SPEAR: a Security Protocol Engineering & Analysis Resource

J.P. Bekmann, P. de Goede and A.C.M. Hutchison
Data Network Architectures Laboratory,
Department of Computer Science,
University of Cape Town,
Rondebosch 7701,
South Africa
E-mail {*hutch@cs.uct.ac.za*}

Abstract

SPEAR, the Security Protocol Engineering and Analysis Resource, is a protocol engineering tool which focuses on security protocols, with the specific aims of *secure and efficient design outcomes* and *support for the 'production' process*. SPEAR offers developers of cryptographic protocols an environment in which to design, analyze and generate security protocols. SPEAR supports protocol specification via a graphical user interface in the style of message sequence charts (MSCs). Security analysis based on cryptographic logics (currently BAN) is facilitated. Production code (currently Java) can be generated once a design has been analyzed. Meta-execution of code allows performance evaluation of cryptographic protocols from within the controlled environment of SPEAR. This paper documents the rationale for, and integration of, these various factors.

1 Introduction

The use of open, unreliable networks is increasing as more companies become connected to internetworks. Communications security of these systems is critical and while cryptographic algorithms exist, their security alone is not sufficient to ensure the integrity of a whole system.

One of the most important elements of a cryptographic system is the distributed nature of communicating parties. Since virtually all communication is done via insecure channels, exchange of information can be augmented by cryptographic techniques to provide security via *cryptographic* (or *security*) *protocols*.

It is widely recognized that the engineering of cryptographic protocols is a very subtle task which can easily be deceptive to developers. SPEAR, the Security Protocol Engineering and Analysis Resource, is a protocol engineering tool which focuses on security protocols, with the specific aims of *secure and efficient design outcomes* and *support for the 'production' process*.

SPEAR attempts to marry protocol engineering and formal logical analysis in a tool which greatly simplifies the design, analysis and generation of security protocols.

2 Security Protocol Engineering and Analysis

Burrows, Adabi and Needham recognize that it is very difficult to design security protocols that achieve their intended function and degree of security [4]. As a result formal methods have been developed to aid in the design and analysis of authentication and other security protocols. The development of logics to analyze security protocols provides one technique for ensuring the correctness of protocols [1]. Logic systems have been used to detect flaws in protocols previously accepted as correct [4, 10].

By combining aspects of a formal protocol specification and cryptographic logics SPEAR allows a user to design a security protocol in such a way that there is enough specification for the automatic generation of code that implements the protocol. By integrating aspects of cryptographic logics into this process, logical analysis is naturally incorporated into this design process. The combined expressiveness of this type of specification provides users with a tool which automates the most laborious aspects of the design process and offers a powerful and flexible test-bed for security protocol design.

There are languages available that allow the formal specification of protocols and tools which have been developed in the research of cryptographic logics (e.g. NRL analyzer [18], the Interrogator [16], HOL based Cryptographic logic tool [3]). To our knowledge, however, there are no tools that allow for the easy specification of cryptographic protocols combining both the logics and the formal protocol specifications in such a way as to distil the critical issues and present the user with a higher level design overview. Our tool aims to give users the freedom to explore the critical concepts of security in a controlled and expressive environment without having to actually do an implementation or make use of multiple tools to analyze different facets of a protocol.

3 Selection of Formalism

In selecting a formalism for representing the specification of a security protocol we primarily contemplated the use of the Specification and Description Language (SDL) [6, 2, 8]. SDL is in widespread use and has been used extensively in other projects undertaken by our research group, so this was an obvious consideration for us.

SDL is a specification language used to describe communicating systems such as telecommunication protocols. It is an International Telecommunication Union[1] standard, and as such is widely recognized and supported. Recently the textual version of SDL (SDL/PR) has also been augmented with a graphical means of specification (SDL/GR). In this regard SDL is superior to other formal description techniques such as Estelle [14, 7] and LOTOS [12] which have only textual representations. There are existing tools such as editors, correctness analyzers, simulators and code-generators for specifications given in SDL. Other members of our group have

[1]Formerly the Comité Consultatif International Telegraphique et Telephonique.

developed a system called SPECS II (the *S*DL *P*erformance *E*valuation of *C*oncurrent *S*ystems [5, 11]) which takes SDL input and performs correctness and performance analysis of protocol input. SPECS II also performs code generation from an SDL specification, and in this respect is similar to other proprietary SDL tools such as Geode [19].

