Kokichi Futatsugi
Fumio Mizoguchi
Naoki Yonezaki (Eds.)

LNCS 3233

Software Security – Theories and Systems

Second Mext-NSF-JSPS International Symposium, ISSS 2003
Tokyo, Japan, November 2003
Revised Papers

Springer

Kokichi Futatsugi Fumio Mizoguchi
Naoki Yonezaki (Eds.)

Software Security – Theories and Systems

Second Mext-NSF-JSPS International Symposium, ISSS 2003
Tokyo, Japan, November 4-6, 2003
Revised Papers

Springer

Volume Editors

Kokichi Futatsugi
Japan Advanced Institute of Science and Technology (JAIST)
1-1 Asahidai, Tatsunokuchi, Nomi, Ishikawa 923-1292, Japan
E-mail: kokichi@jaist.ac.jp

Fumio Mizoguchi
Tokyo University of Science, Information Media Center
2641 Yamazaki, Noda, Chiba, 278-8510, Japan
E-mail: mizo@ia.noda.tus.ac.jp

Naoki Yonezaki
Tokyo Institute of Technology
Graduate School of Information Science and Engineering
Department of Computer Science
2-12-1-W8-67 Ookayama, Meguro-ku, Tokyo 152-8552, Japan
E-mail: yonezaki@cs.titech.ac.jp

Library of Congress Control Number: 2004114314

CR Subject Classification (1998): C.2.0, D.4.6, E.3, D.3, F.3, K.6.5

ISSN 0302-9743
ISBN 3-540-23635-X Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11340331 06/3142 5 4 3 2 1 0

Preface

Following the success of the International Symposium on Software Security 2002 (ISSS 2002), held in Keio University, Tokyo, November, 2002, ISSS 2003 was held in the Tokyo Institute of Technology, Tokyo, on November 4–6, 2003. This volume is the collection of the papers that were presented at ISSS 2003. The proceedings of ISSS 2002 was published as LNCS 2609.

Although the security and reliability of software systems for networked computer systems are major concerns of current society, the technology for software security still needs to be developed in many directions. Similar to ISSS 2002, ISSS 2003 aimed to provide a forum for research discussions and exchanges among world-leading scientists in the fields of both theoretical and systems aspects of security in software construction.

The program of ISSS 2003 was a combination of invited talks and selected research contributions. It included the most recent visions and researches of the 9 invited speakers, as well as 11 contributions of researches funded by the MEXT grant-in-aid for scientific research on the priority area "Implementation Scheme for Secure Computing" (AnZenKaken). We collected the original contributions after their presentation at the symposium and began a review procedure that resulted in the selection of the papers in this volume. They appear here in final form.

ISSS 2003 required a lot of work that was heavily dependent on members of the program committee, and staffs and graduate students who participated in AnZenKaken. We sincerely thank them for their efforts and time.

June 2004

Kokichi Futatsugi
Fumio Mizoguchi
Naoki Yonezaki

Organization

Program Committee

Kokichi Futatsugi (Co-chair)	JAIST
David Gilliam	Jet Propulsion Laboratory
Masami Hagiya	University of Tokyo
Fumio Mizoguchi (Co-chair)	Tokyo University of Science
Aloysius Mok	University of Texas at Austin
George Necula	University of California at Berkeley
Mitsuhiro Okada	Keio University
John Rushby	SRI International
Etsuya Shibayama	Tokyo Institute of Technology
Andre Scedrov	University of Pennsylvania
Naoki Yonezaki (Co-chair)	Tokyo Institute of Technology

Organization Committee

Kokichi Futatsugi	JAIST
Fumio Mizoguchi	Tokyo University of Science
Mitsuhiro Okada	Keio University
Benjamin Pierce	University of Pennsylvania
Andre Scedrov	University of Pennsylvania
Naoki Yonezaki	Tokyo Institute of Technology
Akinori Yonezawa (Chair)	University of Tokyo

Sponsors

Japanese Ministry of Education, Science and Culture (MEXT)
National Science Foundation (NSF)
Japanese Society for the Promotion of Science (JSPS)

Table of Contents

Verifying Confidentiality and Authentication in Kerberos 5*

Frederick Butler[1,3], Iliano Cervesato[2], Aaron D. Jaggard[1,4], and Andre Scedrov[1]

[1] Department of Mathematics, University of Pennsylvania, 209 South 33rd Street, Philadelphia, PA 19104–6395 USA
scedrov@saul.cis.upenn.edu

