Carnegie Mellon University
Research Showcase

Computer Science Department | School of Computer Science

4-23-2004

A Formal Analysis of Some Properties of Kerberos 5 Using MSR

Frederick Butler
University of Pennsylvania

Iliano Cervesato
ITT Industries

Andre Scedrov
University of Pennsylvania

Aaron D. Jaggard
Tulane University

Follow this and additional works at: http://repository.cmu.edu/compsci

Recommended Citation
Butler, Frederick; Cervesato, Iliano; Scedrov, Andre; and Jaggard, Aaron D., "A Formal Analysis of Some Properties of Kerberos 5 Using MSR" (2004). *Computer Science Department.* Paper 20.
http://repository.cmu.edu/compsci/20

This Technical Report is brought to you for free and open access by the School of Computer Science at Research Showcase. It has been accepted for inclusion in Computer Science Department by an authorized administrator of Research Showcase. For more information, please contact research-showcase@andrew.cmu.edu.

A Formal Analysis of Some Properties of Kerberos 5 Using MSR *

University of Pennsylvania Department of Computer & Information Science Technical Report MS-CIS-04-04

Frederick Butler† Iliano Cervesato‡ Aaron D. Jaggard†¶ Andre Scedrov¶◊

Department of Mathematics
University of Pennsylvania
Philadelphia, PA USA
{fbutler@math,scedrov@saul.cis}.upenn.edu

ITT Industries, Inc.
Advanced Engineering & Sciences
2560 Huntington Avenue
Alexandria, VA 22303
iliano@itd.nrl.navy.mil

Department of Mathematics
Tulane University
New Orleans, LA USA
adj@math.tulane.edu

April 23, 2004

Abstract

We give three formalizations of the Kerberos 5 authentication protocol in the Multi-Set Rewriting (MSR) formalism. One is a high-level formalization containing just enough detail to prove authentication and confidentiality properties of the protocol. A second formalization refines this by adding a variety of protocol options; we similarly refine proofs of properties in the first formalization to prove properties of the second formalization. Our third formalization adds timestamps to the first formalization but has not been analyzed extensively. The various proofs make use of rank and corank functions, inspired by work of Schneider in CSP, and provide examples of reasoning about real-world protocols in MSR. We also note some potentially curious protocol behavior; given our positive results, this does not compromise the security of the protocol.

*†Partially supported by ONR Grant N00014-01-1-0431. ‡Partially supported by NRL under contract N00173-00-C-2086. ¶Partially supported by the DoD University Research Initiative (URI) program administered by the Office of Naval Research under Grant N00014-01-1-0795. ◊Partially supported by NSF Grant CCR-0098096. A short version of this appeared in *S. Schneider, ed.,* 15th *IEEE Computer Security Foundations Workshop, Cape Breton, Nova Scotia, Canada, June 2002, IEEE Computer Society Press, 2002.* Most of this work was carried out and the initial draft of this report was written while Jaggard was at the Department of Mathematics, University of Pennsylvania.

Contents

I Introduction and Background **4**

1 Introduction **4**

2 Overview of the Kerberos 5 Protocol **5**

3 MSR **6**
3.1 Signature . . . 6
3.2 States and roles . . . 7

II Formalizing Kerberos 5 **8**

4 A Level Formalization of Kerberos 5 **8**
4.1 The Authentication Service Exchange . . . 8
4.2 The Ticket-Granting Exchange . . . 10
4.3 The Client/Server Exchange . . . 11
4.4 A level intruder formalization . . . 12
4.4.1 Network, pairing and encryption rules . . . 12
4.4.2 Data generation rules . . . 12
4.4.3 Data access rules . . . 13

5 C Level Formalization of Kerberos 5 **13**
5.1 The Authentication Service Exchange . . . 14
5.2 The Ticket-Granting Exchange . . . 16
5.3 The Client/Server Exchange . . . 17
5.4 The Client/Server Exchange without mutual authentication . . . 18
5.5 C level intruder formalization . . . 19

6 B Level Protocol Formalization **20**
6.1 The Authentication Service Exchange . . . 20
6.2 The Ticket-Granting Exchange . . . 21
6.3 The Client/Server Exchange with mutual authentication . . . 23
6.4 The Client/Server Exchange without mutual authentication . . . 23

III Analyzing Kerberos 5 **27**

7 Anomalous Protocol Behavior **27**
7.1 Ticket anomaly . . . 27
7.2 Anonymous ticket switch anomaly . . . 28
7.3 Encryption type anomaly . . . 30
7.4 Ticket replay anomaly . . . 31
7.5 Possible replays . . . 32

8 Rank and Corank Functions **32**
8.1 Rank . . . 33
8.2 Corank . . . 34

9 Properties of Kerberos 5 **36**
9.1 The Ticket-Granting Exchange . . . 37
9.1.1 Confidentiality of $AKey$ . . . 37
9.1.2 Authentication of ticket-granting ticket and authenticator . . . 38
9.2 The Client/Server Exchange . . . 38
9.2.1 Confidentiality of $SKey$ . . . 38
9.2.2 Authentication of ST and authenticator . . . 39

IV Conclusions and References **40**

10 Conclusions and Future Work **40**
10.1 Conclusions . . . 40
10.2 Future work . . . 40

V Appendices **42**

A Proofs of Protocol Properties **42**
A.1 The Ticket-Granting Exchange . . . 42
A.1.1 Confidentiality of $AKey$ . . . 42
A.1.2 Authentication of TGT and authenticator . . . 43
A.2 The Client/Server Exchange . . . 44
A.2.1 Confidentiality of $SKey$ . . . 44
A.2.2 Authentication of ST and authenticator . . . 45

B Lemmas for Authentication Properties **45**
B.1 General lemmas . . . 45
B.1.1 Lemmas for A level analysis . . . 45
B.1.2 Lemmas for C level analysis . . . 46
B.2 Lemmas for Ticket-Granting Exchange . . . 46
B.2.1 Lemmas for Theorem 3 . . . 46
B.2.2 Lemmas for Theorem 4 . . . 47
B.2.3 Lemmas for Theorem 5 . . . 47
B.2.4 Lemmas for Theorem 6 . . . 48
B.3 Lemmas for Client/Server Exchange . . . 49
B.3.1 Lemmas for Theorem 7 . . . 49
B.3.2 Lemmas for Theorem 9 . . . 50

C Anomalous Traces **52**
C.1 A level trace of ticket anomaly . . . 52
C.2 C level trace of encryption type anomaly . . . 53

D Message Fields **54**

Part I
Introduction and Background

1 Introduction

Kerberos [13, 17, 16, 18] is a widely deployed protocol, designed to repeatedly authenticate a client to multiple application servers based on a single login. The protocol uses various credentials (tickets), encrypted under a server's key and thus opaque to the client, to authenticate the client to the server; this allows the client to obtain additional credentials or to request service from an application server. A formalization of Kerberos 4, the first publicly released version of this protocol, was given in [5] and has since been extended and thoroughly analyzed using an inductive approach [1, 2, 3, 4]. This analysis, through heavy reliance on the Isabelle theorem prover, yielded formal correctness proofs for a fairly detailed specification, and also highlighted a few minor problems. A simple fragment of the latest version, Kerberos 5, has been investigated using the state exploration tool Murφ [14]. This approach proved effective for finding an attack, which the authors of [14] note is unrealizable in a full implementation of Kerberos 5, but came short of proving positive correctness results.

Here we report on a project whose goal is to use the Multi-Set Rewriting (MSR) framework to give a precise specification of Kerberos 5 at various levels of detail, ranging from a minimal account, similar to that used in [14], to a detailed formalization of every behavior encompassed by this complex suite [13, 16]. Our particular objectives include giving a precise and unambiguous description of this protocol, making its operational assumptions explicit, stating the properties it is supposed to satisfy, and proving that it satisfies these properties. This will complement the currently spotty and often vague information in the literature. This project is also intended as a test-bed for MSR on a real-world protocol: we are interested in how easy it is to write large specifications in MSR, in what ways this language can be improved, and whether the insight gained with toy protocols scales up. In this work we have also started exploring forms of reasoning that best take advantage of the linguistic features of MSR.

In this paper we provide three formalizations of Kerberos 5, which we call our A, B, and C level formalizations. The B and C level formalizations add detail to the A level formalization but are not otherwise related. The A level formalization omits most timestamps and all optional features, including only what we believe is needed to provide authentication. It is similar to the formalization of Kerberos 4 in [1, 2, 3], but without timestamps. This level of abstraction is a good starting point to utilize the proof techniques demonstrated within this paper, providing a formalization which is not overly complicated (making proofs feasible), but which retains many properties of the full Kerberos 5 protocol. Our B level formalization adds some timestamps and temporal checks to our A level formalization, thus closely paralleling the formalization of Kerberos 4 in [1, 2, 3]. We have not found any new and interesting properties or anomalies related to the timestamps here; the two features of the B level which are not found in [1, 2, 3]—the single option of mutual authentication and error messages—seemed like the most promising area to focus our efforts. This leads to our C level formalization, which does not include temporal checks or most timestamps. It extends the A level formalization by making mutual authentication optional and adding error messages, along with several low-level aspects of the protocol, namely options, flags, and checksums, none of which has appeared in any previous study of Kerberos. We have focused our investigations on the A and C level formalizations, with the abstraction of the former facilitating reasoning about the protocol and the detail of the latter providing an interesting step on the way to formalizing the protocol in full detail.

We have proved confidentiality and authentication properties [11] for our A level formalization, and have extended some of these proofs to our C level formalization; in each case, we use the notion of rank and corank functions, inspired by [20]. While Kerberos specifically disclaims responsibility for preventing denial of service attacks, we have noticed instances of other potentially curious protocol behavior. The first, which arises in both the A level and C level formalizations, violates properties that were proved to hold for Kerberos 4 [1] and highlights the structural differences between the messages in versions 4 and 5 of the protocol. The other three instances of curious behavior, seen only in our C level formalization, take advantage of protocol options available at this level; the first and third of these are related to the behavior also seen at the A level, while the second is completely unrelated. Our informal analysis of the B level formalization did not reveal any new anomalies.

A shorter, preliminary report on this work appeared in [6]. This paper adds the B level formalization, analysis of the C level Ticket-Granting Exchange and of the A level Client/Server Exchange, and some additional curious

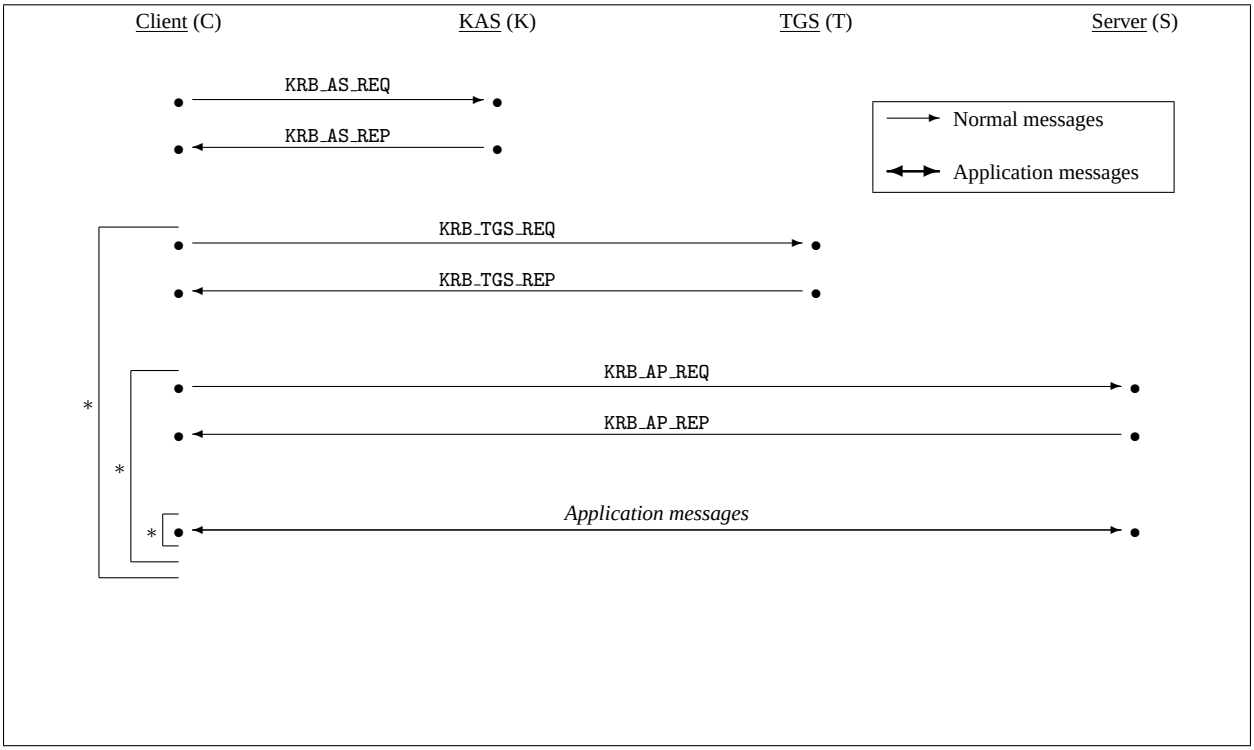

Figure 1: Expected message flow in Kerberos 5

protocol behavior. The A and C level formalizations have been updated in minor ways, as has our analysis of the A level Ticket-Granting Exchange.

The rest of this paper is structured as follows. In Sections 2 and 3 we give an overview of the Kerberos 5 protocol and the MSR formalism. Our A level, C level, and B level formalizations are given in Sections 4, 5, and 6, respectively. In Section 7 we discuss the curious protocol behavior that we have noted; in Section 8 we define the rank and corank functions we use in our analysis of the protocol, and in Section 9 use these classes of functions to prove confidentiality and authentication properties of Kerberos 5. The appendices provide details not included in the text, including the full proofs of protocol properties, MSR traces showing the anomalies discussed here, and a comparison of the network messages formalized here with the full messages specified by [16]

2 Overview of the Kerberos 5 Protocol

The Kerberos 5 protocol allows a client to repeatedly authenticate herself to multiple servers while minimizing the use of the long-term secret key(s) shared between the client and the Kerberos infrastructure. The client starts by obtaining a long-term credential, whose use requires her long term (shared) key, and then uses this to obtain short-term credentials for particular servers. Assume that a client C wishes to authenticate herself to an application server S. A standard run of Kerberos 5 which accomplishes this consists of three successive phases; the expected message flow in these phases is shown in Figure 1 and proceeds as follows.

