

Verifying Confidentiality and Authentication in Kerberos 5 ^{*}

Frederick Butler^{1,3}, Iliano Cervesato², Aaron D. Jaggard^{1,4}, and Andre
Scedrov¹

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 & 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.

* 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.

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.

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 Mur φ [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 provides an overview of MSR and the definition of the rank and corank functions that we use 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 the 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

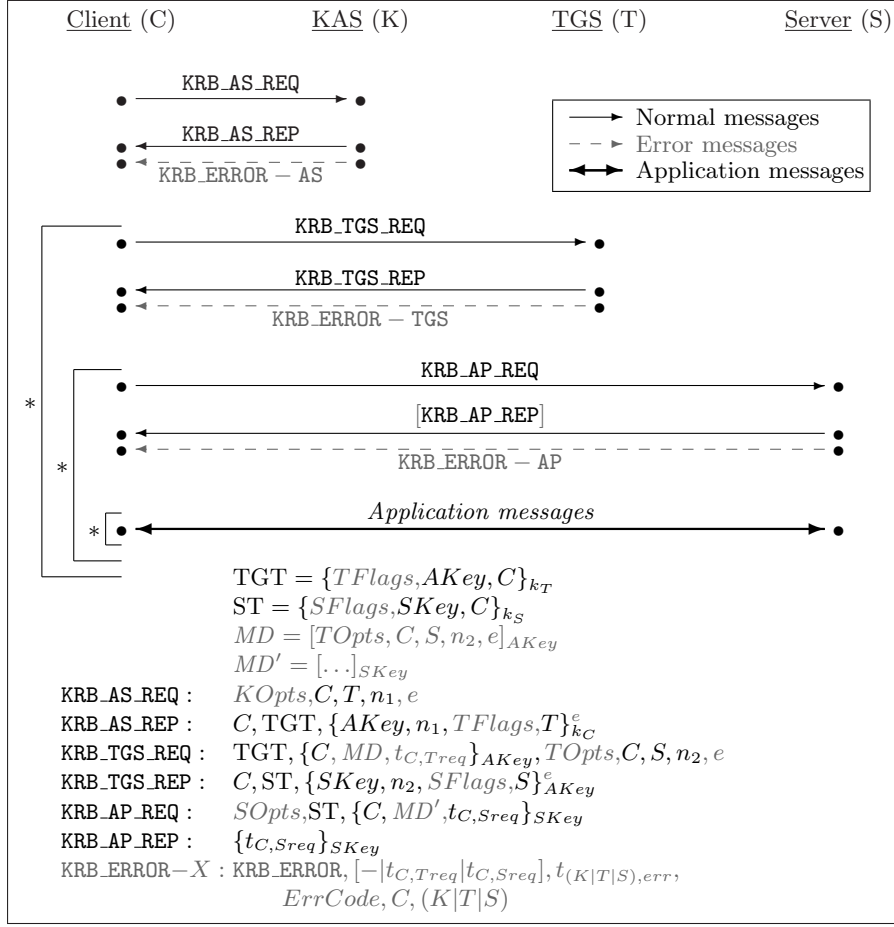


Fig. 1. Kerberos 5 Messages in the A level and C level Formalizations of Kerberos 5

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 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 changes 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 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 detailed in App. C.

3 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 clients, the fourth type of protocol participant. Each of these is subtype of a generic **principal** type, and the names of each may be used in network messages. Thus **principal** $<$: **msg** and **KAS** $<$: **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 : **client** and T : **TGS**, the shared key type **shK** C T is a subtype of **msg** while the database key type **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 : **shK** ^{e} C T or k : **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, **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 **I**(m). We will encounter other memory predicates as we go along. *Role state predicates*, of the form **L**(\dots), 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 $r = lhs \longrightarrow rhs$, 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.

3.3 Rank and Corank Functions

Inspired by Schneider’s analysis of the amended Needham-Schroeder protocol [16], we define rank and corank functions on facts and multisets of facts.

Given a key k and message m_0 , the k -rank relative to m_0 captures the maximum number of nested encryptions (or keyed message digestions in the C level formalization) of the message m_0 using the key k which appear in a term; the innermost encryption/digestion must be exactly $\{m_0\}_k$ or $[m_0]_k$, *i.e.*, m_0 encrypted/digested using k . We then define the rank of a fact $P(t_1, \dots, t_j)$ to be the maximum value of the rank of the t_i ; intuitively, the number of times m_0 is encrypted by k to obtain this fact can be obtained by looking at the arguments to the predicate P . The rank of a multiset of finitely many distinct facts is the maximum rank of a fact in the multiset.

We authenticate the origin of $\{m_0\}_k$ or $[m_0]_k$ by showing that a specified protocol participant may create a fact F of k -rank 1 relative to m_0 but that no other participants, including the intruder, can rewrite a multiset M of facts so that the resulting multiset M' has greater k -rank relative to m_0 than M did. If no facts at the beginning of the trace had positive k -rank, then the appearance of a fact of positive k -rank in some multiset of the trace implies that the protocol participant created F at some point in the trace.

For a set E of keys and atomic message m_0 , the E -corank of a term relative to m_0 is the minimum number of decryptions using keys from E needed to extract the message m_0 from the term. We then define the corank of a fact $P(t_1, \dots, t_j)$ to be the minimum corank of an argument t_i that may be placed back on the network; this allows, *e.g.*, protocol participants to store keys in memory without using facts whose corank equals 0. Corank is defined for a multiset of facts as the minimum corank of a fact appearing in the multiset.

Note that the fact $!m_0$ that corresponds to the intruder’s knowledge of m_0 has E -corank equal to 0 relative to m_0 for every set E of keys. We may thus use corank to prove the confidentiality of some m_0 by showing that for some set E of keys, no facts of E -corank 0 relative to m_0 ever appear in a protocol trace.

4 Properties of Kerberos 5

4.1 Overview

We have established *confidentiality* and *data origin authentication* properties for Kerberos 5. The latter are important because Kerberos claims to provide authentication; the former are used in proving authentication and are also important

because some of the session keys which we show to be confidential may be used for additional (post-protocol run) communication between protocol participants.

