An Environment to Facilitate the Teaching of GNY-Based Security Protocol Analysis Techniques

Elton Saul and Andrew Hutchison
Data Network Architectures Laboratory
University of Cape Town
South Africa
{esaul, hutch}@cs.uct.ac.za

Abstract: The development of cryptographic logics to analyze security protocols has provided one technique for ensuring the correctness of security protocols. However, it is commonly acknowledged that analysis using a modal logic such as GNY tends to be inaccessible and obscure for the uninitiated. In this paper we describe a graphical tree-based specification environment which operates in conjunction with a Prolog-based GNY analyzer. This environment can be used to easily construct GNY statements using dynamically-constructed contextualized pop-up menus. We will show how this environment helps to distance students and protocol engineers from the syntactical element of GNY analysis, allowing them to focus more on the associated semantics and distil the critical issues that arise during protocol analysis. By freeing individuals to focus on an analysis, instead of hampering them with the necessary syntax, we can ensure that the fundamental concepts and advantages related to GNY analysis are kept in mind and applied as well.

Key words: Security protocol analysis, GNY logic, cryptographic protocols, security education and practice

1. INTRODUCTION

Analysis methods for cryptographic protocols have predominantly focused on detecting information leakage, rather than determining whether a protocol attains its stated goals. However, security protocols often fall short

of achieving their intended objectives, usually for very subtle reasons. As a result of this fact, cryptographic logics have been developed to aid in determining whether protocols actually fulfil their intended goals. Using logics to analyze security protocols has a number of advantages:

- The use of logics forces protocol designers to explicitly state the security assumptions which they have made and will require after the protocol has executed.
- Reasoning with logics makes designers think about the use for which each component is intended, thus minimizing redundancy.
- Cryptographic logics can also be used to explicitly bind the evolution of beliefs in a protocol session to message contents, number of messages and message rounds, thus helping to determine the minimum number of messages required to achieve a given set of beliefs and possessions.

Analysis using logics was first popularized in 1989 by the BAN modal logic [1]. BAN and other logic systems have successfully been used to reveal flaws in protocols that were previously accepted as correct [3]. A popular successor of BAN is GNY [4].

However, GNY analyses can appear complicated to uninitiated or non-mathematically inclined individuals. While teaching students how to analyze protocols with GNY, we noticed that many of them balked or got bogged down in syntactic issues, instead of focusing on the actual analysis. This apprehension regarding the GNY syntax, as well as the size of the postulate set, often restrained individuals from effectively utilizing the logic to uncover protocol flaws.

One of the most well-established findings in memory research is that people can recognize material far more easily than they can recall it [6]. This fact has clearly been applied in the design of graphical user interfaces over the past decade. For example, many user interfaces now make use of an extensive range of menus containing text or iconic lists of operations, options, files and so on. Instead of having to recall a name or a particular combination of function keys to perform an operation, users only need to scan through a menu until they recognize the name or the icon representing the operation which is required.

So, if we make use of a graphical user interface with sufficient cognitive aids, we could possibly develop an environment which will facilitate the rapid and accurate construction of GNY statements. In fact, in this paper we will demonstrate that by making use of common GUI components such as tree-views, tabbed panes and pop-up menus, we can create an environment which helps users to structure, organize and create GNY statements with ease. We will also show how this environment helps to distance protocol

engineers from the syntactical element of GNY analysis, allowing them to focus more on the associated semantics and distil the critical issues that arise during protocol analysis.

Through the course of this paper we will describe an environment that has been developed to facilitate and enable high-quality GNY-based cryptographic protocol analysis. This environment is made up of a graphically-based GNY specification environment, known as Visual GNY, which works in conjunction with a Prolog-based GNY analyser and a graphical protocol design environment. During the development of this system, we exposed it to students who had previously worked with GNY and their response was over-whelmingly positive. Instead of forcing individuals to go through the tedium of manually applying postulates and remembering cryptic syntax and notation, the system that we have developed frees them from this task and helps them to distil the more critical issues and focus more on carrying out an analysis correctly and purposefully.

The remainder of this paper is organized as follows. In Section 2 we describe the framework in which this work is taking place, while Section 3 presents a brief overview of the GNY logic. Then, in Section 4 we present our approach for visualizing GNY statements, describing the environment that implements this approach in Section 5 along with a description of how it is used in Section 6. Finally, we present experimental results in Section 7 and conclude in Section 8.

2. SPEAR II FRAMEWORK

The Visual GNY environment which we will describe in this paper is a component of the SPEAR II analysis framework [7]. This framework aims to provide a unified graphically-based environment within which security protocols can be specified, analyzed and then implemented. At present, the framework consists of three primary components::

- A graphical security protocol specification environment named GYPSIE [8].
- A Prolog-based GNY analysis engine named GYNGER based on the analyzer outlined in [5].
- The Visual GNY environment that is used for constructing GNY statements necessary for a protocol analysis [9].

To conduct an analysis the protocol is first specified in GYPSIE. Once this specification phase is complete, the Visual GNY environment is invoked

and GNY statements are constructed that will be used in the protocol analysis. Upon specifying all of the GNY preconditions, the defined GNY statements are exported to GYNGER where they are used for an analysis. Once the analysis is complete, the results are sent back from GYNGER to the Visual GNY environment, where they are displayed.

3. GNY CONCEPTS AND NOTATION

In this section the basic notions underlying the GNY reasoning process will be introduced. We will first briefly describe the notion of a formula and formula extensions, after which we will describe how formulae and GNY operators can be combined to form statements. Finally, we present a brief description of how a GNY analysis is normally conducted.

