# ATTACK ANALYSIS OF CRYPTOGRAPHIC PROTOCOLS USING STRAND SPACES

**Simon Lukell and Andrew Hutchison**

Data Networks Architectures Group
Department of Computer Science
University of Cape Town
Rondebosch
7701

Simon Lukell
slukell@cs.uct.ac.za
021-650 3127

Andrew Hutchison
hutch@cs.uct.ac.za
021-947 5293

ABSTRACT

Network security protocols make use of cryptographic techniques to achieve goals such as confidentiality, authentication, integrity and non-repudiation. However, the fact that strong cryptographic algorithms exist does not guarantee the security of a communications system. In fact, it is recognised that the engineering of security protocols is a very challenging task, since protocols that appear secure can contain subtle flaws and vulnerabilities that attackers can exploit. A number of techniques exist for the analysis of security protocol specifications. Each of the techniques currently available is not capable of detecting every possible flaw or attack against a protocol when used in isolation. However, when combined, these techniques all complement each other and allow a protocol engineer to obtain a more accurate overview of the security of a protocol that is being designed. This fact, amongst others, is the rationale for *multi-dimensional security protocol engineering*, a concept introduced by previous projects in the DNA group. We propose an *attack construction* approach to security protocol analysis within a *multi-dimensional context*. This analysis method complements the method used in the existing *inference construction* analysis tools developed earlier in the group. This paper gives a brief overview of the concepts associated with our project, including a summary of existing security protocol analysis techniques, and a description of the *strand space model*, which is the intended formalism for the analysis.

KEY WORDS

Security protocol, cryptographic protocol, analysis, attack, vulnerability, strand space model

# ATTACK ANALYSIS OF CRYPTOGRAPHIC PROTOCOLS USING STRAND SPACES

## 1 INTRODUCTION

Network security protocols make use of cryptographic techniques to achieve goals such as confidentiality, authentication, integrity and non-repudiation. However, the fact that strong cryptographic algorithms exist does not guarantee the security of a communications system [31]. In fact, it is recognised that the engineering of security protocols is a very challenging task, since protocols that appear secure can contain subtle flaws and vulnerabilities that attackers can exploit [2]. A number of techniques exist for the analysis of security protocol specifications. Each of the techniques currently available is not capable of detecting every possible flaw or attack against a protocol when used in isolation. However, when combined, they complement each other and allow a protocol engineer to obtain a more accurate overview of the security of a protocol that is being designed [16]. A former DNA group project, the Security Protocol Engineering and Analysis Resource (SPEAR) [3], and its successor, SPEAR II [29], introduced the concept of *multi-dimensional security protocol engineering*. Several aspects of cryptographic protocol engineering are collected into one application, which allows an engineer to rapidly construct, analyse and implement secure protocol designs. The aspect of security protocol analysis in these projects was based on the *inference construction* techniques BAN [6] and GNY [13] modal logics respectively.

We propose an automated *attack construction* analysis of security protocols *within a multi- dimensional context*. The primary focus of our investigation is on techniques of effective automatic searches for possible attacks – in form of secrecy and authentication violations – against the protocol. The other objective of the project is to study the relationship between inference construction and attack construction methodologies in order to facilitate combination of these in a unified environment such as SPEAR II. The method of our choice for formal description of security protocols is based on the so-called *strand space model*, first introduced by Thayer Fábrega, Herzog and Guttman [11].

The aims of this paper are:

- to position our proposed research in the context of multi-dimensional security protocol engineering in general and the SPEAR II project in particular,
- to give a brief overview of the existing security protocol analysis techniques in order to put attack construction in context, and
- to present the strand space model, which is the formalism on which we will build the protocol analysis.

The remainder of the paper is organised as follows. In Section 2, multi-dimensional security protocol engineering is described, with a summary of the SPEAR II project. A brief overview of existing protocol analysis techniques is given Section 3, after which the strand space model is presented in Section 4. Finally, the paper is concluded in Section 5.

