecy problem for a syntactic subclass of protocols, the so called tagged protocols. Essentially, these are protocols where the important components of each message have some kind of *type tags* attached to them. The use of tags allows us to prove that for every tagged protocol, there is a run which leaks a secret iff there is a run of bounded length which leaks a secret. This is the key to our decidability result.

We continue the same theme in [61] where we prove that even if we do not place any bound on message length, we can obtain decidability of the secrecy problem for the class of tagged protocols. We achieve this by showing that for tagged protocols, every run is equivalent to a *well-typed run* (under a suitable notion of equivalence which preserves many important security properties). A well-typed run is basically a run in which there is no type-flaw. This means that nonces occurring in the protocol specification are only replaced by nonces in the different sessions of the run, and so on for the other types of data as well. This further means that the length of the messages occurring in a well-typed run is bounded by the length of the messages occurring in the protocol specification. Since every run is equivalent to a well-typed run, the problem reduces in effect to the earlier setting and thus we get our decidability result.

In [60], we also consider a semantic subclass of protocols based on an equivalence relation of finite index on messages, and prove the decidability of the secrecy problem for this semantic subclass, under the assumption that the nonces and keys come from a fixed finite set.

In [63], we look at methods for reasoning about protocols. We define a logic in which several important properties like secrecy and authentication can be naturally specified. We give examples which illustrate how to reason about protocols using the logic. We then show that the undecidability results and the reduction to well-typed runs proved earlier extend to the verification problem for the logic as well. Using the reduction to well-typed runs, we prove the decidability of the verification problem of the logic in a setting where there are no restrictions on the length of messages occurring in runs of a protocol, but where the nonces and keys come from a fixed finite set.

References

[1] Martin Abadi. Secrecy by Typing in Security Protocols. *Journal of the ACM*, 46(5):749–786, 1999.

[2] Martin Abadi, Cédric Fournet, and Georges Gonthier. Secure Implementation of Channel Abstractions. *Information and Computation*, 174(1):37–83, April 2002.

[3] Martin Abadi and Andrew D. Gordon. A Bisimulation Method for Cryptographic Protocols. *Nordic Journal of Computing*, 5(4):267–303, 1998.

[4] Martin Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: the spi calculus. *Information and Computation*, 148(1):1–70, 1999.

[5] Martin Abadi and Roger M. Needham. Prudent engineering practices for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22:6–15, 1996.

[6] Martin Abadi and Phillip Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). In *Proceedings of the IFIP International Conference on TCS (IFIP TCS2000)*, volume 1872 of *Lecture Notes in Computer Science*, pages 3–22, 2000.

[7] Martin Abadi and Mark Tuttle. A Semantics for a Logic of Authentication. In *Proceedings of the Tenth Annual ACM Symposium on Principles of Distributed Computing*, pages 201–216, August 1991.

[8] Rafael Accorsi, David Basin, and Luca Viganò. Modal Specifications of Trace-Based Security Properties. In Klaus Fischer and Dieter Hutter, editors, *Proceedings of the Second International Workshop on Security of Mobile Multiagent Systems*, pages 1–11, July 2002.

[9] Roberto M. Amadio and Witold Charatonik. On name generation and set-based analysis in Dolev-Yao model. Technical Report 4379, INRIA, January 2002. Extended abstract in *Proceedings of CONCUR'02*, Springer-Lecture Notes in Computer Science 2421, pages 499–514, 2002.

[10] Roberto M. Amadio, Denis Lugiez, and Vincent Vanackère. On the symbolic reduction of processes with cryptographic functions. *Theoretical Computer Science*, 290(1):695–740, 2002. Also *INRIA Research Report* 4147, March 2001.

[11] Ross Anderson and Roger M. Needham. Programming Satan's computer. In *Computer Science Today*, volume 1000 of *Lecture Notes in Computer Science*, pages 426–441, 1995.

[12] David E. Bell and Leonard J. LaPadula. Secure computer systems: Mathematical foundations and model. Technical Report M74-244, MITRE Corporation, Bedford, Massachussets, 1973.

[13] Giampaolo Bella. Modelling Security Protocols Based on Smart Cards. In *Proceedings of the International Workshop on Cryptographic Techniques & E-Commerce*, pages 139–146, 1999.

[14] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In D. Stinson, editor, *Advances in Cryptography-Crypto93*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249, 1993.

[15] Pierre Bieber. A logic of communication in a hostile environment. In *Proceedings of 3rd Computer Security Foundations Workshop*, pages 14–22. IEEE Press, 1990.

[16] Dominique Bolignano. Towards a mechanization of cryptographic protocol verification. In *Proceedings of CAV'97*, volume 1254 of *Lecture Notes in Computer Science*, pages 131–142, 1997.

[17] Colin Boyd and Wenbo Mao. On a limitation of BAN logic. In *Proceedings of Eurocrypt'93*, Lecture Notes in Computer Science, pages 240–247, 1993.