- In the first phase, C sends a `KRB_AS_REQ` message to the *Kerberos Authentication Server* (KAS) K requesting a *ticket granting ticket* TGT for use with a particular *Ticket Granting Server* (TGS) T. K is expected to reply with a `KRB_AS_REP` message consisting of the ticket TGT and an encrypted component containing a fresh *authentication key* $AKey$ to be shared between C and T. TGT contains $AKey$ and is encrypted using the secret key k_T of T; the accompanying message is encrypted under C's secret key k_C. Each of k_C and k_T is shared between the named participant and a central key database from which it is accessible by K.

- In the second phase, C forwards TGT, along with an *authenticator* encrypted under $AKey$, to the TGS T as a KRB_TGS_REQ message; this requests a *service ticket* for use with the server S. T is expected to respond with a KRB_TGS_REP message consisting of the service ticket ST and an encrypted component containing a fresh *service key* $SKey$ to be shared between C and S. ST contains $SKey$ and is encrypted under S's secret key k_S, which is shared between S and the central key database accessible by T; the information for C, including $SKey$, is encrypted under $AKey$.

- In the third phase, C forwards ST and a new authenticator encrypted with $SKey$ in a KRB_AP_REQ message to S. If all credentials are valid, this application server will authenticate C and provide the service. The acknowledgment KRB_AP_REP message is optional.

A single ticket-granting ticket can be used to obtain several service tickets, possibly from several application servers, while it is valid. Similarly, a single service ticket for the application server S can be used for repeated service from S before it expires. In both cases, a fresh authenticator is required for each use of the ticket.

Note that the message flow is generally similar to that in Kerberos 4. However, Kerberos 5 includes a multitude of options, some of which we formalize in Section 5, not available in the previous version of the protocol. Additionally, the structure of the KRB_AS_REP and KRB_TGS_REP messages changed between versions 4 and 5 of the protocol. 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, the ticket-granting ticket and the service are sent without further encryption. This enables the cut and paste anomalies which we describe in Section 7 and slightly weakens the properties which were proved for Kerberos 4.

As we formalize different aspects of Kerberos 5, we will modify Figure 1 to show how we represent these protocol messages in MSR.

Finally, we note that the Kerberos 5 protocol has changed from its initial specification [13]; our work here is based on version 10 [16] of the revisions to [13]. Among other things, this adds anonymous tickets (in which the client's name is replaced by a generic username) to the protocol. We discuss curious protocol behavior related to anonymous tickets in Section 7.2; anonymous tickets may or may not be present in future revisions of Kerberos 5 [15], and have been removed from the current version of the protocol description [18]. (The description of the protocol is an IETF Internet Draft, each version of which has a six month lifetime.)

3 MSR

MSR originated as a simple logic-oriented language aimed at investigating the decidability of protocol analysis under a variety of assumptions [9, 10]. 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 [8]; its connections to other protocol analysis methods have been the subject of more recent work [7]. MSR 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 fresh data (*e.g.*, nonces or session keys). It supports an array of useful static checks that include type-checking and data access verification. It has so far been applied to toy protocols such as Needham-Schroeder and Neumann-Stubblebine [8]; one of the aims of this project is to evaluate it on a real-world protocol. We will introduce the syntax and operations of MSR as we go along.

3.1 Signature

In order to specify a protocol in MSR, the protocol entities need to be classified and appropriately (sub)typed. The signature fragment in Figure 2 sets up the typing infrastructure in the case of Kerberos 5, with the 'Types' column summarizing the types used in this work. Italicized types (*e.g.*, *ts* for TGS or server, and *tcs* for *ts* or client) are auxiliary and serve the purpose of making precise the definitions of dbK and shK; a laxer definition could do without them. The 'Subtyping' column expresses the subtyping relations satisfied by these types ($\tau <: \tau'$ means that τ is a subsort of τ'), with indentation used as a visual aid to track dependencies. The declarations shown in black support the A and B level formalizations (Sections 4 and 6) of this protocol, while the grayed-out additions are necessary for the C level specification (Section 5).

	Types	Subtyping	Names
(Messages)	msg : type.		m, X, Y
(Principals)	principal : type.	principal <: msg.	
	KAS : type.	KAS <: principal.	K
	tcs : type	*tcs* <: principal.	
	ts : type	*ts* <: *tcs*.	
	TGS : type.	TGS <: *ts*.	T
	server : type.	server <: *ts*.	S
	client : type.	client <: *tcs*.	C
(Encryption types)	etype : type.	etype <: msg.	e
(Keys)	*key* : etype → type.		
	dbK : etype → *tcs* → type.	$\forall e$: etype,A : *tcs*. dbK^e A <: *key*^{e}.	$k_{\_}$
	shK : etype → client → *ts* → type.	$\forall e$: etype,C : client, A : *ts*. shK^e C A <: *key*^{e}.	$AKey$
		$\forall e$: etype,C : client, A : *ts*. shK^e C A <: msg.	$SKey$
(Nonces)	nonce : type.	nonce <: msg.	n
(Timestamps)	time : type.	time <: msg.	$t_{\_,\_}$
(Options)	Opt : type.	Opt <: msg.	
	KOpt : type.	KOpt <: Opt.	$KOpts$
	TOpt : type.	TOpt <: Opt.	$TOpts$
	SOpt : type.	SOpt <: Opt.	$SOpts$
(Flags)	Flag : type.	Flag <: msg.	
	TFlag : type.	TFlag <: Flag.	$TFlags$
	SFlag : type.	SFlag <: Flag.	$SFlags$

Figure 2: An MSR Signature for the A level and C level Specifications of Kerberos 5

Observe that shared keys (shK _ _) can be part of a message, but database keys (dbK _), *i.e.*, keys shared between *tcs* principals and the key database, cannot. Notice also that the encryption types (needed in the C level specification) parameterize the various keys.

Additional declarations are needed to populate these types. In order to do so, we declare actual clients, servers, database keys, *etc*. Conventional names for various meta-syntactic entities are given in the rightmost column of Figure 2. For example, clients will typically called C. An underscore _ in a name will be appropriately instantiated in the discussion: for example, k_C will represent the database key of a client C and $t_{C,Sreq}$ will stand for a timestamp included by C in a request to S.

The syntax of messages is shown in Figure 3. The first two declarations formalize concatenation and shared-key encryption (with the encryption algorithm potentially depending on the encryption type). The third declaration captures message digests as an implementation of cryptographic hashing; these are declared similarly to shared-key encryption. We will generally keep the encryption type implicit unless we are specifically discussing it (as in Section 7.3).

(Pairing)	_, _ : msg → msg → msg.
(Encryption)	$\{\_\}^{\_}_{\_}$: etype → msg → *key* → msg.
(Message digest)	$[\_]^{\_}_{\_}$: etype → msg → *key* → msg.

Figure 3: Syntax for MSR messages.

3.2 States and roles

Intuitively, MSR represents the state of execution of a protocol as a multiset S of ground first-order formulas. 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 either *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, usually written as $L(\ldots)$, 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 $r = lhs \longrightarrow rhs$, where lhs is a finite multiset of facts and constraints. These constraints, which are not facts, are used by principals to, *e.g.*, 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 fresh data (*e.g.*, nonces or session keys) 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.

Part II
Formalizing Kerberos 5

4 A Level Formalization of Kerberos 5

Our A level formalization of Kerberos 5 has enough detail to prove authentication and confidentiality results (discussed in Section 9) but contains little else. The most notable omission is that of almost all timestamps; the sole one included here prevents the KRB_AP_REP message from being the encryption of an empty message. Bella and Paulson's thorough analysis of Kerberos 4 included consideration of timestamps. The primary differences between Kerberos 4 and Kerberos 5 do not involve timestamps; as we have focused on the unanalyzed details of Kerberos 5, we have omitted timestamps from this formalization of the protocol. However, a natural extension of our work thus far would be a formalization and analysis of Kerberos 5 which includes all (or most) of the timestamps and temporal checks used in this protocol. We leave this for future work.

Figure 4 updates Figure 1 to show how the different protocol messages are represented in this formalization of Kerberos 5.

4.1 The Authentication Service Exchange

Figure 5 shows the client role for the Authentication Service Exchange. When C : client undertakes this role, she may use rule $\alpha_{1.1}$ to send a KRB_AS_REQ message to any K : KAS requesting a ticket granting ticket for any T : TGS. In this formalization, the KRB_AS_REQ message contains C's name, T's name, and a freshly generated nonce n_1. When C sends the request, she also stores the information from the request (C, T, and n_1) in a role state predicate L.

C expects the response from K to be composed of her name, an opaque message (intended to be the ticket granting ticket), and another message encrypted under one of her database keys. This encrypted message is expected to contain a key of type shK C T to be shared between C and T and used in the Ticket Granting Exchange, the nonce n_1 from C's original request, and T's name. If a message of this form appears on the network (C uses the role state predicate L to ensure that the nonce and the name of the TGS in this message match those in her original request), then C may read this message from the network and save the relevant information. She does this using rule $\alpha_{1.2}$, which replaces the facts $\mathsf{N}(C, X, \{AKey, n_1, T\}_{k_C})$ and $L(C, T, n_1)$ with the fact $Auth_C(X, T, AKey)$, a memory predicate. C thus saves the (presumed) ticket X, the name T of the TGS for whom the ticket was requested, and the key $AKey$ to be shared by C and T.

Figure 6 shows the role of the Kerberos Authentication Server for the Authentication Service Exchange. Whenever a valid KRB_AS_REQ message appears on the network, any K : KAS may read that message from the network and respond appropriately. The validity of the KRB_AS_REQ message is determined by some external process (incorporating local policy) modelled by the constraint $Valid_K(C, T, n_1)$. K's response involves generating a fresh key $AKey$: shK C T to be shared by the client C and TGS T named in the KRB_AS_REQ message and then putting a message, intended for C, on the network. This network message contains C's name, the ticket granting ticket to be included in C's later request(s) to T, and data for C encrypted under one of her database keys. The ticket granting ticket contains the key to be shared between T and C and C's name, with these encrypted together using one of T's

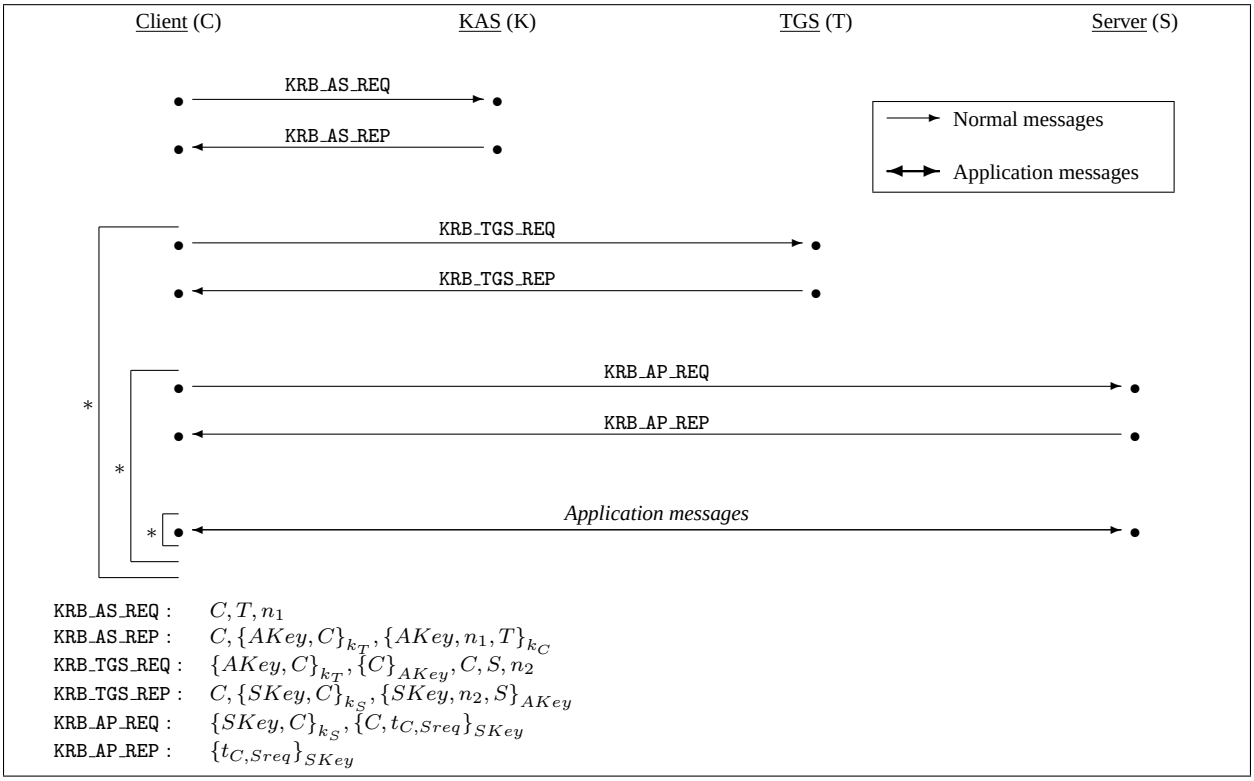

Figure 4: Protocol messages in the abstract formalization

$$\left(\begin{array}{lll} \exists L : \mathsf{client} \times \mathsf{TGS} \times \mathsf{nonce}. & & \\ \forall T : \mathsf{TGS}\,. & \xrightarrow{\alpha_{1.1}} & \exists n_1 : \mathsf{nonce} \\ \forall K : \mathsf{KAS}. & & \mathsf{N}(C, T, n_1) \\ & & L(C, T, n_1) \\ \forall \ldots \quad . & & \\ \forall k_C : \mathsf{dbK}\ C \quad . & \mathsf{N}(C, X, \{AKey, n_1, T\}_{k_C}) & \\ \forall AKey : \mathsf{shK}\ C\ T. & L(C, T, n_1) & \xrightarrow{\alpha_{1.2}} \quad Auth_C(X, T, AKey) \\ \forall X : \mathsf{msg} \quad . & & \\ \forall n_1 : \mathsf{nonce} \quad . & & \end{array}\right)^{\forall C:\mathsf{client}}$$

Figure 5: The client's role in the A level Authentication Service Exchange.

$$\left(\begin{array}{lll} \forall C : \mathsf{client} \quad . & & \\ \forall T : \mathsf{TGS} \quad . & & \\ \forall n_1 : \mathsf{nonce} \quad . & \mathsf{N}(C, T, n_1) & \xrightarrow{\alpha_{2.1}} \quad \exists AKey : \mathsf{shK}\ C\ T \\ \forall k_C : \mathsf{dbK}\ C \quad . & Valid_K(C, T, n_1) & \quad \mathsf{N}(C, \{AKey, C\}_{k_T}, \\ \forall k_T : \mathsf{dbK}\ T \quad . & & \quad \{AKey, n_1, T\}_{k_C}) \\ \forall AKey : \mathsf{shK}\ C\ T. & & \end{array}\right)^{\forall K:\mathsf{KAS}}$$

Figure 6: The authentication server's role in the A level Authentication Service Exchange.

database keys. The encrypted data for C are $AKey$, the nonce from the request to which K is responding, and T's name.