## 2 MULTI-DIMENSIONAL SECURITY PROTOCOL ENGINEERING

Specialised tool support for formal methods can significantly aid protocol engineers in creating and implementing cryptographic protocols that achieve their goals, do not leak information, and are immune to attacks. In fact, protocol design and analysis has become so advanced and complex that we cannot perform certain analyses by hand as they take too long and tend to become tedious and error-prone over time. It has been shown that each of the available techniques is not capable of cutting out every possible flaw in a protocol when used on its own [16]. On the other hand, when used in combination, they complement each other, resulting in a more secure implementation. This *multi-dimensional* approach combines a number of engineering dimensions into one application, aiding the construction, analysis and implementation of protocols. In addition, the SPEAR philosophy aims to *facilitate* cryptographic protocol engineering and aid users in focussing on the critical issues by presenting them with an appropriate level of detail, and by guiding them as much as possible. It is believed that a collection of tools for all aspects of security protocol engineering in a user-friendly environment will assist in producing more secure cryptographic protocols.

### 2.1 The SPEAR II Tool

A schematic overview of the SPEAR II framework is given in Figure 1. Completed modules within the framework are indicated by solid outlines, while possible future additions are denoted by grey outlines. This software engineering view of the tool is an intuitive representation that shows some relationships between its components, but it also indicates how the tool is used. In the figure, the big arrows between the modules indicate a natural work order when developing a security protocol, and the thinner arrows show what kind of information is conveyed between the modules. The figure suggests an iterative approach with the analysis modules feeding back results from analyses to the specification environment.

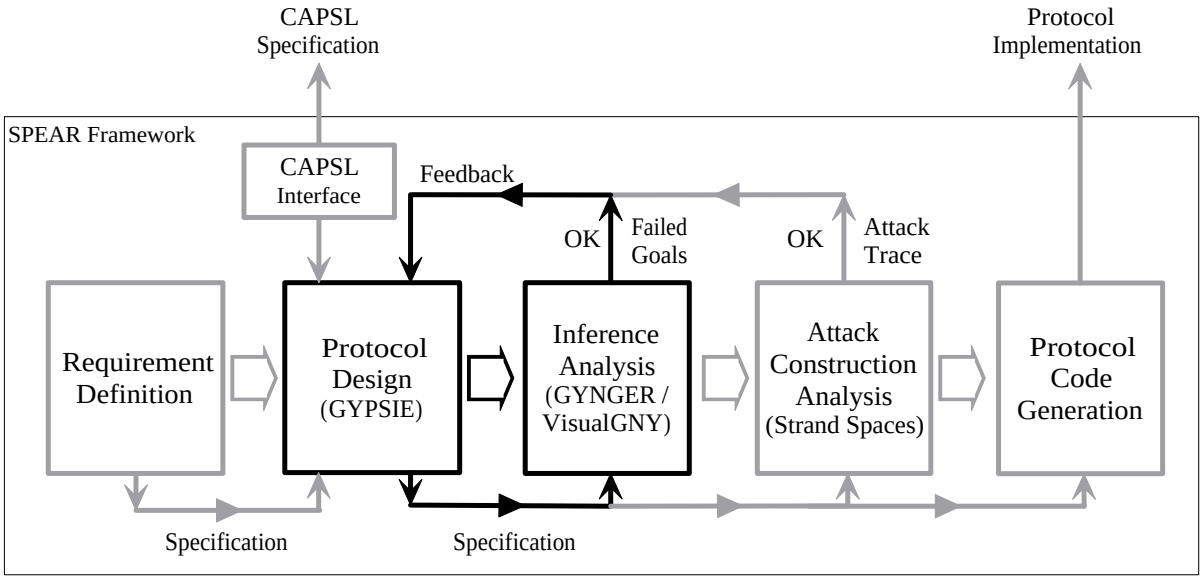


*Figure 1: The existing components of SPEAR II with proposed additions*

### 2.1.1 The Current Status of SPEAR II

Currently SPEAR II consists of four components integrated into one unified graphical interface:

- The GYPSIE protocol specification environment is designed for effective and accurate construction of cryptographic protocols and functions as the main interface of the SPEAR II application. By using three levels of abstraction presented through different views, the GYPSIE environment is able to present a protocol engineer with an appropriate impression of a cryptographic protocol and its operation.
- GYNGER is a Prolog-based analyser that performs automatic analysis of protocols by using the GNY modal logic [13]. The analysis engine employs a forward-chaining approach to mechanise the tedious application of GNY inference rules, allowing all derivable GNY statements to be generated accurately and efficiently.
- The Visual GNY environment was created to facilitate GNY-based protocol analysis and works in close conjunction with GYPSIE. In essence, Visual GNY functions as a user-friendly interface to the GYNGER analyser. To use the Visual GNY environment, users do not need to know the details of the GNY syntax and notation. However, they must be familiar with the semantics and concepts underlying the logic to use it effectively.
- A message rounds calculator (not shown) receives a protocol specification from GYPSIE and returns the messages that can be sent together in parallel, which ensures that the most efficient protocol design in terms of message rounds can be deployed at the implementation stage.

