A GENERIC GRAPHICAL SPECIFICATION ENVIRONMENT FOR SECURITY PROTOCOL MODELLING

Elton Saul
Data Network Architectures Laboratory
University of Cape Town, South Africa
esaul@cs.uct.ac.za

Andrew Hutchison
Data Network Architectures Laboratory
University of Cape Town, South Africa
hutch@cs.uct.ac.za

Abstract Designing and implementing security protocols is a difficult task. A graphical specification environment helps one to cope with this complexity by enabling the visualization of hierarchical message structures and providing suitable abstraction and encapsulation so that designers can retain a high-level perspective while also being free to hone in on the details of the design. The graphical interface framework described in this paper isolates the critical issues in a protocol design and presents the user with an appropriate level of detail. This is accomplished through the use of a high-level view of the message flow and a more detailed component view that shows the structure of each protocol message. Each view can be easily manipulated by using standard graphical interface mechanisms such as drag-and-drop and context specific pop-up menus. An added advantage of this interface is that it is possible to connect to analysis or code generation routines via a GGSE-API.

Keywords: Security Modelling, protocol engineering, CASE tools

1. INTRODUCTION

The design and engineering of security protocols is widely recognized to be a very challenging and difficult task since protocols that appear secure can contain subtle flaws and vulnerabilities which attackers can exploit [2, 1]. As a result of this fact, it is imperative that security CASE tools be developed and used effectively in order to facilitate the creation of more secure and reliable cryptographic protocols.

All CASE tools require an interface to specify the system to be designed, implemented and possibly analyzed. To facilitate the efficient, timely and accurate specification of a security protocol, a usable and expressive graphical interface is required. A graphical specification environment helps designers to cope with the complexity of modern security protocols by enabling the visualization of hierarchical message structures, such as encryptions and hashes, and providing suitable abstraction and encapsulation mechanisms so that designers can retain a high-level perspective on the working of the protocol.

This paper describes a Generic Graphical Specification Environment (GGSE) that was developed for use with security CASE tools. Section 2 gives guidelines for creating a specification environment. In Section 3 we describe the GGSE framework from a potential user's perspective, while Section 4 elaborates on its architectural principles. We conclude in Section 5.

2. DESIGN INTERFACE GUIDELINES

Any tool that is concerned with designing and engineering cryptographic protocols should provide a design module that facilitates the rapid and accurate specification of a protocol, but at the same time is flexible enough to accommodate new types of protocols and security methods.

To define a security protocol, the following two steps are necessary:

1 The *principals* that are involved in the protocol must be declared.

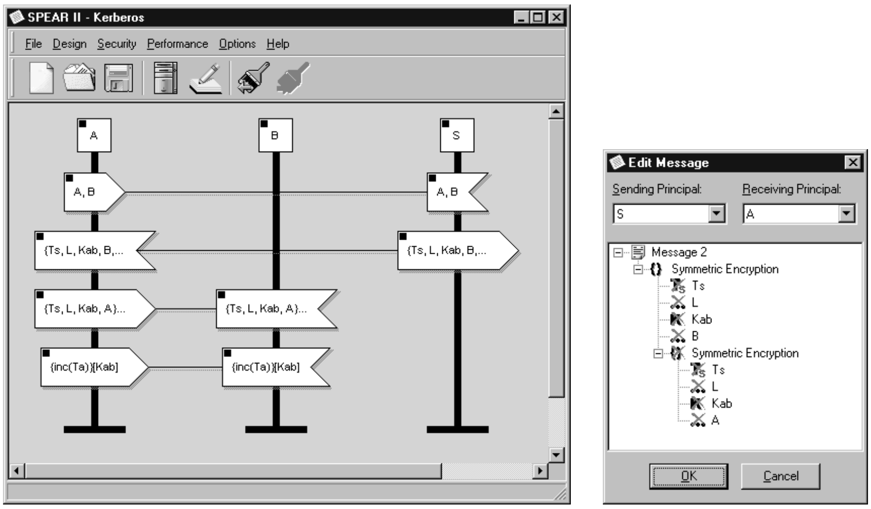
2 The *message passing specification* must be clearly represented and must also accurately describe the working of the protocol.

Secondary to these requirements, the underlying structure of components within each message could be clearly defined to enable source code generation or analyses. External functions that are to be applied to the message components may also be defined. The linkage between these external functions and generated source code must be made explicit. Communications settings, such as transport protocols and instance timeouts, can also be specified, and lastly information specific to security or performance analysis may be declared. This could include details such as initial beliefs and possessions of principals and pre-recorded protocol timings.