3.1 Formulae

A *formula* in a protocol description is a name referring to a bit string which would have a particular value in a session. This is analogous to a variable identifier in a programming language. Principals often exchange formulae to express their current beliefs or transfer information. Beliefs are described by statements, introduced in the next section. Let C range over all statements and let X be a formula. Then $X \sim C$ is also a formula, more specifically, a formula with an *extension*. Statement C is the extension and is considered an integral part of the formula.

3.2 Statements

A basic *statement* reflects some property of a formula, typically reflecting a relation between a principal and a formula. Let P and Q range over principals. The following are statements:

$P \triangleleft X$: P *is told* a formula, X, possibly after performing some computation, such as decryption. Thus, the formula being told is itself, or some computable content thereof.

$P \triangleleft *X$: P is told a formula, X, which he is *not* the first to convey in the current session of the protocol, though he could have transmitted it in a previous session. *Also*, it is the first time that P receives X in the current session.

$P \ni X$: P *possesses* formula X. P is able to repeat this formula in future messages of the current session. At a particular stage of a session, P possesses all the formulae that he has been told, all the formulae he started the session with, and all the ones that he generated during the current

session. In addition, P possesses all the formulae that are computable from formulae he already possesses.

$P \mid\sim X$: P *once conveyed* formula X. X can be a formula explicitly exchanged or some computable content of a formula. Thus, a formula can also be exchanged implicitly.

$\#(X)$: Formula X is *fresh*. A principal should believe that a formula originated by another principal is fresh if it has been constructed after the occurrence of some fresh event. A principal believes anything he has originated to be fresh if he cannot have chosen the same formula for the same purpose before.

$\phi(X)$: Formula X is *recognizable*. A principal would believe X to be recognizable if he has certain expectations about the value or structure thereof. He may recognize a particular value, a particular structure or other forms of redundancy. In either case, he may not possess part or all of the formula.

$P \overset{S}{\leftrightarrow} Q$: S is a suitable *secret* for P and Q. These entities may use S to prove their identities to each other. They may also use it as, or derive from it, an encryption key K to communicate, denoted as $P \overset{K}{\leftrightarrow} Q$. This notation is symmetrical.

$\overset{K}{\mapsto} P$: $+K$ is a suitable *public key* for Q. The matching secret key is given by $-K$.

The only default assumption which we require is that S, K or $-K$ will never be discovered by any principal except the legitimate owners or principals which the owners trust. In the latter case, the trusted principals should never use S, K or $-K$ as a proof of identity or as an encryption key to communicate. Continuing, the following are also statements:

$P \propto X$: P is eligible to convey formula X. P holds the relevant possessions and beliefs. This notation is used to detect inconsistencies in the protocol description.

$P \dashv (X)$: P is *not the first* principal to originate formula X. This formula must first be generated and conveyed by another principal.

Statements are often associated with individual principals to specify their states. Let C range over statements. The following are also statements:

(C_1, C_2): The conjunction of two statements. Conjunctions represent sets and have properties such as associativity and commutativity.

$P \mid\equiv C$: P believes that statement C holds. $P \mid\equiv$ is considered an empty statement.

$P \mid\equiv Q \mid\Rightarrow C$: P believes that Q has *jurisdiction* over statement C. He believes that Q has authority on C and should be trusted in this respect.

$P \mid\equiv Q \mid\Rightarrow Q \mid\equiv *$: P believes that Q has *jurisdiction* over all his beliefs. P considers Q to be competent and honest.

3.3 Conducting an Analysis

An analysis with GNY is very similar to one carried out with BAN. However, one significant improvement of GNY over BAN is that it defines an abstract 'protocol parser' which helps to derive a form of the protocol more suitable for manipulation. The major steps carried out before analyzing a security protocol with GNY are enumerated below:

a) Any implicit information conveyed by a protocol formula that contains a secret is represented logically by the attachment of an extension to the formula.
b) A star is placed in front of all formulae containing secrets that the receiving principal is *not* the first to convey in the current session of the protocol. The star also indicates that it is the first time that the receiving principal receives the formula in the current session.
c) The initial belief and possession sets of each principal are constructed. The possession set consists of all formulae available to the principal, while the belief set includes the current beliefs of the principal.
d) The desired final possession and belief sets for each principal are specified based on the design goals of the protocol.

Once these steps have been performed, an analysis can proceed. Each analysis essentially consists of deriving a series of assertions, each assertion being obtained by the application of the GNY inference rules to the assertions already contained within the belief and possession sets of a principal. After each assertion is derived, it is added to either the belief or possession set of the relevant principal. Once the analysis is complete the belief and possession sets will contain the final state of each principal after the protocol has run to completion. This information can then be compared to the desired final conditions to determine whether the protocol has achieved its intended goals.

4. STRUCTURED TREES AND GNY STATEMENTS

The GNY representation technique which we will now present allows any GNY statement to be viewed as a structured tree. Each node within this tree will have an assigned type, and its position within the tree will be determined by that type. We will see that this approach leads to a clean, concise, consistent and uncluttered graphical representation.

