Proceedings of the 4th WSEAS Int. Conf. on Information Security, Communications and Computers, Tenerife, Spain, December 16-18, 2005 (pp234-239)

Extending CAPSL for Logic-Based Verifications

LIANG TIAN, REINER DOJEN, TOM COFFEY
Data Communications Security Laboratory
Department of Electronic and Computer Engineering
University of Limerick
IRELAND
http://www.dcsl.ul.ie

Abstract: - Cryptographic protocols are designed to provide security services, such as key distribution, authentication and non-repudiation, over insecure networks. The design process of cryptographic protocols is highly complex. In particular, the translation from the informal protocol description to the formal protocol specification is an error-prone step, as the exact meaning of the requirements of the security protocol need to be conveyed. This paper presents a case study on extending the Common Authentication Protocol Specification Language (CAPSL) to enable it to be used with logic-based formal verification tools for security protocols.

Key-Words: - specification of cryptographic protocols, verification of security protocols, CAPSL

1. Introduction

Cryptographic protocols are designed to provide security services, such as key distribution, authentication and non-repudiation, over insecure networks. The design process of cryptographic protocols is particularly complex and error-prone [1]. The surprisingly significant number of published protocols that have subsequently been found to contain various flaws, sometimes several years after the original publication, highlights the complexity of the design process.

Conventionally, informal techniques have been used in the design and verification of cryptographic protocols. However, informal verification alone can lead to subtle protocol flaws and weaknesses remaining unidentified. Conversely, formal verification techniques provide a systematic approach to discovering protocol flaws and weaknesses. Common approaches to formal protocol verification are based on modal logics [1], [2], [3] or state-machines [4], [5], [6], [7]. Automation of such formal verification techniques removes many of the potential error sources in manual verification [8].

Although formal verification has demonstrated great success in discovering protocol flaws, these formal techniques are not foolproof. The complex and difficult translation from the informal protocol description to the formal protocol specification is the critical step in order to convey the exact meaning of security protocol steps [9]. Any misunderstanding of theses steps will result in a flawed formal specification, rendering the formal verification process useless. This is particularly true of logic-based verifications, as the intended message meanings must be formally defined, which requires a thorough understanding of the logic.

CAPSL is a formal language for expressing authentication and key-exchange protocols. Its purpose is to express enough of the abstract features of these protocols to support an analysis for protocol failures. However, due to its original design intention for use with state-machine based verification techniques, CAPSL has difficulty to model certain logic-specific features.

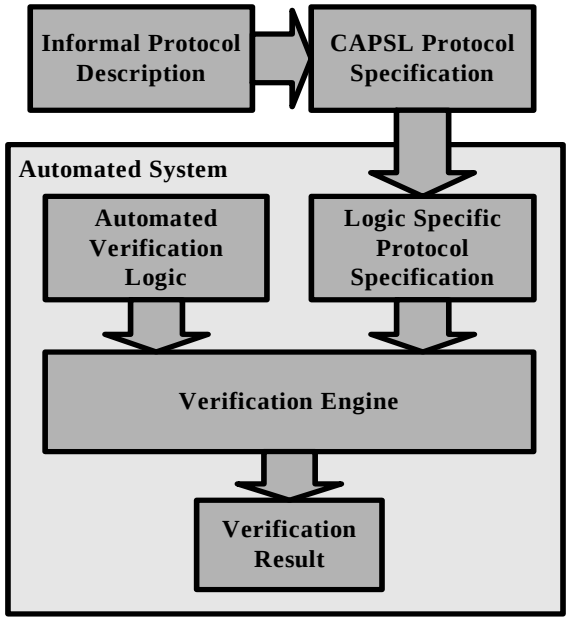

Fig. 1: Automated Protocol Verification

This paper presents a case study on extending CAPSL to enable it to be used in logic-based formal verification of security protocols. Using

these extensions, one can use the CAPSL specification language to model protocols for verification with the GNY logic. Such a formal specification can be used as the input to an automated proving engine [8], [10] as outlined in Figure 1, where the translation from the CAPSL protocol specification into the logic-specific specification is automated.