### 2.1.2 The Future of SPEAR II

In order to increase practical value of the tool, a number of additions can be made to the SPEAR II framework. From a software engineering perspective, a *protocol requirements tool* would assist the user to obtain an initial protocol specification from a set of requirements rather than having to define the specification from the beginning. On the other end, an *implementation generation tool* would complete the protocol engineering process. Development of such a tool is currently under research in the DNA group [34]. One way of increasing the confidence in a protocol specification is to use external analysis tools. The Common Authentication Protocol Specification Language (CAPSL) [23], supports interfaces to several tools. Therefore, a *CAPSL interface* would also be a useful addition to the framework.

As mentioned, the protocol analysis dimension in SPEAR II is based on GNY logic, which is only one of a number of available analysis techniques for security protocols. In order to increase confidence in a protocol specification, it is necessary to incorporate additional analysis methods in the framework. Our chosen method for protocol analysis, attack construction analysis, complements the existing GNY inference construction module in SPEAR II. The strand space model is an intuitive and efficient formalism, with inherent properties favouring an attack construction analysis approach. The results from other work in this area [32, 24] and our own experience of implementing a prototype system [18], favour this approach. The remainder of the paper gives an overview of the existing protocol analysis techniques and a description of the strand space model.

## 3 SECURITY PROTOCOL ANALYSIS TECHNIQUES

The available security protocol analysis techniques can be classified in a number of ways [15, 22, 21, 9]. For our purposes, the methods are classified as shown in Figure 2. The three main analysis methods are *ad hoc analysis*, *inference analysis* and *explicit intruder model analysis*.

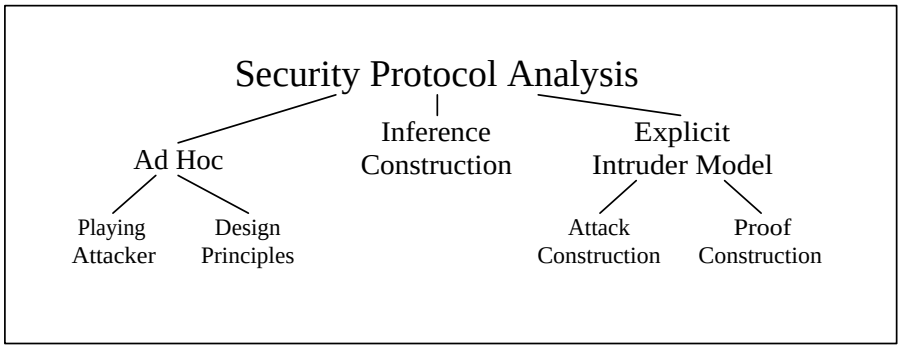


*Figure 2: A classification of security protocol analysis methods*

### 3.1 *Ad Hoc* Analysis

*Ad Hoc* protocol analysis is a set of informal methods of analysing protocol design. Many design flaws can be avoided using this method, in fact, some protocol vulnerabilities can only be detected with the use of *ad hoc* analysis. The biggest disadvantage with this approach is that the analysis only shows that the specification is resistant to each test performed on it. Moreover, due to its informal nature, there is no way of showing that the test itself is complete. *Ad hoc* analysis can be divided into *'playing the attacker'* and *design rules and principles*.

#### 3.1.1 Playing the Attacker

This is an informal way of testing the protocol specification (or implementation) for flaws. As the name indicates, the tester tries to break the protocol with the help of a checklist of previously identified protocol flaws and also applies tests made specifically for the analysed protocol. The method requires a great deal of experience and insight in protocol engineering for it to be effective.

#### 3.1.2 Design Rules and Principles

Through the history of security protocol design theory, a number of practical design rules and principles have been formulated to assist a protocol engineer to avoid design flaws. In many cases these principles, if followed, are shown to rule out attacks of a certain type against the protocol [14, 1, 2]. An important group of design principles are those that deal with issues in the area between the symbolic high-level representation of protocols, and the low-level cryptographic operations. For example, some attacks take advantage of properties of concatenated message components at the bit level [27], which cannot be identified in a formal analysis on the protocol level.