These types of properties hold in both the Ticket-Granting and Client/Server exchanges. As these sub-protocols have similar structure, it is unsurprising that the properties are expressed and proved in very similar ways. Table 1 shows the parallel relationships between the properties that we have established so far—each exchange has a corresponding confidentiality property and a corresponding authentication property. The confidentiality properties give conditions under which an intruder never learns certain information, while the authentication properties state conditions under which, if certain messages appear on the network, then these messages originated with specific principals. These properties are described in more detail for the Ticket-Granting Exchange in Sec. 4.2 and for the Client/Server Exchange in Sec. 4.3. Throughout this work, we assume the presence of a Dolev-Yao intruder. Additionally, we do not intentionally leak keys to this intruder as was done in [7–9].

Table 1. Properties established for Kerberos 5

	Confidentiality	Authentication
T-G Exchange	Property 1	Property 2
C/S Exchange	Property 3	Property 4

The precise statements (in terms of MSR rules and facts) and proofs of these properties are related in much the same way that our formalizations themselves are related—removing some information from the detailed version gives the more abstract version. As a result, we are optimistic that we will be able to extend the properties shown for the Client/Server Exchange to our detailed formalization.

4.2 Properties of the Ticket-Granting Exchange

As the Ticket-Granting Exchange is closer to the beginning of the protocol run, these properties are slightly simpler than for the Client/Server Exchange below. We have proved these properties for both the A and C level formalizations.

Confidentiality of *AKey*. The first property that we have established for Kerberos 5 is the confidentiality of the session key generated by the Authentication Server, *i.e.*, that the intruder does not learn this key.

Property 1. If the intruder does not know the long term secret keys (k_C and k_T) used to encrypt the session key *AKey* generated by the authentication server *K* for use by *C* and *T*, then the intruder cannot learn *AKey*.

Authenticity of the Ticket-Granting Ticket and Authenticator. The second property of Kerberos 5 is data origin authentication of the ticket and authenticator used in the client's request to the ticket granting server.

Property 2. If the intruder does not know the long term key used to encrypt a ticket-granting ticket TGT and this ticket didn't initially exist, then if the TGS processes a request, ostensibly from a client C , containing the ticket-granting ticket TGT and the session key $AKey$, then some Authentication Server created the session key $AKey$ for C to use with the TGS and also generated TGT . Furthermore, if the intruder does not know the long term key that the authentication server used to send $AKey$ to the client, then the authenticator was created by this particular client.

4.3 Properties of the Client/Server Exchange

We now move to properties of the Client/Server Exchange; as this exchange parallels the Ticket Granting Exchange, the properties parallel the properties we have proved for that exchange. These properties build on those stated above and may be viewed as the main positive results that we have obtained. Thus far, we have only proved these properties for our A level formalization, although we expect these to hold for the C level formalization as well.

Confidentiality of $SKey$. The first property for the Client/Server Exchange is that the session key shared by the client and server is not known to the intruder.

Property 3. If the intruder knows neither the long term secret key used by a TGS to encrypt the service ticket ST containing a new session key $SKey$ for a client to use with a server nor the session key used by the client to request the service ticket, then the intruder cannot learn $SKey$.

Authenticity of the Service Ticket and Authenticator. The second property in this exchange is the data origin authentication of the ticket and authenticator included in the client's request to the server.

Property 4. If the intruder does not know the long term key used to encrypt a service ticket ST and this ticket did not initially exist, then if a server S processes a request, ostensibly from a client C , containing the service ticket ST and the session key $SKey$, then some Ticket Granting Server generated the session key $SKey$ for C to use with S and also created ST . Furthermore, if the intruder never learns the session key which the Ticket Granting Server used to encrypt $SKey$ when replying to the client's request for a service ticket, then this client created the authenticator.

5 Anomalies

5.1 Overview of Anomalies

In our analysis of Kerberos 5, we found four anomalies that we review these here. Three of these were described in [4]; we give a more detailed description of the forth in Sect. 5.2. While this behavior deviates from the intended protocol behavior, these anomalies do not compromise the security of the protocol. As noted by Jeffrey [19], some implementations of Kerberos 5 do not sufficiently guard against replays, allowing an intruder to force repeated generation of fresh credentials by replaying requests from a client. This concern is separate from the anomalies noted here, and we leave further discussion of it to the more detailed report [5] of our work.

The first anomaly can be realized in our A level formalization, while the remaining anomalies require the detail of our C level formalization. The ticket option anomaly further requires the detection of replays; this is not explicitly included in the C level, although generic error detection is.

In the ‘ticket anomaly’ an intruder I intercepts the `KRB_AS_REP` message from some KAS K and replaces the ticket-granting ticket with a garbage message. I then intercepts subsequent `KRB_TGS_REQ` messages from the client and restores the original ticket, allowing the protocol to proceed normally. This behavior is undetectable by the client because she expects to be unable to read the encrypted ticket. However, she has false beliefs about the data in her possession, namely that she holds a valid ticket when in fact she does not. While the origin of the ticket and authenticator may be authenticated, unlike in Kerberos 4 [7] we cannot prove that the entirety of a valid `KRB_TGS_REQ` message originated with the client.

In the ‘anonymous ticket switch anomaly’ a client requests two service tickets, one anonymous and one non-anonymous, from a TGS T for use with a server S . An intruder I may intercept the two `KRB_TGS_REP` messages and switch the tickets, forwarding the resulting messages to the client. Exactly one of the tickets the client receives contains her identity (the other is anonymous), but the client is mistaken in belief about which of the tickets this is. The client then sends two `KRB_AP_REQ` messages to S , one with and one without her name (but matched to the wrong ticket) and not requesting mutual authentication in either one. The intruder intercepts both of these messages and replaces the authenticator without the client’s name with the other authenticator, forwarding the resulting messages to S . S may read both tickets, but only the key from the non-anonymous ticket opens the accompanying authenticator. S then accepts the request with the non-anonymous ticket and rejects the request with the anonymous ticket, returning an error message to the client. I may tamper with this message so that it contains the client’s name; she then correctly believes that exactly one of her requests was accepted, but incorrectly believes that S has not seen her name.