2. Logic-Based Verification

The technique of logic-based formal verification is accredited largely to Burrows, Abadi and Needham, developers of the BAN logic [11]. This work initiated intense research in the area of logic-based formal verification. Several logics, such as GNY [12], CS [13] and ZV [14] have been developed on the basis of BAN. These logics can be used to generate concise proofs and have identified a number of flaws in protocols previously considered secure. They incorporate several improvements over BAN and are applicable to a wider range of protocols. In general, logic-based formal verification involves the following steps [1]:

1. Formalisation of the protocol messages
2. Specification of the initial assumptions
3. Specification of the protocol goals
4. Application of the logical postulates

A successfully verified protocol can be considered secure within the scope of the logic. On the other hand, even the results of a failed verification are helpful, as these may point to missing assumptions or weaknesses in the protocol. If a weakness is discovered, the protocol should be redesigned and re-verified.

However, verification logics have their limitations, not least of which is the likelihood of errors in protocol formalisation. Opportunities to make such mistakes abound as the verification process is complicated and requires a thorough understanding of the used logic. During the verification process the semantics of the protocol must be interpreted, in order to specify the meaning that a protocol message is intended to convey. If the formalised protocol does not properly represent the original design, then the proof demonstrates only that the protocol corresponding to this formal description is secure. However, no claims can be made on the security of the original design. The use of a commonly known specification language, like CAPSL, reduces the risk of errors in the specification.

3. The CAPSL Specification Language

CAPSL, acronym for Common Authentication Protocol Specification Language, is a formal language for expressing authentication and key-exchange protocols [15]. Its purpose is to express enough of the abstract features of these protocols to support an analysis for protocol failures. The authors of CAPSL had broadened the applicability of CAPSL further with the extension to MuCAPSL [16] for multicast protocols. However, CAPSL has difficulty to present certain features specific to logic-based verifications, due to its original design intention for use with state-space-based verification techniques. While this case study uses the CAPSL language, the presented extensions can also be applied to MuCAPSL.

CAPSL is defined as a high- level language for applying formal methods to the security analysis of cryptographic protocols. It intends 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. CAPSL also clarifies the distinction between short-term session data and the long-term data associated with persistent entities. This distinction is achieved by applying the general type specification mechanism, together with the novel concepts of private functions and invertibility axioms. CAPSL is modular and extensible, and has a number of syntactic features that are unique to protocol analysis. CAPSL syntax is motivated both by user convenience and by the needs of protocol analysis tools. A CAPSL specification is made up of three kinds of modules: *typespec*, *protocol*, and *environment* specifications, usually in that order. Abstract data type specifications (called *typespecs*) introduce new data types and define cryptographic operators and other functions axiomatically. Standard typespecs are included automatically and others may be supplied by the user. A protocol specification has three principal parts: declarations, messages and goals. With CAPSL, one can express protocols in the message-list form. Figure 2 illustrates a sample protocol specification for the Needham-Schöder Public-Key Protocol.

3.1 Extendibility of CAPSL

One of CAPSL's advantages over other specification languages is its extendibility in the form of abstract data type specifications, called typespecs. Typespecs are used to declare operations axiomatically as abstract data types. Specifications

for the most popular operators, representing the abstract features of cryptosystems, are included in a standard 'prelude' file of typespecs supplied with the CAPSL environment.

Type specifications in CAPSL and their use for introducing new operators and subtypes bring an expanding class of protocols within reach. Further, typespecs can be used to introduce new operators and subtypes as needed when specifying a protocol for a specific verification tool or technique.

```
PROTOCOL NSPK;
VARIABLES
     A,B: PKUser;
     Na,Nb: Nonce, CRYPTO;
ASSUMPTIONS
     HOLDS A: B;
MESSAGES
     1. A -> B: {A,Na}pk(B);
     2. B -> A: {Na,Nb}pk(A);
     3. A -> B: {Nb}pk(B);
GOALS
     SECRET Na
     SECRET Nb;
     PRECEDES A: B | Na;
     PRECEDES B: A | Nb;
END;
```

Fig.2: Sample CAPSL Specification

3.2 Problems in Using CAPSL with Logic-based Formal Verification

CAPSL is intended to serve as an interface supporting a wide range of formal verification techniques for the analysis of cryptographic protocols. However, due to its design goal for use with state-space techniques, it provides little support for many required operators in logic-based verifications. Therefore, the original CAPSL language cannot directly be used to specify security protocols for logic-based verifications. However, typespecs can be used to introduce the required operators for use with different logics. As different logics use distinct operators, a different set of typespec extensions must be used for each individual logic. For example, the concepts of belief and jurisdiction are central to the GNY logic, while the CS logic is based on knowledge and time-indices.

3.3 Using "Typespecs" to Extend CAPSL

CAPSL's extendibility is achieved using the keyword typespec to introduce new data types and functional operators into CAPSL to meet various requirements of different verification methods.