4.2 The Ticket-Granting Exchange

$$\left(\begin{array}{lll} \exists L : \mathsf{client}^{(C)} \times \mathsf{server} \times \mathsf{TGS}^{(T)} \times \mathsf{shK}\ C\ T \times \mathsf{nonce}. & & \\ & & \\ \forall T : \mathsf{TGS} \quad . & & \exists n_2 : \mathsf{nonce} \\ \forall S : \mathsf{server} \quad . & Auth_C(X, T, AKey) & \xrightarrow{\alpha_{3.1}} \quad \mathsf{N}(X, \{C\}_{AKey}, C, S, n_2) \\ \forall AKey : \mathsf{shK}\ C\ T. & & \quad Auth_C(X, T, AKey) \\ \forall X : \mathsf{msg} \quad . & & \quad L(C, S, T, AKey, n_2) \\ & & \\ \forall \ldots \quad . & \mathsf{N}(C, Y, & \\ \forall SKey : \mathsf{shK}\ C\ S. & \quad \{SKey, n_2, S\}_{AKey}) & \xrightarrow{\alpha_{3.2}} \quad Service_C(Y, S, SKey) \\ \forall Y : \mathsf{msg} \quad . & L(C, S, T, AKey, n_2) & \\ \forall n_2 : \mathsf{nonce} \quad . & & \end{array}\right)^{\forall C:\mathsf{client}}$$

Figure 7: The client's role in the A level Ticket-Granting Exchange.

Figure 7 gives the client role for the Ticket-Granting Exchange. If a client C has successfully completed the Authentication Service Exchange to get a ticket and key for T : TGS (as evidenced by the predicate $Auth_C(X, T, AKey)$), she may use rule $\alpha_{3.1}$ to send a KRB_TGS_REQ message to T. (This predicate does not guarantee that X is a ticket for T, only that it was received in the Authentication Service Exchange in the place for the ticket.) In firing rule $\alpha_{3.1}$, C generates a fresh nonce and puts a message on the network containing the presumed ticket X, an authenticator consisting of her name encrypted under $AKey$, her name, the name of the server S for whom she wants a service ticket, and the freshly generated nonce n_2. This rule preserves the predicate $Auth_C$ since tickets may be used multiple times (until they expire, which is not modelled here) and also creates a role state predicate L which contains information (C, S, T, $AKey$, and n_2) related to C's KRB_TGS_REQ message.

C expects the response from T to contain her name, and opaque message (intended to be the service ticket), and additional data encrypted under the key $AKey$ which is shared between C and T. These data are a key to be shared between C and the server S for whom C has requested credentials, the nonce n_2 from C's request to T, and S's name. If a message of this form appears on the network, C may use rule $\alpha_{3.2}$ to process it; as in the Authentication Service Exchange, C uses the role state predicate to ensure that the proper nonce is included in the response she receives. This rule consumes the network message and role state predicate and stores the (presumed) service ticket, server name, and new shared key in the memory predicate $Service_C$.

$$\left(\begin{array}{lll} \forall C : \mathsf{client} \quad . & & \\ \forall S : \mathsf{server} \quad . & \mathsf{N}(\{AKey, C\}_{k_T}, & \\ \forall AKey : \mathsf{shK}\ C\ T. & \quad \{C\}_{AKey}, C, S, n_2) & \xrightarrow{\alpha_{4.1}} \quad \exists SKey : \mathsf{shK}\ C\ S \\ \forall k_T : \mathsf{dbK}\ T \quad . & Valid_T(C, S, n_2) & \quad \mathsf{N}(C, \{SKey, C\}_{k_S}, \\ \forall k_S : \mathsf{dbK}\ S \quad . & & \quad \{SKey, n_2, S\}_{AKey}) \\ \forall n_2 : \mathsf{nonce} \quad . & & \end{array}\right)^{\forall T:\mathsf{TGS}}$$

Figure 8: The ticket granting server's role in the A level Ticket Granting Exchange.

Figure 8 contains the TGS role in the Ticket-Granting Exchange. When a valid KRB_TGS_REQ message appears on the network, the TGS T whose database key is used to encrypt the ticket in this message may process the request. As in the Authentication Service Exchange, the validity of the request is checked by an external process which is modelled here as the constraint $Valid_T$. T may process a valid request message by firing rule $\alpha_{4.1}$, which consumes the network message fact, generates a fresh key to be shared by the client C and server S named in the request, and puts a message intended for C on the network. This message contains C's name, a service ticket to be passed on to S, and data for C encrypted under the key $AKey$ which was included in the ticket-granting ticket and used by C to encrypt the authenticator in the KRB_TGS_REQ message. The service ticket is encrypted under one of S's database

keys and contains the freshly generated key $SKey$ and C's name. The data encrypted for C are the freshly generated key, the nonce from the KRB_TGS_REQ request to which T is responding, and the S's name.

4.3 The Client/Server Exchange

$$\left(\begin{array}{lll} \exists L : \mathsf{client}^{(C)} \times \mathsf{server}^{(S)} \times \mathsf{shK}\ C\ S \times \mathsf{time} \times \mathsf{msg}. & & \\ \forall S : \mathsf{server}\ . & & \\ \forall SKey : \mathsf{shK}\ C\ S. & Service_C(Y,S,SKey) & \alpha_{5.1} \quad \mathsf{N}(Y,\{C,t_{C,Sreq}\}_{SKey}) \\ \forall t_{C,Sreq} : \mathsf{time}\ . & Clock_C(t_{C,Sreq}) & \longrightarrow \quad Service_C(Y,S,SKey) \\ \forall Y : \mathsf{msg}\ . & & \quad L(C,S,SKey,t_{C,Sreq},Y) \\ \forall \ldots & \mathsf{N}(\{t_{C,Sreq}\}_{SKey}) & \alpha_{5.2} \\ & L(C,S,SKey,t_{C,Sreq},Y) & \longrightarrow \quad DoneMut_C(S,SKey) \end{array}\right)^{\forall C:\mathsf{client}}$$

Figure 9: The client's role in the A level Client/Server Exchange (with mutual authentication).

Figure 9 contains the client role for the Client/Server Exchange. Once the client C has obtained a (presumed) service ticket and key for the server S via the Ticket-Granting Exchange (storing these in the memory predicate $Service_C$), she may use the rule $\alpha_{5.1}$ to request service from S. In addition to the predicate $Service_C(Y,S,SKey)$, which stores the data from the ticket-granting exchange, the left side of this rule also contains the constraint $Clock_C(t_{C,Sreq})$. This is satisfied if and only if C's local time is $t_{C,Sreq}$: time. The firing of rule $\alpha_{5.1}$ puts a KRB_AP_REQ message on the network; this consists of the message Y from the $Service_C$ predicate, presumed to be a service ticket for S, and an authenticator. The authenticator is C's name and her current time encrypted together using the key $SKey$ which was stored with Y in $Service_C$. This rule preserves the $Service_C$ predicate for future reuse and also creates a role state predicate containing information about the request. Although not explicitly shown in this formalization, we assume that C requests mutual authentication from the server S; our detailed formalization below allows for C to specify whether or not S should respond.

C expects S to respond by sending a message consisting of $t_{C,Sreq}$ encrypted by the key $SKey$, shared by C and S, which C used to encrypt the authenticator in the KRB_AP_REQ message. If she sees a message of this form on the network (and has sent the matching initial request as indicated by the role state predicate L), C may use rule $\alpha_{5.2}$ to read this KRB_AP_REP message from the network. She then stores the server's name and the shared key $SKey$ in the $DoneMut_C$ memory predicate (indicating that the protocol has finished with mutual authentication used). The $DoneMut_C$ predicate and key $SKey$ would be used by C in additional interactions with S (such as sending messages related to network services provided by S) after authentication has been completed; as this is outside of the Kerberos protocol, we do not formalize it here.

$$\left(\begin{array}{lll} \forall C : \mathsf{client}\ . & \mathsf{N}(\{SKey,C\}_{k_S}, & \\ \forall SKey : \mathsf{shK}\ C\ S. & \quad \{C,t_{C,Sreq}\}_{SKey}) & \alpha_{6.1} \quad \mathsf{N}(\{t_{C,Sreq}\}_{SKey}) \\ \forall t_{C,Sreq} : \mathsf{time}\ . & Valid_S(C,t_{C,Sreq}) & \longrightarrow \quad Mem_S(C,SKey,t_{C,Sreq}) \\ \forall k_S : \mathsf{dbK}\ S\ . & & \end{array}\right)^{\forall S:\mathsf{server}}$$

Figure 10: The end server's role in the A level Client/Server Exchange (with mutual authentication).

The server's role in this exchange is shown in Figure 10. If the network contains a valid KRB_AP_REQ message intended for S (the ticket is encrypted using one of his database keys), then he may process it using rule $\alpha_{6.1}$. As for the other request messages, the validity of the KRB_AP_REQ message is determined by an external process which we formalize as the constraint $Valid_S(C,t_{C,Sreq})$. This rule puts a KRB_AP_REP message on the network; this consists of the timestamp $t_{C,Sreq}$ from the request being processed encrypted by the shared key $SKey$ included in the service ticket. S also stores the relevant information about this request and his response, namely the client's name, the shared key, and the time of the request, in the memory predicate Mem_S. This is analogous to C's predicate $DoneMut$ in that this information may be used after authentication is completed but is not part of the protocol itself and thus not used in our formalizations.

4.4 A level intruder formalization

In this section, we present the rules specifying the Dolev-Yao intruder model for Kerberos 5.

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

- the fairly standard operations of interception/transmission of a network message, decomposition/composition of a pair, and decryption/encryption of a message given a known key (Section 4.4.1);
- the often overlooked action of generating new data (Section 4.4.2);
- and the use of accessible data (Section 4.4.3).

4.4.1 Network, pairing and encryption rules

We present the following pairs of rules describing how the Dolev-Yao intruder can work with data on the network or in her possession; the rules in each pair are symmetric (*e.g.*, encryption and decryption) operations.

The intruder may intercept network messages (INT), removing them from the network, and transmit messages she knows (TRN):

$$\left(\forall m : \mathsf{msg}.\quad \mathsf{N}(m) \xrightarrow{\text{INT}} \mathsf{I}(m)\right)^{\mathsf{I}} \quad \left(\forall m : \mathsf{msg}.\quad \mathsf{I}(m) \xrightarrow{\text{TRN}} \mathsf{N}(m)\right)^{\mathsf{I}}$$

The intruder may decompose (DMC) and compose (CMP) compound messages:

$$\left(\forall m_1, m_2 : \mathsf{msg}.\quad \mathsf{I}(m_1, m_2) \xrightarrow{\text{DMC}} \begin{matrix}\mathsf{I}(m_1)\\ \mathsf{I}(m_2)\end{matrix}\right)^{\mathsf{I}} \quad \left(\forall m_1, m_2 : \mathsf{msg}.\quad \begin{matrix}\mathsf{I}(m_1)\\ \mathsf{I}(m_2)\end{matrix} \xrightarrow{\text{CMP}} \mathsf{I}(m_1, m_2)\right)^{\mathsf{I}}$$

If the intruder knows a shared key, she may decrypt (SDC′) and encrypt (SEC′) messages using this key:

$$\left(\begin{matrix}\forall C : \mathsf{client} & . & \\ \forall A : \text{TS} & . & \mathsf{I}(\{m\}_k)\\ \forall k : \mathsf{shK}\ C\ A. & & \mathsf{I}(k)\\ \forall m : \mathsf{msg} & . & \end{matrix} \xrightarrow{\text{SDC'}} \mathsf{I}(m)\right)^{\mathsf{I}} \quad \left(\begin{matrix}\forall C : \mathsf{client} & . & \\ \forall A : \text{TS} & . & \mathsf{I}(m)\\ \forall k : \mathsf{shK}\ C\ A. & & \mathsf{I}(k)\\ \forall m : \mathsf{msg} & . & \end{matrix} \xrightarrow{\text{SEC'}} \mathsf{I}(\{m\}_k)\right)^{\mathsf{I}}$$

If the intruder knows a database key, she may decrypt (DDC′) and encrypt (DEC′) messages using this key:

$$\left(\begin{matrix}\forall A : \text{TCS} & . & \mathsf{I}(\{m\}_k)\\ \forall k : \mathsf{dbK}\ A. & & \mathsf{I}(k)\\ \forall m : \mathsf{msg} & . & \end{matrix} \xrightarrow{\text{DDC'}} \mathsf{I}(m)\right)^{\mathsf{I}} \quad \left(\begin{matrix}\forall A : \text{TCS} & . & \mathsf{I}(m)\\ \forall k : \mathsf{dbK}\ A. & & \mathsf{I}(k)\\ \forall m : \mathsf{msg} & . & \end{matrix} \xrightarrow{\text{DEC'}} \mathsf{I}(\{m\}_k)\right)^{\mathsf{I}}$$

Finally, the intruder may duplicate (DPM and DPD) and delete (DLM and DLD) any of the data (messages or database keys) that she knows; the deletion rules can be safely omitted from the specification.

$$\left(\forall m : \mathsf{msg}.\quad \mathsf{I}(m) \xrightarrow{\text{DPM}} \begin{matrix}\mathsf{I}(m)\\ \mathsf{I}(m)\end{matrix}\right)^{\mathsf{I}} \qquad \left(\forall m : \mathsf{msg}.\quad \mathsf{I}(m) \xrightarrow{\text{DLM}} \cdot\right)^{\mathsf{I}}$$

$$\left(\begin{matrix}\forall A : \text{TCS} & . & \\ \forall k_A : \mathsf{dbK}\ A. & & \mathsf{I}(k_A)\end{matrix} \xrightarrow{\text{DPD}} \begin{matrix}\mathsf{I}(k_A)\\ \mathsf{I}(k_A)\end{matrix}\right)^{\mathsf{I}} \quad \left(\begin{matrix}\forall A : \text{TCS} & . & \\ \forall k_A : \mathsf{dbK}\ A. & & \mathsf{I}(k_A)\end{matrix} \xrightarrow{\text{DLD}} \cdot\right)^{\mathsf{I}}$$

4.4.2 Data generation rules

In general, the intruder should be able to generate everything an honest principal can generate, often nonces and session keys but nothing else. In the case of Kerberos, we must admit an exception to this rule: because principals forward uninterpreted data, we must also allow the intruder to create garbage, modelled as objects of the generic type msg.