The ‘ticket replay anomaly,’ described in more detail below, has effects similar to the anonymous ticket switch anomaly. Again the intruder intercepts an `KRB_AP_REQ` messages to a server S , but now she simply replays it to produce

a replay error from the server; she may then tamper with this error message at will, essentially using the server to unpack the timestamp from the request to make the new error message appear authentic. If this is done in conjunction with intercepting a second `KRB_AP_REQ` message from the client to S , the client might believe (as in the anonymous ticket switch anomaly) that S has accepted exactly one of her requests but have an incorrect belief about which one.

These first three anomalies all involve separating tickets from the rest of the message containing them. This was not possible in Kerberos 4, where the tickets were included with other data encrypted by a server for the client. This change in message structure between the two versions of the protocol allows these anomalies but does not affect the basic authentication provided by Kerberos.

In the ‘encryption type anomaly’ we assume that a client C has different long-term keys for use with different encryption methods, that one of these has been learned by the intruder I , and that C knows this key has been compromised. When sending a `KRB_AS_REQ` message to some KAS K , C would then request that the response be encrypted using an encryption method other than the one corresponding to the lost key. As this message is unencrypted, I may change the encryption type field to specify the encryption method corresponding to the lost key. K ’s response will then be encrypted using the lost key and thus be readable by I , who may then impersonate C . While I could do this anyway knowing C ’s long-term key, we see that such impersonation is possible even if C takes steps to avoid using the lost long-term key.

5.2 Ticket Replay Anomaly

We now examine the ticket replay anomaly in detail; similar discussions of the other replays we found are given in the detailed report of our work [5].

Figure 2 updates Fig. 1 to show the message flow in this anomaly. The client C initiates and completes the Authentication Service Exchange with a KAS K , obtaining a ticket-granting ticket TGT for the TGS T . She then uses this ticket to make two requests for service tickets for a single server S , requesting different options for these two service tickets.

T receives these two requests and grants two different service tickets ST and ST' with associated session keys $SKey$ and $SKey'$; we assume that the options actually granted by the TGS are different for these two tickets. Recall that the TGS sends a copy of the granted options along with the new session key (both encrypted under the session key shared by the client and the TGS), so the client associates the different granted options with these different keys. The client then sends two requests to the server, one with ST and an authenticator encrypted using $SKey$ and containing a timestamp t and other with ST' , $SKey'$ and t' , respectively. We assume that in both requests, the client does not request mutual authentication from the server, so she expects a response only in case of an error.

The intruder I intercepts these requests. She duplicates the request containing ST , $SKey$, and t and forwards these to the server, who accepts the first and rejects the second because of the replayed authenticator. This prompts an error message, containing t , from the server, which the intruder may intercept, modify,

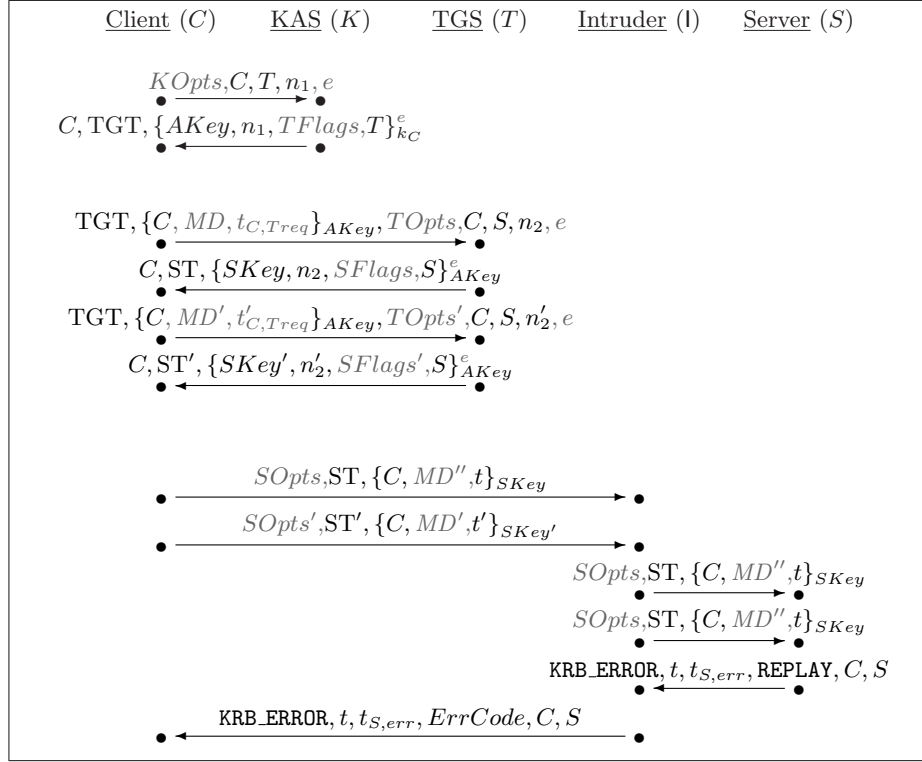


Fig. 2. Message flow in the ticket replay anomaly

and send to the client. The intruder does not send the second request, containing ST' , $SKey'$, and t' , to the server.

As a result, the client receives an error message containing the timestamp t but no response to her request containing ST' , $SKey'$, and t' . She might assume that her first request was rejected while her second was accepted, while the reverse is actually true. This is potentially worrisome because the options on the tickets are different; in the case of anonymous tickets, the client might erroneously assume that her identity has not been seen by the server (if the error is tied to a non-anonymous ticket).

It is unclear whether this anomaly is of practical concern. It does highlight the interactions between the ticket options and other traces; for the anonymous ticket option, these may be particularly undesirable.

Because the formalizations we present here do not include explicit checks for replayed authenticators, this anomaly is not realizable in these formalizations. However, adding such checks does not require other additional detail not in the C level formalization.