Thus, the goal of a design module should be to provide an intuitive interface which distils the critical issues and presents a designer with an appropriate level of detail, thereby allowing for the rapid and accurate specification of security protocols. Essentially, this implies that the interface should be graphical in nature. Components of this GUI should be written to ensure that compulsory steps in the specification process are clear and simple to complete. The importance and relevance of information that is related to other engineering phases should be clearly indicated and required only when necessary.

3. THE GGS ENVIRONMENT

The GGS environment is divided into two views. The *high-level protocol view* shows the overall message flow, indicating clearly and concisely what the protocol messages are and the principals that send and receive them. The more detailed *component view* displays each message as a hierarchical tree and allows the properties of each individual item within the tree to be manipulated. Besides allowing a protocol to be designed in these two views, the specification environment also ensures that a protocol can be saved and exported to various formats, such as text or LaTeX.



Sending Principal: S

Receiving Principal: A

Message 2
Symmetric Encryption
Ts
L
Kab
B
Symmetric Encryption
Ts
L
Kab
A

Figure 1 An illustration of the high-level protocol view on the left and the component view on the right, as implemented in the SPEAR II tool.

3.1. HIGH-LEVEL PROTOCOL VIEW

The high-level protocol view can be considered as a front-end to the protocol design module. Its main purpose is to provide a suitable abstraction and encapsulation mechanism so that designers can retain a high-level perspective of the operation of the protocol. An illustration of an implementation of the high-level view is shown in Figure 1.

The Message Sequence Chart (MSC) [6] syntax forms the basis for the high-level protocol view that is used to represent the exchange of messages within a protocol. In the chosen MSC-type representation, communicating principals are specified as axes. Each message is represented by two message boxes, a convex box and a concave box. The sending and receiving principals are designated through the placement of these message boxes — the convex box indicating the sender, and the concave box indicating the recipient. Messages are ordered sequentially in time, with the earlier messages at the top of a principal axis, and the later messages near the bottom of the axis. Within each message a textual representation is displayed, showing the contents of the message.

The high-level view can be fully manipulated through drag-and-drop operations. A message can be reordered in time by either dragging it up or down along an axis. The sender and receiver can also be changed by dragging either the convex or the concave message box onto a new principal axis to signify the new sender or receiver respectively. The principal axes can also be repositioned to 'neaten' or simplify the appearance of the specification. Note that movement of the principal axes does not change the functioning of the protocol in any respect.

The attributes of a message or principal can be modified by using the message or principal pop-up menu which appears when clicking on the button in the respective graphical representation. Setting the attributes of a message will initiate the component view dialog so that the individual components within the message can be initialized. Using the relevant pop-up menu, the selected message or principal can also be deleted. When a principal is deleted, all the messages that it sends and receives are also erased. The ability to duplicate a message is provided by the message pop-up menu. The high-level view also provides an undo and redo feature to ensure that designers can recover from accidental message and principal moves, deletions and edits.

A problem inherent in this MSC-type representation is that if a message contains many components then the corresponding message box can span too far across or even off the visible canvas area. This problem was solved by allowing the designer to specify a maximum message box width so that any message box exceeding this size would be truncated. However, to allow the designer to identify a message, tooltips were added to the message boxes so that the entire message contents are displayed when hovering over the button embedded in each box.

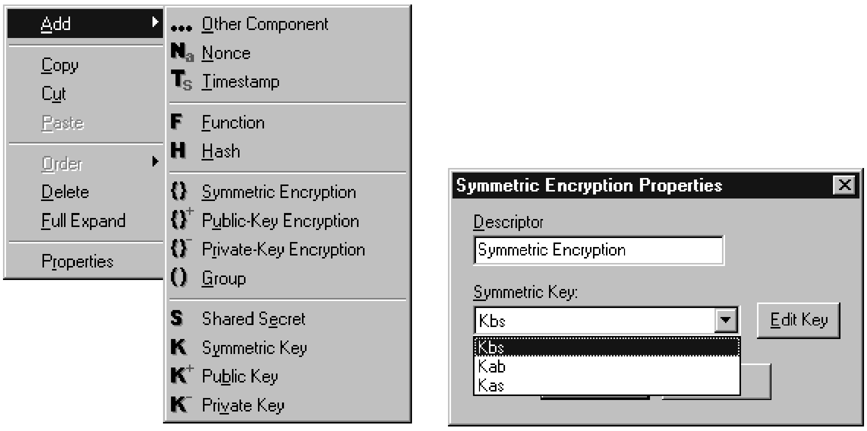
3.2. COMPONENT VIEW

The component view is a drag-and-drop environment that uses a hierarchical tree representation to show the relative structure of components in a message. A sample implementation of this view is illustrated in Figure 1. Each node in the tree represents a component and can either be empty or contain further components. There are thirteen primary components:

- *Non-terminal* components include functions, hashes, symmetric encryptions, public-key encryptions, private-key encryptions and groups.
- *Terminal* components include nonces, timestamps, shared secrets, symmetric keys, public keys, private keys and user-defined components.