After much deliberation and debate we decided against using SDL as our input formalism. Our reason for not selecting SDL as our specification formalism was largely that SDL does not allow a high-level cryptographic specification, but rather provides a low-level functional specification. While recognizing that it is at the functional level that subtle flaws can be identified, we perceive a lot of *common* low level functionality amongst cryptographic protocols. By this we mean that although message recipients and contents vary greatly in security protocols, much of the implementational detail such as connections, time-outs, etc. can be seen as common to many actual protocol realizations. Accepting this, one can view SPEAR as being, in fact, at a level *above* SDL – where designers can concentrate chiefly on logical security protocol design and actual message contents. In this way they can take advantage of analysis techniques at these levels while still having the possibility to generate SDL as an *output* of the modelling tool for further analysis should this be desired.

So we have not discarded SDL outright; what we are suggesting is that a full functional specification as required by SDL for each new protocol was not considered the level at which a tool like SPEAR should operate. With a view to allowing prototyping and experimentation, it was instead decided to use a protocol specification technique close to that of Message Sequence Charts (MSCs) [15]. MSCs capture the exchange of protocol messages at a higher level which we believe is more appropriate to cryptographic protocol design, where a lot of effort (and support) is required for message content construction. In SPEAR we have augmented the MSC-like specification with several features which make protocol specification clearer and more intuitive.

Since analysis of message exchanges is also valuable SPEAR can augment a user's protocol description with commonly used elements (of communication, reliability, etc.) to produce a complete SDL output for further analysis by such tools as SPECS II or Geode, if this is required.

4 Scope of the Tool

Practical experience with developing and deploying cryptographic protocols was the catalyst for developing SPEAR. The time taken to design and implement protocols, and particularly to analyze and tune their implementation, suggested the need for a tool which could encompass all of these aspects. In particular, experience with logic analysis revealed this to be a laborious task when attempted manually. Formulation of a protocol in SDL was again a time-consuming exercise that was not necessarily warranted in terms of, for example, checking for unreachable states or freedom from deadlock.

In this section and those which follow, SPEAR is described at *conceptual* and *architectural* levels, and then at the level of *realization*. At the conceptual level an *overview* of modules is first given after which each conceptual model is described in turn.

4.1 Conceptual Overview

Figure 1 positions SPEAR against the four aspects of security protocol design which it encompasses. Formal protocol specification, security analysis, code generation and meta-execution constitute four distinct, but interconnected modules in SPEAR.