[2] ITT Industries, Inc., Advanced Engineering and Sciences, 2560 Huntington Avenue, Alexandria, VA 22303 USA
iliano@itd.nrl.navy.mil

[3] Department of Mathematics, Armstrong Hall, P.O. Box 6310, West Virginia University, Morgantown, WV 26506-6310 USA

[4] Department of Mathematics, Tulane University, 6823 St. Charles Avenue, New Orleans, LA 70118 USA
adj@math.tulane.edu

Abstract. We present results from a recent project analyzing Kerberos 5. The main expected properties of this protocol, namely confidentiality and authentication, hold throughout the protocol. Our analysis also highlights a number of behaviors that do not follow the script of the protocol, although they do not appear harmful for the principals involved. We obtained these results by formalizing Kerberos 5 at two levels of detail in the multiset rewriting formalism MSR and by adapting an inductive proof methodology pioneered by Schneider. Our more detailed specification takes into account encryption types, flags and options, error messages, and a few timestamps.

1 Introduction

Over the last few years we have pursued a project intended to give a precise formalization of the operation and objectives of Kerberos 5 [1–3], and to determine whether the protocol satisfies these requirements. Our initial results were reported in [4]. A detailed and complete account of this work can be found in [5]. This paper is instead intended as a high-level summary of the goals, methods and outcome of the overall project.

* Scedrov, Butler, and Jaggard were partially supported by the DoD University Research Initiative (URI) program administered by the Office of Naval Research under Grant N00014-01-1-0795, and by NSF Grant CCR-0098096. Cervesato was partially supported by NRL under contract N00173-00-C-2086. This paper was written while Cervesato was visiting Princeton University.

We adopted a hierarchical approach to formalizing such a large and complex protocol, and gave a base specification and two orthogonal refinements of it. These may be thought of as a fragment of a family of refinements of our base specification, including a common refinement whose detail would approach that of pseudocode. Notably, we were able to extend the theorems and proofs for our most abstract formalization to one of our more detailed formalizations by adding detail.

Our base specification, which we refer to as our 'A level' formalization, contains enough detail to prove authentication and confidentiality results but omits many of the details of the full protocol. The first ('B level') refinement of the A level specification adds selected timestamps and temporal checks to the base specification; while these are an important part of the protocol, the B level formalization did not yield as many interesting results and is omitted from further discussion here. See [5] for details. We refer to our second refinement of the base specification as our 'C level' formalization, although it neither refines nor is refined by the B level formalization. The C level description of Kerberos adds encryption types, flags and options, checksums, and error messages to the core exchanges included in the A level.

Our analysis concentrated on the confidentiality of session keys and on the data origin authentication of tickets and other information as the main requirements at the A and C levels. The initial report [4] of this work included some of these properties for the A level; we have extended this work both to other parts of the A level and to parallel theorems in the C level formalization. We also found various anomalies—curious protocol behaviors that do not appear to cause harm, and that do not prevent the protocol from achieving authentication. We summarize those findings here, including details of an anomaly not noted in [4].

Background. Kerberos [1–3] is a widely deployed protocol, aimed at repeatedly authenticating a client to multiple application servers based on a single login. Kerberos makes use of various tickets, encrypted under a server's key and unreadable the user, which when forwarded in an appropriate request authenticate the user to the desired service. A formalization of Kerberos 4, the first publicly released version of this protocol, was given in [6] and then extended and thoroughly analyzed using an inductive approach [7–10]. This analysis, through heavy reliance on the Isabelle theorem prover, yielded formal correctness proofs for a specification with timestamps, and also highlighted a few minor problems. A simple fragment of the more recent version, Kerberos 5, has been investigated using the state exploration tool Murφ [11].

Methodology. We used the security protocol specification language MSR to formalize Kerberos 5 at the various levels of abstraction we intend to consider. MSR [12–15] is a simple, logic-oriented language aimed at giving flexible specifications of complex cryptographic protocols. It uses strongly-typed multiset rewriting rules over first-order atomic formulas to express protocol actions and relies on a form of existential quantification to symbolically model the generation of nonces, session keys and other fresh data. The framework also includes static checks such as type-checking and data access verification to limit specification oversights. In contrast to other meta-security systems, MSR offers an open and uncommitted logical basis that makes it possible to reason about and experiment with a wide variety of specification and verification techniques in a sound manner. While MSR resembles the inductive method used to analyze Kerberos 4 in that both model state transitions in a protocol run, MSR is tailored to specifying protocols because of its primitive notion of freshness and its formal semantics that captures the notion of transition. It is thus of interest to connect this specification language to methods of proof; indeed, this project was also intended as a test-bed for MSR on a real-world protocol prior to embarking in a now ongoing implementation effort, and to explore forms of reasoning that best take advantage of the linguistic features of MSR. The experience gathered during this project was very positive.