Components can only be added to a non-terminal node. To add a component, the component view pop-up menu is used. A sample layout for this menu is illustrated in Figure 2. It is possible to cut, copy, paste and order nodes using either keyboard shortcuts, the pop-up menu or dragging operations with a pointer device. When cutting or copying a non-terminal node, all the nodes contained therein are also cut or copied, allowing entire subtrees to be moved.

Copies of components can be made across messages due to a shared clipboard that holds the copied items. However, when making copies



Descriptor: Symmetric Encryption

Symmetric Key: Kbs

Figure 2 An illustration of a pop-up menu that allows components to be added and modified on the left and a likely properties dialog for a symmetric encryption on the right.

a reference is transferred, not an actual copy of the component. This means that only one instance of a given component will exist in memory, ensuring that consistency is maintained when a component appears in more than one message.

The component view pop-up menu is used to activate the properties dialog for a selected component. Each component type has its own distinct properties dialog that can be used to set its attributes — for example, a possible properties dialog for a symmetric encryption component is illustrated in Figure 2. The attributes for a component can include information specific for code generation, initialization data for further analysis methods, or a host of other records necessary to describe the component.

Certain components can contain other component types as one of their attributes — for example, an encryption must specify the key being used. In these cases, it is essential that these aggregated components be accessible in component properties dialogs. In the case of encryptions, all the keys that exist in the protocol specification are accessible from any given encryption properties dialog. This means that a key can be specified in one message, and then used for an encryption in another message. Essentially the guiding principle is that all components which can be aggregated must be available for inclusion in the properties dialog of the components that can contain them. The illustration in Figure 2 shows how all the symmetric keys within a protocol are accessible from the symmetric encryption properties dialog.

4. THE GGSE FRAMEWORK

Creating a GUI design interface is often a tedious and time-consuming development task. The GGSE provides an existing GUI environment in which a security protocol can be easily and intuitively specified. By leveraging off the GGSE, protocol modelling tools which require a GUI design environment will be much simpler to develop. Information that has been specified through the GGSE GUI can be retrieved by using the supplied GGSE-API. The GGSE framework also provides the ability to expand the functionality already provided by the design environment.

4.1. STORAGE STRUCTURES

Within the GGSE framework, five significant items are stored for later retrieval, manipulation and querying by design and analysis procedures. Interaction with the GGSE revolves around these data storage items. The storage classes, each of which contains appropriate attributes and methods, are listed below:

1 The role-players in the protocol.

2 The messages transmitted during a protocol session.

3 Hierarchical trees which store the components for each message.

4 Components which are subclassed for each cryptographic type and form the nodes in the hierarchical tree.

5 A controller which stores information related to the protocol rendering and also deals with user interaction.

The principals in the protocol are stored within a dynamic list structure. Information concerning messages and their order is stored in a similar construct. Principal objects do not aggregate the messages which they send and receive since this makes it difficult to determine ordering information without including some form of extra information. Instead, the list of messages is an independent and ordered structure and principals contain references to the messages which they originate and receive. A binary sibling-child tree is used to store the message components hierarchically. Most operations on this tree, such as walking, pruning and grafting, then break down to simple recursive functions.

4.2. BASELINE FUNCTIONALITY

Rendering the protocol to the canvas allows the user to interact with the model in memory. Whenever the model is drawn, the GGSE records where each principal and message was placed so that user interaction points can be accurately interpreted and responded to by either allowing the selected object to be dragged or a context-sensitive pop-up menu to be displayed.

Dragging-and-dropping operations are fundamental to the working of the GGSE. On the canvas, dragging-and-dropping is implemented by determining the principal or message to be moved and then drawing an XORed representation before finally determining whether the desired placement is valid and updating the model in memory. The manipulation and rendering of the component tree occurs in a standard tree-view, similar to those provided by the majority of GUI-based programming interfaces and class libraries. Thus, operations such as drag-and-drop, non-terminal node expansion and contraction, and tree rendering can all be performed by using the supplied tree-view API.