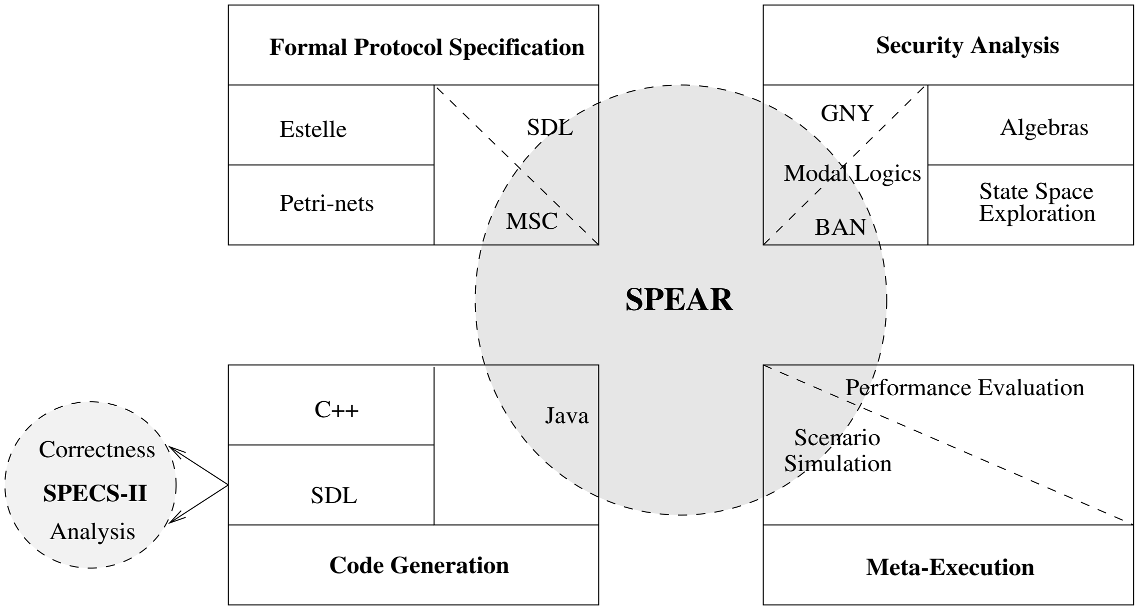

Figure 1: SPEAR Conceptual Overview

The depiction of SPECS II on the left hand side of the figure locates the security work in our research environment against other protocol analysis work and denotes the possibility to conduct SDL simulation and correctness analysis on the basis of existing available resources.

4.2 Protocol Specification

The choice of formalism for protocol specification has been discussed, and Figure 1 reflects that MSCs are the primary formalism with the adoption of some SDL-like features. Figure 2 presents the elements of protocol specification in SPEAR.

The steps necessary to specify a protocol are as follows:

1. Define the *entities* (communicating parties) involved in the protocol.

2. Define the *possessions* (data items) that are used in the protocol.

3. Define the *external functions* that provide a means of assigning initial values to possessions and allow interaction between user, and SPEAR generated, Java source code.

Secondary to these required steps one can:

- Define the *initial* and final *required* BAN beliefs of entities to allow BAN analysis of the protocol. The initial beliefs are those beliefs that the entity holds prior to execution

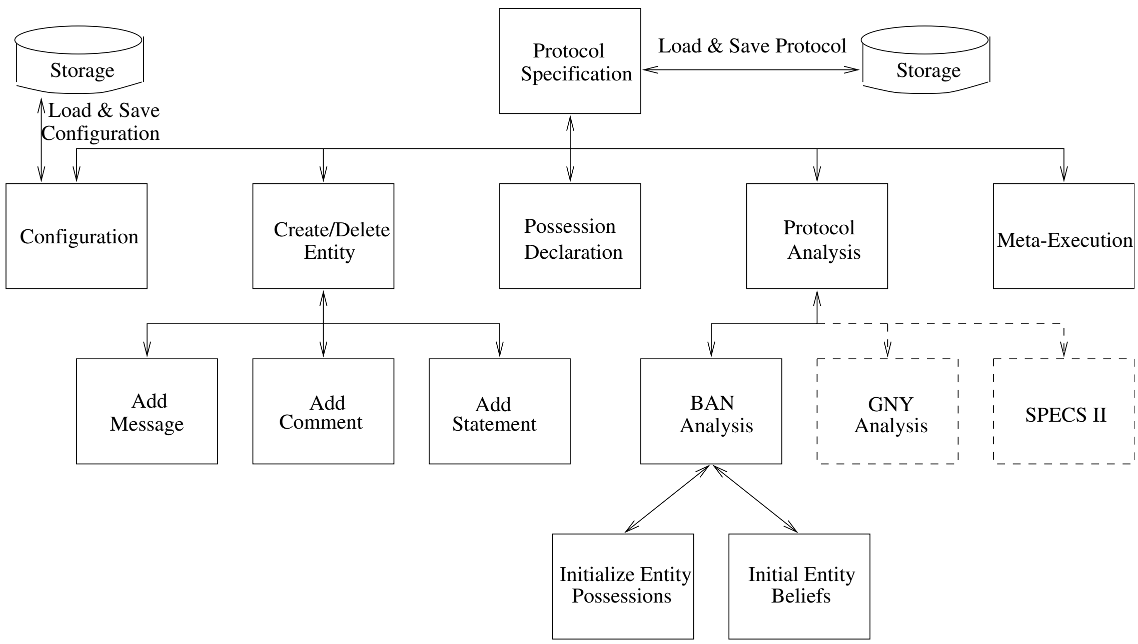

Figure 2: Protocol Specification

of the protocol whereas the required beliefs are those beliefs that should hold as true on completion of the communication exchange.

- Configure the settings of the SPEAR interface and the protocol.

Following these steps one can perform *meta-execution*, *BAN analysis* and *performance analysis* on the protocol. In Figure 2 GNY analysis and SPECS II are shown as dotted rectangles to indicate future links.

4.3 Security Analysis

In terms of security analysis, SPEAR draws on modal logic – presently BAN – to reason about the initial, intermediate and final belief and possession sets of principals.