6 Conclusions and Future Work

In this paper, we have summarized the results of a project specifying and analyzing the Kerberos 5 suite [3]; our initial report on this work was given in [4], and a detailed discussion is contained in [5]. We have concentrated on a subset of this protocol that takes into account encryption types, checksums, flags and options, error messages, and a few timestamps on top of the core message exchange. We have formally shown that, at this level of detail, Kerberos 5 complies with the confidentiality and data origin authentication requirements. We have also observed anomalous behaviors that deviate from the expected script of the protocol, but do not appear to have the potential of causing harm to the principals involved. In order to produce these results, we paired the MSR security protocol specification language [13, 14] with an inductive proof method inspired by the work of Schneider [16].

This work may be extended in two directions. First, our formalizations of the Kerberos 5 may be refined by adding more options (in particular renewable or postdatable tickets), more timestamps and temporal checks, and the seldom analyzed optional subprotocols that comprise the suite (in particular the `KRB_SAFE` and `KRB_PRIV` exchanges and the mechanisms for distributing and updating long-term keys). Second, we may refine the tools used in the analysis to cope with the increasing complexity and the necessity to assess requirements that go beyond confidentiality and authentication. A first step is the ongoing implementation of an MSR environment that will help manage complex specifications and assist with their development. Another part of this is the further development of our verification methods in order to reuse previously proved results as guidelines for new proofs in more refined specifications of Kerberos 5. We are also looking at automation to help for this purpose.

We have received encouraging feedback from the IETF Kerberos Working Group [17, 20].

Acknowledgments

We are grateful to Alan Jeffrey, John Mitchell, Clifford Neuman, and Ken Raeburn for a number of helpful comments on our earlier work.

References

1. Kohl, J., Neuman, C.: The Kerberos Network Authentication Service (V5) (1993) Network Working Group Request for Comments: 1510.
2. Neuman, B.C., Ts'o, T.: Kerberos: An Authentication Service for Computer Networks. *IEEE Communications* **32** (1994) 33–38
3. Neuman, C., Kohl, J., Ts'o, T., Raeburn, K., Yu, T.: The Kerberos Network Authentication Service (V5) (2001) Internet draft, expires 20 May 2002.
4. Butler, F., Cervesato, I., Jaggard, A.D., Scedrov, A.: A Formal Analysis of Some Properties of Kerberos 5 Using MSR. In: Proceedings of the 15th Computer Security Foundations Workshop, IEEE Computer Society (2002) 175–190

5. Butler, F., Cervesato, I., Jaggard, A.D., Scedrov, A.: A formal analysis of some properties of kerberos 5 using *MSR*. Technical Report CIS-MS-04-04, University of Pennsylvania, Department of Computer and Information Science (2004) 59 pages. Available from [ftp://ftp.cis.upenn.edu/pub/papers/scedrov/ms-cis-04-04.\[pdf|ps\]](ftp://ftp.cis.upenn.edu/pub/papers/scedrov/ms-cis-04-04.[pdf|ps]).
6. Bella, G., Riccobene, E.: Formal Analysis of the Kerberos Authentication System. *J. Universal Comp. Sci.* **3** (1997) 1337–1381
7. Bella, G.: Inductive Verification of Cryptographic Protocols. PhD thesis, University of Cambridge (2000)
8. Bella, G., Paulson, L.C.: Using Isabelle to Prove Properties of the Kerberos Authentication System. In Orman, H., Meadows, C., eds.: *Proc. of DIMACS'97, Workshop on Design and Formal Verification of Security Protocols* (CD-ROM). (1997)
9. Bella, G., Paulson, L.C.: Kerberos Version IV: Inductive Analysis of the Secrecy Goals. In: *Proc. of ESORICS '98, Fifth European Symposium on Research in Computer Science*. Number 1485 in *Lecture Notes in Computer Science*, Springer-Verlag (1998) 361–375
10. Bella, G., Paulson, L.C.: Mechanising BAN Kerberos by the Inductive Method. In: *Proc. of CAV98 – Tenth International Conference on Computer Aided Verification*. (1998)
11. Mitchell, J.C., Mitchell, M., Stern, U.: Automated Analysis of Cryptographic Protocols Using $\text{Mur}\varphi$. In: *Proc. of the IEEE Symposium on Security and Privacy*, IEEE Computer Society Press (1997) 141–153
12. Cervesato, I., Durgin, N.A., Lincoln, P., Mitchell, J., Scedrov, A.: A Meta-notation for Protocol Analysis. In: *Proc. of the Twelfth IEEE Computer Security Foundations Workshop*. (1999) 55–69
13. Cervesato, I.: Typed Multiset Rewriting Specifications of Security Protocols. In: *Proc. of the First Irish Conference on the Mathematical Foundations of Computer Science and Information Technology–MFCSIT'00*, Elsevier ENTCS 40 (2000)
14. Cervesato, I.: Typed MSR: Syntax and Examples. In: *Proc. of the First International Workshop on Mathematical Methods, Models and Architectures for Computer Network Security — MMM'01*. Springer-Verlag (2001) St. Petersburg, Russia, 21–23 May 2001.
15. Durgin, N.A., Lincoln, P.D., Mitchell, J.C., Scedrov, A.: Multiset Rewriting and the Complexity of Bounded Security Protocols. Manuscript, 63 pages. (2002)
16. Schneider, S.: Verifying Authentication Protocols in CSP. *IEEE Transactions on Software Engineering* **24** (1998) 741–758
17. Neuman, C.: (2002) Personal communication.
18. Neuman, C., Yu, T., Hartman, S., Raeburn, K.: The Kerberos Network Authentication Service (V5) (2004) Internet draft, expires 15 August 2004. <http://www.ietf.org/internet-drafts/draft-ietf-krb-wg-kerberos-clarifications-05.txt>.
19. Jeffrey, A.: (2002) Personal communication.
20. Raeburn, K.: (2002) Personal communication.