The verification methods we used were inspired by Schneider's notion of rank function [16] and also influenced by the inductive theorem proving method Paulson et al. applied to the verification of Kerberos 4 [7–10]. For each formalization of Kerberos 5 that we analyzed, we defined two classes of functions, rank and corank, to capture the essence of the notions of authentication and confidentiality, respectively. Our experience with these functions suggests that their (currently manual) generation might be automated and that they might be embedded in an MSR-based proof assistant.

Results. We proved authentication properties for both tickets (Ticket-Granting and Service) used in Kerberos 5 and the associated encrypted keys that the client sends with each ticket in a request to a server. In doing so, we used separate confidentiality properties that we proved for the two session keys generated during a standard protocol run. We have proved the confidentiality and authentication properties for the first key and ticket in both our A and C level formalizations, and we have shown that the parallel properties for the second key/ticket pair hold in our A level formalization; we expect these to also hold in the C level formalization. Table 1 (in Sect. 4.1) shows which property statements here correspond to the two property types and two protocol exchanges. In this report we state the protocol properties in language that is applicable to both formalizations; for those properties proved in both, deleting details from the corresponding C level theorem gives the A level theorem statement, and the proofs show similar parallels.

We also uncovered a handful of anomalous behaviors, which are interesting curiosities rather than practical attacks: in the A level "ticket switch" anomaly, an intruder corrupts a ticket on its way to the client but restores it as it is about to be used. This was refined to the "anonymous ticket switch" anomaly at the C level. The "encryption type" anomaly is found only in the C level and has to do with the intruder hijacking requests by manipulating their encryption type fields. After these anomalies were reported in [4], we found the "ticket option anomaly," whose effects generalize those of the anonymous ticket switch anomaly although the actions of the intruder are simpler than those originally described for the anonymous ticket switch anomaly; we describe this anomaly here in addition to reviewing the other anomalies that we found.

Organization of This Paper. We start in Sect. 2 with a high level description of Kerberos 5 and a discussion of which details are included in our A and C level formalizations. Section 3 outlines MSR and the definition of the rank and corank functions that we used in our analysis. In Sect. 4.1 we turn to our positive results on confidentiality and authentication in Kerberos 5; Sect. 5 discusses curious protocol behavior we have seen in our work. Finally, Sect. 6 outlines possible directions for extensions of this work.

The appendices contain MSR rules comprising our formalizations of Kerberos 5. Appendices A and B give the A level description of behavior by honest protocol participants and the intruder, respectively. Appendices C and D do the same for our C level formalization.

2 Overview of the Kerberos 5 Protocol

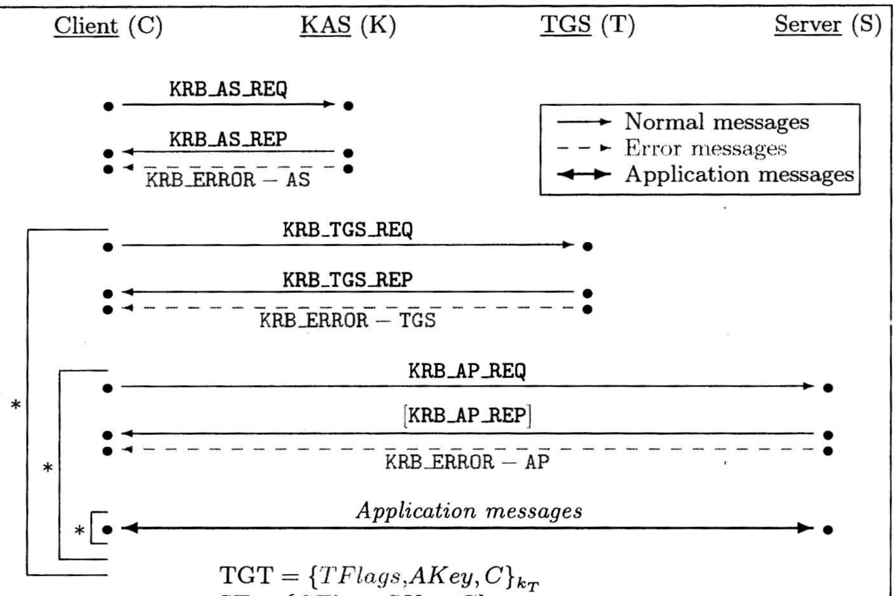
2.1 Intuitive Description

A standard Kerberos 5 run provides a client C with convenient means to repeatedly authenticate herself to an application server S. To this end, she obtains a *service ticket* from a third party known as a *Ticket Granting Server* or TGS. For added flexibility, authentication to TGS is implemented by presenting a *ticket granting ticket* to a fourth party, the *Kerberos Authentication Server* or KAS.