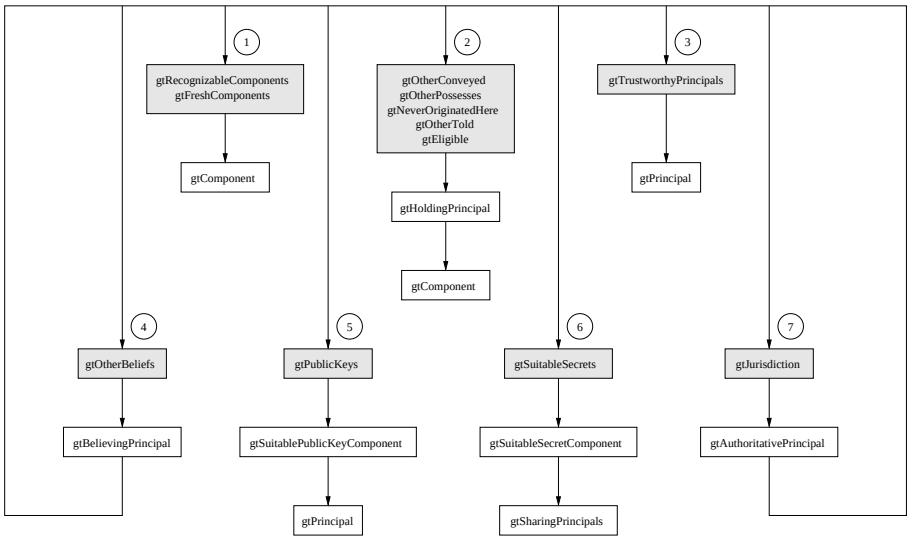

Figure 1. Relative hierarchy of tree nodes within our structured tree representation.

When using our structured tree approach, a set of GNY statements will be represented by of a collection of trees, each of the statements of a particular type belonging to the same tree. For example, a set of statements that only contains freshness, and conveyance statements would be represented by two trees, each containing only freshness, and conveyance statements respectively. Before we describe our structured tree approach any further, we need to partition all possible GNY statements into seven groups. All of the statements within a given group share the same structured tree representation.

- Group 1: $\#(X)$, $\phi(X)$
- Group 2: $P \mid\sim X$, $P \ni X$, $P \dashv (X)$, $P \lhd X$, $P \propto X$
- Group 3: $P \mid\Rightarrow P \mid\equiv *$
- Group 4: $P \mid\equiv C$
- Group 5: $\overset{+K}{\mapsto} P$
- Group 6: $P \overset{S}{\leftrightarrow} Q$
- Group 7: $Q \mid\Rightarrow C$

The illustration in Figure 1 lists the GNY types in each group and the manner in which nodes belonging to a given group are structured. For example, when representing a statement in Group 5, the root node must be of type *gtPublicKeys*, followed by a child node of type *gtSuitablePublicKeyComponent*, terminating with a child node of type

gtPrincipal. GNY types, prefaced by the letters *gt*, are used to identify tree nodes so that the correct captions, icons and pop-up menus will be displayed. Thus, if a node is of type *gtPrincipal*, then the principal's name and a principal icon will be used to represent the node in the tree-view. However, if a node is of type *gtFreshComponents*, then the text 'Fresh Components' will be used as the node caption and the icon for the freshness category will be displayed beside the text. A collection of graphical GNY statements is illustrated in Figure 2.

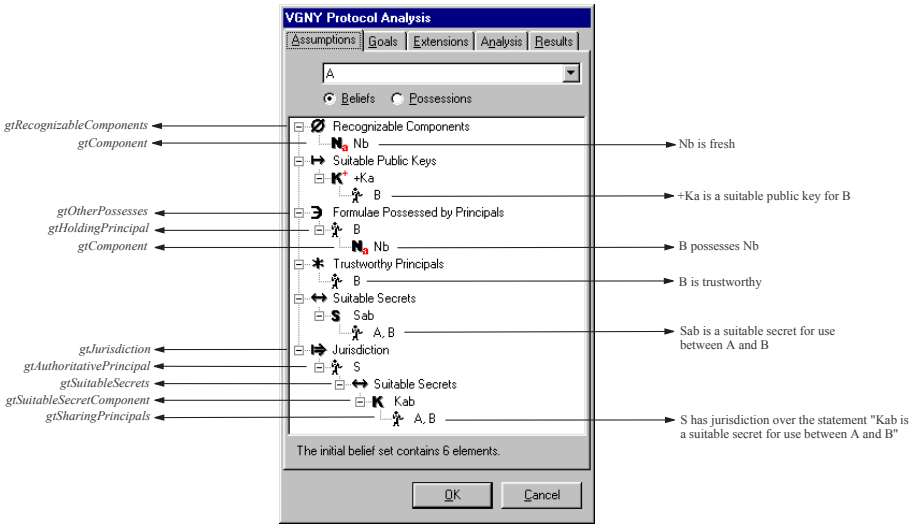

Figure 2. GNY statements specified in our structured tree-view.

The statements represented by the structured trees of Figure 2 form the initial belief set of principal A, as indicated by the tabbed pane, combo-box and radio button selections. Thus, every statement within these trees has the implicit prefix 'A believes that'. When using structured trees to store extensions, there is no implied prefix since an extension statement is merely an expression that must be believed to be true before a formula can be transmitted. An important point to note is that we don't require structured trees to specify the possession set for a given principal, since possession statements only have one operator and two operands, the left operand being the principal name and the right operand a formula. In fact, all that we require is a collection of one-node trees, each node having type *gtComponent*. Thus, when determining the GNY statement represented by a particular node in this collection of trees, we merely need to prefix the statement `A possesses' to the name of the formula represented by each tree node.