### 3.2 Inference Analysis

Inference analysis is a class of analyses that builds a framework of modal logic around properties such as knowledge and beliefs of the participants in a protocol. The first logic system for protocol analysis was the so-called BAN logic devised by Burrows, Abadi and Needham [6]. It assumes that authentication is a function of integrity and freshness, and uses logical rules to trace both of those attributes through the protocol. There are three main stages for the analysis of a protocol using BAN logic. The first step is to express the assumptions and goals as statements in a symbolic notation so that the logic can proceed from a known state to one where it can ascertain whether the goals are in fact reached. The second step is to transform the protocol steps into symbolic notation. Finally, a set of deduction rules called postulates are applied. The postulates should lead from the assumptions, via intermediate formulas, to the authentication goals. The BAN logic has been extended in, amongst others, GNY [13] (the analysis method used in SPEAR II), and SvO [33]. Inference analysis of cryptographic protocol has shown to be a success. A number of protocol flaws have been found by the use of this technique, including Needham-Schröder [25] and CCITT X.509 [7]. However, as stated previously, these methods have limitations. One type of attack that cannot be detected using this kind of analysis is so-called type-flaw attacks, which take advantage of some protocols' vulnerability to message component substitutions.

### 3.3 Explicit Intruder Model Analysis

This class of analysis methods involve an explicit model of the protocol and a model of an intruder. There are many available formalisms that can be used to model the protocol, the participants, the intruder, their actions and the messages that they exchange. Examples of such formalisms is the language used in the NRL protocol analyzer [20], the rank functions used in CSP based analysis [30], and the previously mentioned strand space model. All approaches within this class use essentially the same basic assumptions about network communication and the capabilities of the adversary. These assumptions are based on the model introduced by Dolev and Yao [8], which gives the intruder the following capabilities:

- Read any message and block further transmission
- Decompose a message into parts and remember them
- Generate fresh data as needed
- Compose a new message from known data and send it

It is worth noting here that the intruder is only capable of obtaining encrypted information if he possesses the key to decrypt it with. This is referred to as *perfect encryption assumption* [19], which is a means of isolating the protocol functionality from the cryptographic operations used in it.

There are two different methods of using this model, namely searching the model forwards and searching it backwards. Tools that use a forward search start in an initial state of a protocol environment and search the state space for insecure states exhaustively. This kind of analysis we refer to as *attack construction*. The backward search, called *proof construction*, attempts to prove that a given insecure

state of a protocol is unreachable. Outside the world of security protocol engineering, these methods are called *model checking* and *theorem proving* respectively. The distinction is made here in order to emphasise the fact that the forward search of a model *finds an attack* against a protocol (in form of a trace), whereas a backward search *finds a proof* of a specified attack being possible against a protocol.

#### 3.3.1 Attack Construction

As the name indicates, this kind of methods construct probable attack sets based on the algebraic properties of the protocol's algorithms. Examples of such methods are the NRL Protocol Analyzer [20], and Lowe's method of using the FDR model checker [17]. These methods are targeted towards ensuring authentication, correctness, or secrecy properties of the analysed protocols. Their disadvantage lies in the big number of possible events that must be examined, also referred to the *state space explosion problem*. However, various optimisation techniques exist that limit the search space to a manageable size. Furthermore, in combination with the development of more powerful computer systems, this approach has shown to be viable for modelled systems of a reasonable size. The main advantage of this approach is that it is largely automatic, a property that agrees with the usability philosophy of the SPEAR project.

#### 3.3.2 Proof Construction

Attempts to avoid the exponential searches of attack construction, and to extend analyses that involve arbitrarily large numbers of participants and messages, has given rise to the proof construction approach for the analysis of protocol failures [26, 5, 28]. It has the potential of being as thorough as attack construction in proving possible attacks, while avoiding exponential searches by replacing them with theorems about these searches. This method is completely general, with the disadvantage that it typically requires significant human insight and guidance.

#### 3.3.3 Hybrid Methods

The complementary nature of model checking and theorem proving has led to attempts of combining the two above methods in order to take full advantage of the strength of respective approach. An example of a domain-independent tool that does this is Berezin's Symbolic Model Prover (SyMP) [4]. The model checking aspect of the tool provides the high degree of automation and the theorem proving aspect provides rules for limiting the search space. The challenge in this area is to guarantee termination of the search, without compromising the (practical) completeness of it.

