9 Automated Analysis of Cryptographic Protocols Based on Trusted Freshness

Abstract Logics can be systematically applied to reasoning about the working of protocols. However, the process of applying and reapplying the inference rules is often tedious and error-prone when carried out manually, while automation method will improve this problem. An automated logic-based analysis tool based on the freshness principle is introduced and developed, which uses the belief multiset formalism to analyze the security of cryptographic protocols.

The freshness principle presented in Chapter 4 has proved to be an efficient and easy idea in analyzing the security of cryptographic protocols from its capacity in distinguishing whether a message is fresh or not based on already trusted freshness identifier. The inherent appeal in using modal logic to instance the freshness principle stems from logic's simplicity and effectiveness for analyzing cryptographic protocols[1–4]. The modal logic of security analysis based on trusted freshness, the belief multiset formalism, has been presented in Chapter 7. Logic can be systematically applied to reasoning about the working of protocols, often helping to reveal missing assumptions, deficiencies or redundancies. This can then lead to the protocol, the assumptions or the original goals being re-evaluated, after which the inference rules can be reapplied to determining whether the goals are attainable after these modifications have been made. However, the process of applying and reapplying the inference rules is often tedious and error-prone when carried out manually. Another problem is that protocol has become so advanced and complex that we often cannot perform certain security analysis by hand, and we may accidentally miss conclusions drawn from inference rules, the freshness principle or the informal principles based on trusted freshness. Hence, specialized tool support for formal methods can significantly aid protocol engineers in creating and implementing cryptographic protocols which do not leak information, do achieve security goals and are immune to replay attacks[5].

In this chapter, we will give an automated logic-based analysis tool based on the freshness principle (see Chapter 4), and this Prolog-based analyzer uses the belief multiset formalism (see Chapter 7) to analyze protocols and

L. Dong et al., *Cryptographic Protocol*

it is developed mainly based on the concepts introduced during the SPEAR II Framework[5, 6].

9.1 Previously known methods for automated analysis

The cryptographic protocol automated analysis tools are useful for keeping track of whether or not a cryptographic protocol is secure or not.

9.1.1 Automated analysis tool based on logic

The symbolic manipulation correctness analysis approaches can greatly simplify the analysis of protocols, which consist of theoretic computer scientists in formal method area, and the security properties are expressed as a set of abstract symbols which can be manipulated, sometimes by a formal logic system, sometimes by an mechanical tool called a theorem prover, toward a YES/NO result[7].

SPEAR I, the Security Protocol Engineering and Analysis Resource[8], was developed by J.P. Bekmann, P. de Goede and A. Hutchison in 1997 to aid in the design and analysis of cryptographic protocols. The two primary goals of SPEAR I are to enable secure and efficient protocol design and to support the generation of protocol source code. SPEAR I offers developers of cryptographic protocols an environment in which security protocols are designed, analyzed and generated. Protocols are specified using a graphical user interface in the style of Event Trace diagrams and the security analysis based on the BAN logic[9] is facilitated.

SPEAR II, the Security Protocol Engineering and Analysis Resource II[6], is a protocol engineering tool built on the foundation of previous experience garnered during the SPEAR I project. The goal of the SPEAR II tool is to facilitate cryptographic protocol engineering and to aid users in distilling the critical issues during an engineering session by presenting them with an appropriate level of detail and guiding them as much as possible. The SPEAR II tool consists of four components that have been integrated into one consistent and unified graphical interface: a protocol specification environment (GYPSIE), a GNY statement construction interface (Visual GNY), a Prolog-based GNY analysis engine (GYNGER) and a message rounds calculator. GYNGER is a Prolog-based analyzer that performs automated analysis of protocols by using the GNY modal logic. Recall that the GNY logic could find the reflection replay attack (Suppose the message sender could recognize the message sent by itself), but it is hard for the GNY logic to find the transfer replay attacks and the direct replay attacks. The analysis engine employs a forward chaining approach to mechanize the tedious application of GNY in-

ference rules, allowing all derivable GNY statements to be generated quickly, accurately and efficiently. To conduct an analysis with GYNGER a protocol engineer needs to specify a protocol's messages, initial assumptions and target goals in a Prolog-style GNY syntax. The GNY rule set is then imported and employed in the analysis, after which a proof is generated in an English-style GNY syntax for every successful goal that was specified. Visual GNY functions as a user-friendly interface to the GYNGER analyzer. SPEAR II provides a graphically based protocol security analysis environment, and it can find the security flaws in a lot of authentication protocols. The idea of SPEAR II to implement GNY logic could be referenced for automation of protocol security analysis based on the logic – belief multiset formalism.

9.1.2 Automated analysis tool based on model checking

9.1.2.1 FDR tool

Failures Divergences Refinement (FDR)[10−12] is a model checker tool that is tailored to check CSP processes for refinement relations, where CSP stands for Communication Sequential Process[13]. CSP is particularly suitable for modeling and describing the behavior of concurrency and communication systems, and this feature has inspired some researchers for its using for formal analysis of authentication protocols. FDR models a complex system, such as an authentication protocol, into a (finite) state system and the properties of a state system can be expressed by some state satisfaction relations. FDR allows the refinement relation to be checked mechanically for finite state processes and it can be used to clarify whether or not certain properties of a complex system will be satisfied. In 1995, Lowe applied the FDR model checker and successfully uncovered a previously unknown error in the Needham-Schroeder Public-key Authentication Protocol, where this flaw has not been discovered for seventeen years since the publication of this protocol in 1978[11]. In FDR, a principal (it may be an attacker) in a protocol is considered as a concurrent CSP process, and a variety of attacks (such as eavesdropping, imitation or replaying) could be applied by an attacker process. The security of a protocol is modeled as the sequences of principal's events, and the FDR is used to check whether or not certain sequence of a principal's events is satisfied. The FDR model checker for CSP has achieved the success in analyzing the Needham-Schroeder Public-key Authentication Protocol.

9.1.2.2 NRL protocol analyzer

The NRL protocol analyzer[14] is a PROLOG-based protocol model-checking tool developed by Meadows, where "NRL" stands for Naval Research Laboratory of the United States of America. The NRL Protocol Ana-

lyzer is also based on the Dolev and Yao threat model of communications[15]. In Dolev-Yao model, an adversary could observe all message traffic over the network, intercept, read, modify or destroy messages, perform transformation operations on the intercepted messages (such as encryption or decryption, as long as he has in his possession of the correct keys), and send his messages to other principals by masquerading as some principals. Since an adversary's computational capability is polynomially bounded in the Dolev-Yao model, after an execution of the protocol, the adversary could not learn any information of the secret messages or cryptographic keys for which a protocol is meant to protect.

In Model-checking techniques the analysis of the behavior of a system usually involves a state space exploration to check whether or not certain properties will be satisfied. The main algorithm used in the NRL Protocol Analyzer settles a state reachability problem. It is well known that such algorithms are not guaranteed to terminate. Therefore a limit is placed on the number of recursive calls allowed for some of the checking routines. Using the tool seems to require quite a high level of user expertise in accurately coding the transition rules for a protocol and in specifying insecure state. The tool also has an inherent limitation on being particularly applicable to protocols for key establishment[7].

The NRL protocol analyzer has been used to analyze a number of authentication protocols and has successfully found or demonstrated known flaws in some of them. These protocols include the Needham-Schroeder Public-key Authentication Protocol[16] (for the analysis of this protocol Meadows provided a comparison between the analysis using the NRL protocol analyzer and Lowe's analysis using the model checker FDR in [11]), the Internet Key Exchange protocol (IKE, a reflection attack is found in the signature-based "Phase 2" exchange protocol)[17,18] and the Secure Electronic Transaction protocols (SET)[19].

9.1.2.3 The Murφ

Murφ[20] is a protocol verification tool that has been successfully applied to several industrial protocols, especially in the domains of multiprocessor cache coherence protocols and multiprocessor memory models. The Murφ language is a simple high-level language for describing nondeterministic finite-state machines. To use Murφ for verification, one has to model the protocol in the Murφ language and augment this model with a specification of the desired properties. The *state* of the model consists of the values of all global variables. The transition from one state to another is performed by *rules.* The desired properties of a protocol can be specified in Murφ by invariants, which are Boolean conditions that have to be true in every reachable state. The Murφ system automatically checks, by explicit state enumeration, if all reachable states of the model satisfy the given specification. Most Murφ models are nondeterministic since states typically allow execution of more than one ac-

tion. Murφ can only guarantee correctness of the down-scaled version of the protocol, but not correctness of the general protocol. If a state is reached in which some invariant is violated, Murφ prints an error trace – a sequence of states from the start state to the state exhibiting the problem. Murφ proves that there exist attacks on some known protocols such as Needham-Schroeder Public-key Authentication Protocol, TMN protocol and a simplified version of Kerberos V5[21–23].