The security of a protocol refers to the secureness of the actual message interchange, the applicability of the respective cryptographic methods, the type and quantity of data sent across insecure channels and the possible attacks that the protocol may be susceptible to. The security analysis module encompasses these aspects and provides automation support. SPEAR provides an automatic means of checking many of the security aspects of a protocol, for example using the previously mentioned logics. The main aim of this is *to provide an increased degree of confidence in the protocol and the knowledge that certain types of attacks are not possible, and certain flaws are not exhibited.*

4.4 Code Generation

Generation of fully functional, portable source code which can be used as actual client and server software is another goal of the SPEAR tool. This removes the networking programming and most, if not all, of the protocol coding. The output is in the form of Java source code classes which can be incorporated into applications that make use of the designed protocol.

4.5 Meta-Execution and Performance Evaluation

The SPEAR tool makes a test-bed available to the designer where errors can be detected without having to comprise the security of a real system. One of the features here is execution of the code generated by the code-generation module. This code is augmented with hooks to the SPEAR tool, in such a way that execution can be done in a controlled environment and monitored for performance evaluation.

A longer term aim is also to simulate attacks at a meta-execution level to determine whether protocols are susceptible to certain types of attack (e.g. parallel session or replay attacks).

5 Architectural Overview

In this section a more detailed overview of the architecture of SPEAR is presented.

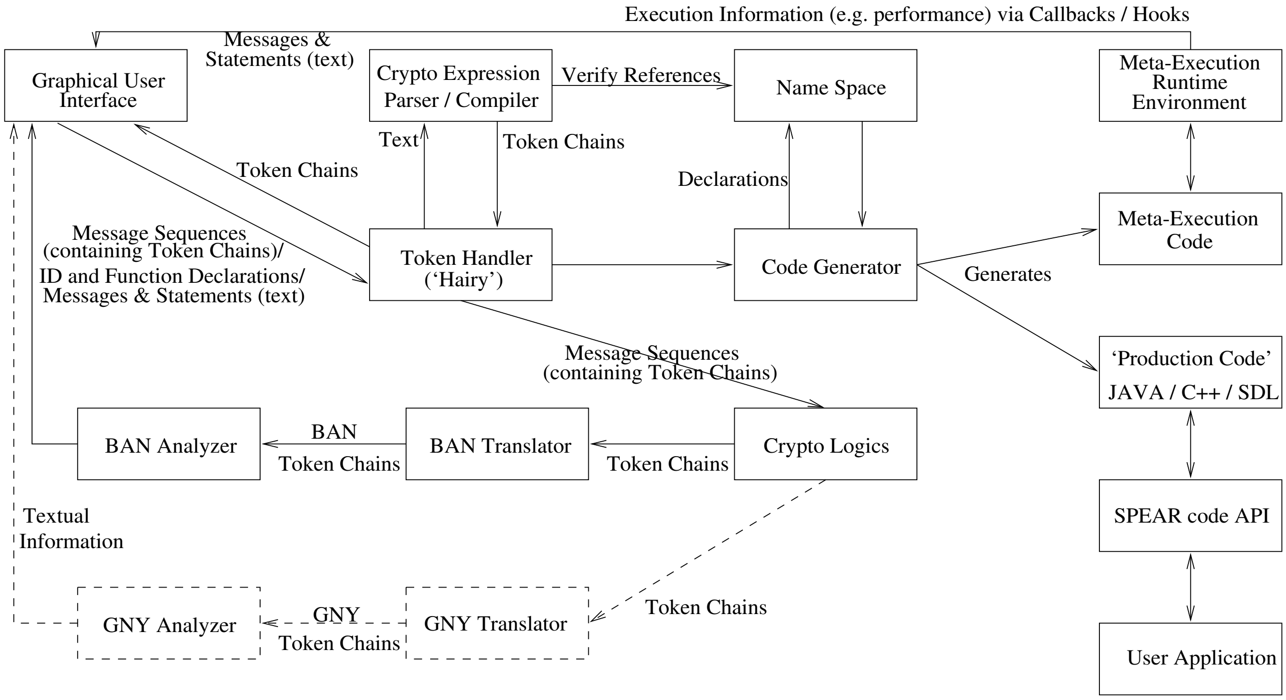

Figure 3: Architectural Overview

Messages are entered in the GUI via textual fields to form a protocol specification. As these messages are entered they are passed to the Crypto-Expression Parser / Compiler via the Token Handler. They are parsed by the Crypto-Expression Parser and compiled into a token format. Any syntactic or semantic errors are highlighted at this stage.