5. OVERVIEW OF VISUAL GNY

The structured tree approach which we have developed is implemented in the Visual GNY environment which is part of the SPEAR II framework. For each principal within a given protocol specification, up to four sets of structured trees are created, two for the storage of initial beliefs and possessions, and another two for the storage of target beliefs and possessions. A further four sets of trees can be used to store the successful beliefs, successful possessions, failed beliefs and failed possessions for each principal upon the completion of a successful GNY analysis. A set of structured trees is also created for every formula that has extensions, these extensions being defined in the Visual GNY environment.

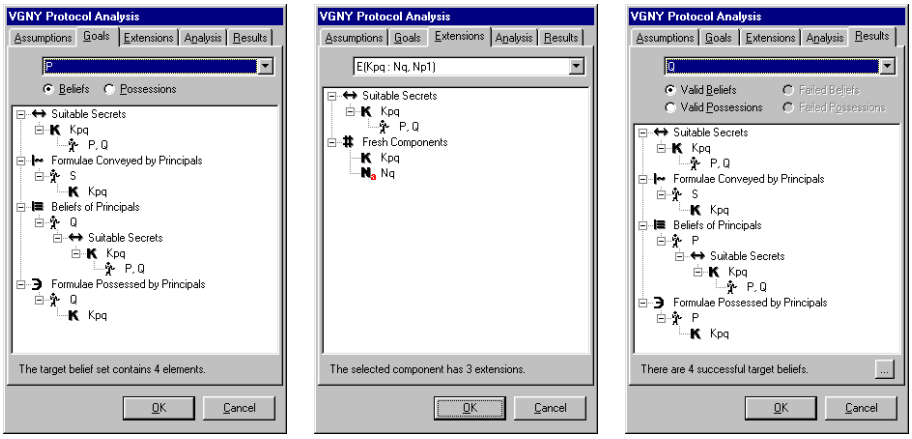

Figure 3. The Goals, Extensions and Results Panes from the Visual GNY Environment.

5.1 The Visual GNY Interface

The Visual GNY interface is composed of five tabbed panes. Within each of these tabbed panes, a drop-down combo-box and a selection of radio buttons are used to select the appropriate set of structured trees to modify or view. The currently selected set of structured trees is displayed in a tree-view component centered within the client area of the tabbed pane. Changing either the combo-box or radio button selection changes the set of structured trees being displayed in the tree-view. A label situated below the tree-view indicates the number of GNY statements represented by the set of structured trees displayed in the tree view. All interaction with the structured tree takes

place through pop-up menus that are dynamically constructed depending on the selected tree node.

The *Assumptions* tabbed pane, illustrated in Figure 3, is used to specify the initial belief and possession sets of principals involved in a protocol. The *Goals* tabbed pane, displayed in Figure 3, is structured in the same way as the *Assumptions* pane, but is used to store the target belief and possession sets of a given principal. In both the *Goals* and the *Assumptions* panes, the radio buttons are used to switch between the belief and possession sets, while the combo-box is used to select the believing or possessing principal. The *Extensions* tabbed pane, also shown in Figure 3, allows a user to specify extension statements that are attached to a formula specified in the protocol messages. This tabbed pane does not include any radio buttons, as only one set of structured trees is ever used to store the extension statements. The combo-box is used to select the formula to which a user wishes to append extensions.

The *Analysis* pane allows one to invoke a GNY analysis using the GYNGER protocol analyzer. Within this tabbed pane information such as the location of the Prolog interpreter, working directories, results files and the location of the GNY rules in Prolog format must be supplied. Finally, the *Results* tabbed pane, shown in Figure 3, displays the outcome of a GNY analysis. Radio buttons are used to switch between the valid beliefs, valid possessions, failed beliefs and failed possession sets for the principal selected by the drop-down combo-box. If one of these sets is missing, then the radio button for that set is disabled, thereby allowing one to obtain a quick indication of what occurred during the analysis. The proof for a valid goal can be obtained by right-clicking its structured tree representation, while all of the derived statements are obtained by clicking the button in the lower right corner of the *Results* pane.

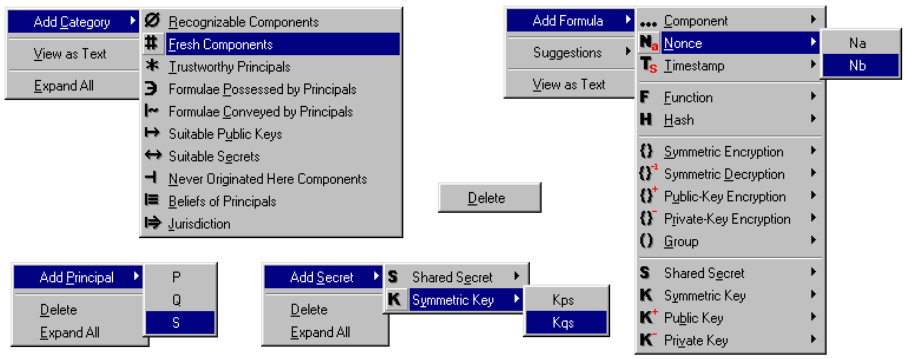

Figure 4. Some pop-up menus used in the Visual GNY environment.