A Abstract Level Protocol Formalization

We now give a more detailed description of the A level formalization of Kerberos 5; Figs. 3–6 show some of the MSR roles and rules comprising this formalization. We believe that the A level formalization contains the minimum

$$\left(\begin{array}{l}
\exists L : \text{client} \times \text{KOpt} \times \text{TGS} \times \text{nonce} \times \text{etype}. \\
\forall T : \text{TGS} \quad . \\
\forall K : \text{KAS} \quad . \\
\forall \text{KOpts} : \text{KOpt}. \\
\forall e : \text{etype} \quad . \\
\\
\forall \dots \quad . \\
\forall k_C : \text{dbK } C \quad . \\
\forall \text{AKey} : \text{shK } C T. \quad \text{N}(C, X, \{\text{AKey}, \\
\quad \quad \quad n_1, \text{TFlags}, T\}_{k_C}) \\
\forall X : \text{msg} \quad . \\
\forall n_1 : \text{nonce} \quad . \\
\forall \text{TFlags} : \text{TFlag} \quad . \\
\\
\forall \dots \quad . \\
\forall \text{ErrorCode} : \text{msg}. \quad \text{N}(\text{KRB_ERROR}, t_{K, \text{err}}, \\
\quad \quad \quad \text{ErrorCode}, C, K) \\
\forall t_{K, \text{err}} : \text{time} \quad . \quad L(C, \text{KOpts}, T, n_1, e)
\end{array} \right) \xrightarrow{\alpha\delta_{1.1}} \left(\begin{array}{l}
\exists n_1 : \text{nonce} \\
\text{N}(\text{KOpts}, C, T, n_1, e) \\
L(C, \text{KOpts}, T, n_1, e) \\
\\
\text{Auth}_C(X, \text{TFlags}, \\
T, \text{AKey}) \\
\\
\text{ASError}_C(\\
\text{KRB_ERROR}, t_{K, \text{err}}, \\
\text{ErrorCode}, K)
\end{array} \right) \xrightarrow{\delta_{1.2'}}$$

$\forall C : \text{client}$

Fig. 3. The client's role in the Authentication Service Exchange

amount of detail needed to capture the Kerberos 5 protocol. Although our proofs related to this formalization do not use nonces, omitting them would remove the connection between the `KRB_AS_REQ` and `KRB_AS_REP` messages.

A.1 Authentication Service Exchange

Recall that the Authentication Service Exchange allows a client C to obtain from a KAS K credentials to be used in the Ticket Granting Exchange with a TGS T . C 's actions in this exchange are formalized in Fig. 3, K 's in Fig. 4. Note that K 's rules fill in the gaps between C 's rules; for the later exchanges of the protocol, we will omit the various server rules because they are essentially recoverable from the client rules.

C asks K for credentials for the server T using rule $\alpha_{1.1}$, sending a `KRB_AS_REQ` message with her name C , the name T of the TGS for which she wishes to obtain credentials, and a fresh nonce n_1 , and storing these in the role state predicate L . Rule $\alpha_{1.2}$ allows C to process the `KRB_AS_REP` message which K sends in response to her initial request. This message is expected to contain C 's name, an opaque ticket X to be passed on to T , and, encrypted under C 's long-term key k_C , a session key AKey for use with T , the nonce n_1 from the original request, and the name T of the TGS. If the message is of this form and if C , T , and n_1 , match the data from the original request (stored in L), C removes the `KRB_AS_REP` message from the network, deletes the role state predicate L , and stores the relevant data in the persistent memory predicate Auth_C . In the abstract formalization, C does not process any other (*i.e.*, error) messages which K may return as defined in [3].

$$\left(\begin{array}{l}
\forall C : \text{client} \quad . \\
\forall T : \text{TGS} \quad . \\
\forall n_1 : \text{nonce} \quad . \text{N}(K\text{Opts}, C, T, n_1, e) \\
\forall k_C : \text{dbK } C \quad . \text{Valid}(K\text{Opts}, C, \\
\forall k_T : \text{dbK } T \quad . \quad T, n_1, e) \\
\forall AKey : \text{shK } C T. \text{SetAuthFlags}(K\text{Opts}, \\
\forall K\text{Opts} : \text{KOpt} \quad . \quad T\text{Flags}) \\
\forall e : \text{etype} \quad . \\
\forall T\text{Flags} : \text{TFlag} \quad . \\
\\
\forall \dots \quad . \text{N}(K\text{Opts}, C, T, n_1, e) \\
\forall \text{ErrorCode} : \text{msg}. \text{Invalid}(K\text{Opts}, C, \\
\forall t_{K, \text{err}} : \text{time} \quad . \quad T, n_1, e) \\
\quad \quad \quad . \text{Clock}_K(t_{K, \text{err}})
\end{array} \right) \xrightarrow{\alpha\delta_{2.1}} \left(\begin{array}{l}
\exists AKey : \text{shK } C T \\
\text{N}(C, \{T\text{Flags}, \\
AKey, C\}_{k_T}, \\
\{AKey, n_1, \\
T\text{Flags}, T\}_{k_C}) \\
\\
\text{N}(\text{KRB_ERROR}, t_{K, \text{err}}, \\
\text{ErrorCode}, C, K)
\end{array} \right) \quad \forall K : \text{KAS}$$

Fig. 4. The authentication server's role in the Authentication Service Exchange

If there is a `KRB_AS_REQ` message from C on the network, and if it is valid (as determined by the external process *Valid*), rule $\alpha_{2.1}$ allows K to generate a fresh session key $AKey$ for use between C and T and to send this key in a `KRB_AS_REP` message to C . This message consists of C 's name, the ticket for T , and, encrypted under k_C , the key $AKey$, the nonce n_1 from C 's request, and the name T of the TGS. The ticket for T is encrypted with T 's long-term key k_T and contains $AKey$ and the name C of the client.

A.2 The Ticket-Granting Exchange

The third and fourth messages shown in Fig. 1 comprise the Ticket-Granting Exchange. Here the client C presents credentials previously obtained from an authentication server (via the Authentication Service Exchange) to a TGS T in order to obtain a service ticket for an application server S .

The client's actions in this exchange are formalized in Fig. 5. If, as indicated by the memory predicate $\text{Auth}_C(X, T, AKey)$, the client C has completed the authentication service exchange to get credentials for the TGS T , rule $\alpha_{3.1}$ allows her to initiate an exchange with T to obtain credentials for the application server S . In doing so, she chooses a new nonce n_2 and sends a `KRB_TGS_REQ` message to T consisting of the previously obtained ticket X , an authenticator (C encrypted under the session key $AKey$), her name C , the name S of the server for which C wishes to obtain credentials, and the new nonce n_2 . C stores the information about this request in the role state predicate L , and retains the memory predicate Auth_C for use in future exchanges with T .