5.1 Token Format

The token format consists of a number of classes which are used to represent a parse tree. For example there is a *set* class, a *function call* class, and an *identifier* class. These classes act as 'containers' such that after linking their instantiations (forming '*chains*') will combine to form a parse tree.

For example:

RSA:Ka_public(nonce_a, MD5(msg), Kab) (denoting the encryption of a nonce, a, an MD5 hashed message and a session key, Kab, with a public RSA key called Ka_public) is represented internally in SPEAR as:

new FunctionCall("RSA", "Ka_public", new Set(new Id("nonce_a"),
new FunctionCall("MD5",null, new Id("msg")), new Id("Kab")))

Note that here "RSA", "Ka_public", "nonce_a", "MD5" and "msg" would have been defined in the Name Space earlier.

5.2 Token Handler Interactions

The Token Handler (nicknamed "Hairy") acts as the interface to all semantic action of SPEAR. Thus the GUI accesses the Crypto-Expression Parser, Code Generator, logic analyzer ('Crypto Logics'), Meta-Execution and Name Space modules via the Token Handler. The Token Handler module receives declarations of identifiers and functions which it forwards to the Crypto-Expression Parser that verifies names in the expressions with the Name Space module. The Name Space module contains defined identifiers and functions (these are specified via the GUI). The Token Handler also receives tokens from the GUI (which were generated by the Crypto-Expression Parser) and placed within the context of specific protocol entities and message sequences.

5.3 The Code Generator Module

Using the defined properties of items (such as entities, data types, full function references etc.) in the Name Space and message sequences, the Code Generator is able to generate the code necessary for message composition and transactions (sending and receiving). This output is currently Java source code, but in future versions will also generate SDL/PR and optionally C++. This is the "production code".

In order to use this "production code" in an application, the user would access the functionality of this code via the SPEAR code API which provides the cement for a clean interface and runtime support for the generated code. In order to do performance analysis, code is generated which contains hooks to the GUI for visual presentation of execution information (e.g. rounds and bounds [9]). This is the "meta-execution" code. The "meta-execution" code is executed within the context of the meta-execution runtime environment which provides the framework in a similar way as the user environment for the "production code".

5.4 The Crypto Logic Module

The Token Handler calls the Crypto Logics module and supplies the relevant entity definitions and message sequences (defined using the aforementioned token chains). These token chains are translated into a context usable for cryptographic logics such as BAN or GNY. In this release of SPEAR, BAN is implemented, but GNY is planned as a future extension. For the inclusion of GNY a more elaborate translation module is required since GNY has additional rules which are also more complex than those for BAN. The BAN analyzer uses the rules of the BAN logic to derive properties such as possessions and beliefs in the protocol. This information is fed back to the GUI in textual form for presentation to the user.

6 Protocol Specification Using SPEAR

In this section we show how protocol specification is conducted when using SPEAR.