Therefore, the core of the protocol consists of a succession of three phases outlined in Fig. 1 (the color conventions and the various fields will be explained shortly). We will now describe them in succession:

- In the first phase, C sends a request KRB_AS_REQ to a KAS K for a ticket-granting ticket TGT, for use with a particular TGS T. K replies with a message KRB_AS_REP consisting of the ticket TGT and an encrypted component containing a fresh *authentication key* $AKey$ to be shared between C and T. TGT is encrypted using the secret key k_T of T; the accompanying message is encrypted under C's secret key k_C.
- In the second phase, C forwards TGT, along with an *authenticator* encrypted under $AKey$, to the TGS T (message KRB_TGS_REQ). T responds in KRB_TGS_REP by sending a service ticket ST encrypted under the secret key k_S of the application server S, and a component containing a *service key* $SKey$ to be shared between C and S, encrypted under $AKey$.
- In the third phase, C forwards ST and a new authenticator encrypted with $SKey$, in message KRB_AP_REQ to S. If all credentials are valid, this application server will authenticate C and provide the service. The acknowledgment message KRB_AP_REP is optional.

A single ticket-granting ticket can be used to obtain several service tickets, possibly for several application servers, while it is valid. Similarly, one service



$$\mathrm{TGT} = \{TFlags, AKey, C\}_{k_T}$$
$$\mathrm{ST} = \{SFlags, SKey, C\}_{k_S}$$
$$MD = [TOpts, C, S, n_2, e]_{AKey}$$
$$MD' = [\ldots]_{SKey}$$

KRB_AS_REQ : $KOpts, C, T, n_1, e$
KRB_AS_REP : $C, \mathrm{TGT}, \{AKey, n_1, TFlags, T\}^{e}_{k_C}$
KRB_TGS_REQ : $\mathrm{TGT}, \{C, MD, t_{C,Treq}\}_{AKey}, TOpts, C, S, n_2, e$
KRB_TGS_REP : $C, \mathrm{ST}, \{SKey, n_2, SFlags, S\}^{e}_{AKey}$
KRB_AP_REQ : $SOpts, \mathrm{ST}, \{C, MD', t_{C,Sreq}\}_{SKey}$
KRB_AP_REP : $\{t_{C,Sreq}\}_{SKey}$
KRB_ERROR—X : KRB_ERROR, $[-|t_{C,Treq}|t_{C,Sreq}], t_{(K|T|S),err},$
$ErrCode, C, (K|T|S)$

Fig. 1. Kerberos 5 Messages in the A level and C level Formalizations of Kerberos 5

ticket can be used to repeatedly request service from S before it expires. In both cases, a fresh authenticator is required for each use of the ticket.

The protocol run described above is very similar to that of Kerberos 4. The primary difference between the two versions (aside from some options available in version 5 and not in version 4) is the structure of the KRB_AS_REP and KRB_TGS_REP messages. In version 4 the ticket-granting ticket is sent by the KAS as part of the message encrypted under the client's secret key k_C, and the service ticket sent by the TGS is likewise encrypted under the shared key$AKey$. In version 5, we see that the ticket-granting ticket and the service ticket are sent without further encryption; this enables the cut and paste anomalies which we describe below.

Note that the Kerberos 5 protocol has changed since the initial specification [1]. Here we use version 10 [3] of the revisions to [1]; changes include the addition of anonymous tickets, although these may or may not be present in future revisions of the protocol [17]. The description of the protocol is an IETF Internet Draft, each version of which has a six month lifetime. The current version is [18].

2.2 A and C Level Formalizations

The fragment of Kerberos 5 we adopt for our A level specification consists of the minimum machinery necessary to implement the above presentation. It is summarized in black ink in Fig. 1 (please ignore the parts in gray) and fully formalized in [5], with salient features reported in App. A.

We adopt standard conventions and use the comma "," to denote concatenation of fields, and expressions of the form $\{m\}_k$ for the result of encrypting the message m with the key k. We do not distinguish encryption algorithms at this level of detail. We usually write n, possibly subscripted, for nonces, and t, similarly decorated, for timestamps.