5.2 Contextualized Pop-Up Menus

When using the Visual GNY interface, all that is required to specify a GNY statement is a pointing device, such as a mouse or trackball. The power of the Visual GNY interface stems from the fact that a user is 'guided' while constructing a GNY statement. This guidance is implemented through the use of pop-up menus. For example, to add the statement $P \mid\equiv Q \ni K_{pq}$ to principal P's target belief set, the user right-clicks in an open area of the tree-view representing P's target beliefs. A pop-up menu will then be displayed from which she can specify that she wants to add a `Formulae Possessed by Principals' belief category. A parent-less tree node of type *gtOtherPossesses* will then be inserted in the tree-view. When right-clicking on this node, the user will be presented with a pop-up menu that contains a list of principals. She selects principal Q from the list and a tree node of type *gtHoldingPrincipal* is added to the tree-view, the node of type *gtOtherPossesses* being its parent. Then, when right-clicking on this node, a list of formulae that have been specified in the GYPSIE environment are listed. Upon selecting K_{pq} from the list of shared keys, a tree node of type *gtComponent* is added as a child of the node of type *gtHoldingPrincipal*. At this point, right-clicking on the formula node only presents a *Delete* option. Also, once the formula node has been added, the statement counter at the bottom of the dialog is updated. A selection of the pop-up menus used to construct a structured tree representing a GNY statement are presented in Figure 4. When right-clicking on a tree node, the pop-up menu created to service that node will be displayed. This pop-up menu selection is based on the GNY type of the tree node.

An important point to note about the pop-up menus is that their content is updated dynamically. For example, assume we create a node with type *gtRecognizableComponents* that will serve as the root of a structured tree to store recognizability statements. Also, assume that there are two nonces defined in the protocol specification, namely N_a and N_b. When initially right-clicking on the *gtRecognizableComponents* node both nonces will be visible in the resultant pop-up menu. Assume that we click on N_a so that it is added as a child to the *gtRecognizableComponents* node. Now, the next time we right-click on the *gtRecognizableComponents* node only N_b will be displayed, since N_a has already been added to the *gtRecognizableComponents* node. Essentially, the principle used is that a formula, principal or belief category is only available for selection from a pop-up menu if it has not yet been added as a child of the node which was right-clicked. As a result of this fact, individuals cannot create duplicate GNY statements in a structured tree.

5.3 Enforcing Syntactic and Semantic Correctness

A significant advantage of the Visual GNY environment is that it ensures that syntactically correct statements are constructed. Because the pop-up menus enforce the predefined order of the nodes in the structured tree, it is not possible to specify a tree node that has an inappropriate type or to graft a node into an incorrect location. In fact, it is also not possible to specify an incomplete tree, since a user will not be allowed to change to another structured tree set, or press the *OK* button, unless all of the structured trees are complete. If a given tree is incomplete, then the node requiring a child will be highlighted and a message box indicating this fact will be displayed. The dynamic construction of the contextualized pop-up menus also ensures that no duplicate GNY statements can be generated. This elimination of duplication helps to ensure that an efficient and more optimal set of GNY statements are exported to a GNY protocol analyzer. Besides this, it also helps to eliminate confusion by ensuring that there is only one copy of any given statement.

Enforcing semantic correctness is a lot more difficult than ensuring syntactic compliance. Some semantic checking has been added into the Visual GNY environment, but it is not capable of eliminating every semantic error. Type correctness is enforced for the suitable secret and suitable public key statements by ensuring that a user is only able to use components of the appropriate type when constructing these expressions with pop-up menus. Because formula and principal names are imported from the protocol specification, all constructed GNY statements should refer to components that exist in the protocol specification. However, if a user imports a formula into a structured tree, and then removes all of its instances from the protocol specification, a bright red question mark is displayed as the structured tree node icon and the statement represented by the node is not considered as valid or exportable.

5.4 Exporting Visual GNY Statements

The ability to convert structured trees into a format which is compatible with an external GNY analysis tool or usable by a protocol engineer is fundamental to the operation of the Visual GNY environment. The structured trees defined within the Visual GNY interface can be exported to text, Latex and Prolog-style formats. The textual format displays each GNY statement in an English-style syntax, so that a statement such as $P \mid\equiv \#(N_a)$ is represented by the text string “A believes that Na is fresh”. When exporting to Latex, each of the structured trees is translated into native GNY mathematical notation. As can be imagined, this feature is exceptionally

useful for type-setting Latex documents which contain GNY statements. Finally, the Prolog-style output is directly compatible with GYNGER, allowing all of the GNY statements constructed in the visual interface to be used for automated analysis without any tedious manual translation. Along with the GNY statements defined in the Visual GNY environment, the list of protocol messages is also output using being-told statements. This list is created through interaction with the GYPSIE protocol specification environment wherein the messages, their receivers and relative order are defined. The GYPSIE API calls allow this protocol output to be generated with or without stars. In fact, the protocol parsing and appending of stars is all carried out by GYPSIE since it is easily automated. No user-interaction is required for protocol parsing, ensuring that this analysis phase is free from errors.

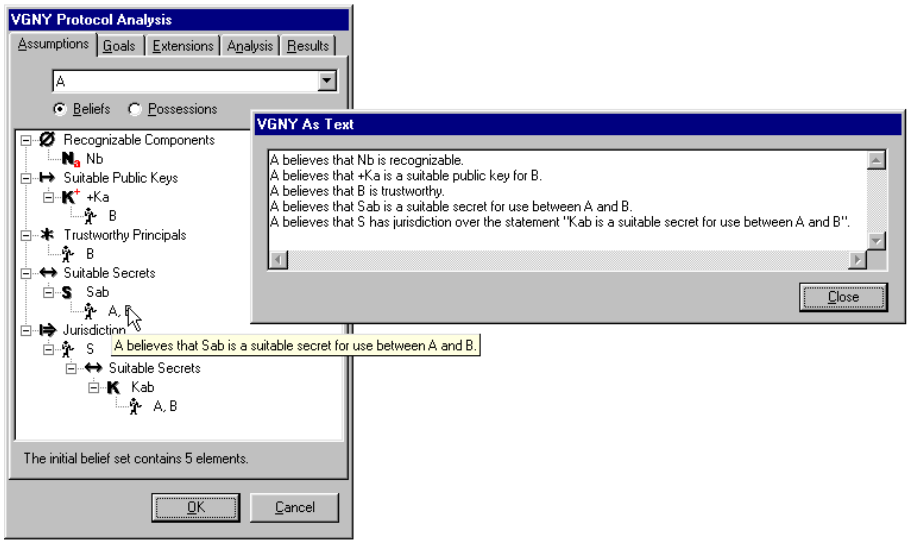

Figure 5. The 'View as Text' and Visual GNY tooltip in action.