The classical use of typespec in CAPSL is to declare cryptographic operators, hash functions, and other operations axiomatically as abstract data types. For example, in the CAPSL prelude file the basic symmetric-key type was introduced as shown in Figure 3.

```
TYPESPEC SKEY;
IMPORTS FIELD;
TYPES Skey;
FUNCTIONS
     sha(Field): Skey;
     mac(Skey,Field): Skey;
END;
```

Fig.3: CAPSL Symmetric-key Type

The only operators given in this typespec are a hash function and a keyed hash. The new type KKEY now can be used to declare a variable or a subtype in a protocol description.

To utilize CAPSL in logic-based verification of security protocols, it must be extended for each individual logic, as many of the corresponding logic operators are not supported in CAPSL. While the use of typespec was intended to introduce operations used for the execution of cryptographic protocols, such as encryption and decryption, it also can be used to include verification operators not provided by the original CAPSL language. For example, Figure 4 presents a type specification for the GNY logic in regard to its jurisdiction operator. By importing this type into a protocol specification file, the GNY jurisdiction property can be modelled.

```
TYPESPEC JURIS;
FUNCTIONS
 Jurisdict(Principal,Field):Atom;
END;
```

Fig.4: Type for GNY Jurisdiction Operator

4. Case Study: A CAPSL Extension for the GNY Logic

This section investigates the extension of CAPSL for the GNY logic. While CAPSL provides a number of common constructs, which are applicable to cryptographic protocols, it is not able to model all of GNY's operators. The proposed extensions in this paper allow the use of CAPSL with GNY-based verifications.

4.1 The GNY Logic

The GNY logic [12] is used to reason about cryptographic protocols. GNY is a direct successor

of the BAN logic and is quite powerful in its ability to uncover even subtle protocol flaws. Discussions of the virtues and limitations of the logic can be found in [17] and [18].

In GNY, message extensions are added to the protocol description during protocol formalisation, so that principals can communicate their beliefs and thus reason about each other's beliefs. The use of message extensions enables the logic to deal with different levels of trust between protocol principals. As such, it is considered an improvement over BAN, which assumes that all principals are honest and competent. This development is noteworthy as many protocol attacks are performed by dishonest principals. As an example of a message extension, consider the following: P→Q:{K,P}Ks- is formally stated as $Q \triangleleft *\{*K,*P\}Ks- \sim> S \mid\equiv P \xleftrightarrow{K} Q$. This means that principal Q is told a session key, K, and an identity, P, encrypted under the private key of principal S. Each component is marked with a not-originated-here asterisk. Also, Q is told that S believes K is a suitable shared secret for P and Q.

The postulates of the GNY logic are used to deduce whether the protocol goals can be derived from the initial assumptions and protocol steps. If such a derivation exists, the protocol is successfully verified.

4.2 Extending CAPSL for GNY

The following concepts of the GNY logic need to be modelled by CAPSL:

- **Convey:** When a principal conveys some formula X, X can be the message itself or something computable from such a message. i.e. a formula can be conveyed implicitly. This operator is not directly supported in CAPSL.
- **Posession:** This indicates what a principal possesses, or is capable of possessing. At any particular stage of a protocol run, this includes all the formulae that a principal has been told, all the formulae it started the session with, as well as all the ones it has generated during the course of the run. Also, the principal possesses or is capable of possessing everything that is computable from the formulae that it already possesses. Available through CAPSL `HOLDS` operator.
- **Freshness:** When a formula is declared as fresh, it implies that this formula has not been used for the same purpose at any time before the current run of the protocol e.g. a pseudo-random number generator can produce formulae that a principal can safely believe to be fresh. CAPSL only supports freshness of variables, but not of messages/statements.
- **Recognisable:** This implies that a principal would recognise a formula if the principal has certain expectations about the contents of the formula. The principal may recognise its own identifier e.g. its own identifier, a particular structure e.g. the format of a timestamp, or a particular form of redundancy. This operator is not directly supported in CAPSL.
- **Belief:** The reasoning system uses rules about how belief is propagated to establish new beliefs. Belief is considered useful in evaluating trust that may be placed in a security protocol or another principal. If a principal P believes X, then the principal P acts as if X is true. Available through CAPSL `BELIEVES` operator.
- **Secrecy:** A secrecy assertion implies that the value of a variable generated by its nominal originator cannot be obtained by an intruder (unless, that is, that the intruder is playing the role of a legitimate). Available through CAPSL `SECRET` operator.
- **Jurisdiction:** The notion of jurisdiction also represents trust. Saying that a principal has jurisdiction over some piece of data implies that the principal is an authority on this data and should be trusted on this matter. This operator is not directly supported in CAPSL.
- **Message Extension:** Protocol specifications often include verbal descriptions to the effect that a principal should only proceed if certain conditions hold or only if the principal holds certain beliefs. This can be regarded as a precondition. In GNY, if a statement C is the precondition for a formula being conveyed, C is called a message extension