9.1.2.4 Interrogator

The Interrogator[24] is a Prolog program developed by Jonathan Millen, Sidney Clark and Sheryl Freedman in 1985. Using the Interrogator, a protocol engineer can search for security vulnerabilities in network protocols for automatic cryptographic key distribution. Given a formal specification of a protocol, the Interrogator searches for message modification attacks that defeat the protocol objective and reveal secret information. The current version of the Interrogator assumes that the adversary is trying to learn private information, and the only way in which he can get that information is to read a message in which it is transmitted as a data item. A black-box view of the Interrogator is simple: for input it receives a protocol specification and a target data item; its output is a message history, consistent with the protocol specification, showing how the adversary could obtain the data item, if this is possible. The Interrogator and its associated graphical interface were implemented using LM-Prolog on an LISP machine. The user interface takes advantage of the windowing, graphics and mouse capabilities of the LISP machine. Within the Interrogator, protocols are modeled using a state-transition approach, principals being represented as communicating finite-state machines. This method allows a wider class of protocols to be supported and permits variations in message sequencing. The Interrogator interface has two main components: a preprocessor that converts textual protocol specifications into an internal Prolog form, and a display interface for graphical user interaction. To conduct an analysis, a protocol is specified in a textual format, edited with normal LISP machine facilities, parsed and loaded. The interactive graphical display is then used to establish penetration objectives. If a flaw is found, it is displayed in the form of a message sequence, showing messages before and after modification by an adversary. The Interrogator has been developed to the extent where it has succeeded in finding a multiple-modification penetration of the Needham-Schroeder protocol and some others with known vulnerabilities. Given a protocol specification and a target component to uncover, the Interrogator searches for a scenario involving adversary actions which reveal the target. The history of messages sent and modified is displayed, allowing the user to examine how the attack was carried out and evaluate it for feasibility and possible counter-measures.

9.1.3 Automated analysis tool based on theorem proving

SyMP[25] stands for "Symbolic Model Prover" and it is a general purpose prover generator for generating special purpose theorem provers in various application domains. SyMP is proposed by S. Berezin and A. Groce of Carnegie Mellon University. The core of the tool is a generic prover which is connected to several proof system modules. Each such module defines an input specification language, a proof system, and a rule application mechanism, and the generic prover provides all the proof management and an interactive user interface. Note that the SyMP prover does not have a built-in model checker

SyMP has two proof systems: the default proof system, and Athena. The default proof system implements a general framework for combining model checking and theorem proving, and has a hardware-oriented specification language. The main purpose of the language is to provide a convenient environment for fast and clean prototyping of new (mostly hardware) verification methodologies based on model checking with some elements of theorem proving. It can also be used as an intermediate representation in translation between other specification languages.

The Athena proof system is specialized in verifying security protocols, uses Strand Spaces[26] as the basis for the protocol representation and is based on the Athena[27] technique developed by D. Song. Each protocol in this framework is a set of roles, and each role is a sequence of actions where actions are separated by an optional semicolon. Two built-in actions, "send" and "receive", send and receive messages to and from the environment. An instance of a role with concrete parameters defines a strand, or a particular run of a particular principal in the protocol. Properties are specified in a propositional logic that specifies which strands must or must not appear in any protocol execution.

9.1.4 CAPSL specification language

CAPSL, the Common Authentication Protocol Specification Language[28], is a high-level language intended to support the analysis of cryptographic protocols using formal methods. The development of CAPSL started in 1996 and is being managed by Jonathan Millen. Its goal is to permit a protocol to be specified once in a form that is usable as an interface to any type of analysis tool or technique, given appropriate translation software. The CAPSL Intermediate Language (CIL) acts as an interface to analysis tools, allowing protocols specified in CAPSL to be examined by these tools. CIL is designed to make the translation to tool-specific representations as easy as possible. A CAPSL specification is parsed and translated into CIL, and at that point a different translator can convert from CIL to whatever form required for each

tool. The translator from CAPSL to CIL can deal with the universal aspects of input language processing, such as parsing, type checking, and unraveling a message-list protocol description into the underlying separate processes.

Messages in cryptographic authentication protocols are constructed using cryptographic operators and functions. In principle, all functions used in CAPSL, and the data types they operate on, must be specified axiomatically with abstract data type specifications, called typespecs. Several commonly used data types and operators are defined in a standard prelude. Type specifications in this prelude are considered built-in, and do not need to be supplied by a designer or imported explicitly. When a protocol is being analyzed or simulated, the analyst may have to specify which principals are to be run. Other run-specific information, such as the initial knowledge of the attacker, may also have to be supplied. A CAPSL environment contains specifications detailing this kind of information. Environment specifications, like type specifications, are separated from the definition of a protocol. The content and interpretation of an environment specification depend on the analysis tool. However, CAPSL does provide syntax, keywords and organization so that different tools can take advantage of the CAPSL parser. Declarations to name principals and other constants can be placed in an environment, and sessions can be defined. More than one session may be declared and these sessions will run concurrently by default.

9.2 Automated cryptographic protocol analysis based on trusted freshness

As we have seen, conducting an analysis manually is often tedious, error-prone and not very productive in the majority of situations. To facilitate the effective use and application of a logic system, it is needed to make the logic analysis automated and the user interface graphical.

In this section, we put forward a cryptographic protocol analysis analyzer based on the belief multiset formalism logic (for short, the BMF logic), the BMF analyzer, which is also a completely graphic-based environment. A benefit of the BMF analyzer is that it would make protocol analysis operations accessible to a wider range of individuals, since it would remove the requirement that protocol designers need to be experts in a large number of specialized engineering techniques.

9.2.1 Analyzer frame based on belief multiset formalism

The BMF analyzer consists of 5 parts: visual BMF, BMF analyze engine, BMF result view, BMF rule engine and BMF attack engine, and it involves

operations on the database of facts, goals, rules and attacks. The parts BMF analyze engine, BMF rule engine and BMF attack engine are the core of the BMF analyzer, while the parts visual BMF and BMF result view are the user-friendly interfaces to the BMF analyzer. The frame of the BMF analyzer illustrates in Fig. 9.1.

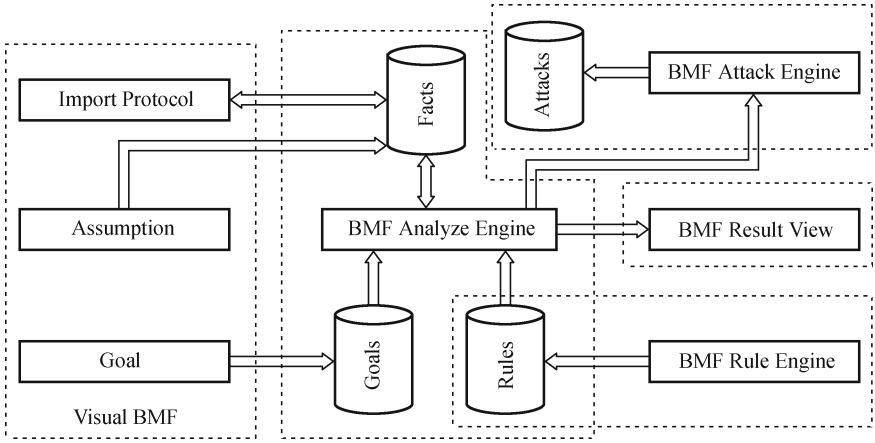

Fig. 9.1 The frame of the BMF analyzer.

1. Visual BMF

The visual BMF functions as a user-friendly interface to the BMF analyzer, and the input components include protocol messages, premises and goals. These components are constructed to BMF statements necessary for analysis via the visual BMF interface, and then passed on to BMF analyze engine. The protocol message import interface supports the input of the messages including the principals, nonce etc., supports the chosen of cryptographic schemes, and the construction of the BMF language component and the terms. From the view point of a participant in a protocol run, the terms owned by each principal are completely different. The premise import interface supports the input of the initial security assumptions of the public-key and private key in public-key case, the long-term key in the shared-key case, and also principal which has generated the freshness identifier used in the protocol messages. The goal import interface supports the security goal configuration of an input protocol, such as UA-secure, MA-secure, UK-secure MK-secure.

2. BMF analyze engine

The BMF analyze engine supports the formal presentations of terms, initial assumptions, security goals, inference rules, and it functions as a protocol security analyzer that generates all of the BMF beliefs and possessions that can result from the systematic application of the inference rules to a set of

initial assumptions and message steps. The BMF analyze engine mainly includes the database of premises, the database of goals, the database of rules, and the analyzing engine based on the trusted freshness.

3. BMF result view

The BMF result view subsystem supports the presentations of the security analysis result of a cryptographic protocol based on the belief multiset formalism.

4. BMF rule engine

The BMF rule engine is an interface of a new belief multiset formalism inference rule input. Based on the freshness principle, the belief multiset formalism inference rules could be extended to meet variety applications in the real world, such as new liveness rules, new confidential rules, new freshness rules, new association rules, also other new rules about non-repudiation, fairness etc. A new rule could be inputted into the BMF analyzer via a user-friendly interface, and then it is converted to a standard format rule with a series of operations via BMF analyze engine, and then it is passed on to the rules database.

5. BMF attack engine

Recall that the security analysis of a protocol based on the belief multiset formalism can either establish the correctness of the protocol when it is in fact correct, or identify the absence of the security properties and the structure to construct attacks based on the absence. The BMF attack engine supports the presentations of attacks in the belief multiset formalism, and supports the construction of attacks from the absence of the security properties.