The intruder may generate fresh nonces (NG), session keys (KG′), and generic messages (MG) using the following rules:

$$\left(\cdot \xrightarrow{\text{NG}} \exists n : \mathsf{nonce}\ \mathsf{I}(n)\right)^{\mathsf{I}}$$

$$\left(\begin{array}{ll}\forall C : \mathsf{client}. & \\ \forall A : \mathsf{TS} & .\end{array} \quad \cdot \quad \xrightarrow{\text{KG'}} \quad \exists k : \mathsf{shK}\ C\ A\ \mathsf{I}(k)\right)^{\mathsf{I}}$$

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

The intruder is not allowed to generate any other kind of data—principal names of any kind (the introduction of new agents happens out-of-band), long-term keys (they are distributed out-of-band), or timestamps (they are generated by an external clock, not by any principal)—as that would open the door to countless false attacks. Note that MG does not allow the generation of database keys (which are not subtypes of msg), nor does it generate terms which may be typed as anything other than msg. In particular, the messages freshly generated by MG are not the encryption of any other messages; in Section 8, we restate this in terms of the rank functions defined there as Axiom 1.

4.4.3 Data access rules

The intruder is entitled to look up the same data that any other principal may; she may store these data in the $\mathsf{I}(\_)$ predicate for later use.

The intruder has access to the name of any principal (client, server, TGS, or KAS):

$$\left(\forall A : \mathsf{principal}. \quad \cdot \quad \xrightarrow{\text{PA}} \quad \mathsf{I}(A)\right)^{\mathsf{I}}$$

The intruder also has access to any defined timestamp. As timestamps are guessable, they are thus qualitatively different from nonces here. We note that this may provide the intruder with more power than she would reasonably have.

$$\left(\forall t : \mathsf{time}. \quad \cdot \quad \xrightarrow{\text{TA}} \quad \mathsf{I}(t)\right)^{\mathsf{I}}$$

The intruder is entitled to lookup any session key she owns. This is modelled by the following two slightly asymmetric rules.

$$\left(\begin{array}{ll}\forall A : \mathsf{TS} & . \\ \forall k : \mathsf{shK}\ \mathsf{I}\ A. & \end{array} \quad \cdot \quad \xrightarrow{\text{SA1'}} \quad \mathsf{I}(k)\right)^{\mathsf{I}}$$

$$\left(\begin{array}{ll}\forall C : \mathsf{client} & . \\ \forall k : \mathsf{shK}\ C\ \mathsf{I}. & \end{array} \quad \cdot \quad \xrightarrow{\text{SA2'}} \quad \mathsf{I}(k)\right)^{\mathsf{I}}$$

It should be possible to prove that these rules are redundant since the intruder, like any other principal, is handed her session keys by the KAS or the TGS; these rules could then be eliminated.

Finally, the intruder may access any of her long-term (database) keys:

$$\left(\forall k : \mathsf{dbK}\ \mathsf{I}. \quad \cdot \quad \xrightarrow{\text{DA'}} \quad \mathsf{I}(k)\right)^{\mathsf{I}}$$

5 C Level Formalization of Kerberos 5

Our C level formalization is closer to the full Kerberos 5 specification than is our A level formalization in Section 4. Figure 11 updates Figure 1 to show the protocol messages in the C level formalization, with those details not in the A level formalization shown here in gray type.

In this formalization we extend the A level formalization by adding the message field which allows a client to request various options (including ANONYMOUS tickets where implemented) from a TGS as well as the message field (of type etype) which the client uses to request particular encryption method(s); we note curious behavior involving these details of the protocol in Section 7. We now include message digests as specified by Kerberos 5; one topic for further work is their utility in defending against the anomalous behavior discussed in Section 7. Here we add the message field (of type SOpt) which allows C to specify whether a KRB_AP_REP response from S is requested and have also incorporated error messages. These will allow investigation of protocol runs which would not appear in the previous analyses of Kerberos. Although we do not make use of it, we have also added the option field of type KOpt to parallel those of types TOpt and SOpt and various flag fields corresponding to the option fields already discussed.

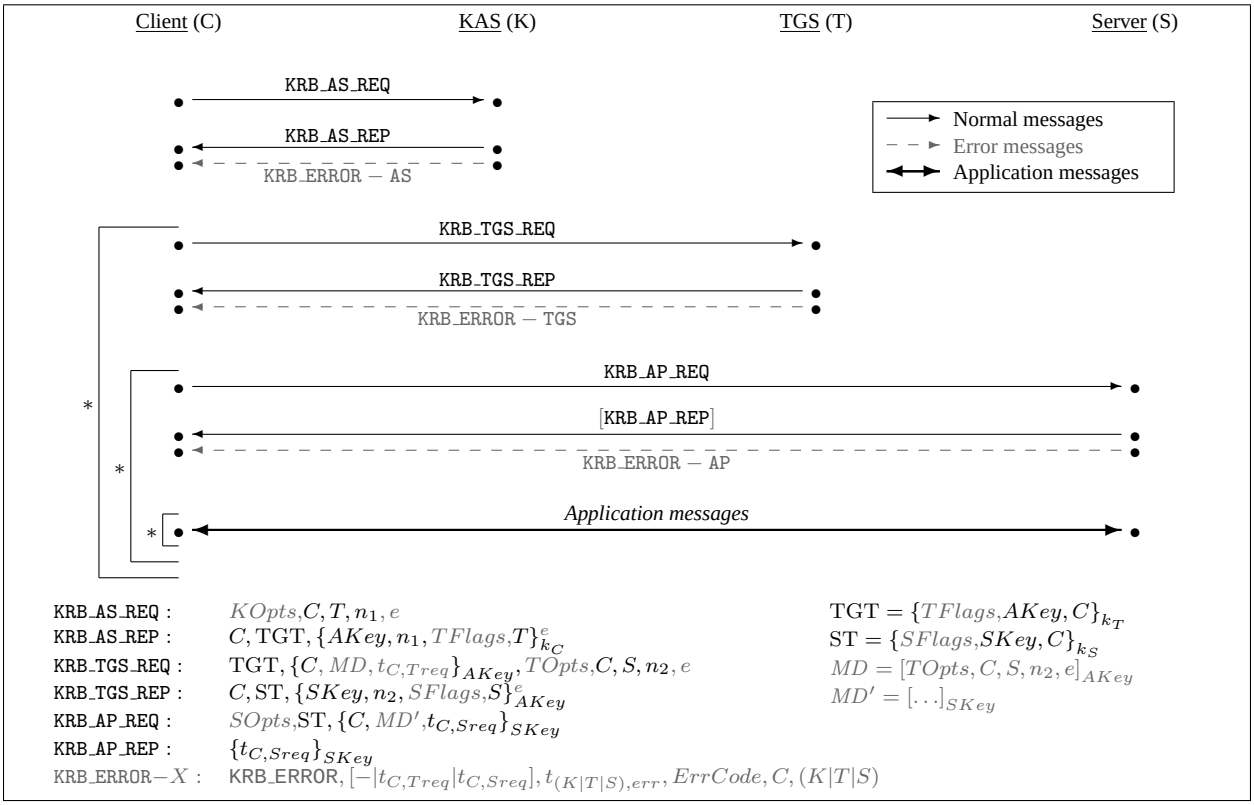

Figure 11: Protocol messages in the C level formalization

The C level formalization allows the various servers to send error messages in response to requests. In order to associate error messages with the corresponding requests, the authenticators sent to T and S now include the timestamps $t_{C,Treq}$ and $t_{C,Sreq}$; we have not, however, added any temporal checks involving these timestamps. Note that error messages are completely unencrypted and do not contain any information which was originally sent in encrypted form. These can be generated at will by the intruder.

Figures 12–19 give the non-intruder roles in the C level formalization. The gray text indicates detail which appears here but not in the A level formalization. Figures 18 and 19 are entirely gray as these roles are used only when a client does not request mutual authentication from a server, a protocol option not included in the A level formalization. The rule corresponding to rule $\alpha_{i.j}$ in the A level formalization is denoted by $\gamma_{i.j}$; alternative rules are indicated by priming j (*e.g.*, for error handling) or i (for the Client/Server Exchange without mutual authentication).

While fields specifying encryption type appear in several messages in this level, and should technically appear for every encrypted message that occurs (following to Section 3), we explicitly include these only in the Authentication Service Exchange rules (Section 5.1) unless we are discussing encryption types in particular (as in Section 7.3).

5.1 The Authentication Service Exchange

Figure 12 shows the client role for the Authentication Service Exchange in our detailed formalization. Rule $\gamma_{1.1}$ allows the client C to initiate the authentication process by sending a message to a KAS K. This extends Rule $\alpha_{1.1}$ by adding fields for options ($KOpts$) and encryption type (e) to both the request message placed on the network and the role state predicate L.

Rule $\gamma_{1.2}$ parallels rule $\alpha_{1.2}$ and allows C to process the expected response from K. In addition to accounting for $KOpts$ and e from the original request, this rule also accepts and stores the field $TFlags$ which describes the options actually granted by K (which may or may not match those requested by C).

$$
\left(
\begin{array}{lll}
\exists L : \mathsf{client} \times \mathsf{KOpt} \times \mathsf{TGS} \times \mathsf{nonce} \times \mathsf{etype}. & & \\
\forall T : \mathsf{TGS} \quad . & & \exists n_1 : \mathsf{nonce} \\
\forall K : \mathsf{KAS} \quad . & \xrightarrow{\gamma_{1.1}} & \mathsf{N}(KOpts, C, T, n_1, e) \\
\forall KOpts : \mathsf{KOpt}. & & L(C, KOpts, T, n_1, e) \\
\forall e : \mathsf{etype} \quad . & & \\
\forall \ldots \quad . & & \\
\forall k_C : \mathsf{dbK}\ C \quad . & & \\
\forall AKey : \mathsf{shK}\ C\ T. & \mathsf{N}(C, X, \{AKey, & \\
\forall X : \mathsf{msg} \quad . & n_1, TFlags, T\}^{e'}_{k_C}) & \xrightarrow{\gamma_{1.2}} \quad Auth_C(X, TFlags, \\
\forall e' : \mathsf{etype} \quad . & L(C, KOpts, T, n_1, e) & \quad T, AKey) \\
\forall n_1 : \mathsf{nonce} \quad . & & \\
\forall TFlags : \mathsf{TFlag}\ . & & \\
\forall \ldots \quad . & \mathsf{N}(\texttt{KRB\_ERROR}, t_{K,err}, & \\
\forall ErrorCode : \mathsf{msg}. & ErrorCode, C, K) & \xrightarrow{\gamma_{1.2'}} \quad ASError_C(\texttt{KRB\_ERROR}, \\
\forall t_{K,err} : \mathsf{time} \quad . & L(C, KOpts, T, n_1, e) & \quad t_{K,err}, ErrorCode, K)
\end{array}
\right)^{\forall C:\mathsf{client}}
$$

Figure 12: The client's role in the C level Authentication Service Exchange.

Finally, rule $\gamma_{1.2'}$ allows C to process generic error messages returned; this has no analogue in the A level formalization. If C has an AS request pending (as evidenced by the existence of the role state predicate L) and she sees an error message on the network which includes her name, then she may read the error message from the network and store the information contained in it in the memory predicate $ASError$ (which is new in this formalization). We do not currently use the $ASError$ predicate beyond this, but it might be used to allow more extensive error processing by the client. Note that the nonce n_1 is not returned in the error message; if C has multiple requests pending with a KAS K, each with its own nonce, there is no way for C to associate the error message with the request that generated the error.

Figure 13 shows the authentication server's role in the detailed formalization of the Authentication Service Exchange. Rule $\gamma_{2.1}$ parallels rule $\alpha_{2.1}$, adding details which allow the processing of the requested options $KOpts$ (the options actually granted are described in $TFlags$) and encryption type e. The constraint $Valid_K$ incorporates the additional details $KOpts$ and e added to the request. We use constraint $SetAuthFlags_K$ to implement K's granting of options in the ticket-granting ticket; we allow these to depend on the requested options and the names of the principals who will share the key generated by this rule. Like $Valid_K$, $SetAuthFlags$ may depend on local policy and is not explicitly described in the formalization. The constraint $SetETypes_K$ implements K's policies for selecting an encryption type e' for the data for C (taking into account her requested encryption type(s) e) and an encryption type e'' for the ticket for T.

$$
\left(
\begin{array}{lll}
\forall C : \mathsf{client} \quad . & & \\
\forall T : \mathsf{TGS} \quad . & & \\
\forall n_1 : \mathsf{nonce} \quad . & & \\
\forall e, e', e'' : \mathsf{etype} \quad . & \mathsf{N}(KOpts, C, T, n_1, e) & \exists AKey : \mathsf{shK}\ C\ T \\
\forall k_C : \mathsf{dbK}^{e'}\ C \quad . & Valid_K(KOpts, C, T, n_1, e) & \xrightarrow{\gamma_{2.1}} \quad \mathsf{N}(C, \{TFlags, AKey, C\}^{e''}_{k_T}, \\
\forall k_T : \mathsf{dbK}^{e''}\ T \quad . & SetAuthFlags_K(KOpts, C, T, TFlags) & \quad \{AKey, n_1, TFlags, T\}^{e'}_{k_C}) \\
\forall AKey : \mathsf{shK}\ C\ T. & SetETypes_K(C, e, e', T, e'') & \\
\forall KOpts : \mathsf{KOpt} \quad . & & \\
\forall TFlags : \mathsf{TFlag}\ . & & \\
\forall \ldots \quad . & \mathsf{N}(KOpts, C, T, n_1, e) & \\
\forall ErrorCode : \mathsf{msg}. & Invalid_K(KOpts, C, T, n_1, e) & \xrightarrow{\gamma_{2.1'}} \quad \mathsf{N}(\texttt{KRB\_ERROR}, t_{K,err}, \\
\forall t_{K,err} : \mathsf{time} \quad . & Clock_K(t_{K,err}) & \quad ErrorCode, C, K)
\end{array}
\right)^{\forall K:\mathsf{KAS}}
$$

Figure 13: The authentication server's role in the C level Authentication Service Exchange.