5.5 Reducing Complexity

Within the Visual GNY environment, we have tried to create an environment that facilitates simple and straight-forward construction of GNY statements. Now, while working in one of the tree-views, a user might need to know the GNY statement which is represented by a specific set of nodes. To facilitate such a query, a feature which displays the GNY statement represented by a given node through the use of tooltips has been created. Thus, when hovering over a valid terminal node of a structured tree,

the GNY statement which this node represents is displayed, as illustrated in Figure 5. This feature prevents users from having to navigate a tree and derive the GNY statement which it represents. Tooltips also give an indication of what the implicit prefix for a given set of structured is. For example, the implicit prefix of the structured trees illustrated in Figure 5 is 'A believes that'. To view all of the GNY statements represented by the set of structured trees, a user can right-click in any open area and select the *View as Text* option from the resultant pop-up menu. A dialog containing an English-style list of GNY statements in an edit box is then displayed, as illustrated in Figure 5.

Another way in which the Visual GNY environment 'guides' a user is by helping her to structure and order the analysis process. The tabbed panes give an indication of the information required for an analysis, and are roughly laid out in the order that this information would be supplied. Belief and possession sets for a given principal are grouped into a single tabbed pane, only one being visible at a time through the selection of radio buttons. The statement counter at the bottom of a tabbed pane also helps to give an indication of the number of GNY statements already specified. Nodes within a given structured tree can be expanded or collapsed as required. If a node contains children then a clickable token is displayed to its left. Clicking on this token allows the node to be collapsed or expanded, thus allowing a user to control the amount of information which is presented. In this way the level of detail provided by the interface can be varied appropriately, allowing a user to control any possible disorientation to some degree. Also, because only one type of statement occurs in any given structured tree, the user does not have to 'search around' for similar statements, as is the case with some manual paper-based analyses. Presentation of the analysis results is also well laid out, allowing the user to see at a glance whether there are any valid or failed possessions and beliefs. This is accomplished through the use of enabled and disabled radio buttons in the *Results* pane. All of these results sets can be viewed by selecting the appropriate radio button, if it is enabled.

6. CONDUCTING A PROTOCOL ANALYSIS

In Figure 6 we sketch the steps that are undertaken during a typical analysis session. Such a session normally begins by specifying the principals, messages and formulae of the protocol in question within the GYPSIE specification environment. Once this phase has been completed, the Visual GNY environment is invoked and the initial assumptions and goals of each principal are specified as required. Extensions are also appended to

formulae. Once all of the necessary preconditions have been defined, details such as the location of the Prolog interpreter, the location of the GNY rules Prolog source, working directories and output files are defined within the *Analysis* tabbed pane. Upon the initiation of the analysis process, the structured GNY trees are all translated into a GYNGER-compatible Prolog syntax and then run through the analyzer. The Visual GNY environment monitors the analysis thread, and when it is complete, retrieves the results from the output files, parses these results, and then constructs the appropriate structured trees to display in the *Results* pane. Proofs and the list of all derived statements are also stored.

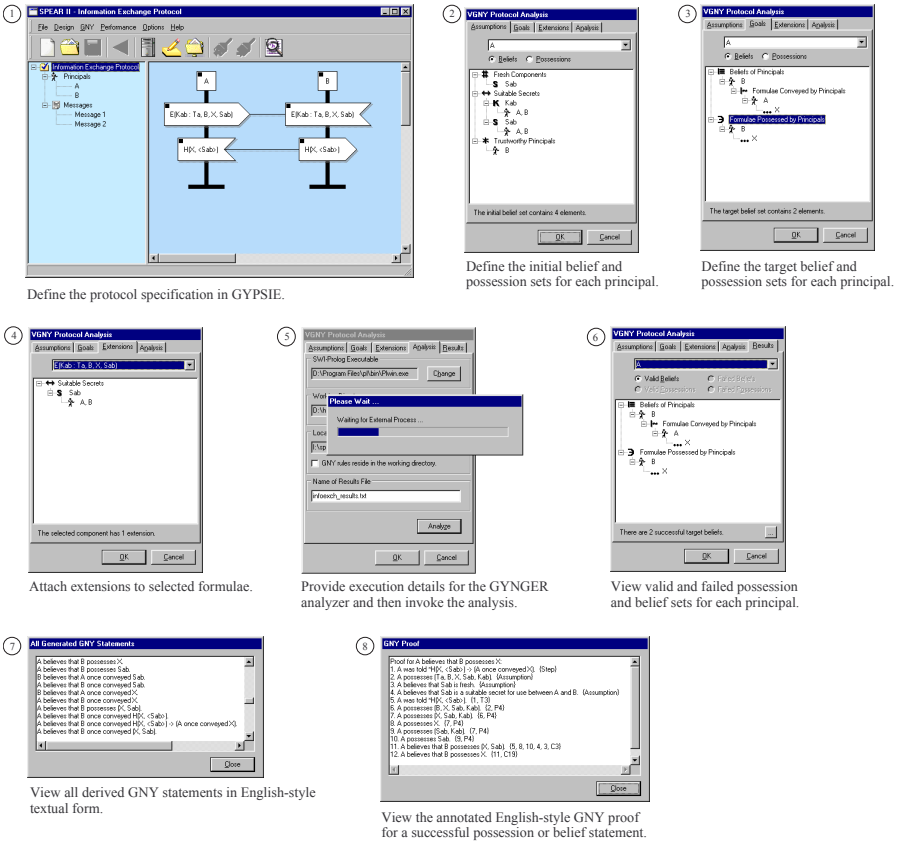

Figure 6. Steps undertaken when conducting a GNY protocol analysis.