## 4 THE STRAND SPACE MODEL

In this section we informally introduce the strand space model. For a detailed formal account for the model, refer to the original papers [11, 10, 12]. Firstly, the fundamentals of the model and the basic terminology is introduced, after which a simple example is given of how the model can be used to describe a known protocol flaw. Finally, a description of a modelled intruder using strands is given.

4.1 Basic Notions

The strand space approach is also based on the Dolev-Yao intruder model. It is a graph-based method that is used to to prove properties of arbitrary combinations of protocols running at the same time. In Figure 3, $P_1$, $P_2$, $P_3$ and $P_4$ are principals. The actions of the principals are modelled as a number of sequential threads put in parallel. These threads are are called *strands*. The nodes on the strands are the actions that the principal performs, in this case $+a$ means that message $a$ is sent, and $-a$ means that the same message is received. The sequence of actions along the strand is referred to as its *trace*. The nodes on a single strand are causally related (denoted with the $\Rightarrow$ operation). Between the strands there is in general the sending and receiving of messages ($\rightarrow$), so also between a sending and a receiving node there is a causal relationship. In the figure, strand $P_1$ sends message $a$ at a certain point, and strand $P_3$ expects message $a$ at another point, so the two strands can be hooked together at these points. A *bundle* consists of a number of strands (legitimate or not) hooked together where one strand sends a message and another strand receives that same message. A strand is a *linear structure*, a sequence of one principal's message transmissions and receptions, whereas a bundle is a *graph-structured* entity, representing the communication between a number of strands.

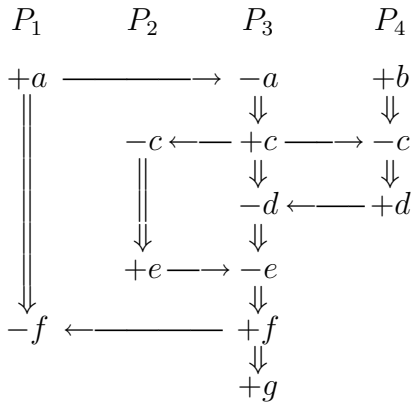


*Figure 3: A bundle*

A *strand space* is the set of all combinations of strands (all possible bundles) reflecting the activity of honest principals involved in a protocol, together with a number of strands of the intruder. The nodes on all the strands together form a partial order when provided with the causality relation induced by the sequentiality on a single strand ($\Rightarrow$) and the sending and receiving of messages ($\rightarrow$). Typically, for a protocol to be *correct*, each bundle must contain one strand for each of the legitimate principals participating in the session (i.e. are part of the bundle), all agreeing on the principals, nonces and session keys. Penetrator strands or stray legitimate strands may also be part of a bundle, even in a correct protocol, but they should not prevent the legitimate participants from agreeing on the data values, or from maintaining the secrecy of the chosen values.

A strand space models the assumption that some values occur only freshly by including only one strand *originating* that data by initially sending a message containing it. Many strands, however, may combine with the originating strand by receiving the message and process its contents further. A strand space also models the assumption that some values are impossible for a penetrator to guess.

In fact, the space simply lacks any penetrator strand in which a value is sent without it first having been received. The model allows several instances of the same trace simply by introducing identical strands representing the same trace being executed at different times.

### 4.2 An Attack Using Strands

We will illustrate the use of the model with an example with the Needham-Schröder public key protocol:

$$A \rightarrow B : \{N_a, A\}_{K_B}$$
$$B \rightarrow A : \{N_a, N_b\}_{K_A}$$
$$A \rightarrow B : \{N_b\}_{K_B}$$

The protocol is described in the so-called *standard notation* for security protocol descriptions. Each line defines a message in the protocol. The first message is sent from principal $A$ to $B$, denoted by $A \rightarrow B$, and the contents of the message is defined after the colon. The message is a concatenation of a nonce generated by principal $A$, $N_a$, and the identity of $A$. These two values are then encrypted with the public key of $B$, $K_B$.