Rule $\gamma_{2.1'}$ allows the KAS K to send an error message in response to an invalid message request. The invalidity of a request is determined by the constraint $Invalid_K$, which is not defined in the formalization but which is assumed to hold when the request is invalid for the reason given by the error code $ErrorCode$. (If there are multiple reasons

why a request is invalid, we assume that $Invalid_K$ conforms to implementation-specific rules about which error code to return.) Rule $\gamma_{2.1'}$ also makes use of the constraint $Clock_K$ to generate a timestamp for the error message; as for $Clock_C$ in rule $\alpha_{5.1}$, this constraint is satisfied exactly when its argument matches K's local time.

5.2 The Ticket-Granting Exchange

Figure 14 shows the C level client role in the Ticket-Granting Exchange. The client C now includes a timestamp $t_{C,Treq}$ in her request to the TGS T, which she places on the network using rule $\gamma_{3.1}$. She uses $TOpts$ to specify the options she would like set on the new ticket, possibly including the request for an ANONYMOUS ticket if this option has been implemented as in [16]. C also uses the key $AKey$ which she shares with T to construct a keyed checksum $[TOpts, C, S, n_2, e]_{AKey}$; following the protocol specification, the checksum is not included in the data (the KRB-REQ-BODY part of the message) over which the checksum is taken. The role state predicate has been expanded from the A level version to store the additional information from her request in this formalization, namely $TOpts$, $t_{C,Treq}$, and e.

$$\left(\begin{array}{lll}
\exists L : \mathsf{client}^{(C)} \times \mathsf{TOpt} \times \mathsf{server} \times \mathsf{TGS}^{(T)} \times \mathsf{shK}\ C\ T \times \mathsf{nonce} \times \mathsf{time}. & & \\
\forall T : \mathsf{TGS}\ . & & \exists n_2 : \mathsf{nonce} \\
\forall S : \mathsf{server}\ . & & \mathsf{N}(X, \{C, [TOpts, C, S, n_2, e]_{AKey}, \\
\forall AKey : \mathsf{shK}\ C\ T. & & t_{C,Treq}\}_{AKey}, TOpts, C, S, n_2, e) \\
\forall X : \mathsf{msg}\ . & Auth_C(X, TFlags, T, AKey) & \xrightarrow{\gamma_{3.1}} \\
\forall t_{C,Treq} : \mathsf{time}\ . & Clock_C(t_{C,Treq}) & Auth_C(X, TFlags, T, AKey) \\
\forall TFlags : \mathsf{TFlag}\ . & & L(C, TOpts, S, T, \\
\forall TOpts : \mathsf{TOpt}\ . & & \quad AKey, n_2, t_{C,Treq}, e) \\
\forall e : \mathsf{etype}\ . & & \\
\forall \ldots\ . & \mathsf{N}(C, Y, & \\
\forall SKey : \mathsf{shK}\ C\ S. & \quad \{SKey, n_2, SFlags, S\}_{AKey}) & \xrightarrow{\gamma_{3.2}} \\
\forall Y : \mathsf{msg}\ . & L(C, TOpts, S, T, & Service_C(Y, SFlags, \\
\forall n_2 : \mathsf{nonce}\ . & \quad AKey, n_2, t_{C,Treq}, e) & \quad S, SKey) \\
\forall SFlags : \mathsf{SFlag}\ . & & \\
\forall \ldots\ . & \mathsf{N}(\mathtt{KRB_ERROR}, t_{C,Treq}, t_{T,err}, & \\
\forall ErrorCode : \mathsf{msg}. & \quad ErrorCode, C, T) & \xrightarrow{\gamma_{3.2'}} \\
\forall t_{T,err} : \mathsf{time}\ . & L(C, TOpts, S, T, & TGSError_C(T, t_{T,err}, ErrorCode) \\
 & \quad AKey, n_2, t_{C,Treq}, e) &
\end{array}\right)^{\forall C:\mathsf{client}}$$

Figure 14: The client's role in the C level Ticket-Granting Exchange.

Rule $\gamma_{3.2}$ parallels rule $\alpha_{3.2}$, adding the $SFlags$ field in the expected response from T (this indicates which options were actually granted on the ticket created by T) and extending the memory predicate $Service$ to store this information.

The client C processes error messages from T using rule $\gamma_{3.2'}$. As formalized here, the processing of these messages consists only of reading them from the network, deleting the role state predicate associated with the original request, and storing data about the error in the memory predicate $TGSError$. An extension of this formalization might make further use of this predicate to allow a more nuanced response by C to error messages.

Figure 15 shows the TGS role in the Ticket-Granting Exchange. Rule $\gamma_{4.1}$ allows the TGS T to process a valid request in the C level formalization from a client C, including the new message fields discussed for the client role. We also add the $SetServFlags_T$ constraint, which implements T's policies in granting options in response to those requested by C via the $TOpts$ field; here, we allow the granted options to depend upon the properties of the ticket-granting ticket ($TFlags$) as well as the principals C and S who will share the new key generated by this rule. Another constraint verifies the keyed checksum included in the request.

Rule $\gamma_{4.1'}$ allows T to generate an error message in response to an invalid request for a service ticket. The $Invalid_T$ constraint ensures that an appropriate error code is included in this message; note that $Invalid_T$ has sufficient information to determine whether the checksum included in the request message is correct. The $Clock_T$ constraint is used, as elsewhere, to obtain the local time.

$$
\left(\begin{array}{lll}
\forall C : \mathsf{client} \;. & & \\
\forall S : \mathsf{server} \;. & & \\
\forall AKey : \mathsf{shK}\ C\ T. & & \\
\forall k_T : \mathsf{dbK}\ T \;. & \mathsf{N}(\{TFlags, AKey, C\}_{k_T}, & \\
\forall k_S : \mathsf{dbK}\ S \;. & \quad \{C, ck, t_{C,Treq}\}_{AKey}, & \exists SKey : \mathsf{shK}\ C\ S \\
\forall n_2 : \mathsf{nonce} \;. & \quad TOpts, C, S, n_2, e) & \mathsf{N}(C, \{SFlags, SKey, C\}_{k_S}, \\
\forall t_{C,Treq} : \mathsf{time} \;. & Valid_T(TOpts, C, S, n_2, e, t_{C,Treq}) \quad \xrightarrow{\gamma_{4.1}} & \quad \{SKey, n_2, SFlags, S\}_{AKey}) \\
\forall TOpts : \mathsf{TOpt} \;. & SetServFlags_T(TOpts, TFlags, C, S, SFlags) & \\
\forall e : \mathsf{etype} \;. & ck = [TOpts, C, S, n_2, e]_{AKey} & \\
\forall SFlags : \mathsf{SFlag} \;. & & \\
\forall ck : \mathsf{msg} \;. & & \\
\forall TFlags : \mathsf{TFlag} \;. & & \\
 & \mathsf{N}(\{TFlags, AKey, C\}_{k_T}, & \\
\forall \ldots \;. & \quad \{C, ck, t_{C,Treq}\}_{AKey}, & \\
\forall ErrorCode : \mathsf{msg}. & \quad TOpts, C, S, n_2, e) & \mathsf{N}(\mathtt{KRB\_ERROR}, t_{C,Treq}, t_{T,err}, \\
\forall t_{T,err} : \mathsf{time} \;. & Invalid_T(TOpts, C, S, n_2, \quad \xrightarrow{\gamma_{4.1'}} & \quad ErrorCode, C, T) \\
 & \quad e, t_{C,Treq}, ck) & \\
 & Clock_T(t_{T,err}) &
\end{array}\right)^{\forall T:\mathsf{TGS}}
$$

Figure 15: The ticket granting server's role in the C level Ticket Granting Exchange.

5.3 The Client/Server Exchange

Figure 16 shows the client role for the Client/Server Exchange when the client C would like mutual authentication from the server S. Rule $\gamma_{5.1}$ parallels rule $\alpha_{5.1}$, adding a few new details. The $SOpts$ field allows C to request particular behavior on the part of S and is now stored in the role state predicate L. One of the options controlled by $SOpts$ is mutual authentication (a reply from S in response to a valid request from C). If this option is requested, the constraint $Mutual(SOpts)$ holds. We assume that this is the case here, and treat the other case separately below. The network message generated by C now also includes a keyed checksum. Since the contents of this are not specified by [13], we leave this as $[\ldots]_{SKey}$ here. If the properties of a checksum over a particular set of data are of interest, this can be specified in the formalization.

$$
\left(\begin{array}{lll}
\exists L : \mathsf{client}^{(C)} \times \mathsf{SOpt} \times \mathsf{server}^{(S)} \times \mathsf{shK}\ C\ S \times \mathsf{time} \times \mathsf{msg}. & & \\
\forall S : \mathsf{server} \;. & & \\
\forall SKey : \mathsf{shK}\ C\ S. & Service_C(Y, SFlags, S, SKey) & \mathsf{N}(SOpts, Y, \{C, [\ldots]_{SKey}, t_{C,Sreq}\}_{SKey}) \\
\forall t_{C,Sreq} : \mathsf{time} \;. & Mutual(SOpts) \quad \xrightarrow{\gamma_{5.1}} & Service_C(Y, SFlags, S, SKey) \\
\forall Y : \mathsf{msg} \;. & Clock_C(t_{C,Sreq}) & L(C, SOpts, S, SKey, t_{C,Sreq}, Y) \\
\forall SFlags : \mathsf{SFlag} \;. & & \\
\forall SOpts : \mathsf{SOpt} \;. & & \\
\forall \ldots. & \mathsf{N}(\{t_{C,Sreq}\}_{SKey}) \quad \xrightarrow{\gamma_{5.2}} & DoneMut_C(S, SKey) \\
 & L(C, SOpts, S, SKey, t_{C,Sreq}, Y) & \\
\forall \ldots \;. & \mathsf{N}(\mathtt{KRB\_ERROR}, t_{C,Sreq}, & \\
\forall ErrorCode : \mathsf{msg}. & \quad t_{S,err}, ErrorCode, C, S) \quad \xrightarrow{\gamma_{5.2'}} & APError_C(S, t_{S,err}, ErrorCode) \\
\forall t_{S,err} : \mathsf{time} \;. & L(C, SOpts, S, SKey, t_{C,Sreq}, Y) &
\end{array}\right)^{\forall C:\mathsf{client}}
$$

Figure 16: The client's role in the C level Client/Server Exchange with mutual authentication.

Rule $\gamma_{5.2}$ allows C to process a message whose form matches that of the expected response from S; this extends rule $\alpha_{5.2}$ to treat the addition of $SOpts$. Rule $\gamma_{5.2'}$ allows C to process an error message from S. As with error messages in the other exchanges, this deletes the relevant role state predicate and stores the currently unused error information in a memory predicate (here $APError$).

Figure 17 shows the server role for processing requests which require mutual authentication. If the network message is a valid request, the server S uses rule $\gamma_{6.1}$ to process it. The network message now includes the $SFlags$

field in the ticket and other fields described under the client's role. As for the client's role, we use the constraint $Mutual$ to ensure that mutual authentication has been requested (the case where it is not is treated below). The $Valid_S$ constraint has been extended to the new fields of the message. S also verifies the keyed checksum; as noted above, the data over which this is taken is unspecified by both the protocol and our formalization.

$$
\left(
\begin{array}{lll}
\forall C : \mathsf{client} \;. & & \\
\forall SKey : \mathsf{shK}\ C\ S. & \mathsf{N}(SOpts, \{SFlags, SKey, C\}_{k_S}, & \\
\forall t_{C,Sreq} : \mathsf{time} \;. & \quad \{C, ck, t_{C,Sreq}\}_{SKey}) & \xrightarrow{\gamma_{6.1}} \quad \mathsf{N}(\{t_{C,Sreq}\}_{SKey}) \\
\forall k_S : \mathsf{dbK}\ S \;. & Mutual(SOpts) & \qquad\qquad\quad Mem_S(C, SKey, t_{C,Sreq}) \\
\forall ck : \mathsf{msg} \;. & Valid_S(C, SOpts, SFlags, t_{C,Sreq}) & \\
\forall SOpts : \mathsf{SOpt} \;. & ck = [\ldots]_{SKey} & \\
\forall SFlags : \mathsf{SFlag} \;. & & \\
 & & \\
 & \mathsf{N}(SOpts, \{SFlags, SKey, C\}_{k_S}, & \\
\forall \ldots \;. & \quad \{C, ck, t_{C,Sreq}\}_{SKey}) & \\
\forall ErrCode : \mathsf{msg}. & Mutual(SOpts) & \xrightarrow{\gamma_{6.1'}} \quad \mathsf{N}(\mathtt{KRB\_ERROR}, t_{C,Sreq}, \\
\forall t_{S,err} : \mathsf{time} \;. & Invalid_S(C, SOpts, & \qquad t_{S,err}, ErrCode, C, S) \\
 & \quad SFlags, SKey, t_{C,Sreq}, ck) & \\
 & Clock_S(t_{S,err}) &
\end{array}
\right)^{\forall S:\mathsf{server}}
$$

Figure 17: The end server's role in the C level Client/Server Exchange with mutual authentication.

Rule $\gamma_{6.1'}$ allows S to respond to an invalid request appearing on the network. The invalidity is determined by the $Invalid_S$ constraint; this includes the possibility of an incorrect checksum. S uses the $Clock_S$ constraint to obtain the correct timestamp for the error message and places this message on the network.

5.4 The Client/Server Exchange without mutual authentication

The rules for the Client/Server Exchange when mutual authentication is not requested have the same level of detail as the rules for requests involving mutual authentication. The differences are solely to complete the exchange after the client's message to the server without having the client wait for a response.