The client's second rule, $\alpha_{3.2}$, allows her to read a `KRB_TGS_REP` message that matches her request to T . This message consists of C 's name, an opaque ticket Y to be passed to the application server, and, encrypted under the session key $AKey$, a session key $SKey$ for use by C and the application server, the

$$\left(\begin{array}{l}
\exists L : \text{client} \times \text{TOpt} \times \text{server} \times \text{TGS} \times \text{nonce} \times \text{time}. \\
\forall T : \text{TGS} \quad . \\
\forall S : \text{server} \quad . \\
\forall AKey : \text{shK } C T. \\
\forall X : \text{msg} \quad . \quad \text{Auth}_C(X, TFlags, \\
\quad \quad \quad T, AKey) \quad \alpha\delta_{3.1} \\
\forall t_{C,Treq} : \text{time} \quad . \quad \text{Clock}_C(t_{C,Treq}) \quad \longrightarrow \\
\forall TFlags : \text{TFlag} \quad . \\
\forall TOpts : \text{TOpt} \quad . \\
\forall e : \text{etype} \quad . \\
\quad \quad \quad \text{Auth}_C(X, TFlags, \\
\quad \quad \quad T, AKey) \\
\quad \quad \quad L(C, TOpts, S, T, \\
\quad \quad \quad n_2, t_{C,Treq}, e) \\
\\
\forall \dots \quad . \\
\forall SKey : \text{shK } C S. \quad \text{N}(C, Y, \{SKey, n_2, \\
\quad \quad \quad SFlags, S\}_{AKey}) \quad \alpha\delta_{3.2} \\
\forall Y : \text{msg} \quad . \quad L(C, TOpts, S, \\
\quad \quad \quad T, n_2, t_{C,Treq}, e) \quad \longrightarrow \quad \text{Service}_C(Y, SFlags, \\
\quad \quad \quad S, SKey) \\
\forall n_2 : \text{nonce} \quad . \\
\forall SFlags : \text{SFlag} \quad . \\
\\
\quad \quad \quad \text{N}(\text{KRB_ERROR}, t_{C,Treq}, \\
\forall \dots \quad . \quad t_{T,err}, \text{ErrorCode}, \\
\forall \text{ErrorCode} : \text{msg}. \quad C, T) \quad \delta_{3.2'} \\
\forall t_{T,err} : \text{time} \quad . \quad L(C, TOpts, \\
\quad \quad \quad S, T, n_2, t_{C,Treq}, e) \quad \longrightarrow \quad \text{TGSError}_C(T, \\
\quad \quad \quad t_{T,err}, \text{ErrorCode})
\end{array} \right) \quad \forall C : \text{client}$$

Fig. 5. The client's role in the Ticket-Granting Exchange

nonce n_2 , and the application server's name S . C , S , and n_2 must match the stored information about the original request to T in order for C to process the `KRB_TGS_REP` message. If C does process the message using $\alpha_{3.2}$, she stores the ticket Y , server name S , and session key $SKey$ in the memory predicate Service_C .

Because of its close correspondence to the rules in Fig. 5, we omit the TGS rule from our A level. If a `KRB_TGS_REQ` message, such as that in the right-hand side of rule $\alpha_{3.1}$, appears on the network, the TGS rule allows the TGS T to read it from the network and process it; this includes a validity check that constrains the firing of the rule. In firing the rule, T produces a fresh session key for the client to share with the server named in the `KRB_TGS_REQ` message and places a `KRB_TGS_REP` message (like that in the left-hand side of rule $\alpha_{3.2}$) onto the network.

A.3 The Client/Server Exchange

The fifth and sixth messages in Fig. 1 form the Client/Server Exchange. Here the client C presents credentials, previously obtained from a TGS, to the application server S . In the abstract level formalization we assume that mutual

$$\left(\begin{array}{l}
\exists L : \text{client}^{(C)} \times \text{SOpt} \times \text{server}^{(S)} \times \text{shK } C \ S \times \text{time} \times \text{msg}. \\
\forall S : \text{server} \quad \cdot \quad \text{N}(\text{SOpts}, Y, \{C, \\
\forall SKey : \text{shK } C \ S. \text{Service}_C(Y, SFlags, \\
\forall t_{C,Sreq} : \text{time} \quad \cdot \quad S, SKey) \quad \alpha\delta_{5.1} \quad \begin{array}{l} [\dots]_{SKey}, \\ t_{C,Sreq}\}_{SKey} \\ \text{Service}_C(Y, SFlags, \\ S, SKey) \\ L(C, \text{SOpts}, S, SKey, \\ t_{C,Sreq}, Y) \end{array} \\
\forall Y : \text{msg} \quad \cdot \quad \text{Mutual}(\text{SOpts}) \\
\forall SFlags : SFlag \quad \cdot \quad \text{Clock}_C(t_{C,Sreq}) \\
\forall \text{SOpts} : \text{SOpt} \quad \cdot \\
\forall \dots \quad \begin{array}{l} \text{N}(\{t_{C,Sreq}\}_{SKey}) \\ L(C, \text{SOpts}, S, SKey, \\ t_{C,Sreq}, Y) \end{array} \quad \alpha\delta_{5.2} \quad \text{DoneMut}_C(S, SKey) \\
\forall \dots \quad \cdot \quad \text{N}(\text{KRB_ERROR}, \\
\forall t_{C,Sreq}, t_{S,err}, \\
\forall \text{ErrorCode} : \text{msg}. \quad \text{ErrorCode}, C, S) \quad \delta_{5.2'} \quad \text{APError}_C(S, \\
\forall t_{S,err} : \text{time} \quad \cdot \quad L(C, \text{SOpts}, S, \\ SKey, t_{C,Sreq}, Y) \quad \text{t}_{S,err}, \text{ErrorCode})
\end{array} \right) \quad \forall C : \text{client}$$

Fig. 6. The client's role in the Client/Server Exchange with mutual authentication

authentication is requested by C ; thus the sixth message (of type KRB_AP_REP) is required for the protocol to finish.