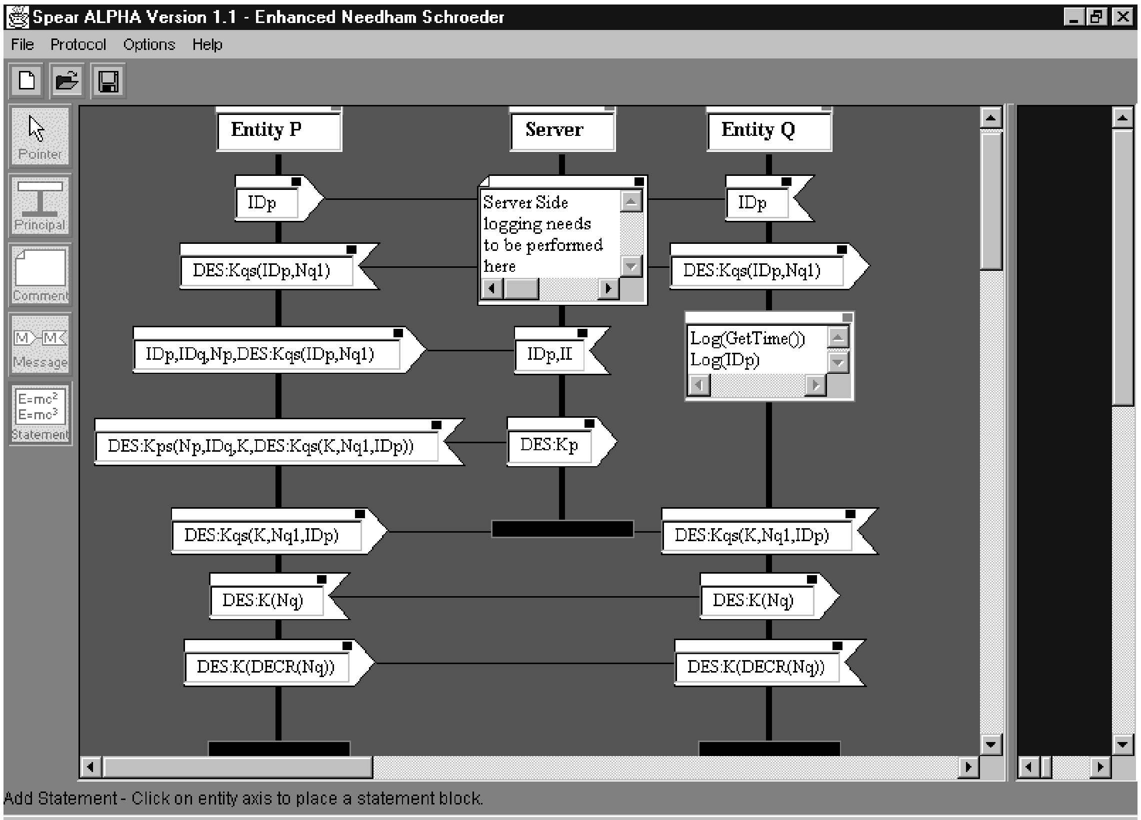

Figure 4: Enhanced Needham–Schroeder Protocol on SPEAR

Figure 4 shows the SPEAR tool and reveals its major components by depicting a version of the Enhanced Needham-Schroeder protocol, as it would be defined in SPEAR.

The toolbar on the left hand side of the interface shows the protocol elements that can be added to the protocol canvas – the central section.

These elements include:

- **entities**, which are the communicating parties (or security principals)
- **comments**, which are merely user notes and have no functional role (they can be attached to entities or float on the canvas)
- **messages**, which are the actual content sent to other communicating entities. Where the right hand side is 'convex' this indicates a message *sent*, while 'concave' indicates a message received. Finally provision is also made for
- **statements**, which are general commands that can be issued (for example to call user-defined functions, assign values to data items or perform other general commands)

By clicking on the axis (black vertical line) of an entity, protocol elements can be placed on them. In the case of messages, a line is extended from the sender to the mouse pointer which, after clicking on the axis of another entity, results in a line being placed to that entity as receiver. Currently messages can only be sent to single receivers, but support for multiple receivers (as envisaged in, for example, [13]) is a future goal.

The menubar consists of File, Protocol, Options and Help menus. The File menu is concerned with standard file operations while the Protocol menu contains options to modify the protocol as a whole. This includes setting the protocol name, declaring the possessions available to the entities participating in the protocol, declaring the functions that are called to initialize these possessions and general layout functions. The Options menu has elements to configure SPEAR. The Help menu contains help and information regarding SPEAR. The toolbar on the top provides easy access to menu options.

The empty area on the right of the protocol canvas contains information windows which give information about the BAN analysis of the protocol, current status of the meta-execution of the protocol and performance related information. It is predominantly used to convey transient information about the protocol or current status of the protocol to the user.

6.1 Declaring Possessions

Before sending messages one needs to declare the possessions that are used in the protocol, and the user functions which will be called. Possessions are then initialized at an entity level – with each entity using the defined function calls to provide the possessions with initial values. The values of possessions within an entity can also be initialized by receiving messages. For instance a server entity could send the address (IDa) of an authentication entity to a client. The possession IDa would then be defined for the client after receiving the message. Thus although the possessions are declared at protocol level, they are given values at an entity level. In addition the function calls are defined at a protocol level but need not be called (or even defined in every entity).