Figure 18 gives the client role for the Client/Server Exchange when the client C does not request mutual authentication. Rule $\gamma_{5'.1}$ differs from rule $\gamma_{5.1}$ only in that $SOpts$ is not set for mutual authentication (thus satisfying the $NoMutual$ constraint) and C's use of the $DoneNoMut$ predicate to store information about her request; as with the $DoneMut$ predicate, this formalization does not make further use of this data. C does keep the role state predicate in order to tie the processing of error messages to the original request.

$$
\left(
\begin{array}{lll}
\exists L : \mathsf{client}^{(C)} \times \mathsf{SOpt} \times \mathsf{server}^{(S)} \times \mathsf{shK}\ C\ S \times \mathsf{time} \times \mathsf{msg}. & & \\
 & & \\
\forall S : \mathsf{server} \;. & & \\
\forall SKey : \mathsf{shK}\ C\ S. & & \mathsf{N}(SOpts, Y, \{C, [\ldots]_{SKey}, t_{C,Sreq}\}_{SKey}) \\
\forall t_{C,Sreq} : \mathsf{time} \;. & Service_C(Y, SFlags, S, SKey) & \\
\forall Y : \mathsf{msg} \;. & NoMutual(SOpts) & \xrightarrow{\gamma_{5'.1}} \quad Service_C(Y, SFlags, S, SKey,) \\
\forall SFlags : \mathsf{SFlag} \;. & Clock_C(t_{C,Sreq}) & \quad L(C, SOpts, S, SKey, t_{C,Sreq}, Y) \\
\forall SOpts : \mathsf{SOpt} \;. & & \quad DoneNoMut_C(S, SKey) \\
 & & \\
\forall \ldots \;. & \mathsf{N}(\mathtt{KRB\_ERROR}, t_{C,Sreq}, & \\
\forall ErrCode : \mathsf{msg}. & \quad t_{S,err}, ErrCode, C, S) & \xrightarrow{\gamma_{5'.2'}} \quad APError_C(S, t_{S,err}, \\
\forall t_{S,err} : \mathsf{time} \;. & L(C, SOpts, S, SKey, t_{C,Sreq}, Y) & \qquad ErrCode) \\
 & &
\end{array}
\right)^{\forall C:\mathsf{client}}
$$

Figure 18: The client's role in the C level Client/Server Exchange without mutual authentication.

Figure 19 gives the corresponding server role. As for the client, the $Mutual$ constraint is replaced by the $NoMutual$ constraint. When processing a valid request using rule $\gamma_{6'.1}$, the server S does not produce a network message, but still creates the memory predicate Mem. Error messages are as in the mutual authentication case.

$$\left(\begin{array}{lll}
\forall C : \mathsf{client} \quad . & & \\
\forall SKey : \mathsf{shK}\ C\ S. & \mathsf{N}(SOpts, \{SFlags, SKey, C\}_{k_S}, & \\
\forall t_{C,Sreq} : \mathsf{time} \quad . & \quad \{C, ck, t_{C,Sreq}\}_{SKey}) & \\
\forall k_S : \mathsf{dbK}\ S \quad . & NoMutual(SOpts) & \xrightarrow{\gamma_{6'.1}} \quad Mem_S(C, SKey, t_{C,Sreq}) \\
\forall ck : \mathsf{msg} \quad . & Valid_S(C, SOpts, SFlags, t_{C,Sreq}) & \\
\forall SOpts : \mathsf{SOpt} \quad . & ck = [\ldots]_{SKey} & \\
\forall SFlags : \mathsf{SFlag} \quad . & & \\
 & & \\
 & \mathsf{N}(SOpts, \{SFlags, SKey, C\}_{k_S}, & \\
\forall \ldots \quad . & \quad \{C, ck, t_{C,Sreq}\}_{SKey}) & \\
\forall ErrCode : \mathsf{msg}. & NoMutual(SOpts) & \xrightarrow{\gamma_{6'.1'}} \quad \mathsf{N}(\mathtt{KRB_ERROR}, t_{C,Sreq}, \\
\forall t_{S,err} : \mathsf{time} \quad . & Invalid_S(C, SOpts, & \quad t_{S,err}, ErrCode, C, S) \\
 & \quad SFlags, SKey, t_{C,Sreq}, ck) & \\
 & Clock_S(t_{S,err}) &
\end{array}\right)^{\forall S:\mathsf{server}}$$

Figure 19: The end server's role in the C level Client/Server Exchange without mutual authentication.

5.5 C level intruder formalization

The admissible Dolev-Yao intruder actions are updated to reflect the added detail in the C level principal roles and additions to the syntax of the MSR specification.

The intruder 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. The necessary extensions to the network, pairing, and encryption rules are as follows.

$$\left(\begin{array}{lll}
\forall C : \mathsf{client} \quad . & & \\
\forall A : \mathsf{TS} \quad . & \mathsf{I}(\{m\}^e_k) & \\
\forall e : \mathsf{etype} \quad . & \mathsf{I}(k) & \xrightarrow{\text{SDC'}} \quad \mathsf{I}(m) \\
\forall k : \mathsf{shK}^e\ C\ A. & & \\
\forall m : \mathsf{msg} \quad . & &
\end{array}\right)^{\mathsf{I}} \quad
\left(\begin{array}{lll}
\forall C : \mathsf{client} \quad . & & \\
\forall A : \mathsf{TS} \quad . & \mathsf{I}(m) & \\
\forall e : \mathsf{etype} \quad . & \mathsf{I}(k) & \xrightarrow{\text{SEC'}} \quad \mathsf{I}(\{m\}^e_k) \\
\forall k : \mathsf{shK}^e\ C\ A. & & \\
\forall m : \mathsf{msg} \quad . & &
\end{array}\right)^{\mathsf{I}}$$

$$\left(\begin{array}{lll}
\forall A : \mathsf{TCS} \quad . & & \\
\forall e : \mathsf{etype} \quad . & \mathsf{I}(\{m\}^e_k) & \\
\forall k : \mathsf{dbK}^e\ A. & \mathsf{I}(k) & \xrightarrow{\text{DDC'}} \quad \mathsf{I}(m) \\
\forall m : \mathsf{msg} \quad . & &
\end{array}\right)^{\mathsf{I}} \quad
\left(\begin{array}{lll}
\forall A : \mathsf{TCS} \quad . & & \\
\forall e : \mathsf{etype} \quad . & \mathsf{I}(m) & \\
\forall k : \mathsf{dbK}^e\ A. & \mathsf{I}(k) & \xrightarrow{\text{DEC'}} \quad \mathsf{I}(\{m\}^e_k) \\
\forall m : \mathsf{msg} \quad . & &
\end{array}\right)^{\mathsf{I}}$$

$$\left(\begin{array}{lll}
\forall A : \mathsf{TCS} \quad . & & \\
\forall e : \mathsf{etype} \quad . & \mathsf{I}(k_A) & \xrightarrow{\text{DPD}} \quad \begin{array}{l}\mathsf{I}(k_A) \\ \mathsf{I}(k_A)\end{array} \\
\forall k_A : \mathsf{dbK}^e\ A. & &
\end{array}\right)^{\mathsf{I}} \quad
\left(\begin{array}{lll}
\forall A : \mathsf{TCS} \quad . & & \\
\forall e : \mathsf{etype} \quad . & \mathsf{I}(k_A) & \xrightarrow{\text{DLD}} \quad . \\
\forall k_A : \mathsf{dbK}^e\ A. & &
\end{array}\right)^{\mathsf{I}}$$

We need to update the data generation rule KG' as follows.

$$\left(\begin{array}{lll}
\forall C : \mathsf{client}. & & \\
\forall A : \mathsf{TS} \quad . & \cdot & \xrightarrow{\text{KG'}} \quad \exists k : \mathsf{shK}^e\ C\ A\ \mathsf{I}(k) \\
\forall e : \mathsf{etype} \ . & &
\end{array}\right)^{\mathsf{I}}$$

The data access rules need to be updated as follows.

$$\left(\begin{array}{lll}
\forall A : \mathsf{TS} \quad . & & \\
\forall e : \mathsf{etype} \quad . & \cdot & \xrightarrow{\text{SA1'}} \quad \mathsf{I}(k) \\
\forall k : \mathsf{shK}^e\ \mathsf{I}\ A. & &
\end{array}\right)^{\mathsf{I}} \quad
\left(\begin{array}{lll}
\forall C : \mathsf{client} \quad . & & \\
\forall e : \mathsf{etype} \quad . & \cdot & \xrightarrow{\text{SA2'}} \quad \mathsf{I}(k) \\
\forall k : \mathsf{shK}^e\ C\ \mathsf{I}. & &
\end{array}\right)^{\mathsf{I}} \quad
\left(\begin{array}{lll}
\forall e : \mathsf{etype} \ . & & \\
\forall k : \mathsf{dbK}^e\ \mathsf{I}. & \cdot & \xrightarrow{\text{DA'}} \quad \mathsf{I}(k)
\end{array}\right)^{\mathsf{I}}$$

The C level intruder also makes use of rules which do not extend rules of the A level intruder. Here the intruder can construct a message digest as long as she knows the proper key. However, there is no disassembling rule for message

digests since (cryptographic) hashing does not permit recovering a message.

$$\left(\begin{array}{lll} \forall C : \mathsf{client} & . & \\ \forall A : \mathsf{TS} & . & \mathsf{I}(m) \\ \forall e : \mathsf{etype} & . & \mathsf{I}(k) \\ \forall k : \mathsf{shK}^e\ C\ A. & & \\ \forall m : \mathsf{msg} & . & \end{array} \xrightarrow{\text{MD}} \mathsf{I}([m]_k^e)\right)^{\mathsf{I}}$$

The updates to the generation rules are limited to allowing the intruder to choose the encryption type of any session key she 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 Section 4.4.

Data access rules are subject to similar changes. However, we treat the new data types, encryption types, options and flags, similarly to timestamps: each of them range over a limited number of legal values, each being public knowledge. As for timestamps, these rules make encryption types, options and flags guessable.

$$\left(\forall e : \mathsf{etype}. \quad \cdot \quad \xrightarrow{\text{EA}} \quad \mathsf{I}(e)\right)^{\mathsf{I}}$$

$$\left(\forall o : \mathsf{Opt}. \quad \cdot \quad \xrightarrow{\text{OA}} \quad \mathsf{I}(o)\right)^{\mathsf{I}}$$

$$\left(\forall f : \mathsf{Flag}. \quad \cdot \quad \xrightarrow{\text{FA}} \quad \mathsf{I}(f)\right)^{\mathsf{I}}$$

Observe that, by virtue of subtyping, the last two inference figures apply to each of the subsorts of Opt and Flag.

Other information that was inaccessible in the A level specification of the intruder remains inaccessible.

6 B Level Protocol Formalization

Our B level formalization extends our A level formalization by adding different details than we use in our C level formalization. Figure 20 updates Figure 1 to show the protocol messages in the B level formalization, with those details not in the A level formalization shown here in gray type.

The primary new detail here is the addition of timestamps and other time data to the protocol messages. As in our C level formalization, we also make mutual authentication by the end server optional and add error messages (following the full protocol specification, these are again unencrypted).

The MSR signature of the B level formalization is the same as for the A level formalization; the B level intruder rules are unchanged from those for the A level.

6.1 The Authentication Service Exchange

The client's actions in Authentication Service Exchange are formalized in Figure 21. Rule $\beta_{1.1}$ allows C to initiate the Authentication Service Exchange with some K : KAS; this leaves the corresponding A level rule ($\alpha_{1.1}$) unmodified.

Rule $\beta_{1.2}$ allows C to process a KRB_AS_REP message naming her and including the nonce she previously sent to K. This formalization adds two time data to the KRB_AS_REP message; these are discussed in connection with K's rule $\beta_{2.1}$. As in the A level formalization, C uses the $Auth_C$ predicate to save information about this exchange (here also including the two time fields) for future use.

If C has an AS request pending, indicated by the role state predicate L, and an error message containing her name appears on the network, then she may use rule $\beta_{1.2'}$ to process the message. In doing so, C stores the relevant information in the $ASError_C$ memory predicate. Here an error message consists of the message type (KRB_ERROR), time that the error occurred ($t_{K,err}$), description of the error ($ErrorCode$), and the names of the client and KAS involved.

The role of the KAS in this exchange is shown in Figure 22. Rule $\beta_{2.1}$ allows K to process a valid KRB_AS_REQ message. As in the A level formalization, the validity of this message is determined by the $Valid_K$ external process. If the request is valid, K reads the current time from his local clock via the $Clock_K$ constraint, uses the

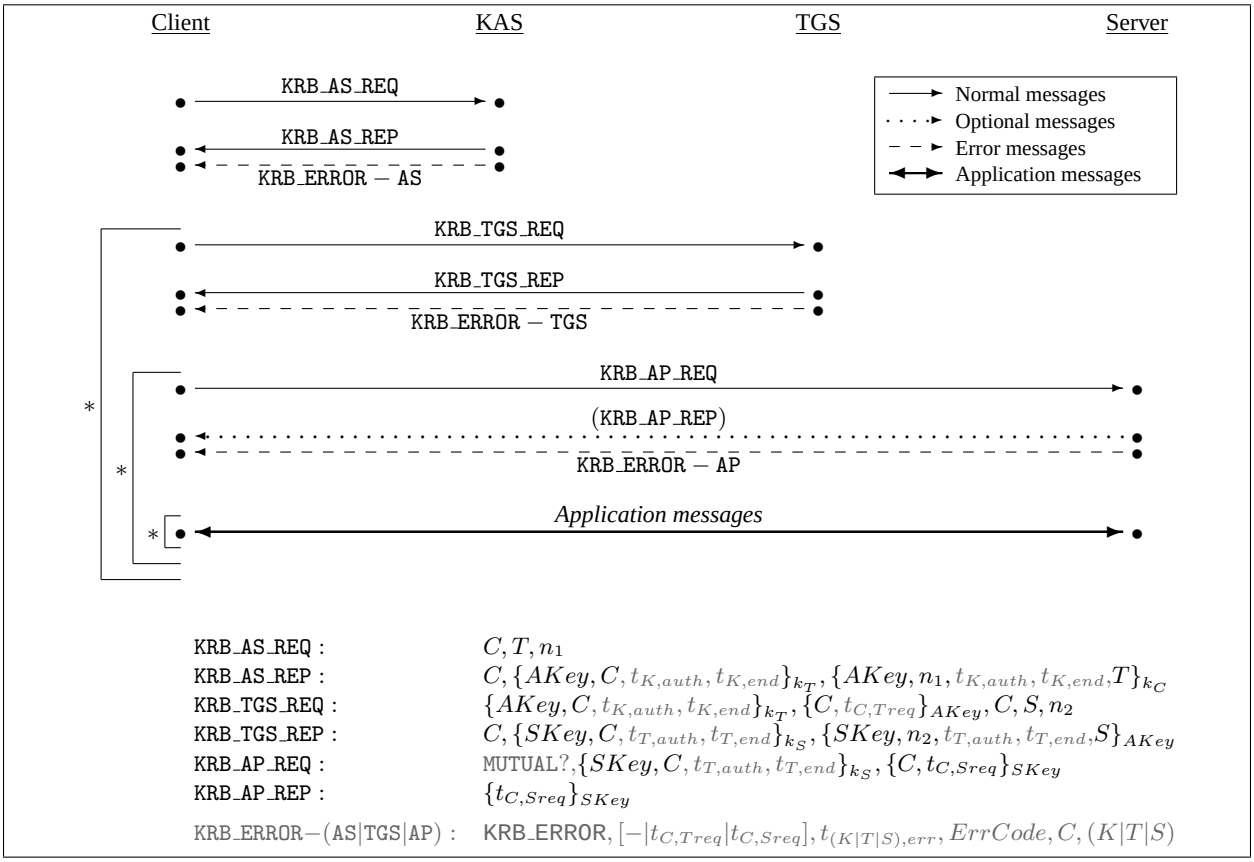

Figure 20: Kerberos 5 Messages in the B level Formalization

$ATicketExp$ constraint to determine an appropriate ticket expiration time ($t_{K,end}$), and sends a KRB_AS_REP message which extends the A level version by adding these two time values. Rule $\beta_{2.1'}$ formalizes K's response to an invalid KRB_AS_REQ (as determined by the $Invalid_K$ constraint); this error message also includes K's current local time.