[18] Michael Burrows, Martin Abadi, and Roger M. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, Feb 1990.

[19] Iliano Cervesato, Nancy A. Durgin, Patrick D. Lincoln, John C. Mitchell, and Andre Scedrov. A Meta-notation for Protocol Analysis. In P. Syverson, editor, *Proceedings of the 12th IEEE Computer Security Foundations Workshop*, pages 35–51. IEEE Computer Society Press, 1999.

[20] Iliano Cervesato, Catherine A. Meadows, and Paul F. Syverson. Dolev-Yao is no better than Machiavelli. In P. Degano, editor, *Proceedings of WITS'00*, pages 87–92, July 2000.

[21] John Clark and Jeremy Jacob. A survey of authentication protocol literature. Available at `http://www.cs.york.ac.uk./~jac`, 1997.

[22] Hubert Comon and Véronique Cortier. Tree automata with one memory, set constraints, and cryptographic protocols. *Theoretical Computer Science*, 2003. To appear.

[23] Hubert Comon, Véronique Cortier, and John C. Mitchell. Tree automata with One Memory, Set Constraints, and Ping-Pong Protocols. In *Proceedings of ICALP 2001*, volume 2076 of *Lecture Notes in Computer Science*, pages 682–693, 2001.

[24] Hubert Comon and Vitaly Shmatikov. Is it possible to decide whether a cryptographic protocol is secure or not? *Journal of Telecommunications and Information Technology*, 4:5–15, 2002.

[25] Mourad Debbabi, Mohamed Mejri, Nadia Tawbi, and Imed Yahmadi. Formal automatic verification of authentication protocols. In *Proceedings of the First IEEE International Conference on Formal Engineering Methods (ICFEM97)*, pages 50–59. IEEE Press, 1997.

[26] Dorothy E. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5):236–243, May 1977.

[27] Dorothy E. Denning and Giovanni M. Sacco. Timestamps in Key Distribution Protocols. *Communications of the ACM*, 24(8):533–536, August 1981.

[28] Whitfield Diffie and Martin E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, November 1976.

[29] Danny Dolev, Shimon Even, and Richard M. Karp. On the Security of Ping-Pong Protocols. *Information and Control*, 55:57–68, 1982.

[30] Danny Dolev and Andrew Yao. On the Security of public-key protocols. *IEEE Transactions on Information Theory*, 29:198–208, 1983.

[31] Nancy A. Durgin, Patrick D. Lincoln, John C. Mitchell, and Andre Scedrov. The undecidability of bounded security protocols. In *Proceedings of the Workshop on Formal Methods and Security Protocols (FMSP'99)*, 1999.

[32] Nancy A. Durgin and John C. Mitchell. Analysis of security protocols. In *Calculational System Design*, volume 173 of *Series F: Computer and System Sciences*, pages 369–395. IOS Press, 1999.

[33] F. Javier Thayer Fábrega, Jonathan Herzog, and Joshua Guttman. Strand Spaces: Proving Security Protocols Correct. *Journal of Computer Security*, 7:191–230, 1999.

[34] Joseph Goguen and Josè Meseguer. Security policies and security models. In *Proceedings of the 1982 IEEE Symposium on Research in Security and Privacy*, pages 11–20. IEEE Computer Society Press, 1982.

[35] Dieter Gollmann. *Computer Security*. John Wiley & Sons Ltd., 1999.

[36] Li Gong, Roger Needham, and Raphael Yahalom. Reasoning About Belief in Cryptographic Protocols. In Deborah Cooper and Teresa Lunt, editors, *Proceedings 1990 IEEE Symposium on Research in Security and Privacy*, pages 234–248. IEEE Computer Society, 1990.

[37] Jean Goubault Larrecq. A method for automatic cryptographic protocol verification. In *Proceedings of the 15th IPDPS Workshops 2000*, volume 1800 of *Lecture Notes in Computer Science*, pages 977–984, 2000.

[38] Michael Harrison, Walter Ruzzo, and Jeffrey Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, 1976.

[39] Nevin Heintze and Doug Tygar. A model for secure protocols and their composition. *IEEE Transactions on Software Engineering*, 22:16–30, 1996.

[40] Jonathan Herzog. Computational soundness for formal adversaries. Master's thesis, Massachussets Institute of Technology, October 2002.

[41] Jonathan Herzog. A Computational Interpretation of Dolev-Yao Adversaries. In R. Gorrieri, editor, *Proceedings of WITS'03*, pages 146–155, April 2003.

[42] Darrell Kindred and Jeannette Wing. Fast, automatic checking of security protocols. In *Proceedings of the 2nd USENIX workshop on e-commerce*, pages 41–52, 1996.

[43] Butler W. Lampson. A note on the confinement problem. *Communications of the ACM*, 16(10):613–615, October 1973.

[44] Butler W. Lampson. Protection. *ACM Operating Systems Review*, 8(1):18–24, 1974.