The Possession Declaration dialog box (Figure 5) is used to declare *all* possessions available to entities during the run of the protocol. Since this operation is not specific to any entity using these possessions, the dialog is invoked from the Protocols menu option. Each entity then initializes the possessions that it wishes to use as described.

There are currently six types of possessions:

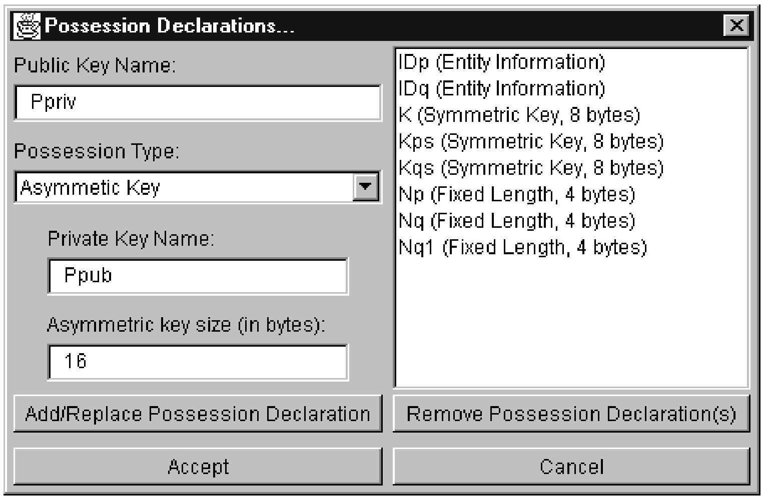

Figure 5: Possession Declaration Dialog Box

- Asymmetric Keys
- Symmetric Keys
- Fixed Length data (an array of bytes)
- Variable Length data (size not constant e.g. mail message contents)
- Delimited data (data separated with a given delimiter e.g. SMTP messages) and
- Entity Information data (contains information about the current entity that is being communicated with e.g. IP address, port number, etc.).

All sizes of data items are specified in bytes.

6.2 Building Expressions

The Expression Builder dialog (Figure 6) is used to construct messages that are sent across the network. The currently defined possessions are listed on the left and the defined functions are listed on the right. Double-clicking on the left list will replace the current selection with the data item (possession) selected, while double-clicking on the function list will apply the function to the selected area as the argument to that function. Once the message has been constructed one can check the semantic and syntactic correctness of the message. This will highlight incorrectly used functions, non-existent functions and other errors.

7 Evaluation

To demonstrate the usefulness of the SPEAR tool, a number of security protocols commonly known and used in industry are being extensively dealt with. In addition to evaluation using

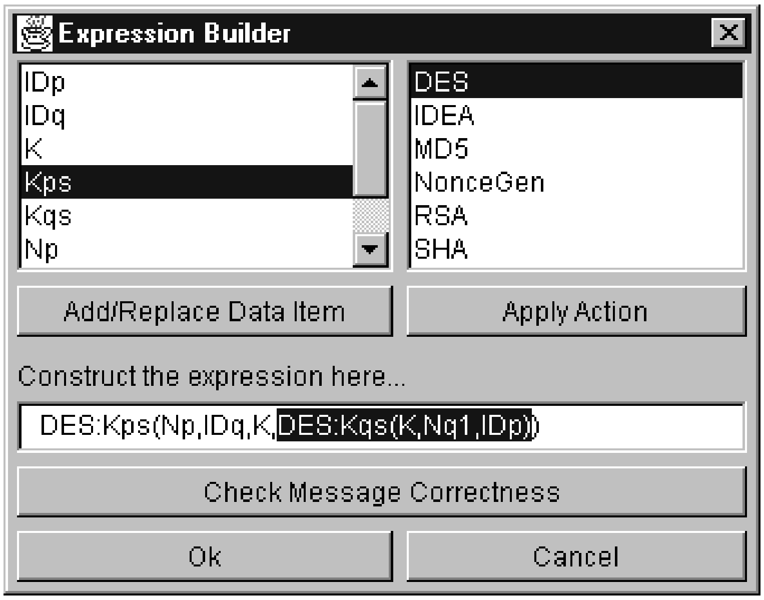

Figure 6: Message Builder Dialog

protocols such as the Enhanced Needham-Schroeder Protocol [17] (as shown in Figure 4), Kerberos and X.509, we will learn from experience gained through designing our own security protocols.