6.2 The Ticket-Granting Exchange

The client's role in the Ticket-Granting Exchange is shown in Figure 23. Rule $\beta_{3.1}$ extends rule $\alpha_{3.1}$ so that C now includes a timestamp (the value of her local clock) in the authenticator she sends to the TGS; the predicate $Auth_C$ is extended as discussed under rule $\beta_{1.2}$, and the role state predicate L now stores the timestamp added in the B level. Rule $\beta_{3.2}$ updates rule $\alpha_{3.2}$ to store the two new time values in the KRB_TGS_REP message in the predicate $Service_C$.

Rules $\beta_{3.2'}$ and $\beta_{3.2''}$ allow C to handle error messages that appear to relate to a KRB_TGS_REQ message she has sent (as evidenced by the role state predicate L). These cover errors due specifically to an expired Ticket-Granting Ticket ($\beta_{3.2'}$) and generic errors ($\beta_{3.2''}$); in both cases the contents of the error message are stored in the $TGSError_C$ predicate, although these data are not used elsewhere in this formalization.

The role of the TGS in this exchange is shown in Figure 24. This formalization adds the time data $t_{T,end}$, the expiration time of the service ticket obtained via the constraint $STicketExp$, and $t_{T,auth}$, the current time on T's local clock obtained via $Clock_T$. T also performs an explicit temporal check, given by the constraint $t_{T,auth} < t_{K,end}$, to ensure that the ticket-granting ticket has not expired; this check is not performed by the constraint $Valid_T$, which does not have access to the local time $t_{T,auth}$.

Rule $\beta_{4.1'}$ allows T to respond to a KRB_TGS_REQ message which contains an expired ticket-granting ticket (in

$$
\left(
\begin{array}{lll}
\exists L : \mathsf{client} \times \mathsf{TGS} \times \mathsf{nonce}. & & \\
\forall T : \mathsf{TGS}\,. & \xrightarrow{\beta_{1.1}} & \begin{array}{l} \exists n_1 : \mathsf{nonce} \\ \mathsf{N}(C, T, n_1) \\ L(C, T, n_1) \end{array} \\
\forall K : \mathsf{KAS}. & & \\
\forall \ldots \quad . & & \\
\forall k_C : \mathsf{dbK}\ C \quad . & & \\
\forall AKey : \mathsf{shK}\ C\ T. & \mathsf{N}(C, X, \{AKey, n_1, t_{K,auth}, t_{K,end}, T\}_{k_C}) & \xrightarrow{\beta_{1.2}} \quad Auth_C(X, T, AKey, t_{K,auth}, t_{K,end}) \\
\forall X : \mathsf{msg} \quad . & & \\
\forall n_1 : \mathsf{nonce} \quad . & L(C, T, n_1) & \\
\forall t_{K,auth} : \mathsf{time} \quad . & & \\
\forall t_{K,end} : \mathsf{time} \quad . & & \\
\forall \ldots \quad . & \mathsf{N}(\mathtt{KRB\_ERROR}, t_{K,err}, ErrorCode, C, K) & \xrightarrow{\beta_{1.2'}} \quad ASError_C(\mathtt{KRB\_ERROR}, t_{K,err}, ErrorCode, K) \\
\forall ErrorCode : \mathsf{msg}. & & \\
\forall t_{K,err} : \mathsf{time} \quad . & L(C, T, n_1) &
\end{array}
\right)^{\forall C:\mathsf{client}}
$$

Figure 21: The client's role in the B level Authentication Service Exchange.

$$
\left(
\begin{array}{lll}
\forall C : \mathsf{client} \quad . & & \\
\forall T : \mathsf{TGS} \quad . & \mathsf{N}(C, T, n_1) & \\
\forall n_1 : \mathsf{nonce} \quad . & Valid_K(C, T, n_1) & \\
\forall k_C : \mathsf{dbK}\ C \quad . & Clock_K(t_{K,auth}) & \xrightarrow{\beta_{2.1}} \quad \begin{array}{l} \exists AKey : \mathsf{shK}\ C\ T \\ \mathsf{N}(C, \{AKey, C, t_{K,auth}, t_{K,end}\}_{k_T}, \\ \quad \{AKey, n_1, t_{K,auth}, t_{K,end}, T\}_{K_C}) \end{array} \\
\forall k_T : \mathsf{dbK}\ T \quad . & ATicketExp(C, T, t_{K,auth}, t_{K,end}) & \\
\forall AKey : \mathsf{shK}\ C\ T. & & \\
\forall t_{K,end} : \mathsf{time} \quad . & & \\
\forall t_{K,auth} : \mathsf{time} \quad . & & \\
\forall \ldots \quad . & \mathsf{N}(C, T, n_1) & \\
\forall ErrorCode : \mathsf{msg}. & Invalid_K(C, T, n_1, ErrorCode) & \xrightarrow{\beta_{2.1'}} \quad \mathsf{N}(\mathtt{KRB\_ERROR}, t_{K,err}, ErrorCode, C, K) \\
\forall t_{K,err} : \mathsf{time} \quad . & Clock_K(t_{K,err}) &
\end{array}
\right)^{\forall K:\mathsf{KAS}}
$$

Figure 22: The authentication server's role in the B level Authentication Service Exchange.

$$
\left(
\begin{array}{lll}
\multicolumn{3}{l}{\exists L : \mathsf{client}^{(C)} \times \mathsf{server} \times \mathsf{TGS}^{(T)} \times \mathsf{shK}\ C\ T \times \mathsf{time} \times \mathsf{nonce}.} \\
\forall T : \mathsf{TGS} \quad . & & \\
\forall S : \mathsf{server} \quad . & & \exists n_2 : \mathsf{nonce} \\
\forall AKey : \mathsf{shK}\ C\ T. & Auth_C(X, T, AKey, & \mathsf{N}(X, \{C, t_{C,Treq}\}_{AKey}, \\
\forall X : \mathsf{msg} \quad . & \quad t_{K,auth}, t_{K,end}) & \quad C, S, n_2) \\
\forall t_{C,Treq} : \mathsf{time} \quad . & Clock_C(t_{C,Treq}) \quad \xrightarrow{\beta_{3.1}} & Auth_C(X, T, AKey, \\
\forall t_{K,auth} : \mathsf{time} \quad . & & \quad t_{K,auth}, t_{K,end}) \\
\forall t_{K,end} : \mathsf{time} \quad . & & L(C, S, T, AKey, t_{C,Treq}, n_2) \\
 & & \\
\forall \ldots \quad . & & \\
\forall SKey : \mathsf{shK}\ C\ S. & \mathsf{N}(C, Y, \{SKey, n_2, t_{T,auth}, & \\
\forall Y : \mathsf{msg} \quad . & \quad t_{T,end}, S\}_{AKey}) & \\
\forall n_2 : \mathsf{nonce} \quad . & L(C, S, T, AKey, t_{C,Treq}, n_2) \quad \xrightarrow{\beta_{3.2}} & Service_C(Y, S, SKey, \\
\forall t_{T,auth} : \mathsf{time} \quad . & & \quad t_{T,auth}, t_{T,end}) \\
\forall t_{t,end} : \mathsf{time} \quad . & & \\
 & & \\
\forall \ldots \quad . & \mathsf{N}(\mathtt{KRB\_ERROR}, t_{C,Treq}, & \\
\forall \mathtt{KRB\_ERROR} : \mathsf{msg}. & \quad t_{T,err}, \mathtt{TKT\_EXP}, C, T) & \\
\forall \mathtt{TKT\_EXP} : \mathsf{msg} \quad . & L(C, S, T, AKey, t_{C,Treq}, n_2) \quad \xrightarrow{\beta_{3.2'}} & TGSError_C(t_{C,Treq}, t_{T,err}, \mathtt{TKT\_EXP}, T) \\
\forall t_{T,err} : \mathsf{time} \quad . & & \\
 & & \\
\forall \ldots \quad . & \mathsf{N}(\mathtt{KRB\_ERROR}, t_{C,Treq}, & \\
\forall ErroCode : \mathsf{msg}. & \quad t_{T,err}, ErrorCode, C, T) & \\
 & L(C, S, T, AKey, t_{C,Treq}, n_2) \quad \xrightarrow{\beta_{3.2''}} & TGSError_C(t_{C,Treq}, t_{T,err}, ErrorCode, T)
\end{array}
\right)^{\forall C:\mathsf{client}}
$$

Figure 23: The client's role in the B level Ticket-Granting Exchange.

which case the constraint $t_{T,err} \geq t_{K,end}$ is satisfied). Note that the $Valid_T$ constraint (which checks the validity of every aspect of the request except that the ticket is not expired) is satisfied. Everything else is similar to the error messages sent by K in the Authentication Service exchange. Rule $\beta_{4.1''}$ allows T to respond to a KRB_TGS_REQ message which is invalid for some other reason (as determined by the $Invalid_T$ external process).

6.3 The Client/Server Exchange with mutual authentication

The client's role in this exchange is shown in Figure 25. Rules $\beta_{5.1}$ and $\beta_{5.2}$ extend the A level rules $\alpha_{5.1}$ and $\alpha_{5.2}$ by adding the time data $t_{T,auth}$ and $t_{T,end}$ to the predicate $Service_C$ and by explicitly setting the MUTUAL_REQUIRED bit in the KRB_AP_REQ message. Recall that $t_{C,Sreq}$ was the only timestamp already included in the A level formalization. Rules $\beta_{5.2'}$ and $\beta_{5.2''}$ are essentially the same as rules $\beta_{3.2'}$ and $\beta_{3.2''}$ in the Ticket-Granting Exchange.

The server's role in this exchange is shown in Figure 26. Rule $\beta_{6.1}$ extends rule $\alpha_{6.1}$ to account for the time data now in the KRB_AP_REQ message (including in the validity check by $Valid_S$ and the data saved in Mem_S); S also ensures that the service ticket has not expired using the constraints $Clock_S(t_{S,now})$ and $t_{S,now} < t_{T,end}$. Note that if S accepts C's request, he sends a confirmation message because the MUTUAL_REQUIRED bit was set in the KRB_AP_REQ message.

Rule $\beta_{6.1'}$ parallels rule $\beta_{4.1'}$ in the Ticket-Granting Exchange and allows S to send an error message in response to a KRB_AP_REQ message which is valid except for an expired ticket (so that the constraints $Valid_S(C, t_{T,auth}, t_{C,Sreq})$ and $t_{S,err} \geq t_{T,end}$ hold). Rule $\beta_{6.1''}$ allows S to send the appropriate error message in response to a KRB_AP_REQ message which is invalid for some other reason.

6.4 The Client/Server Exchange without mutual authentication

The client's role in this exchange is shown in Figure 27. Rule $\beta_{5'.1}$ differs from rule $\beta_{5.1}$ only in that the MUTUAL_NON_REQUIRED bit is set in the KRB_AP_REQ message (with a corresponding change in the information saved in L) and the data for

$$
\left(\begin{array}{lll}
\begin{array}{l}
\forall C : \mathsf{client} \quad . \\
\forall S : \mathsf{server} \quad . \\
\forall AKey : \mathsf{shK}\ C\ T. \\
\forall k_T : \mathsf{dbK}\ T \quad . \\
\forall k_S : \mathsf{dbK}\ S \quad . \\
\forall n_2 : \mathsf{nonce} \quad . \\
\forall t_{K,auth} : \mathsf{time} \quad . \\
\forall t_{K,end} : \mathsf{time} \quad . \\
\forall t_{T,auth} : \mathsf{time} \quad . \\
\forall t_{T,end} : \mathsf{time} \quad . \\
\forall t_{C,Treq} : \mathsf{time} \quad .
\end{array}
&
\begin{array}{l}
\mathsf{N}(\{AKey, C, \\
\quad t_{K,auth}, t_{K,end}\}_{k_T}, \\
\{C, t_{C,Treq}\}_{AKey} \\
\quad C, S, n_2) \\
Valid_T(C, S, n_2, t_{K,auth}, t_{C,Treq}) \\
STicketExp(C, S, t_{S,end}) \\
Clock_T(t_{T,auth}) \\
t_{T,auth} < t_{K,end}
\end{array}
&
\xrightarrow{\beta_{4.1}}
&
\begin{array}{l}
\exists SKey : \mathsf{shK}\ C\ S \\
\mathsf{N}(C, \{SKey, C, \\
\quad t_{T,auth}, t_{T,end}\}_{k_S}, \\
\quad \{SKey, n_2, t_{T,auth}, \\
\quad t_{T,end}, S\}_{AKey})
\end{array} \\
\begin{array}{l}
\forall \ldots \quad . \\
\forall \mathsf{KRB\_ERROR} : \mathsf{msg}. \\
\forall \mathtt{TKT\_EXP} : \mathsf{msg} \quad . \\
\forall t_{T,err} : \mathsf{time} \quad .
\end{array}
&
\begin{array}{l}
\mathsf{N}(\{AKey, C \\
\quad t_{K,auth}, t_{K,end}\}_{k_T} \\
\{C, t_{C,Treq}\}_{AKey}, \\
\quad C, S, n_2) \\
Valid_T(C, S, n_2, t_{K,auth}, t_{C,Treq}) \\
Clock_T(t_{T,err}) \\
t_{T,err} \geq t_{K,end}
\end{array}
&
\xrightarrow{\beta_{4.1'}}
&
\begin{array}{l}
\mathsf{N}(\mathsf{KRB\_ERROR}, t_{C,Treq}, \\
\quad t_{T,err}, \mathtt{TKT\_EXP}, C, T)
\end{array} \\
\begin{array}{l}
\forall \ldots \quad . \\
\forall ErrorCode : \mathsf{msg}.
\end{array}
&
\begin{array}{l}
\mathsf{N}(\{AKey, C \\
\quad t_{K,auth}, t_{K,end}\}_{k_T} \\
\{C, t_{C,Treq}\}_{AKey}, \\
\quad C, s, n_2) \\
Invalid_T(C, S, n_2, t_{K,auth}, t_{C,Treq}, ErrorCode) \\
Clock_T(t_{T,err})
\end{array}
&
\xrightarrow{\beta_{4.1''}}
&
\begin{array}{l}
\mathsf{N}(\mathsf{KRB\_ERROR}, t_{C,Treq}, \\
\quad t_{T,err}, ErrorCode, C, T)
\end{array}
\end{array}\right)^{\forall T:\mathsf{TGS}}
$$