9.2.2 Comparision of two initial implementations of BMF

The focus of the automation of a cryptographic protocol analyzer is to give a correct analysis result, and to remove as much of the complexity and tedium surrounding protocol analyzing as possible and to provide a user-friendly, effective and powerful environment that can be used by the researchers. The automation of the belief multiset formalism based on the freshness principle could be implemented with the object-oriented development language like C#, C++, Java, etc.; or with artificial intelligence language like Prolog. To determine which is more suitable for the implementation of the belief multiset formalism, under the prerequisite to ensure the correctness of the analysis results, we compare the efficiency, usability of the automation tools developed under the object-oriented development language C# and the Prolog-based implementation language. The initial implementations of these two development environments mainly focus on the visual BMF part, the BMF analyze engine part and the BMF result view part of the BMF analyzer.

BMF analyzer 1[29] is developed using the object-oriented development language C#. The corresponding classes based on the belief multiset formalism include the Principal Class, FreshnessComponent Class, Expectation Class, Fragment Class, Timestamp Class, BeliefMultisetsPresentation class, Key class (AsymmetricKey Class, SymmetricKey Class, SharedKey Class), Belief Class (PrincipalBelief Class, MarkBelief Class, KeyBelief Class, KeyKnown Class), as illustrated in Fig. 9.2.

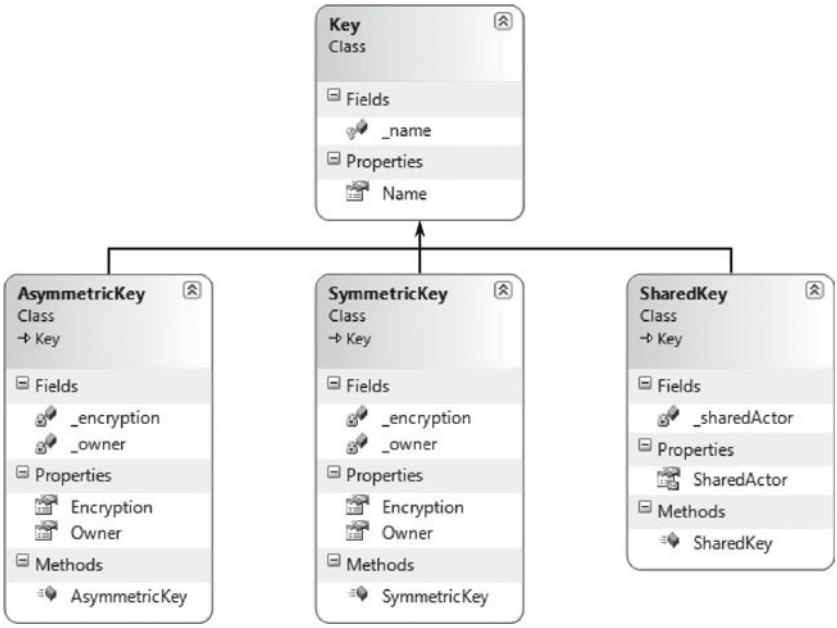

Fig. 9.2 Class illustrations of BMF analyzer 1.

The configuration and the analysis results, which are in the XML format, are stored in the database of facts and the database of goals, and are provided to the user with a completely graphic-based environment for message input and result show, as illustrated in Fig. 9.3.

The BMF analyze engine in the BMF analyzer 1 supports optimized rule search, rule class operations, rule search method triggered by new generated beliefs, remove of plenty of useless intermediate results. Applying the strategy model and abstract factory model to define the database of rules, the belief multiset formalism inference rules could be modeled, and the operations on the rules are independent of the custom applying the database of rules. E.g., for associate class (shared key subclass, shared key with TTP subclass, private key subclass), the operation of the class implements the inference function of rules.

The BMF analyzer 1 is implemented under the object-oriented development language C#. The implementation under the object-oriented develop-

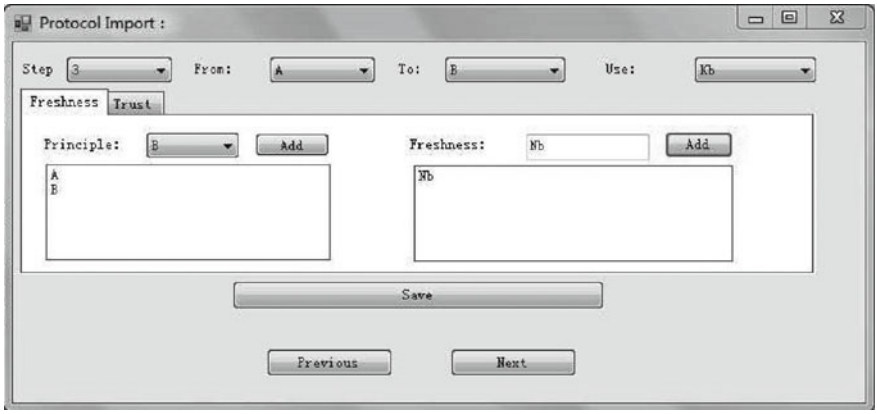

Fig. 9.3 User interface of BMF analyzer 1.

ment environment is model-based, hence it is clear and convenient to develop and debug this analyzer. However, it is difficult to add new classes and to extend the belief multiset formalism rules in the analyzer since they are embedded within the analyzer.

The BMF analyzer 2[30] is implemented based on the artificial intelligence language Prolog that is simple and reliant as an implementation language. The BMF analyzer 2 is similar to the GYNGER in the SPEAR II Framework[6]. The initial BMF analyzer 2 includes three parts: the BMF statement construction interface (visual BMF), a Prolog-based BMF analyze engine (BMFGER) and the result presentation interface (BMF result view). Similar to the GNY analysis engine GYNGER in SPEAR II, the Prolog-based analysis engine BMFGER relies on a forward-chaining inference engine to generate all of the BMF beliefs that can result from the systematic application of the inference rules to a set of initial assumptions and message steps. The visual BMF functions as a user-friendly interface for the input of the protocol, the initial assumptions and the goals. The BMF result view functions as a user-friendly interface for the output of the analysis results of a specific protocol. The initial assumptions goals and analysis results are presented as the belief multiset formalism statements in the Prolog-style, and the belief multiset formalism statements are represented using a tree-like approach. The messages and initial assumptions pertaining to the protocol to be analyzed are specified in the form of fact/3 predicates, and the target goals are specified in the form of goal/2 predicates. The BMF result view converts the analysis results in the Prolog-style into an English-style BMF syntax for every successful goal that was specified, as illustrated in Fig. 9.4.

Note that the rules, the protocol messages in the BMF analyzer 2 are all presented in the Prolog style, hence the database of rules could be stored independently from the BMF analyzer 2, so we could extend the belief multiset rules and add these rules to the BMF analyze engine conveniently in the future, even by a text editor.

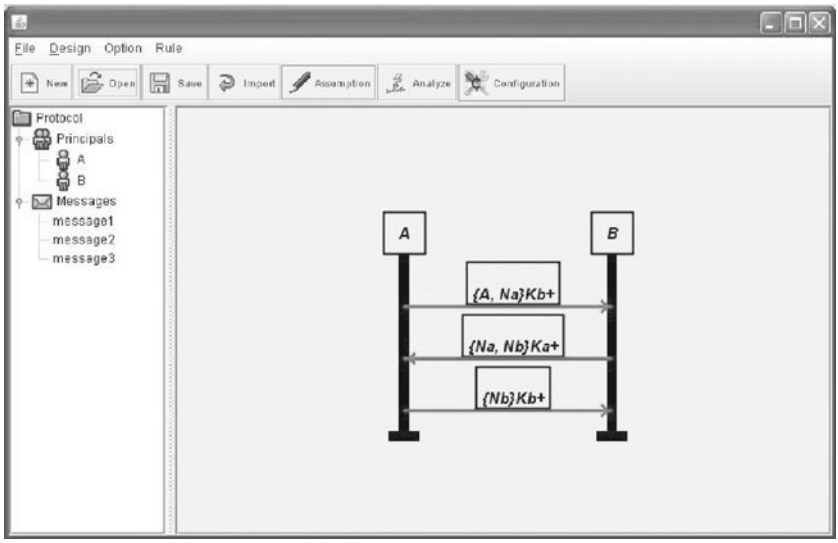

Fig. 9.4 User interface of BMF analyzer 2.

After taking security analysis test of cryptographic protocols under the above two environments, we know that the cryptographic protocol analysis result could be given definitely under both implementations of the BMF analyzers, and the time for analyzing under both cases is short. However, the BMF analyzer 1 which is implemented with embedded belief multiset formalism rules is more difficult to extend than the BMF analyzer 2 which has an independent rule database from the analyzer.

9.2.3 Implementation of the belief multiset formalism

The focus of the BMF analyzer is to provide a user-friendly, effective and powerful environment that can be used to facilitate the analysis of the existing cryptographic protocols and the creation of secure cryptographic protocols based on the notion of the trusted freshness.