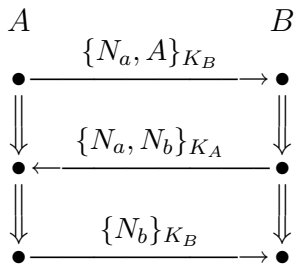


*Figure 4: Needham-Schröder*

In the strand space formalism, the protocol described above is given in the bundle of Figure 4. The column below $A$ represents the strand consisting of the initiator's activity during the exchange, while the column under $B$ represents the strand of the responder's activity. In this abbreviated form of the Needham-Schröder public key protocol, we assume that each principal has acquired the other's public key. The initiator generates a nonce, concatenates this to his name and encrypts this with the intended respondent's public key. The respondent generates a nonce of his own, sending it and the initiator's nonce back, encrypted with the initiator's public key. He has this way answered the initiator's challenge by showing that he could read the first message. Finally, the initiator returns the respondent's nonce encrypted with the respondent's public key for the same reason.

The intended result of the protocol is that the two principals should end up sharing access to the values $N_a$ and $N_b$, each associating these values with the other participant, and no other party should be in possession of them. The protocol might be used in a context where the two values are hashed together

to provide a shared symmetric key for an encrypted session. In fact, the protocol fails to achieve this goal [17]. Figure 5 shows a bundle that provides a counterexample (an attack against the protocol) and illustrates what can go wrong in the protocol. In the figure, the penetrator $P$ has two moments of activity, each represented by a short strand. The initiator $A$ intends to have a session with $P$ or some other principal whose key $P$ controls. $P$ uses this opportunity to impersonate $A$ to $B$.

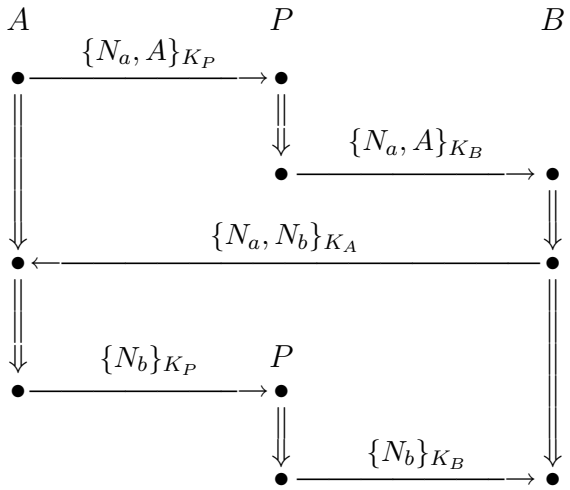


*Figure 5: Needham-Schröder Infiltrated*

The result of the attack is that $B$ believes that $A$ has been authenticated. $B$ also believes that the secret values $N_a$ and $N_b$ are only known by $A$ and $B$ just as in the intended exchange from Figure 4, while in reality $B$ is sharing the values with the penetrator $P$. The protocol flaw can (and was) eliminated by making $B$ include its identity in the second message ($\{N_a, N_b, B\}_{K_A}$). In Figure 5, $A$ is expecting a message from $P$ (the intended responder), but the modified message reveals the identity of the real sender $B$, and the attack fails.

### 4.3 The Intruder

The intruder's capabilities are decided by two factors, namely the set of keys known initially to the intruder, and a set of penetrator strands that allow the intruder to generate new messages from messages he intercepts. A *penetrator set* consists of a set of keys $K_P$. Typically, it contains all public keys, all private keys held by penetrator or his accomplices, and all symmetric keys initially shared between the penetrator and principals playing by the protocol rules. It may also contain "lost keys" that became known to the penetrator previously, perhaps because he succeeded in some cryptanalysis.

The atomic actions available to the penetrator are encoded in a set of penetrator traces. They summarise the ability to discard messages, generate well-known messages, piece messages together, and apply cryptographic operations using keys that become available. A protocol attack requires hooking together several of these atomic actions.

The existing penetrator traces are:

**M** Text message: $\langle +t \rangle$ where $t$ is a known component
**F** Flushing: $\langle -g \rangle$
**T** Tee (duplication): $\langle -g, +g, +g \rangle$
**C** Concatenation: $\langle -g, -h, +gh \rangle$
**S** Separation: $\langle -gh, +g, +h \rangle$
**K** Key: $\langle +K \rangle$ where $K \in K_P$
**E** Encryption: $\langle -K, -h, +\{h\}_K \rangle$
**D** Decryption: $\langle -K^{-1}, -\{h\}_K, +h \rangle$

These capabilities of the intruder correspond directly with the intruder capabilities of the Dolev-Yao model described earlier. Figure 6 shows an example of how these penetrator strands can be hooked together to provide the behaviour of the first step in Figure 5. The open circles in the figure (○) show the two points with which the diagram connects with the first nodes of $A$ and $B$'s strands at the top of Figure 5. The label above each strand shows which kind of penetrator strand it is.