The ability to edit and manipulate the component tree is vital. However, this tree must be ‘attached’ to the GUI tree-view before any editing can occur. Thus, an important function which the GGSE provides is the ability to associate each item stored in the component tree with one of the nodes displayed in the tree-view. When the user manipulates the tree-view, the GGSE ensures that the component tree remains in synchronization. This ensures that pruning and grafting of the tree-view will be reflected in the organization of components within the hierarchical component tree.

Maintaining the tree model involves many standard tree operations, such as node or subtree additions, deletions and repositionings. The underlying API for the component tree model attempts to be as extensive as possible, ensuring that programmers can easily walk the tree, modify and shift nodes around and retrieve and present the information which they require.

As we have seen, components which can be aggregated within other components should be accessible from all the appropriate properties dialog boxes. To facilitate this objective, the GGSE provides functions to extract all the components of a given type from the component tree. These components are placed into a list, allowing their details to be retrieved through simple method calls. This retrieval ability could also be used to aid in future analysis methods that need to examine only specific types of components to draw conclusions.

4.3. EXPANDING FUNCTIONALITY

The functionality of any interface should be easily upgradeable so that further information can be provided to aid in other analysis techniques. To upgrade the GGSE so that a user can specify a protocol more extensively the following dialogs must be extended:

- The necessary component dialogs.
- The message properties dialog which is accessed in the component view by selecting the root tree node.
- The principal properties dialog.
- The protocol properties dialog which is accessible from a pull-down or pop-up menu.

Each of these dialogs would then update the relevant data structures when closed. Finally, API calls would then have to be added to the affected classes so that the new data can be extracted.

4.4. GGSE-API

The API that is provided to the 'outside world' allows a programmer to retrieve all the information specified in the high-level and component views. The following are some of the major API calls which are provided:

- Methods to return the list of principals and then query each of these principals.
- Procedures to obtain the list of messages along with methods to query each of these messages.
- Methods to return the component tree and then to query this tree and extract terminal and non-terminal nodes in a random-access fashion, or in the order in which they have been specified. The ability to obtain a flattened version of the component tree is also provided for display purposes.
- Routines to determine the type of each component in the tree and to obtain all the relevant information describing it.

Additional API calls provide for the ability to manipulate the principal and message lists, export protocol data to other formats and streams to provide for interoperability with other tools and, lastly, to customize display settings on the display canvas.

5. CONCLUSION

In order to facilitate the efficient, timely and accurate specification of a security protocol, a design interface is required that will distil the critical issues and present the user with an appropriate level of detail. This will allow protocol designers to concentrate on the issues at hand, instead of having to battle with a cryptic design environment.

The GUI framework presented in this paper achieves this goal by creating two distinct protocol views, each providing sufficient information to facilitate the rapid and efficient specification of a protocol. The *high-level protocol view* shows the overall message flow, while the more detailed *component view* clearly displays the structure of each message. Manipulating either of these views is achieved through simple graphical operations and facilities such as dragging and dropping and context specific pop-up menus. The design environment is also supported by full redo and undo functionality as well as file save, load and export facilities.

The GGSE which has been described in this paper has been implemented in the SPEAR II security CASE tool [5, 3]. At present we are investigating the integration of other security tools and techniques with the GGSE framework, using SPEAR II as a reference implementation. An exciting project that we envisage is allowing interoperability with the CAPSL specification language [4]. This would entail allowing a protocol defined in the GGSE to be exported to CAPSL and vise versa.

From our work with the SPEAR II project, we can conclude that a security interface such as the one which we have proposed in this paper is of tremendous benefit. The GGSE essentially provides a flexible and extensible way in which a security protocol designer can interact with a number of diverse and distinct engineering and analysis services, while at the same time providing each of these with the necessary information to run to completion. In the complex arena of security protocol design, an environment such as this is sorely needed.

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. In *Computer Science Today*, volume 100, pages 426–441. Springer-Verlag, 1995.

[3] J.P. Beckmann, 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. Brackin, C. Meadows, and J. Millen. CAPSL Interface for the NRL Protocol Analyzer. In *IEEE Symposium on Application-Specific Systems and Software Engineering Technology '99*, Dallas, March 1999.

[5] E. Saul and A.C.M. Hutchison. SPEAR II: The Security Protocol Engineering and Analysis Resource. In *Second Annual South African Telecommunications, Networks and Applications Conference*, pages 171 – 177, Durban, South Africa, September 1999.

[6] International Telecommunication Union, Geneva. *ITU-TS Recommendation Z.120: Message Sequence Chart (MSC)*, 1993.