Another project currently underway in our laboratory is development of *micropayment protocols* and the intention with that project is that once protocol design begins this work will be done on SPEAR. This will provide us with a valuable case study from which to evaluate the utility of our design tool based on the experience of objective users.

8 Conclusion

Cryptographic protocols are crucial to the increasing use of computer networks for commercial and private communication. For this reason it is imperative that during their design such protocols are systematically scrutinized and refined using the most appropriate techniques. SPEAR is a design environment which supports protocol specification in such a way that *security analysis*, *code generation* and *meta-execution* of a cryptographic protocol can be conducted. SPEAR is intended to fulfill a useful and important function as an *integrator* and *enabler* in the discipline of security protocol design.

The SPEAR prototype can be downloaded from: **http://www.cs.uct.ac.za/Research/DNA/SPEAR**.

9 Acknowledgements

This work has been supported by the University of Cape Town Research Committee, the South African Foundation for Research Development and Telkom South Africa.

References

[1] R. Safavi-Naini A.M. Mathuria and P.R. Nickolas. On the Automation of GNY Logic. *Australian Computer Science Communications*, 17(1):370–379, December 1995.

[2] Ferenc Belina and Dieter Hogrefe. The CCITT-Specification and Description Language SDL. In *Computer Networks and ISDN Systems 16*, pages 311–341. 1988/89.

[3] S.H. Brackin. A HOL extension of GNY for automatically analyzing cryptographic protocols. In *Proceedings of Ninth IEEE Computer Security Foundations Workshop*, Co. Kerry, Ireland, 1996.

[4] M. Burrows, M. Abadi, and R.M. Needham. A Logic of Authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, February 1990.

[5] M Butow, M Mestern, C Schapiro, and PS Kritzinger. Performance modelling with the formal specification language SDL. In *Proceedings of the FORTE/PSTV'96 Conference on Formal Description Techniques*, Kaiserslautern, Germany, 1996.

[6] CCITT. *Recommendation Z.100: Specification and Description Language SDL, Blue Book, Vol X.1-X.5*. ITU General Secretariat, Geneva, 1988.

[7] M. Diaz, J.P. Ansart, P. Azema, and V. Chari. *The Formal Description Technique Estelle*. North Holland, 1989.

[8] Ove Faergemand and Anders Olsen. Introduction to SDL-92. In *Computer Networks and ISDN Systems 26*, pages 1143–1167. 1994.

[9] L. Gong. Lower Bounds on Messages and Rounds for Network Authentication Protocols. *Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia*, pages 176–183, November 1993.

[10] L. Gong, R. Needham, and R. Yahalom. Reasoning About Belief in Cryptographic Protocols. *Proceedings of the IEEE 1990 Symposium on Security and Privacy, Oakland, California*, pages 234–248, May 1990.

[11] C Henning, , C Vermeulen, J Zurcher, N de Villiers, C Schapiro, and PS Kritzinger. SDL performance evaluation of concurrent systems: A CASE tool for the development of communicating concurrent systems. In *Proceedings of Teletraffic '96*, Durban, South Africa, 1996.

[12] D. Hogrefe. *Estelle, LOTOS and SDL*. Springer-Verlag, 1989.

[13] A. Hutchison and K. Bauknecht. Cryptographic Key Distribution and Authentication Protocols for Secure Group Communication. In *Proceedings of the Twelfth International Information Security Conference (IFIP/SEC '96), Samos, Greece*, May 1996.

[14] ISO/IEC ISO9074. Estelle, A formal Description Technique Based on an Extended State Transition Model, 1989.

[15] ITU. *ITU-TS Recommendation Z.120: Message Sequence Chart (MSC)*. ITU-TS, Geneva, 1996.

[16] J.K. Millen, S.C. Clark, and S.B. Freedman. The Interrogator: Protocol Security Analysis. *IEEE Transactions on Software Engineering*, SE-13(2), 1987.

[17] R.M. Needham and M.D. Schroeder. Authentication Revisited. *ACM SIGOPS Operating Systems Review*, 21(1):7, 1987.

[18] R. Kremmerer and C. Meadows and J. Millen. Three Systems for Cryptographic Protocol Analysis. *Journal of Crytology*, 7(2), 1994.

[19] Verilog. *GEODE Reference Manual*. Verilog, 1995.