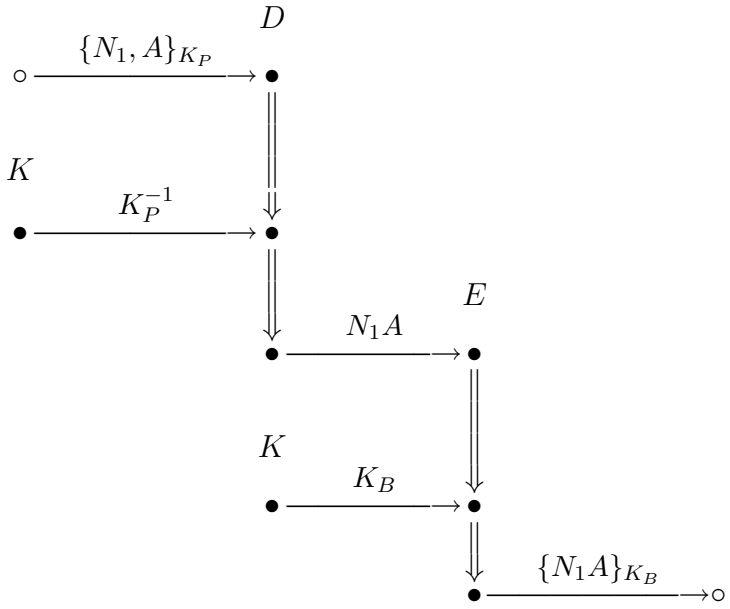


*Figure 6: Needham-Schröder: Penetrator's first step*

The strand space model is an appealing, clean model of security protocols that brings together many techniques that have been used in the field of security protocol analysis. The model considers the *strand* the basic unit as opposed to the *state* of the modelled principals, which makes it more economical in both specification and analysis.

## 5 CONCLUSION

We have presented the concept of multi-dimensional security protocol engineering with the SPEAR II project as an example thereof, and we have positioned our proposed research concerning attack analysis in this context. A summary of the existing analysis methods was given, and finally, in order to

complete the overview of our project, the strand space formalism was presented.

The strand space model is a good start in implementing an efficient automatic attack analysis tool, but other optimisation techniques must be considered in order to maximise the value of it. Furthermore, considering the context of the tool, it is of importance to adapt it as far as possible so that it complements the available inference analysis in an optimal manner. Also, in accordance with the SPEAR usability philosophy, the analysis must be made accessible to a user without expert knowledge in the field of security protocol engineering.

If these challenges are met, the analysis dimension will be able to detect a larger set of protocol vulnerabilities than is currently possible, which would be a valuable step towards a powerful multi-dimensional cryptographic protocol engineering system.

**REFERENCES**


[1] M. Abadi and R. Needham. Prudent Engineering Practice for Cryptographic Protocols. *IEEE Transactions on Software Engineering*, 22(1):6 – 15, January 1996.

[2] R. Anderson and R. Needham. Programming Satan's Computer. *Computer Science Today, Springer LNCS*, 1000:426–441, 1995.

[3] J.P. Bekmann, P. De Goede, and A.C.M. Hutchison. SPEAR: Security Protocol Engineering and Analysis Resources. In *DIMACS Workshop on Design and Formal Verification of Security Protocols*. Rutgers University, September 1997.

[4] S. Berezin. *Model Checking and Theorem Proving: a Unified Framework*. PhD thesis, Carnegie Mellon University, 2002.

[5] S.H. Brackin and R.W. Lichota. CASE for High Assurance: Utilizing Commercial Technology for Automated Cryptographic Protocol Analysis. In *Proceedings of the Sixth Annual Dual-Use Technologies and Applications Conference*, June 1996.

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

[7] CCITT. CCITT X.509. The Directory - An Authentication Framework, 1988.

[8] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198 – 208, 1983.