Figure 6 shows the role of the client C in the exchange. If she has previously obtained credentials (a ticket Y and session key $SKey$, stored in the memory predicate Service_C) for use with S she may fire rule $\alpha_{5.1}$. This places a KRB_AP_REQ message containing the ticket and an authenticator (obtained by encrypting C and the current time $t_{C,Sreq}$ on C 's system, given by the external process Clock_C , under the session key $SKey$) on the network and stores the relevant information about this request in the role state predicate L . C 's second rule, $\alpha_{5.2}$, may be fired when the network contains a KRB_AP_REP message consisting of $t_{C,Sreq}$ encrypted under $SKey$. This rule reads the message from the network and stores the server name S and the session key $SKey$ in the DoneMut_C predicate. Although not modelled in the abstract level formalization, this information is intended to be used in additional communications with S .

We omit the formal statement of the server's MSR rule, which is essentially reconstructible from the client's rules. If a KRB_AP_REQ message for a server S appears on the network, S may fire a rule that reads it off of the network and replies with a KRB_AP_REP message. Paralleling the TGS rule, this is constrained by a validity check on the KRB_AP_REQ message. S stores the data from the request in a memory predicate; while we do not make use of this predicate here, the session key now known to both S and the client may be used for future communication.

B A Level Intruder Formalization

In this section, we present the rules specifying the Dolev-Yao intruder model for Kerberos 5. We ask the reader to ignore for the moment the grayed-out text as it describes additions needed for the more detailed (C level) intruder. We will come back to them in App. D.

We divide the actions available to the intruder into three categories:

- the fairly standard operations of interception/transmission of a message over the network, decomposition/composition of a pair, and decryption/encryption of a message given a known key (App. B.1);
- the often overlooked action of generating new data (App. B.2);
- and the use of accessible data (App. B.3).

B.1 Network, Pairing and Encryption Rules

We have the following rules describing how the Dolev-Yao intruder can intercept/transmit messages, decompose/compose pairs, and decrypt/encrypt messages under the various types of known keys. We have also some administrative rules that permit the duplication and deletion of deleted data.

The intruder is allowed to read messages from the network (removing them from the network in the process) and put messages that she knows onto the network. She may also break compound messages (*e.g.*, m_1, m_2) into their constituent parts and combine separate messages into a single message via concatenation.

The following two rules are samples of the intruder rules; these model decryption (SDC') and encryption (SEC') with a shared key. The type ts is one of the auxiliary types mentioned in Sect. 3 and includes both TGS and server.

$$\left(\begin{array}{l} \forall C : \text{client} \quad . \\ \forall A : ts \quad . \\ \forall e : \text{etype} \quad . \\ \forall k : \text{shK}^e C A. \\ \forall m : \text{msg} \quad . \end{array} \right)^! \left(\begin{array}{l} l(\{m\}_k^e) \xrightarrow{\text{SDC}'} l(m) \end{array} \right)$$

$$\left(\begin{array}{l} \forall C : \text{client} \quad . \\ \forall A : ts \quad . \\ \forall e : \text{etype} \quad . \\ \forall k : \text{shK}^e C A. \\ \forall m : \text{msg} \quad . \end{array} \right)^! \left(\begin{array}{l} l(m) \xrightarrow{\text{SEC}'} l(\{m\}_k^e) \end{array} \right)$$

Similar rules allow the intruder to decrypt and encrypt using a long-term key that she knows.

Finally, the intruder is allowed to duplicate or delete any information that she knows; the ability to delete data might be dropped from the intruder specification without adverse effects.

B.2 Data Generation Rules

In general, the intruder should be able to generate everything an honest principal can generate—*i.e.*, nonces and session keys—and no more. In the case of Kerberos, we must admit an exception to this rule: because principals forward

uninterpreted data, we must allow the intruder to create garbage, modelled as objects of the generic type `msg`.

We omit the intruder rules for generating nonces and session keys, but include the following rule for generating generic messages because this merits some discussion.

$$\left(\cdot \xrightarrow{\text{MG}} \exists m : \text{msg } l(m) \right)^l$$

Rule MG does not allow `l` to generate the long-term key of a principal because `dbK A` is never a subtype of `msg`. Note also that although the intruder may generate fresh messages, she may not type these as anything other than `msg`. The intruder is not allowed to generate any other kind of data, not principal names of any kind (the introduction of new agents happens out-of-band), not long-term keys (they are distributed out-of-band), and not timestamps (they are generated by an external clock, not by any principal). Allowing the intruder to generate data of these forms is incorrect since it would open the doors to countless false attacks.

B.3 Data Access Rules

The intruder is entitled to look up the same data as any other principal. She therefore has access to the names of all entities of type `principal`, to her keys (long-term and session), and to timestamps. Note that in this respect timestamps differ from nonces; we do not allow an intruder to guess the latter. We note that the ability to guess timestamps may give unreasonable strength to the intruder.

No other piece of information is accessible out of thin air by the intruder: unless she has intercepted this information otherwise, she should not be able to guess the nonces generated by other principals, or keys that do not belong to her, or clearly generic messages.

C C Level Protocol Formalization

Our C level formalization extends the A level formalization by adding additional elements of the full protocol specification. In particular, we now include options in messages sent by a `client` (allowing her to request particular encryption methods and `ANONYMOUS` tickets and to specify whether a `server` should provide mutual authentication); the replies to these messages now include flag fields specifying which options were actually granted. Message digests now appear as specified by the protocol. We have also added error messages, and the authenticator in the `KRB_TGS_REQ` message now includes a timestamp which may be sent back to the client in an error message. We do not add any temporal checks, however.

The grayed-out portions of Fig. 1 and Figs. 3—6 are the details which are being added to the A level formalization to obtain the C level. When we refer to a C level rule by name we will now use δ with an appropriate subscript as given in the various figures; this will mean the entire rule depicted in the figure, including the grayed-out portions. While fields specifying encryption type appear in several

messages in this level, and should technically appear for every encrypted message that occurs (according to Sect. 3), we make the convention that we will omit the etype in this capacity unless we are explicitly discussing it.