7. EXPERIMENTS WITH VISUAL GNY

In order to examine the suitability of the Visual GNY environment for specifying GNY statements, we decided to conduct a number of user experiments. Within these experiments, we decided to pit the Visual GNY environment against manual, hand-written GNY statement construction. Our major objective was to determine whether Visual GNY facilitates the effective construction of syntactically and semantically correct GNY expressions. We also wanted to gain an understanding of how users who had never been schooled in GNY analysis techniques would be able to use the structured tree approach, as opposed to those who had completed some vestige of a course in security protocol analysis. The subjects involved in the experiments consisted of two groups of users, namely those educated in GNY analysis techniques and those who had never even heard of GNY. Each of the GNY novices had completed a course in network security principles. None of those who had been educated in GNY analysis techniques had carried out a GNY analysis for the last six months. In total, we involved fifteen educated users, and five novice users in the experiments. The number of novice users was rather small. However, this did not present a problem as we were more interested in how the educated users responded to the environment, since they are more representative of those who will use GNY analysis in the workplace or research arena due to their prior exposure. In effect, the novice users were merely tested for comparative reasons.

The experiment that we developed took the form of three tests. The first test required users to translate a given set of English-style GNY statements into both mathematical GNY notation and structured trees. During this test, individuals made use of Visual GNY to construct the trees. The second test required the translation of GNY statements in mathematical-style notation into English-style expressions. Finally, during the third test individuals had to convert structured trees into equivalent English-style GNY statements. In each test, we ensured that every type of GNY statement was exercised. Before testing the novices, we gave them a brief five minute crash course in GNY analysis techniques. Both groups of users were also briefly instructed in how to use the Visual GNY interface. The Visual GNY tooltips feature and the ability to view all of the structured trees as English-style text were both disabled.

Prior to conducting the experiments, we realized that certain individuals might not be able to recall or remember the meaning of a large portion of the GNY notation. Such a situation might totally bias the results in favour of Visual GNY. So, we decided to give the test subjects the option of using the Visual GNY pop-up menu captions as a reference for the mathematical-style GNY notation. Since the pop-up menus only provide descriptive text and a

mathematical-style GNY icon for each type of GNY statement, the examination of an individual's ability to recall the mathematical-style GNY syntax, construct coherent GNY statements and understand them semantically was not biased by allowing this type of referencing. In fact, when presented with the option of using Visual GNY as a reference, every one of the test subjects accepted, indicating their apprehension regarding the GNY notation. In this respect, we can say that the Visual GNY environment assisted in the construction and interpretation of the hand-written mathematical-style statements to some degree. The results for each of the tests appears in Table 1.

Table 1. Results of experiments pertaining to GNY statement construction.

	15 Educated Users		**5 Novice Users**	
	Sample Mean	**95% Confidence Interval for Population Mean**	**Sample Mean**	**95% Confidence Interval for Population Mean**
English to GNY	78.46%	(76.91%, 80.02%)	72.31%	(71.47%, 73.15%)
English to VGNY	98.46%	(98.15%, 98.77%)	100.00%	(100.00%, 100.00%)
GNY to English	87.22%	(86.36%, 88.08%)	85.00%	(84.75%, 85.25%)
VGNY to English	87.78%	(86.82%, 88.73%)	91.67%	(91.28%, 92.06%)

Listed within Table 1 are the average scores that were obtained by users from each group for the respective tests. From the data obtained, we also calculated the 95% confidence interval indicating where the population mean should lie. This computation assumes that the set of users are a representative sample of our envisaged user base. Since the sample size for our set of novice users consisted of only five individuals, the confidence intervals obtained are not as accurate as those of the educated users, which had a larger sample of fifteen individuals.

An important conclusion that we can derive from the first two tests is that Visual GNY effectively helps users to construct GNY statements. All of the statements that were specified with Visual GNY turned out to be syntactically correct. Those individuals who did not score perfect results for the English to Visual GNY test committed semantic errors, specifically the use of incorrect formulae within expressions. The reason why users always constructed syntactically correct structured trees was because the Visual GNY environment did not allow them to exit or change tabs unless all of the trees were complete. The fact that the novice users all got 100% of the Visual GNY statements correct and the educated users only got 98% correct is not very significant. It merely indicates that the novice users concentrated more closely on the formulae which they inserted into the structured tree. With a larger sample of novice users, we would have definitely encountered someone who would have made a substitution error. What's interesting to

note is that using Visual GNY produces significantly better results than specifying GNY statements by hand. When using Visual GNY individuals scored almost 20% higher. Essentially, what the Visual GNY environment has done is to totally remove the syntactical and notational issues associated with the construction of GNY statements, thus allowing individuals to concentrate on the actual protocol analysis process, which is far more fundamental and important.