Our C level specification extends the A level by adding additional fields and messages, displayed in gray) in Fig. 1. These additions include error messages, option fields (used to by the client in her requests), flag fields (used by servers to describe the options granted), keyed checksums, and explicit consideration of encryption types. The C level formalization is explored in App. C.

3 Protocol Analysis in MSR

MSR originated as a simple logic-oriented language aimed at investigating the decidability of protocol analysis under a variety of assumptions [12, 15]. It evolved into a precise, powerful, flexible, and still relatively simple framework for the specification of complex cryptographic protocols, possibly structured as a collection of coordinated subprotocols [14]. We will introduce the syntax and operations of MSR as we go along.

A central concept in MSR is that of a *fact*—protocols are expressed in MSR as rules which take multisets of these objects, whose precise definition we omit for the moment, and rewrite them as different multisets. MSR facts are a formal representation of network messages, stored information, and the internal states of protocol participants. Protocol runs are represented as sequences of multisets of facts, each obtained from the previous one by applying one of the rules in the formalization of the protocol.

3.1 Protocol Signature

A protocol specification in MSR requires type declarations classifying the terms that might appear in the protocol. We include the types KAS, TGS, and server for authentication, ticket-grating, and applications servers, respectively; these interact with $\mathsf{clients}$, the fourth type of protocol participant. Each of these is subtype of a generic $\mathsf{principal}$ type, and the names of each may be used in network messages. Thus $\mathsf{principal} <: \mathsf{msg}$ and $\mathsf{KAS} <: \mathsf{principal}$, *etc.*

Other types we use include nonces, timestamps, and keys. We use some auxiliary types to allow shared keys to be included in messages while prohibiting long-term database keys from being sent over the network. For example, given $C : \mathsf{client}$ and $T : \mathsf{TGS}$, the shared key type $\mathsf{shK}\ C\ T$ is a subtype of msg while the database key type $\mathsf{dbK}\ C$ is not.

Our C level formalization uses an expanded signature which includes options (subtypes of Opt), which a client uses to modify the default request to a server, and flags (subtypes of Flag), which describe the options actually granted by a server. The C level also adds encryption types, which allow a client to specify the encryption method(s) she would like to use; in this formalization, the encryption type is an additional parameter to the different types of keys.

Syntactically, A level non-atomic messages are either the concatenation m_1, m_2 of two other messages or the result of (symmetrically) encrypting another message m_1 with a key k; we denote the resulting message by $\{m_1\}_k$. As noted above, the C level formalization includes message digests (cryptographic hashes). The digest and encryption of m_1 using k are both parameterized by an encryption type e and denoted by $[m_1]_k^e$ and $\{m_1\}_k^e$, respectively; here k and e must be compatible, *e.g.*, $k : \mathsf{shK}^e\ C\ T$ or $k : \mathsf{dbK}^e\ C$. We will keep the encryption type implicit unless we are specifically discussing it. In our present formalizations, we only use shared keys and not database keys to construct message digests.

3.2 State and Roles

Intuitively, MSR represents the state of execution of a protocol as a multiset S of ground first-order formulas (the 'facts' mentioned earlier). Some predicates are universal: in particular, $\mathsf{N}(m)$ indicates that message m is transiting through the network. Other predicates are protocol-dependent and are classified as memory or role state predicates. *Memory predicates* are used to store information across several runs of a protocol, to pass data to subprotocols, and to invoke external modules. The intruder I stores intercepted information m in the predicate $\mathsf{I}(m)$. We will encounter other memory predicates as we go along. *Role state predicates*, of the form $L(\ldots)$, allow sequentializing the actions of a principal.

Principals cause local transformations to this global state S by non-deterministically executing *multiset rewriting rules* of the form $r = \mathit{lhs} \longrightarrow \mathit{rhs}$, where *lhs* is a finite multiset of facts and some number of constraints (which are not facts). These constraints are used by principals to check system clocks or determine the validity of requests via external processes not explicitly modelled here. Whenever the facts in *lhs* are contained in S and the constraints are all satisfied, rule r can replace these facts with those from *rhs*. The actual definition is slightly more general in the sense that rules are generally parametric and *rhs* may specify the generation of nonces or other data before rewriting the state.

The rules comprising a protocol or a subprotocol are collected in a *role* parameterized by the principal executing it. Rules in a role are threaded through using role state predicates declared inside the role.