From the comparison in Subsection 2, we decide to develop the BMF analyzer mainly on the BMF analyzer 2, that is, using Prolog to develop the BMF analyze engine and using Java source code generation to develop the visual BMF and the BMF result view.

Prolog is a general purpose logic programming language associated with artificial intelligence and computational linguistics, and it remains among the most popular such languages today, with many free and commercial implementations available. The Prolog program logic is expressed in terms of relations, represented as facts and rules, and a computation is initiated by running a query over these relations. Prolog is well suited for implementing a

BMF analyzer as it is straight forward to map all of the BMF constructs into suitable Prolog counterparts which can be easily manipulated and queried.

The BMF analyzer based on Prolog helps the protocol engineers to derive all possible BMF statements applicable to a given protocol and to determine whether a given protocol achieves its design objectives.

Java is an object-orientated technique that is employed in order to facilitate expansion and understanding of the source code. It is clear and convenient to develop and debug user-friendly interfaces like the visual BMF and the BMF result view.

The BMF analyzer is a graphically based application that incorporates cryptographic protocol modeling, automated BMF-based protocol security analysis and the analysis results presentation. The BMF analyzer is implemented similarly to the SPEAR II Framework except the analysis logic is the belief multiset formalism but GNY, and it incorporates a number of enhancements. Because of the architectural and security analysis logic differences between the SPEAR II and the BMF analyzer, the Java code generation and scenario simulation features are not the same in the BMF analyzer implementation.

To conduct an analysis with the BMF analyzer, a protocol engineer needs to specify a protocol's messages, initial assumptions and target goals in a Prolog-style BMF syntax via the visual BMF interface. The Prolog-based BMF rules are then imported and employed in the analysis, where a forward-chaining inference engine generates all of the BMF beliefs that can result from the systematic application of the inference rules to the set of initial assumptions, message steps and the cryptographic goals. A proof is then generated in an English-style syntax for every successful goal that was specified. This English-style proof lists all of the statements involved in the derivation of the successful goal, indicating the postulates that were used and the premises which were employed in the postulate's application.

The BMF analyzer is extensible, it allows extended security analysis of a protocol such as non-repudiation analysis for future use, and it also allows further engineering and design techniques to be incorporated, since the inference rules are stored separately from the BMF analyzer itself.

For interest of concision, some atoms and the predicates in the BMF analyzer are only briefly introduced. For strict or inquisitive readers, please refer to [5, 6] or contact us for detailed information.

9.2.3.1 Facts, goals and rules

In belief multiset formalism, the relations in Prolog include: assumptions, terms, medium results, goals and rules. These important concepts are represented as Prolog-style predications, and stored in the database of facts, the database of goals and the database of rules respectively.

The Prolog-style predications are formulae, statements, etc. Formulae are the components which are used to construct protocol messages and typically contain constants such as principal names, nonce, shared keys, shared parts

of keys, etc. The statements will be used to specify the initial and target beliefs and possessions of principals as well as extensions to be appended to the message components.

Facts

The idealized protocol messages, initial assumptions, terms and medium results are represented as *fact/3*, and are all stored as instances of fact/3 in the database of facts. Before conducting an automated BMF-based analysis, the messages and initial assumptions pertaining to the protocol to be analyzed must be specified in a Prolog file in the form of fact/3 predicates.

The predicate fact/3, which defines an inference step, appears as follows:

$$\text{fact}(\textit{Index}, \textit{Statement}, \textit{Reason}(\textit{PremiseList}, \textit{Rule}))$$

The integer *Index* is used to reference instances of this fact/3, while the argument *Statement* is bound to a derived statement, including the idealized protocol messages, initial assumptions, terms and medium results. The last argument *Reason(PremiseList*, *Rule*) is the reason to derive this fact/3, the parameter *PremiseList* is a list containing the indices of the premises that were used in deriving this *Statement* through the application of *Rule*, and the parameter *Rule* represented by characters enclosed in single quotes is the applied rule to derive this fact/3. If the statement represents terms or initial assumptions, the *PremiseList* would be empty and *Rule* would be either "*Term*" or "*Assumption*". For example, fact(*Index*, *Statement*, Reason([], "*Term*")) and fact(*Index*, *Statement*, Reason([], "*Assumption*")).

There are two important predicates for representing "send" and "receive" terms:

$$\text{send}(\textit{Identity}, \textit{Statement}, \textit{Step})$$

and

$$\text{receive}(\textit{Identity}, \textit{Statement}, \textit{Step})$$

which means that the principal '*Identity*' has sent or received the term '*Statement*' in this step '*Step*' respectively, where the integer '*Step*' is used to reference the time point to send or receive this term.

New added fact/3 involved in the analysis process must be extracted and sorted in ascending order by their indices, and any duplicates in this list must also be removed. The predicate *getMaxFactIndex/1* collects all of the indices within fact/3 into a list and then finds the maximum in this list, and the maximum index is returned in the argument *MaxIndex*.

Goals

In order to perform a protocol security analysis based on the belief multiset formalism, a designer must know what goals the protocol under inspection is expected to achieve. The security analysis will essentially involve a researcher determining the class of the protocol that he wishes to analyze, and then ensuring that the expected goals are fulfilled. For an authentication

protocol, one should verify the identities of participants and then ensures that they agree on an encryption key for later use.

The protocol goals are represented in a similar way to fact/3 with the *goal/2* predicate and stored in the database of goals. The goal/2 predicate appears as follows:

$$\text{goal}(\textit{Index}, \textit{Statement})$$

The integer *Index* is used to reference instances of this goal/2, and the argument *Statement* is bound to the anticipated security objectives (the security properties in the belief multisets, including the beliefs about principals and the beliefs about a freshness identifier). A belief is presented as the predicate:

$$\textit{Belief}\,(\textit{Identity}, \textit{Trust})$$

it means that the principal *Identity* believes the *Trust*. Any target goals must be specified in the same file by using goal/2 predicates.

Rules

The BMF inference rules are all specified through the use of a *rules/0* predicate and stored in the database of rules. For each BMF rule, there is at least one instance of the rules/0 predicates, some requiring more because of multiple conclusions. The basic pattern followed in a typical instance of the rules/0 predicate is to first check that all of the premises of the respective BMF rule are true and then to assert the conclusion in the Prolog database if it has not yet been asserted. After asserting the conclusion, the *addedFacts* atom is also asserted in the database to indicate that a conclusion was derived during the current cycle. If addedFacts has been asserted, then all the remaining instances are retracted from the database, and the *done/0* predicate fails. Otherwise, if addedFacts was not asserted then done/0 would succeed and forward-chaining would not commence. The done/0 predicate checks whether any new beliefs or possessions have been added to the Prolog database in the current cycle.

Example 9.1 Recall that the fragment rule A1(a) in the belief multiset formalism is:

$$\textbf{A1(a)}\ -\{...N, N'...\}_{K_{P_iP_j}} \wedge B_{P_i,t}(\langle 11K_{P_iP_j}P_iP_j\rangle) \wedge B_{P_i,t}(\langle ...1N...\rangle)$$
$$\Rightarrow B_{P_i,t}(\sim \{...N, N'...\}_{K_{P_iP_j}})$$

The code for the rules/0 predicate of the *Fragment Rule 1* in the BMF analyzer appears below:

```
rules :-
    fact(PremiseIndex1, told(P,encrypt(List, shared(K))),_),
    is_list(List),
    length(List, LengthOfList),
    LengthOfList > 1,
    fact(PremiseIndex2, believes(P, secret(shared(K))), _),
    fact(PremiseIndex3, believes(P,fresh(shared(K))), _),
    fact(PremiseIndex4, believes(P, associate(shared(K),P, Q)), _),
```

```
listMemberIsFresh(P, List, PremiseIndex5),
Conclusion = believes(P, bound(List)),
not(fact(_,Conclusion,_)),
getMaxFactIndex(MaxIndex),NewIndex is MaxIndex + 1,
PremiseIndices = [PremiseIndex1, PremiseIndex2, PremiseIndex3,
                  PremiseIndex4, PremiseIndex5],
asserta(fact(NewIndex,Conclusion,reason(PremiseIndices,'Fragment1'))),
asserta(addedFacts).
```

9.2.3.2 Visual BMF

The visual BMF provides graphical interface for the import of the protocol messages, initial assumptions, and security goals.

The protocol messages

Similar to SPEAR II, the visual BMF environment represents BMF statements using a tree-like structure combined with pop-up menus to allow easy interaction and to produce a meaningful representation of BMF information.

All statements of the same type form part of the same tree structure, a heterogeneous set of BMF statements are represented by a collection of separate trees. These representation techniques help users to easily create syntactically correct BMF statements without the need to be acquainted with the BMF syntax and notation. These structured trees representing the protocol messages could be exported and read as English-style text.

With a graphical interface, users have to remember few details about a system's structure and functionality, since this information is available within the interface. Furthermore, to use the visual BMF environment, users do not need to be familiar with the semantics and concepts underlying the belief multiset logic, however, they'd better be familiar with the logic to use it effectively.