The final two tests revealed some interesting results. The scores for reading off mathematical and structured tree-style GNY statements were almost identical for the educated users, and not significantly different in the case of novice users. This seems to indicate that the structured tree representation of a given GNY statement is not any more readable than its corresponding mathematical-style rendering. Difficulties encountered when reading from a structured tree can be attributed to having to jump from node to node, and sometimes having to skip nodes and only return to them later. The fact that mathematical-style GNY is primarily structured in a linear fashion means that it is not that difficult to interpret once the symbols have been understood. Since the novice users had never used GNY before, they did not have any preconceived notions as to how it should be written or structured. This fact might offer a possible explanation as to why they scored better than the educated users when reading from the tree. An interesting point to note is that the test subjects found it easier to read mathematical-style GNY statements than to construct them. This could be because construction of these statements requires recalling the function of each symbol and then stringing these symbols together correctly, while writing out the meaning of mathematical-style GNY statements merely requires one to have an idea of what each symbol represents.

During the course of this brief experimental analysis, we have noticed that many individuals struggle to recall the mathematical-style GNY notation if they have not been using it for some time. As a result of this fact, individuals will not immediately make use of GNY to analyze protocols, since their notational ineptitude would serve as a hindrance to the specification of assumptions and goals. Because of this issue, we chose to develop Visual GNY, empowering those who have used GNY in the past to use it again with ease. The experiments which we have carried out have confirmed that the construction of GNY statements in the Visual GNY environment is a straight-forward and painless operation, producing high-quality syntactically correct statements. However, reading GNY statements from the structured tree is not necessarily a simple task, sometimes confusing individuals. For this reason, the addition of the tooltips and *View as Text* features are exceptionally useful, since they help to create a system which virtually ensures that users construct error free statements — the pop-

up menus accelerating and aiding the construction process, and the tooltips and *View as Text* features being used to validate, verify and view the constructed statements.

8. CONCLUSION

Individuals who study network security techniques and then graduate to become security protocol engineers need to be familiarized with security protocol analysis techniques. However, we have to realize that to be useful, an analysis method must also be usable. To expect individuals to remember the syntax associated with a modal logic such as GNY or the plethora of inference rules used in an analysis is bordering on the nonsensical. Instead, the associated semantic issues and an understanding of how an analysis occurs should be emphasized and taught, tools and reference material being used for the rest.

There are a number of tools that can be used to carry out automated GNY protocol analysis [5, 2]. However, an impediment to using most of these is the construction of the specification which describes the protocol messages, formulae, initial beliefs and possessions, and target goals. Supplying this information is not always a simple and straight-forward task and its prompt, efficient and error-free delivery often depends on the type of software being used. For this reason, the use of software that helps to distance protocol engineers from the syntactical element of protocol analysis, allowing them to focus more on the underlying critical issues, should be encouraged and taught.

A formal analysis method should not just be studied and forgotten. Instead, the security community should be encouraged to develop tools that facilitate and encourage its use by a broad spectrum of individuals. When creating such tools, we should bear in mind that they should promote information recall, not require it. A tremendous amount of research has been carried out on security protocol analysis techniques [3], but how much of this research actually gets used in the field by the engineers who work there? Let's not allow good techniques to go unused. By encouraging more protocol analysis techniques to be applied, we will encourage the development of more robust and secure protocols.

Thus, by leveraging specially developed tools and techniques, a large portion of the difficulties that individuals encounter when using formal methods can be resolved. We have developed a visual environment within which GNY protocol analysis can be conducted. A number of experiments which we conducted on Masters and Honours level students indicates that

the approach which we have used in this environment helps to facilitate the construction of GNY statements, thus freeing individuals to focus more on the analysis and the issues related thereto, instead of having them bogged down in syntax and tedious inference rule application. We hope to continue development of the SPEAR II framework by adding more analysis techniques and ensuring that these techniques can be used by students and protocol engineers alike when implementing and learning about network security techniques.

9. REFERENCES

1. M. Abadi, M. Burrows and R. Needham. A Logic of Authentication. In *Proceedings of the Royal Society,* Series A, 426, 1871, pages 233—271, December 1989.
2. S.H. Brackin. Deciding Cryptographic Protocol Adequacy with HOL: The Implementation. In *The 1996 International Conference on Theorem Proving in Higher Order Logics*, pages 61—76, Turku, Finland, August 1996.
3. P. Georgiadis, S. Gritzalis and D. Spinellis. Security Protocols Over Open Networks and Distributed Systems: Formal Methods for Their Analysis, Design and Verification. *Computer Communications*, 22(8):695—707, May 1999.
4. L. Gong, Cryptographic Protocols for Distributed Systems, PhD thesis, University of Cambridge, April, 1990.
5. A. Mathuria, R. Safavi-Naini and P. Nickolas, On the Automation of GNY Logic. In *Proceedings of the 18th Australian Computer Science Conference*, volume 17, pages 370—379, Glenelg, South Australia, Fenruary, 1995.
6. J. Preece, Y. Rodgers, H. Sharp, D. Benyon, S. Holland and T. Carey. *Human-Computer Interaction.* Addison-Wesley, 1994.
7. 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.
8. E. Saul and A.C.M. Hutchison. A Generic Graphical Specification Environment for Security Protocol Modelling. In *Proceedings of the Sixth Annual Working Conference on Information Security*. pages 311—320, Beijing, China, August 2000. Kluwer Academic Publishers.
9. E. Saul and A.C.M. Hutchison. A Graphical Environment for the Facilitation of Logic-Based Security Protocol Analysis. *South African Computer Journal*. (26):196 —200, November 2000.