[45] K. Rustan M. Leino and Rajeev Joshi. A Semantic Approach to Secure Information Flow. *Science of Computer Programming*, 37(1–3):113–138, 2000.

[46] Gavin Lowe. Breaking and fixing the Needham-Schroeder public key protocol using FDR. In *Proceedings of TACAS'96*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166, 1996.

[47] Gavin Lowe. Towards a completeness result for model checking of security protocols. *Journal of computer security*, 7:89–146, 1999.

[48] Gavin Lowe and Bill Roscoe. Using CSP to detect errors in the TMN protocol. *IEEE Transactions of Software Engineering*, 23(10):659–669, 1997.

[49] John McLean. Security models. In John Marciniak, editor, *Encyclopedia of Software Engineering*. Wiley & Sons Inc., 1994.

[50] Catherine A. Meadows. Formal Verification of Cryptographic Protocols: (A Survey). In *Asiacrypt '94*, volume 917 of *Lecture Notes in Computer Science*, pages 133–150, 1995.

[51] Catherine A. Meadows. Analyzing the Needham-Schroeder public-key protocol. In E. Bertino, H. Kurth, G. Martella, and E. Montolivo, editors, *ESORICS'96*, volume 1146 of *Lecture Notes in Computer Science*, pages 351–364, 1996.

[52] Catherine A. Meadows. The NRL Protocol Analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131, 1996.

[53] Jonathan K. Millen and Vitaly Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *ACM Conference on Computer and Communications Security*, pages 166–175, 2001.

[54] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes: parts I and II. *Information and Computation*, 100(1):1–77, 1992.

[55] John C. Mitchell, Mark Mitchell, and Ulrich Stern. Automated analysis of cryptographic protocols using Murφ. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 141–153, 1997.

[56] David Monniaux. Abstracting cryptographic protocols with tree automata. In *Static analysis symposium*, volume 1694 of *Lecture Notes in Computer Science*, pages 149–163, 1999.

[57] Roger M. Needham and Michael D. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM*, 21(12):993–999, 1978.

[58] D. M. Nessett. A critique of the Burrows, Abadi and Needham logic. *ACM Operating systems review*, 24(2):35–38, 1990.

[59] Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of computer security*, 6:85–128, 1998.

[60] R. Ramanujam and S.P. Suresh. An equivalence on terms for security protocols. In Ramesh Bharadwaj, editor, *Proceedings of AVIS'03*, pages 45–56, Poland, Warsaw, April 2003.

[61] R. Ramanujam and S.P. Suresh. Tagging makes secrecy decidable for unbounded nonces as well. In *Proceedings of 23rd FST&TCS*, volume 2914 of *Lecture Notes in Computer Science*, pages 363–374, Mumbai, India, December 2003.

[62] R. Ramanujam and S.P. Suresh. Undecidability of secrecy for security protocols. Technical report, IMSc, July 2003.

[63] R. Ramanujam and S.P. Suresh. A modal logic for reasoning about security protocols. Technical report, IMSc, 2004.

[64] R. Ramanujam and S.P. Suresh. Decidability of context-explicit security protocols. *Journal of Computer Security*, 2004.

[65] Ronald L. Rivest, Adi Shamir, and Leonard M. Adelman. A Method for Obtaining Digital Signatures and Public-key Cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[66] Michaël Rusinowitch and Mathieu Turuani. Protocol Insecurity with Finite Number of Sessions and Composed Keys is NP-complete. *Theoretical Computer Science*, 299:451–475, 2003.

[67] Steve Schneider. Security properties and CSP. In *Proceedings of the IEEE Computer Society Symposium on Security and Privacy*, pages 174–187, 1996.

[68] Steve Schneider. Verifying Authentication Protocols in CSP. *IEEE Transactions on Software Engineering*, 24(9):741–758, 1998.

[69] Bruce Schneier. *Applied Cryptography*. John Wiley & Sons, second edition, 1996.

[70] Dawn Xiaodong Song, Sergey Berezin, and Adrian Perrig. Athena: A Novel Approach to Efficient Automatic Security Protocol Analysis. *Journal of Computer Security*, 9(1/2):47–74, 2001.

[71] Scott D. Stoller. A Bound on Attacks on Authentication Protocols. In *Proceedings of the 2nd IFIP International Conference on Theoretical Computer Science*, pages 588–600, 2002. An extended version appeared as Indiana University, CS Dept., Technical Report 526, July 1999 (revised January 2001).

[72] Paul F. Syverson and Iliano Cervesato. The logic of authentication protocols. In Ricardo Focardi and Roberto Gorrieri, editors, *Foundations of Security Analysis and Design*, volume 2171 of *Lecture Notes in Computer Science*, pages 63–106, 2001.

[73] Paul F. Syverson and P.C. van Oorschot. On unifying some cryptographic protocol logics. In *Proceedings of the 13th IEEE Symposium on security and privacy*, pages 14–28. IEEE Press, 1994.

[74] Dennis Volpano, Geoffrey Smith, and Cynthia Irvine. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(3):1–21, 1996.