Figure 5 details the proposed "Typespec" extensions to the CAPSL language for GNY specifications.

The CAPSL specification detailed in Figure 6 models the Needham-Schröder Symmetric Key protocol using the extensions outlined above.

5. Conclusions

This paper reviewed the CAPSL specification languages and its usage with logic-based formal verification of cryptographic protocols. The existing problems when using CAPSL as a specification language for verifications with formal logics are detailed. As a case study, the GNY logic was selected to investigate the extendibility of CAPSL for use with logic-based verifications.

```
// Message Extension
TYPESPEC STATEMENT;
TYPES
   Operator : Atom;
   Stmt: Atom;
FUNCTIONS
st(Principal,Operator,Field):Stmt;
END;

TYPESPEC MSGEXT;
IMPORT STATEMENT;
TYPES
   Msg : Field;
FUNCTIONS
   ext(Msg,Stmt):Atom;
END;

// Convey Operator
TYPESPEC CONVEY;
FUNCTIONS
   convey(Principal, Field):Atom;
END;

// Freshnes Operator
TYPESPEC FRESHNESS;
FUNCTIONS
   fresh(Field):Atom;
END;

// Not Originated Here Formulae
TYPESPEC NOTHERE;
FUNCTIONS
   nothere(Field):Atom;
END;

// Recognizable Operator
TYPESPEC RECON;
FUNCTIONS
   reconised(Field):Atom;
END;

// Jurisdiction Operator
TYPESPEC JURIS;
FUNCTIONS
   jurisdiction(Principal,
                     Field):Atom;
END;
```

Fig. 5: Proposed GNY extensions for CAPSL

```
PROTOCOL NeedhamSchroederSymmetric;
IMPORTS MSGEXT;
IMPORTS JURIS;
IMPORTS FRESHNESS;
IMPORTS RECON;
IMPORTS NOTHERE;
VARIABLES
  A,B: Client;
  S: Server;
  Na,Nb: Nonce, FRESH, CRYPTO;
  Ka,Kb: Skey;
  allStatement: Atom;
  Kas: Skey, FRESH, CRYPTO;
  Believe: Operator;
  Stmt1,Stmt2,Stmt3: Stmt;
DENOTES
    Stmt1= st(S, Believe,
              shareKey(A,Kab, B));
    Stmt2= st (S, Believe,
               shareKey(B,Kab,A));
ASSUMPTIONS
  HOLDS A:Kas;
  BELIEVES A: SECRET Kas:A,S;
  HOLDS A: Na;
  BELIEVES A: fresh(Na);
  BELIEVES A: jurisdiction(A,
      st(A,Believe,allStatement));
  BELIEVES A: jurisdiction(A,
      shareKey(A,Kab,B));
  HOLDS B: Kbs;
  BELIEVES B: shareKey(B,Kbs,S);
  HOLDS B :Nb;
  BELIEVES B: fresh(Nb);
  BELIEVES B: jurisdiction(S,
      st(A,Believe,allStatement));
  BELIEVES B: jurisdiction( S,
               shareKey(A,Kab,B));
  BELIEVES A: recognised(B);
  BELIEVES B: recognised(A);

MESSAGES
 A->S: A,B,Na;
 S->A: notHere({ext(Na,B,Kab,{ext
    (Kab,A,Stmt2)}Kbs,Stmt1)}Kas);
 A->B: notHere({ext(Kab,A,
                      Stmt2)}Kbs);
 B->A: notHere({Nb}Kab);
 A->B:notHere({decrement(Nb)}Kab);

GOALS
 HOLDS A: Kab;
 BELIEVES A: shareKey(A,Kab,B);
 BELIEVES A: convey(B, {Nb}Kab);
 BELIEVES A: fresh(B, {Nb}Kab);
 HOLDS B: Kab;
 BELIEVES B: shareKey(B,Kab,A);
 BELIEVES B:convey(A,{
               decrement(Nb)}Kab);
END;
```

Fig. 6: Sample Extended CAPSL Specification

This paper proposed extensions to the CAPSL specification language in the form of typespecs. These extensions allow the use of CAPSL as a specification language for GNY-based verifications. Such a formal specification can be used as the input to an automated proving engine, where the translation from the CAPSL protocol specification into the logic-specific specification is

automated. Similar extensions of CAPSL can be developed for other verification logics, to allow the use of CAPSL as a specification language for these logics.