The protocol's messages are specified in an optimum order as follows: the principals involved in the protocol, the role selection of each principal (sender, receiver and the trusted freshness), the cryptographic mechanism in the protocol, the actual communication between the principals etc. Once the protocol specification is complete, the protocol will be translated into graphical display as shown in Fig. 9.5.

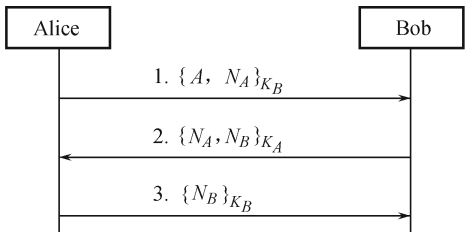

Fig. 9.5 View of a cryptographic protocol in the BMF analyzer.

Here, Alice means the principal A, Bob means the principal B.

The initial assumptions

The initial assumptions, also called the initial BMF beliefs, should be declared at the start of the protocol analysis, and any amendments could be made at a later stage. If a principal has generated a freshness identifier used in the protocol messages, the belief about the freshness of this identifier by the generator is also an initial assumption.

Once the protocol specification is complete, an analyzer can definitely make recommendations as to what the initial conditions for an analysis should be in certain cases, that is, the initial assumptions could be derived directly from the import protocol messages, as shown in Fig. 9.6.

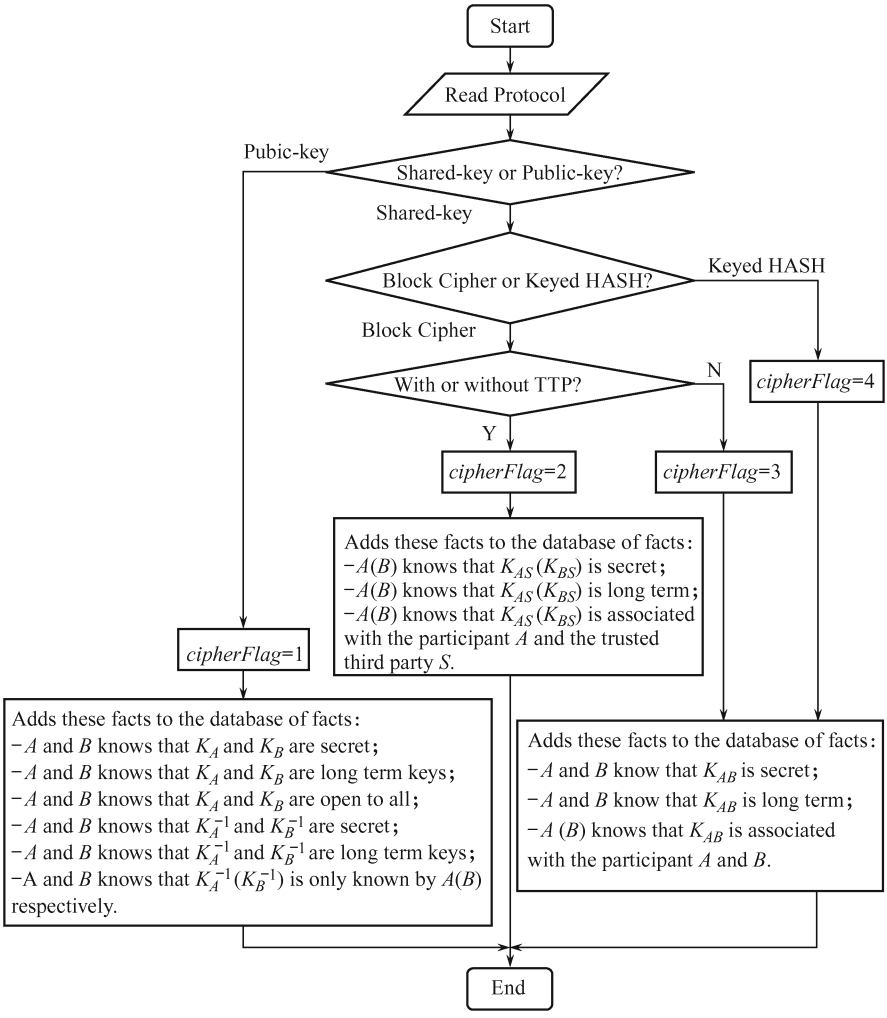

Fig. 9.6 The premise set of BMF analysis.

The *cipherFlag* is the flag of the cryptographic mechanism applied in this protocol:

— for *cipherFlag*=1, the cryptographic mechanism applied is Public-key scheme;
— for *cipherFlag*=2, the cryptographic mechanism applied is shared-key scheme with the trusted third party;
— for *cipherFlag*=3, the cryptographic mechanism applied is shared-key scheme without the trusted third party;
— for *cipherFlag*=3, the cryptographic mechanism applied is keyed hash scheme.

The extraction of terms

In general, the extraction of terms could be done meanwhile with the protocol message specification. If there exist nests of the cryptographic one-way transformations, the condition of repetition of the terms should be prudently considered. If the maximum terms of different messages are the same, there may exist the replay attack, hence the designed protocol should be reconstructed, as shown in Fig. 9.7 for the extraction of the terms, where *maxStep* is the largest number of the protocol message steps.

The security goals

The goal import interface supports the security goal configuration of an input protocol, such as UA-secure, MA-secure, UK-secure and MK-secure. The target goals are specified in the form of goal/2 predicates, that is *Belief*(*Identity*, *Trust*). The *Trust* beliefs are predicates about the security properties, including *Existing*(*Identity*), *Secret*(*Identifier*), *Fresh*(*Identifier*) and *Associate*(*Identifier, Identity*). The predicate *Existing*(*Identity*) means that a principal has the trust about the lively communication of this *Identity* principal. The predicate *Secret*(*Identifier*) means that a principal has the trust about the security of this *Identifier*. The predicate *Fresh*(*Identifier*) means that a principal has the trust that the freshness *Identifier* is a new generated TVP for this protocol. The predicate *Associate*(*Identifier, Identity*) means that a principal has the trust that the freshness *Identifier* is a TVP for a protocol related with the principal *Identity*.

Example 9.2 Here is an illustration of the security goals of a protocol in the BMF analyzer. Suppose the two communication principals are A and B, the new session key they want to establish is k_{AB}.

As for a UA-secure authentication protocol, if A wants to authenticate the principal B, then A has the security goal to achieve: *Belief*(A, *Existing*(B)).

As for an MA-secure authentication protocol, if A and B want to authenticate each other, the security goals to achieve are *Belief*(A, *Existing*(B)) and *Belief*(B, *Existing*(A)).

As for a UK-secure authentication protocol, the security goals to achieve are *Belief*(A, *Existing*(B)), *Belief*(A, *Secret*(k_{AB})), *Belief*(A, *Fresh*(k_{AB})), *Belief*(A, Associate(k_{AB}, A)) and *Belief*(A, *Associate*(k_{AB}, B)).

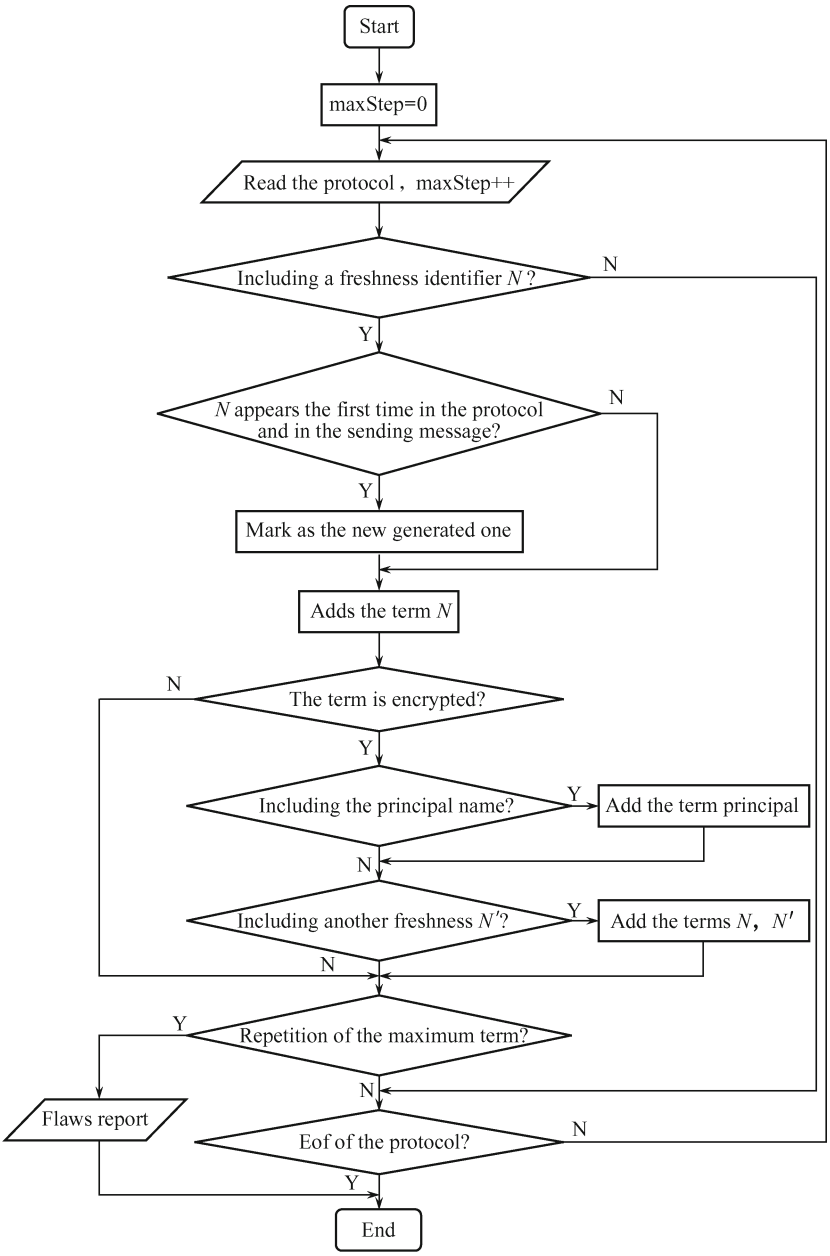

Fig. 9.7 Abstraction of BMF term generation.