[9] N. A. Durgin and J. C. Mitchell. Analysis of security protocols. In *Calculational System Design*, pages 369–395. IOS Press, 1999.

[10] F. J. Thayer Fábrega, J. C. Herzog, and J. D. Guttman. Honest ideals on strand spaces. In *1998 Computer Security Foundations Workshop*, June 1998.

[11] F. J. Thayer Fábrega, J. C. Herzog, and J. D. Guttman. Strand spaces: Why is a security protocol correct? In *Proceedings of 1998 IEEE Symposium on Security and Privacy*, IEEE Comput. Soc, May 1998.

[12] F. J. Thayer Fábrega, J. C. Herzog, and J. D. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 15(7):191–230, 1999.

[13] L. Gong, R. Needham, and R. Yahalom. Reasoning about Belief in Cryptographic Protocols. In *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy*, pages 234 – 248, Oakland, California, 1990. IEEE Computer Society Press.

[14] L. Gong and P.F. Syverson. Fail-Stop Protocols: An Approach to Designing Secure Protocols. In *The Fifth International Working Conference on Dependable Computing for Critical Applications*, pages 44 – 55. Springer-Verlag, September 1995.

[15] S. Gritzalis, D. Spinellis, and P. Georgiadis. Security protocols over open networks and distributed systems: Formal methods for their analysis, design, and verification. *Computer Communications*, 22(8):695–707, 1999.

[16] N. J. Hopper, S. A. Seshia, and J. M. Wing. Combining theory generation and model checking for security protocol analysis. In *Post-CAV Workshop on Formal Methods in Computer Security*, July 2000.

[17] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1055, pages 147–166. Springer-Verlag, Berlin Germany, 1996.

[18] S. Lukell, C. Veldman, and A. Hutchison. Automated attack analysis and code generation in a unified, multi-dimensional security protocol engineering framework. Technical Report CS02-15-00, Department of Computer Science, University of Cape Town, October 2002.

[19] W. Marrero, E. Clarke, and S. Jha. A model checker types for authentication protocols. In *DIMACS Workshop on Design and Formal Verification of Security Protocols*. Rutgers University, September 1997.

[20] C. Meadows. The NRL protocol analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131, 1996.

[21] C. Meadows. Invariant generation techniques in cryptographic protocol analysis. In *PCSFW: Proceedings of The 13th Computer Security Foundations Workshop*. IEEE Computer Society Press, 2000.

[22] C.A. Meadows. Formal Verification of Cryptographic Protocols: A Survey. In *Advances in Cryptology - Asiacrypt '94*, pages 133 – 150. Springer-Verlag, 1995.

[23] J. Millen. CAPSL: Common authentication protocol specification language. Technical Report MP 97B48, The MITRE Corporation, 1997.

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

[25] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[26] L. C. Paulson. Mechanized proofs of security protocols: Needham-Schroeder with public keys. Technical Report 413, University of Cambridge, Computer Laboratory, 1997.

[27] O. Pereira and J.-J. Quisquater. On the perfect encryption assumption. In *Proceedings of the Workshop on Issues in the Theory of Security*, pages 42–45, 2000.

[28] A. W. Roscoe. Modelling and verifying key-exchange protocols using CSP and FDR. In *8th IEEE Computer Security Foundations Workshop*, pages 98–107, 1995.

[29] E. Saul. Facilitating the modelling and automated analysis of cryptographic protocols. Master's thesis, DNA Research Group, Computer Science Department, University of Cape Town, 2001.

[30] B. Schneier. Verifying authentication protocols with CSP. In *PCSFW: Proceedings of The 10th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1997.

[31] B. Schneier. Why cryptography is harder than it looks. *Information Security Bulletin*, 2(2):31 – 36, March 1997.

[32] D. X. Song, S. Berezin, and A. Perrig. Athena: A novel approach to efficient automatic security protocol analysis. *Journal of Computer Security*, 9(1/2):47–74, 2001.

[33] P.F. Syverson and P.C. van Oorschot. On Unifying Some Cryptographic Protocol Logics. In *Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy*, pages 14 – 29, Oakland, California, May 1994. IEEE Computer Society Press.

[34] B. Tobler and A. Hutchison. Generation, analysis and verification of cryptographic protocol implementations. In *3rd annual Information Security South Africa (ISSA) conference*, Sandton Convention Centre, Sandton, Gauteng, July 2003. ISSA. To Appear.