6. Acknowledgements

The authors gratefully acknowledge the support of the Irish Research Council for Science, Engineering and Technology who funded this work under the IRCSET Basic Research Award Scheme (grant SC02/237).

References:

[1] Coffey, T., Dojen, R. and Flanagan, T., "Formal Verification: An Imperative Step in the Design of Security Protocols", Computer Networks Journal, Elsevier Science, Vol. 43, No.5, December 2003, pp.601-618.

[2] Kessler, V. and Wendel, G., "AUTLOG – An advanced logic of authentication", Proceedings of 7th IEEE Computer Security Foundations, Menlo Park, California, USA, August 1994, pp.90-99.

[3] Brackin, S., "Automatically detecting most vulnerabilities in Cryptographic protocols", DARPA Information Survivability Conference and Exposition Vol.1, Hilton Head, South Carolina, USA, January 2000, pp.222-236.

[4] Sverson P. and Meadows C., "Formal requirements for Key distribution protocols", Proceedings of Advances in Cryptology - EUROCRYPT'94, Perugia, Italy, May 1995, pp.320-331.

[5] Huima, A. "Efficient infinite-state analysis of security protocols", Proceedings of FLOC'99 Workshop on Formal Methods and Security Protocols, July 1999

[6] Gürgens, S. and Rudolph, C., "Security Analysis of (Un-)Fair Non-repudiation Protocols", Formal Aspects of Security, LNCS 2629, 2002, pp.97-114.

[7] Healy, K., Coffey, T. and Dojen, R., "A Comparative Analysis of State-Space Tools for Security Protocol Verification", WSEAS Transactions on Information Science and Applications, Volume 1, Issue 5, November 2004, pp.1256-1261.

[8] Dojen, R. and Coffey, T., "Layered Proving Trees: A Novel Approach to the Automation of Logic-Based Security Protocol Verification", ACM Transactions on Information and System Security (TISSEC), Volume 8, Issue 3, August 2005, pp.287-311.

[9] Meadows, C., "What Makes a Cryptographic Protocol Secure? The Evolution of Requirements Specification in Formal Cryptographic Protocol Analysis", Proceedings of ESOP 03, Springer-Verlag, April 2003.

[10] Dojen, R. and Coffey, T., "A Novel Approach to the Automation of Logic-Based Security Protocol Verification", WSEAS Transactions on Information Science and Applications, Vol. 1, No. 5, November 2004, pp.1243-1247.

[11] Burrows, M. Abadi, M, and Needham, R., "A logic of authentication", ACM Operating System Review Vol. 23, No.5, 1989, pp.1–13.

[12] Gong, L., Needham, R. and Yahalom, R., "Reasoning about belief in cryptographic protocols", Proceedings of IEEE Computer Society Synopsis on Research in Security and Privacy, 1990, pp.234–248.

[13] Coffey, T. and Saidha, P., "A logic for verifying public key cryptographic protocols", IEE Journal of Proceedings—Computers and Digital Techniques Vol. 144, No.1, 1997, pp.28–32.

[14] Zhang, Y. and Varadharajan, V., "A logic for modeling the dynamics of beliefs in cryptographic protocols", Australasian Computer Science Conference, 2001.

[15] Millen J., "CAPSL: Common Authentication Protocol Specification Language", Technical Report MP 97B48, The MITRE Corporation, 1997.

[16] Millen, J. and Denker, G., "MuCAPSL", DISCEX III, DARPA Information Survivability Conference and Exposition. IEEE Computer Society, 2003, pp.238–249.

[17] Gong, L. "Handling infeasible specifications of cryptographic protocols", 4th Computer Security Foundation Workshop, 1991, pp.99-102.

[18] Mathuria, A., Safavi-Naini, R. and Nickolas, P., "Some remarks on the logic of Gong, Needham and Yahalom", International Computer Symposium, Hsinchu, Taiwan, R.O.C, Vol. 1, 1994, pp.303-308.