As for an MK-secure authentication protocol, the security goals to achieve for the principal A are $Belief(A, Existing(B))$, $Belief(A, Secret(k_{AB}))$, $Belief(A, Fresh(k_{AB}))$, $Belief(A, Associate(k_{AB}, A))$ and $Belief(A, Associate$

(k_{AB}, B)); the security goals to achieve for the principal B are *Belief*(B, *Existing*(B)), *Belief*(B, *Secret*(k_{AB})), *Belief*(B, *Fresh*(k_{AB})), *Belief*(B, *Associate*(k_{AB}, A)) and *Belief*(B, *Associate*(k_{AB}, B)).

9.2.3.3 BMF analyze engine

Principals, messages, terms and initial assumptions specified in visual BMF are exported to BMF analyze engine in Prolog-style statements for analysis, and all of the BMF statements derived during an analysis, as well as the proofs for successful goals, are stored in the database of facts and are accessible through the result view pane.

Before conducting an automated BMF-based analysis, the messages, terms and initial assumptions pertaining to the protocol to be analyzed have been specified in the form of fact/3 predicates in the database of facts. Any target goals have also been specified in the form of goal/2 predicates in the database of goals.

Similar to the forward-chaining inference engine in SPEAR II, the BMF analyze engine applies all the inference rules to the set of statements consisting of the protocol messages, initial assumptions and medium results, until all of the statements which are derivable have been generated. If there exist new generated fact/2 predicates in the database of facts, the addedFacts atom is also asserted in the database to indicate that a conclusion was derived during the current cycle. If addedFacts has been asserted, the BMF analyze engine applies all the inference rules to the set of the term statements in this step, until there are not any new generated fact/3 predicates inserted into the database of facts. Then the analysis results are compared to determine whether one or more statements describing the goals of a specific protocol are derivable from a given set of initial assumptions. If the security goals are met, a proof can be generated for this security goal in the database, showing all of the steps and inference rules that were required to generate the result. Results from a BMF analysis conducted by the BMF analyze engine are returned to the BMF result view environment and appropriately displayed in English-style BMF syntax. A formal proof will then be constructed to show that a finite number of conclusions in a finite number of steps can be derived from using the inference rules based on the initial assumptions and messages of a given protocol.

Figure 9.8 illustrates the analysis procedure of the BMF analyze engine.

In Chapter 4, we have illustrated the security analysis procedure using the belief multiset formalism. As we have seen, the useful deriving statements, medium results, are developed through the application of the belief multiset inference rules manually, so the analysis procedure is relatively simple. While in the case of automated analysis, the analyzer will generate a lot of medium results, including not only the useful facts but also many needless medium results. The key problem to improve the performance of the analyzer is to discard the needless medium results effectively.

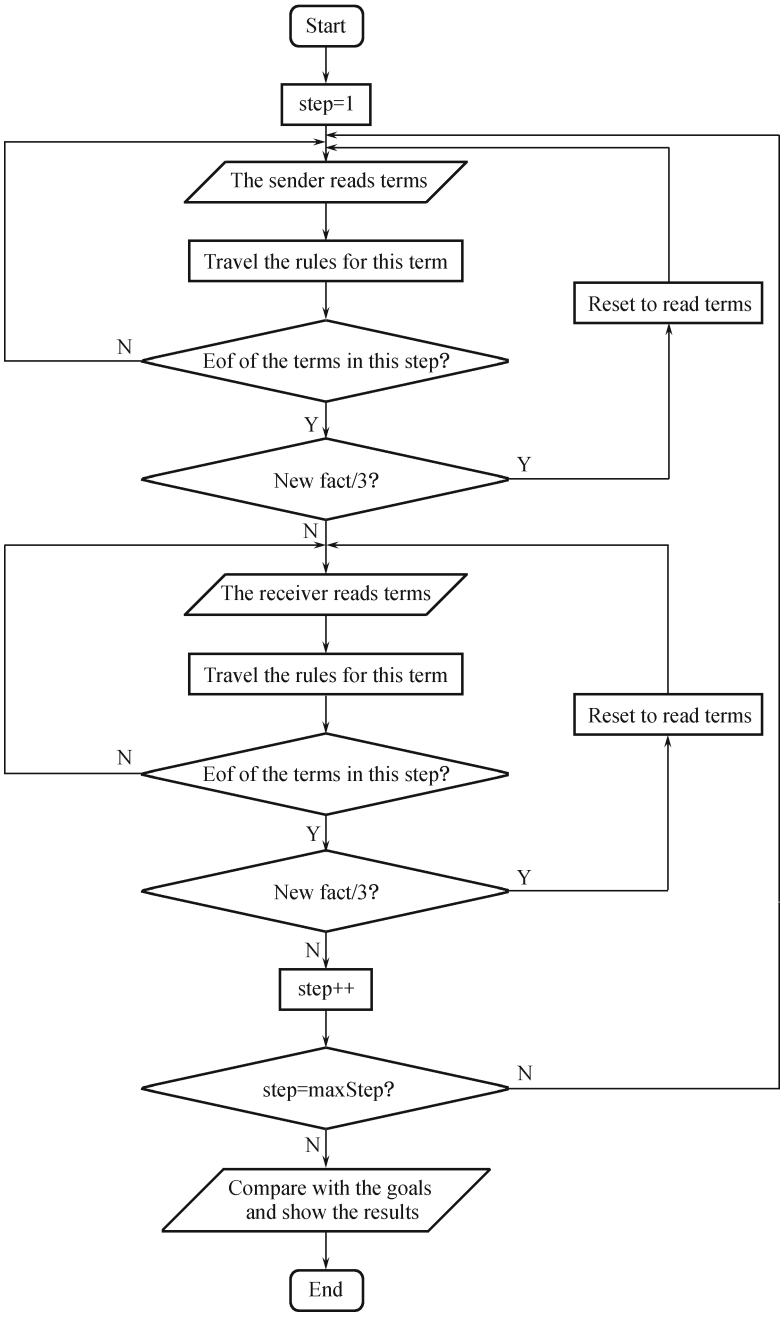

Fig. 9.8 BMF analyze engine.

9.2.3.4 BMF result view

The BMF result view provides a graphical result view environment, including protocol security analysis results showing the failed and successful beliefs in English-style BMF syntax, and the English-style proof list of a successful protocol goal.

The BMF result view ensures that an analysis statement representing in a Prolog-style formula is converted to an appropriate textual representation. The fact that the proof is in an English-style syntax makes it more readable and comprehensible.

This English-style proof lists all of the statements involved in the derivation of the successful goal, indicating the postulates that are used and the premises which are employed in the postulates' application. If a goal fails, a proof cannot be generated, and then the text 'FAILED!' appears instead of a proof.

9.2.3.5 BMF rule engine

The BMF rule engine provides a graphical environment for the new added belief multiset formalism inference rule input. As we know, the belief multiset rules presented in Chapter 4 are for authentication protocols which apply traditional cryptographic mechanisms, the researchers need to extend the inference rules used in the BMF analyze engine to meet variety applications in the real world.

The BMF rule engine includes the following steps to construct a new inference rule in the BMF analyzer:

1) Give the general rules/0 representation of the rule to be inserted.

```
rules   :-
        fact(PremiseIndex1, Statement1ToBeInserted, _),
        fact(PremiseIndex2, Statement2ToBeInserted, _),
         ...  ...
        Conclusion =ConclusionToBeInserted
        not(fact(_, Conclusion, _)),
        getMaxFactIndex(MaxIndex), NewIndex is MaxIndex + 1,
        PremiseIndices = [PremiseIndex1, PremiseIndex2, ... ],
        asserta(fact(NewIndex, Conclusion, reason(PremiseIndices,
                    'RuleNameToBeInserted'))),
        asserta(addedFacts).
```

2) Convert the new inference rule in the belief multiset formalism into the inference rule of the BMF analyzer in Prolog-style.

Replace the *Statement1ToBeInserted*, *Statement2ToBeInserted*, etc. with

the conditions of the new inference rule in the belief multiset formalism; replace the *ConclusionToBeInserted* with the conclusions of the new inference rule in the belief multiset formalism; replace the *RuleNameToBeInserted* with the new inference rule name.