C.1 Authentication Service Exchange

We return to Fig. 3 to see the client’s actions in the Authentication Service Exchange, now looking at both the black and the gray type. Rule $\delta_{1.1}$ allows C to send a `KRB_AS_REQ` message, and extends rule $\alpha_{1.1}$ by adding the options field $KOpts$ and the field e containing the requested encryption type(s) for the response. Rule $\delta_{1.2}$ allows the client to process the response; this now includes the $TFlags$ field indicating which options were granted by the KAS.

Rule $\delta_{1.2'}$ shows C ’s error processing of a generic error message, formalized by C storing relevant information in the memory predicate $ASError_C$. The $ErrorCode$ describes the reason why the `KRB_AS_REQ` failed.

The actions of the KAS K are formalized in Fig. 4, including both the black and gray type. Rule $\delta_{2.1}$ is similar to rule $\alpha_{2.1}$, except the validity check performed by K also covers the added message fields $KOpts$ and e . The external process $SetAuthFlags$ is used to determine which of the options requested in $TOpts$ should be approved; those granted are described by the flags $TFlags$.

If C ’s request is not valid for any reason (as determined by the external process $Invalid$, which we assume to hold iff the $Valid$ check fails), then K reads the current time $t_{K,err}$ from the local clock via the external process $Clock_K$. When rule $\delta_{2.1'}$ is fired, K sends an error message consisting of the time the error occurred ($t_{K,err}$) and the appropriate $ErrorCode$, along with the names C and K .

C.2 The Ticket-Granting Exchange

Considering both colors of type in Fig. 5 gives the C level client rules for the Ticket-Granting Exchange. Rule $\delta_{3.1}$ allows the client to send a `KRB_TGS_REQ` message and updates rule $\alpha_{3.1}$. This message now includes: the current time $t_{C,Treq}$ on the client’s local clock, obtained via the external process $Clock_C$; a message digest, keyed by $AKey$, of the unencrypted portion of the `KRB_TGS_REQ`; the options field $TOpts$; and the requested encryption type(s) e . $TOpts$ may be used by the client to request an `ANONYMOUS` ticket-granting ticket.

Rule $\delta_{3.2}$ extends rule $\alpha_{3.2}$ and again allows the client to read from the network a `KRB_TGS_REP` that corresponds to her previous `KRB_TGS_REQ` message. The added details here are the flags in the response and the options, timestamp, and encryption type stored in the role state predicate. Note that if `ANONYMOUS` is one of the set flags in the `KRB_TGS_REP` message, then instead of the name C the dummy identifier `USER` will appear in the ticket Y in rule $\delta_{3.2}$.

The C level formalization adds error messages; rule $\delta_{3.2'}$ allows C to process these in the same manner as rule $\delta_{1.2'}$ did in the Authentication Service Exchange. Note that the names C and T in the error message must match those in the role state predicate L .

We again omit the formal MSR rules for a TGS as these are essentially reconstructible from the client rules for the Ticket-Granting Exchange. A TGS T now sets flags in response to the options requested by a client and also verifies the checksum included in the `KRB_TGS_REQ` message. The other notable difference from the A level TGS behavior is the addition of a rule to send error messages. This is essentially the same as rule $\delta_{2.1'}$ in the Authentication Service Exchange.

C.3 The Client/Server Exchange

Figure 6, including both colors of type, shows the role of the client C in this exchange. Rule $\delta_{5.1}$ is the obvious extension of rule $\alpha_{5.1}$; here we use the constraint *Mutual* to indicate that the `MUTUAL_REQUIRED` option is being requested by C . Note that if the ticket Y stored in the $Service_C(-)$ predicate is `ANONYMOUS` (indicated by the presence of `ANONYMOUS` in the *SFlags* field, also stored in $Service_C(-)$), then C will send the generic identifier `USER` in place of her name in the authenticator sent in rule $\delta_{5.1}$. We denote the message digest here by $[\dots]_{SKey}$ because [3] specifies that this optional checksum is “application specific.”

Rule $\delta_{5.2}$ is virtually identical to rule $\alpha_{5.2}$, the only difference being the presence of *SOpts* in the role state predicate L . Rule $\delta_{5.2'}$ models C 's handling of error messages in exactly the same way as in the previous exchanges with KAS and TGS, with C storing relevant information sent in the error message in the *APError* memory predicate.

As for the first two exchanges, we omit the rules for processing the client's request. The server's response to a `KRB_AP_REQ` message requesting mutual authentication is constrained by a check of this option and a verification of the checksum (in addition to the general validity check used in the A level). There is no change in the `KRB_AP_REP` message, although the server is now allowed to send error messages in case of an invalid request from the client; these are essentially the same as in the other exchanges.

Our C level formalization allows for the client to request no mutual authentication from the application server S . In this case, C views the exchange as finished as soon as she sends her `KRB_AP_REQ` message, although she may still process error messages from S . Similarly, S does not send a reply to a valid request from C , although he may send error messages if needed.

D C Level Intruder

The C level intruder rules update those of the A level to reflect the added detail of this formalization. The rules for interception/transmission, decomposition/composition, and decryption/encryption with a known key change only to the extent that we must take encryption types into account in the rules that involve cryptographic primitives. There is no disassembling rule for message digests because (cryptographic) hashing does not permit recovering a message.

However, the intruder can construct a message digest as long as she knows the proper key; this is captured by a rule similar to SEC' above.

The updates to the generation rules are limited to allowing the intruder to choose the encryption type of any session key he may generate. None of the new data types introduced at this level of detail can be generated by the intruder (or any other principal). Therefore there are no additional data generation rules beyond those we presented in App. B.2.

Data access rules are subject to similar changes. However, the new data types, encryption types, options and flags, are treated similarly to timestamps: each of them range over a limited number of legal values, each being public knowledge. As for timestamps, we assume that encryption types, options and flags are guessable using rules parallel to TA. Other information that was inaccessible in the A level specification of the intruder remains inaccessible in the C level.