3) Show the new inference rule to the rule design researcher, and allow him to improve this rule until the researchers submit this rule to the system.

To ensure that the new generated rule produces the correct results for different inputs, the rule should be tested individually by specifying all of the rule premises using fact/3 predicates, running the analyzer, and then examining the results.

Recall that the belief multiset inference rules are in the style of Prolog which is a programming language associated with artificial intelligence, and the rule database is stored independently from the BMF analyzer and it is dynamically loaded when the security analysis is ongoing. Hence, it is convenient to edit the inference rules outside the system, and the changing could be applied immediately. That is, besides the insertion of the new inference rule in the graphical environment of the BMF analyzer, the inference rules could even be edited, deleted and inserted in a text editor such as the Notepad in the Windows system. Therefore, the flexibility of the BMF analyzer has been greatly improved.

9.2.3.6 BMF attack engine

The attack engine supports the construction of attacks from the absence of the security properties derived from the security analysis results based on the belief multiset formalism. As we all know, the automation construction of an attack requires quite a high level of user expertise in a large number of specialized engineering techniques, while the manually construction of the attack may seem relatively simple. Here we only give a naive attack engine model for attack construction based on the trusted freshness approach, as shown in Fig. 9.9.

1) Indicate the principal to be deceived from the absence of the security properties of this protocol being analyzed.

2) Construct the first message to cheat the principal whose security properties about this protocol are not met.

3) Complement other messages to form an instance of the protocol run with full messages.

4) Find the messages that couldn't be constructed in this instance. If any, continue Step 5; if not, terminate this protocol construct procedure, and this protocol instance is the attack that we want to construct on this flawed protocol.

5) Start another instance of this flawed protocol in order to generate the key messages that couldn't be constructed from the instance in Step 4. Complement other messages to form this interleaved instance with full messages. Thus, these two interleaved instances, that are the instance in Step 4 and the instance in Step 5, construct the attack on this flawed protocol.

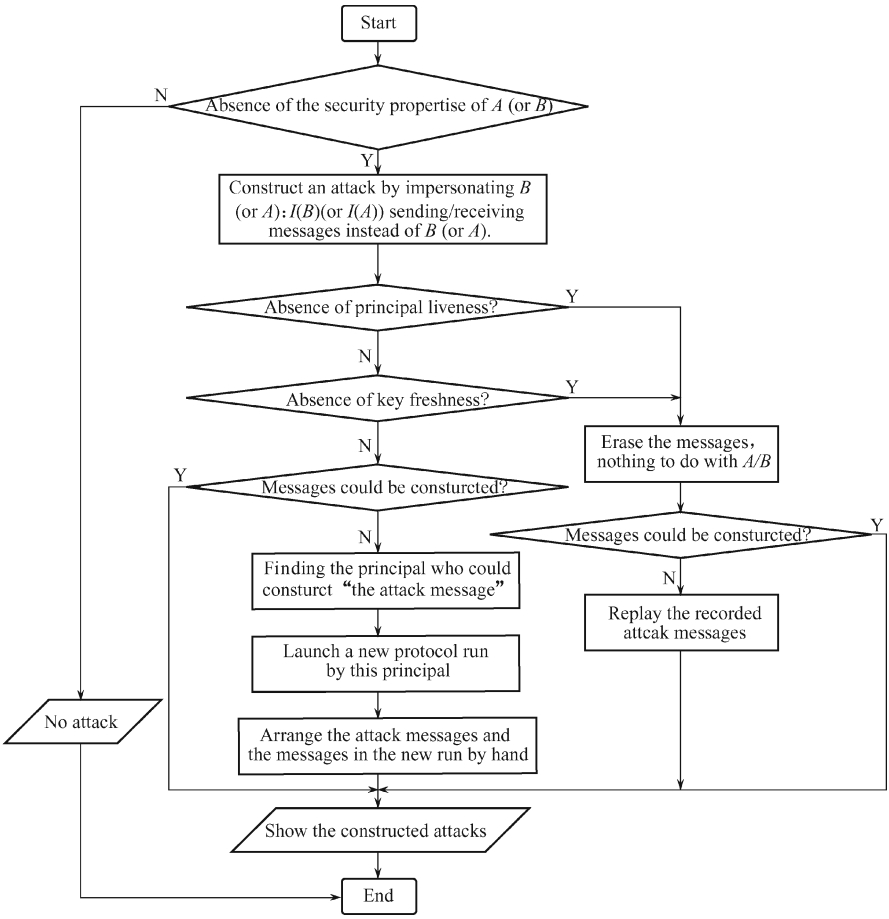

Fig. 9.9 BMF attack engine.

6) Arrange the order of the messages in the two interleaved instances, and then terminate this protocol construct procedure.

Let's review the Needham-Schroeder public-key protocol in Example 1.2. From the security analysis of the Needham-Schroeder protocol using the belief multiset formalism in Subsection 5, A believes that B is in lively correspondence in this protocol run, and the shared parts of both N_A and N_B are secret, fresh, and also associated with the principal A and the principal B. However, although B believes that A is in lively correspondence in this protocol run, N_B is secret, fresh, and associated with the principal A and the principal B, but B has not gotten any corroborative evidence that N_A is fresh and is associated with the principal A and the principal B.

Example 9.3 Here is an illustration of the attack construction procedure of the Needham-Schroeder public-key protocol in the BMF attack engine.

Here, Alice means the principal A, Bob means the principal B, while Malice means the adversary I.

1) It is the principal B whose security properties are absent, we need to construct an attack to cheat B.

2) Construct the first message Message 1 to cheat B.

Message 1 $\quad I(A) \rightarrow B: \quad \{A, N_A\}_{K_B}$

3) Complement Message 2 and Message 3 to form an instance of Needham-Schroeder public-key protocol with full messages.

Message 1 $\quad I(A) \rightarrow B: \quad \{A, N_A\}_{K_B}$

Message 2 $\quad B \rightarrow I(A): \quad \{N_A, N_B\}_{K_A}$

Message 3 $\quad I(A) \rightarrow B: \quad \{N_B\}_{K_B}$

4) Message 3 couldn't be constructed in the above instance since the adversary I impersonating A namely $I(A)$ could not get the freshness identifier N_B.

If the adversary I wants to generate the message $\{N_B\}_{K_B}$, I must know N_B. Since N_B appears only in Message 2 and Message 3, while Message 2 is a one-way transformation sent from the victim B to A and I does not have the knowledge of the decryption key K_A^{-1}, hence the adversary I could not get N_B from Message 2, hence I could only get N_B from Message 3. Note that Message 3 is encrypted under the receiver's public-key, so the adversary I must be the receiver to perform a one-way transformation in order to get N_B from Message 3 using I's private key K_I^{-1}. Hence, there should exists:

Message $3'$ $\quad ?? \rightarrow I: \quad \{N_B\}_{K_I}$

5) Since N_B appears only in Message 2 which is a one-way transformation sent from the victim B to A, only A could get N_B from Message 2, so the Message $3'$ in Step 4 could only be exchanged between the principal A and the adversary I, that is:

Message $3'$ $\quad A \rightarrow I: \quad \{N_B\}_{K_I}$

Hence, the new instance to generate the key Message $3'$ is between A and I, and this idea is consistent with the security property that A is in lively correspondence in this protocol run.

Complement Message $1'$ and Message $2'$ to form this new instance:

Message $1'$ $\quad A \rightarrow I: \quad \{A, ??\}_{K_I}$

Message $2'$ $\quad I \rightarrow A: \quad \{??, N_B\}_{K_A}$

Message $3'$ $\quad A \rightarrow I: \quad \{N_B\}_{K_B}$

If the adversary I wants to generate Message $3'$, then I should know N_B, since N_B is encrypted under A's public-key, I could only replay the recorded message Message 2 $\{N_A, N_B\}_{K_A}$ including $\{??, N_B\}_{K_A}$ in order to get N_B, that is:

Message $2'$ $\quad I \rightarrow A: \quad \{N_A, N_B\}_{K_A}$

To make the principal A believe that Message $2'$ is really from I, then the unknown "??" in Message $1'$ could only be the freshness identifier N_A that

is the same as that in Message 2′. Hence we have:

Message 1′ $A \rightarrow I:$ $\{A, N_A\}_{K_I}$

6) From the above construction procedure, we have the first instance

Message 1 $I(A) \rightarrow B:$ $\{A, N_A\}_{K_B}$

Message 2 $B \rightarrow I(A):$ $\{N_A, N_B\}_{K_A}$

Message 3 $I(A) \rightarrow B:$ $\{N_B\}_{K_B}$

and the second instance

Message 1′ $A \rightarrow I:$ $\{A, N_A\}_{K_I}$

Message 2′ $I \rightarrow A:$ $\{N_A, N_B\}_{K_A}$

Message 3′ $A \rightarrow I:$ $\{N_B\}_{K_B}$

Arrange the order of the messages in the two interleaved instances, and then we have the attack on the Needham-Schroeder public-key protocol, as shown in Fig. 9.10.

Message 1′ $A \rightarrow I:$ $\{A, N_A\}_{K_I}$

Message 1 $I(A) \rightarrow B:$ $\{A, N_A\}_{K_B}$

Message 2 $B \rightarrow I(A):$ $\{N_A, N_B\}_{K_A}$

Message 2′ $I \rightarrow A:$ $\{N_A, N_B\}_{K_A}$

Message 3′ $A \rightarrow I:$ $\{N_B\}_{K_I}$

Message 3 $I(A) \rightarrow B:$ $\{N_B\}_{K_B}$

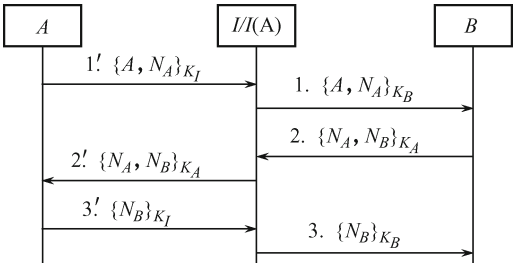

Fig. 9.10 View of attacks in BMF analyzer.

The attack involves two simultaneous runs of the Needham-Schroeder public-key protocol. In the first run, A establishes a valid session with the adversary I; in the second run, I impersonating A tries to establish a bogus session with B. Upon the termination of this attack, B believes that B has correctly established a session with A and they shared exclusively the secret nonce N_A and N_B to generate the new session key.

As we have seen, the attack construction procedure is complex, and the formalization of the intelligence activities in the attack construction is difficult. The BMF attack engine presented is still naive, and a lot of jobs need to be done.

In the above chapters, we have presented security definition, security specifications, freshness principle, manual analysis method, belief multiset formalism and the automation tools based on the trusted freshness for analyzing cryptographic protocols, which clarify whether a cryptographic protocol is secure or not.

The central ingredient in the trusted freshness approach is the observation that a participant's beliefs about key exchange security should depend only on the received fresh and confidential messages and the beliefs already possessed by this party. Analysis based on trusted freshness captures exact authentication information of each principal, which suggests the correctness of a protocol or the way to construct attacks intuitively from the absence of security properties.

First, Chapter 4 presents the security definitions, the security specifications based on the indistinguishability approach and matching conversation, which check whether a cryptographic protocol is secure or not; Chapter 6 makes a more rigorous proof to assess that the indicated security specifications are not only necessary but also substantial under the computational model. This specific security adequacy captures the peculiarities of key exchange protocols that involve different sessions. Chapter 7 presents a belief multiset formalism for analysis of cryptographic protocols based on trusted freshness.

Chapter 4, Chapter 5 and Chapter 7 have exemplified the usability and the efficiency of the security specifications to guarantee the protocol security and the belief multiset formalism via a set of well-known protocols. The absence of certain security properties suggests the instant construction of many attacks (not only one) on the protocol or suggests the correction of the protocol. For example, in Kerberos pair-key protocol in DSNs (see Subsection 5), B could not guarantee the freshness of k_{AB} and the liveness of sensor node A, so the adversary can construct an attack by impersonating A and confuse B to regard an old key k'_{AB} as a new session key between B and A. From the absence of the association k_{AB} with A and B, the adversary can construct an attack and confuse B to believe that B shares a new session key k_{AB} with A, but in deed B shares k_{AB} with the attacker I.

The proofs of security based on trusted freshness are simple and precise, which can be easily accomplished not only by hand (Chapter 4 and Chapter 5) but also by formalism (Chapter 6 and Chapter 7). Moreover, the analysis process based on trusted freshness is rigorous and amenable for design (Chapter 8) and automation (Chapter 9).

References

[1] Dong L, Chen K, Zheng Y, Hong X (2008) The Guarantee of Authentication Protocol Security. Journal of Shanghai JiaoTong University, 42(4): 518 – 522 (in Chinese)

[2] Chen K, Dong L, Lai X (2008) Security Analysis of Cryptographic Protocols Based on Trusted Freshness. Journal of Korea Institute of Information Security and Cryptology, 18(6B): 1 – 13

[3] Dong L, Chen K, Lai X (2009) Belief Multisets for Cryptographic Protocol Analysis. Journal of Software, 20(11): 3060 – 3076 (in Chinese)

[4] Dong L, Chen K, Lai X, Wen M (2009) When is a Key Establishment Protocol Correct? Security and Communication Networks, 2(6): 567 – 579

[5] Saul E (2001) Facilitating the Modelling and Automated Analysis of Cryptographic Protocols. Master Thesis, University of Cape Town

[6] Saul E and Hutchison A (1999) SPEAR II: The Security Protocol Engineering and Analysis Resource. Second South African Telecommunications, Networks, and Applications Conference. http://pubs.cs.uct.ac.za/archive/00000128/01/saul1999_SPEAR_SATNAC.pdf. Accessed 5 May 2011

[7] Mao W (2004) Modern Cryptography: Theory and Practice. Prentice Hall, New Jersey

[8] Bekmann J, Goede P. de and Hutchison A (1997) SPEAR: a Security Protocol Engineering & Analysis Resource. In DIMACS Workshop on Design and Formal Verification of Security Protocols. http://dimacs.rutgers.edu/Workshops/Security/program2/hutch/spear.html. Accessed 5 May 2011

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

[10] Lowe G (1996) Breaking and Fixing the Needham-Schroeder Public-key Protocol Using FDR. In: TACAS'96 Proceedings of the 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Passau, 27 – 29 Mar 1996. Lecture Notes in Computer Science (Lecture Notes in Software Configuration Management), vol 1055. Springer-Verlag, Heidelberg, pp 147 – 166

[11] Lowe G (1995) An Attack on the Needham-Schroeder Public Key Authentication Protocol. Information Processing Letters, 56(3): 131 – 133, 1995

[12] Lowe G (1996) Some new Attacks Upon Security Protocols. In Proceedings of the 9th IEEE Computer Security Foundations Workshop, pages 162 – 169, Jun. 1996

[13] Roscoe A W (1994) Model Checking CSP. In: Roscoe A W (ed) A Clasical Mind: Essays in Honour of C.A.R Hoare. Prentice-Hall, 1994

[14] Meadows C (1996) The NRL Protocol Analyzer: an Overview. Journal of Logic Programming, 26(2): 113 – 131

[15] Dolev D, Yao AC (1983) On the Security of Public Key Protocols. IEEE Transactions on Information Theory 29(2): 198 – 208

[16] Meadows C (1994) A Model of Computation for the NRL Protocol Analyzer. In: Proceedings of the 1994 Computer Security Foundations Workshop, Franconia, 14 – 16 June 1994

[17] Harkins D and Carrel D (1998) The Internet Key Exchange Protocol (IKE). IETF RFC 2409, Available at http://www.ietf.org/rfc/rfc2409.txt. Accessed 5 May 2011

[18] Kaufman C (2005) Internet Key Exchange (IKEv2) Protocol. IETF RFC 4306, Available at http://tools.ietf.org/html/rfc4306. Accessed 5 May 2011

[19] SET (1997) Secure Electronic Transaction. The SET Standard Specification. http://www.setco.org/set-specifications. Accessed 5 May 2011

[20] Mitchell J.C, Mitchell M and Stern U (1997) Automated Analysis of Cryptographic Protocols Using MurΦ. Proc. of 1997 IEEE Symposium on Security and Privacy, Oakland, California, pp 141 – 153

[21] Needham RM, Schroeder MD (1978) Using Encryption for Authentication in Large Network of Computers. Communication of the ACM 21(12): 993 – 999

[22] Tatebayashi M, Matsuzaki N, Newman D (1990) Key Distribution Protocol for Digital Mobile Communication Systems, CRYPTO'89, LNCS435

[23] Neuman C, Ts'o T (1994) Kerberos: An Authentication Service for Computer Networks, IEEE Communications Magazine, 32(9): 33 – 38

[24] Millen JK, Clark SC, Freedman SB (1987) The Interrogator: Protocol Security Analysis. IEEE Trans. Software Eng. 13(2): 274 – 288

[25] Berezin S, Groce A. SyMP: Symbolic Model Prover. Model Checking@CMU. http://www.cs.cmu.edu/ modelcheck/symp.html. Accessed 5 May 2011

[26] Fabrega FJT, Herzog JC, Guttman JD (1998) Strand Spaces: Why is a Security Protocol Correct? In: Proceedings of IEEE Symposium on Security and Privacy, Oakland, 3 – 6 May 1998

[27] Song D (1999) Athena: A New Efficient Automatic Checker for Security Protocol Analysis. Proceedings of the 1999 IEEE Computer Security Foundations Workshop, IEEE Computer Society Press, pp 192 – 202

[28] Millen J K (1996) CAPSLS: Common Authentication Protocol Specification Language. http://www.csl.sri.com/users/millen/capsl. Accessed 5 May 2011

[29] Wei MQ (2008) Automated Verification of Security Protocol. BE Thesis (in Chinese), Shanghai Jiaotong University

[30] Wang EJ (2008) Research of Automated Verification Tool for Security Protocol. BE Thesis (in Chinese), Shanghai Jiaotong University