Figure 24: The ticket granting server's role in the B level Ticket-Granting Exchange.

$$
\left(\begin{array}{lll}
\exists L : \mathsf{client}^{(C)} \times \mathsf{server}^{(S)} \times \mathsf{shK}\ C\ S \times \mathsf{time} \times \mathsf{msg} \times \mathsf{msg}. & & \\
\begin{array}{l}
\forall S : \mathsf{server} \quad . \\
\forall SKey : \mathsf{shK}\ C\ S. \\
\forall Y : \mathsf{msg} \quad . \\
\forall t_{C,Sreq} : \mathsf{time} \quad . \\
\forall t_{T,auth} : \mathsf{time} \quad . \\
\forall t_{T,end} : \mathsf{time} \quad .
\end{array}
&
\begin{array}{l}
Service_C(Y, S, SKey, t_{T,auth}, t_{T,end}) \\
Clock_C(t_{C,Sreq})
\end{array}
&
\xrightarrow{\beta_{5.1}}
&
\begin{array}{l}
\mathsf{N}(\mathtt{MUTUAL\_REQUIRED}, \\
\quad Y, \{C, t_{C,Sreq}\}_{SKey}) \\
Service_C(Y, S, SKey, t_{T,auth}, t_{T,end}) \\
L(C, S, SKey, t_{C,Sreq}, Y, \\
\quad \mathtt{MUTUAL\_REQUIRED})
\end{array} \\
\forall \ldots. &
\begin{array}{l}
\mathsf{N}(\{t_{C,Sreq}\}_{SKey}) \\
L(C, S, SKey, t_{C,Sreq}, Y, \\
\quad \mathtt{MUTUAL\_REQUIRED})
\end{array}
&
\xrightarrow{\beta_{5.2}}
&
DoneMut_C(S, SKey) \\
\begin{array}{l}
\forall \ldots \quad . \\
\forall \mathsf{KRB\_ERROR} : \mathsf{msg}. \\
\forall \mathtt{TKT\_EXP} : \mathsf{msg} \quad . \\
\forall t_{S,err} : \mathsf{time} \quad .
\end{array}
&
\begin{array}{l}
\mathsf{N}(\mathsf{KRB\_ERROR}, t_{C,Sreq}, t_{S,err}, \\
\quad \mathtt{TKT\_EXP}, C, S) \\
L(C, S, SKey, t_{C,Sreq}, Y)
\end{array}
&
\xrightarrow{\beta_{5.2'}}
&
APError_C(t_{C,Sreq}, t_{S,err}, \mathtt{TKT\_EXP}, S) \\
\begin{array}{l}
\forall \ldots \quad . \\
\forall ErrorCode : \mathsf{msg}.
\end{array}
&
\begin{array}{l}
\mathsf{N}(\mathsf{KRB\_ERROR}, t_{C,Sreq}, t_{S,err}, \\
\quad ErrorCode, C, S) \\
L(C, S, SKey, t_{C,Sreq}, Y)
\end{array}
&
\xrightarrow{\beta_{5.2''}}
&
APError_C(t_{C,Sreq}, t_{S,err}, ErrorCode, S)
\end{array}\right)^{\forall C:\mathsf{client}}
$$

Figure 25: The client's role in the B level Client/Server Exchange with mutual authentication.

$$
\left(\begin{array}{lll}
\forall C : \mathsf{client} \;. & \mathsf{N}(\mathtt{MUTUAL\_REQUIRED} & \\
\forall SKey : \mathsf{shK}\ C\ S. & \quad \{SKey, C, t_{T,auth}, t_{T,end}\}_{k_S}, & \\
\forall t_{C,Sreq} : \mathsf{time} \;. & \quad \{C, t_{C,Sreq}\}_{SKey}) & \mathsf{N}(\{t_{C,Sreq}\}_{SKey}) \\
\forall k_S : \mathsf{dbK}\ S \;. & Clock_S(t_{S,now}) \quad \xrightarrow{\beta_{6.1}} & Mem_S(C, SKey, t_{C,Sreq}, \\
\forall t_{S,auth} : \mathsf{time} \;. & Valid_S(C, t_{T,auth}, t_{C,Sreq}) & \quad t_{T,auth}, t_{T,end}) \\
\forall t_{T,auth} : \mathsf{time} \;. & t_{S,auth} < t_{T,end} & \\
\forall t_{T,end} : \mathsf{time} \;. & & \\
 & & \\
\forall \ldots \;. & \mathsf{N}(\mathtt{MUTUAL\_REQUIRED}, & \\
\forall \mathsf{KRB\_ERROR} : \mathsf{msg}. & \quad \{SKey, C, t_{T,auth}, t_{T,end}\}_{k_S}, & \\
\forall \mathtt{TKT\_EXP} : \mathsf{msg} \;. & \quad \{C, t_{C,Sreq}\}_{SKey}) & \mathsf{N}(\mathsf{KRB\_ERROR}, t_{C,Sreq}, \\
\forall t_{S,err} : \mathsf{time} \;. & Valid_S(C, t_{T,auth}, t_{C,Sreq}) \quad \xrightarrow{\beta_{6.1'}} & \quad t_{S,err}, \mathtt{TKT\_EXP}, C, S) \\
 & Clock_S(t_{S,err}) & \\
 & t_{S,err} \geq t_{T,end} & \\
 & & \\
 & \mathsf{N}(\mathtt{MUTUAL\_REQUIRED}, & \\
\forall \ldots \;. & \quad \{SKey, C, t_{T,auth}, t_{T,end}\}_{k_S}, & \\
\forall ErrorCode : \mathsf{msg}. & \quad \{C, t_{C,Sreq}\}_{SKey}) & \mathsf{N}(\mathsf{KRB\_ERROR}, t_{C,Sreq}, \\
 & Invalid_S(C, t_{T,auth}, t_{C,Sreq}, ErrorCode) \quad \xrightarrow{\beta_{6.1''}} & \quad t_{S,err}, ErrorCode, C, S) \\
 & Clock_S(t_{S,err}) & \\
 & &
\end{array}\right)^{\forall S:\mathsf{server}}
$$

Figure 26: The end server's role in the B level Client/Server Authentication Exchange with mutual authentication.

further communication with S (which is not modelled in this formalization) are stored in the $DoneNoMut_C$ predicate. There is no analogue of rule $\beta_{5.2}$ because C does not require a response from S. Rules $\beta_{5'.2'}$ and $\beta_{5'.2''}$ simply update rules $\beta_{5.2'}$ and $\beta_{5.2''}$ to account for the change from `MUTUAL_REQUIRED` to `MUTUAL_NON_REQUIRED` .

The server's role in this exchange is shown in Figure 28. Rule $\beta_{6'.1}$ is rule $\beta_{6.1}$ with the (here undesired) response from S omitted. Rules $\beta_{6'.1'}$ and $\beta_{6'.1''}$ are the same as rules $\beta_{6.1'}$ and $\beta_{6.1''}$ except that they handle bad requests in which the `MUTUAL_NON_REQUIRED` bit is set.

$$
\left(
\begin{array}{lll}
\exists L : \mathsf{client}^{(C)} \times \mathsf{server}^{(S)} \times \mathsf{shK}\ C\ S \times \mathsf{time} \times \mathsf{msg} \times \mathsf{msg}. & & \\
\forall S : \mathsf{server}\ . & & \mathsf{N}(\mathtt{MUTUAL\_NON\_REQUIRED}, \\
\forall SKey : \mathsf{shK}\ C\ S. & Service_C(Y, S, SKey & \quad Y, \{C, t_{C,Sreq}\}_{SKey}) \\
\forall Y : \mathsf{msg}\ . & \quad t_{T,auth}, t_{T,end}) & Service_C(Y, S, SKey \\
\forall t_{C,Sreq} : \mathsf{time}\ . & Clock_C(t_{C,Sreq}) & \quad t_{T,auth}, t_{T,end}) \\
 & & \xrightarrow{\beta_{5'.1}} \quad L(C, S, SKey, t_{C,Sreq}, Y \\
 & & \quad \mathtt{MUTUAL\_NON\_REQUIRED}) \\
 & & DoneNoMut_C(S, SKey) \\
\forall \ldots\ . & \mathsf{N}(\mathsf{KRB\_ERROR}, t_{C,Sreq}, t_{S,err}, & \\
\forall \mathsf{KRB\_ERROR} : \mathsf{msg}. & \quad \mathtt{TKT\_EXP}, C, S) & \xrightarrow{\beta_{5'.2'}} \quad APError_C(t_{C,Sreq}, t_{S,err}, \\
\forall \mathtt{TKT\_EXP} : \mathsf{msg}\ . & L(C, S, SKey, t_{C,Sreq}, Y, & \quad \mathtt{TKT\_EXP}, S) \\
\forall t_{S,err} : \mathsf{time}\ . & \quad \mathtt{MUTUAL\_NON\_REQUIRED}) & \\
 & \mathsf{N}(\mathsf{KRB\_ERROR}, t_{C,Sreq}, t_{S,err}, & \\
\forall \ldots\ . & \quad ErrorCode, C, S) & \xrightarrow{\beta_{5'.2''}} \quad APError_C(, t_{C,Sreq}, t_{S,err}, \\
\forall ErrorCode : \mathsf{msg}. & L(C, S, SKey, t_{C,Sreq}, Y, & \quad ErrorCode, S) \\
 & \quad \mathtt{MUTUAL\_NON\_REQUIRED}) &
\end{array}
\right)^{\forall C:\mathsf{client}}
$$

Figure 27: The client's role in the B level Client/Server Exchange without mutual authentication.

$$
\left(
\begin{array}{lll}
\forall C : \mathsf{client}\ . & & \\
\forall SKey : \mathsf{shK}\ C\ S. & \mathsf{N}(\mathtt{MUTUAL\_NON\_REQUIRED} & \\
\forall t_{C,Sreq} : \mathsf{time}\ . & \quad \{SKey, C, t_{T,auth}, t_{T,end}\}_{k_S}, & \\
\forall k_S : \mathsf{dbK}\ S\ . & \quad \{C, t_{C,Sreq}\}_{SKey}) & \xrightarrow{\beta_{6'.1}} \quad Mem_S(C, SKey, t_{C,Sreq}, \\
\forall t_{S,auth} : \mathsf{time}\ . & Clock_S(t_{S,auth}) & \quad t_{T,auth}, t_{T,end}) \\
\forall t_{T,auth} : \mathsf{time}\ . & Valid_S(C, t_{T,auth}, t_{C,Sreq}) & \\
\forall t_{T,end} : \mathsf{time}\ . & t_{S,auth} < t_{T,end} & \\
 & \mathsf{N}(\mathtt{MUTUAL\_NON\_REQUIRED}, & \\
\forall \ldots\ . & \quad \{SKey, C, t_{T,auth}, t_{t,end}\}_{k_S}, & \\
\forall \mathsf{KRB\_ERROR} : \mathsf{msg}. & \quad \{C, t_{C,Sreq}\}_{SKey}) & \xrightarrow{\beta_{6'.1'}} \quad \mathsf{N}(\mathsf{KRB\_ERROR}, t_{C,Sreq}, t_{S,err}, \\
\forall \mathtt{EXP\_ERR} : \mathsf{msg}\ . & Valid_S(C, t_{T,auth}, t_{C,Sreq}) & \quad \mathtt{TKT\_EXP}, \mathtt{C}, \mathtt{S}) \\
\forall t_{S,err} : \mathsf{time}\ . & Clock_S(t_{S,err}) & \\
 & t_{S,err} \geq t_{T,end} & \\
 & \mathsf{N}(\mathtt{MUTUAL\_NON\_REQUIRED}, & \\
\forall \ldots\ . & \quad \{SKey, C, t_{T,auth}, t_{t,end}\}_{k_S}, & \\
\forall ErrorCode : \mathsf{msg}. & \quad \{C, t_{C,Sreq}\}_{SKey}) & \xrightarrow{\beta_{6'.1''}} \quad \mathsf{N}(\mathsf{KRB\_ERROR}, t_{C,Sreq}, t_{S,err}, \\
 & Invalid_S(C, t_{T,auth}, t_{C,Sreq}, ErrorCode) & \quad ErrorCode) \\
 & Clock_S(t_{S,err}) &
\end{array}
\right)^{\forall S:\mathsf{server}}
$$

Figure 28: The end server's role in the B level Client/Server Exchange without mutual authentication.

Part III
Analyzing Kerberos 5

7 Anomalous Protocol Behavior

In this section we describe some anomalous protocol behavior that we have noted as we have analyzed Kerberos 5. This does not pose a fundamental threat to the security of the protocol—in Section 9 we prove that Kerberos 5 enjoys a number of confidentiality and authentication properties—so the traces described in this section may be viewed as 'interesting curiosities.'

We believe that the attack found by Mitchell, Mitchell, and Stern [14] against a simplified version of Kerberos 5 does not appear in our formalization because in their encoding the KRB_TGS_REP message from T to C did not include S encrypted under $AKey$.

7.1 Ticket anomaly

The primary structural difference between versions 4 and 5 of Kerberos is the manner the KAS and the TGS transmit the ticket-granting and service tickets. In Kerberos 4, the client receives these tickets as part of the data encrypted under either her long term (dbK) key or a session key that she knows. We saw that version 5 sends the tickets as a separate component without additional encryption. Thus it is possible for the intruder to take advantage of this new message structure to tamper with the unprotected ticket (although she is unable to cause serious problems by doing so). Figure 29 updates figure 4 to illustrate the message flow in one such scenario. An MSR trace realizing this anomaly in our A level formalization is given in Appendix C.1.

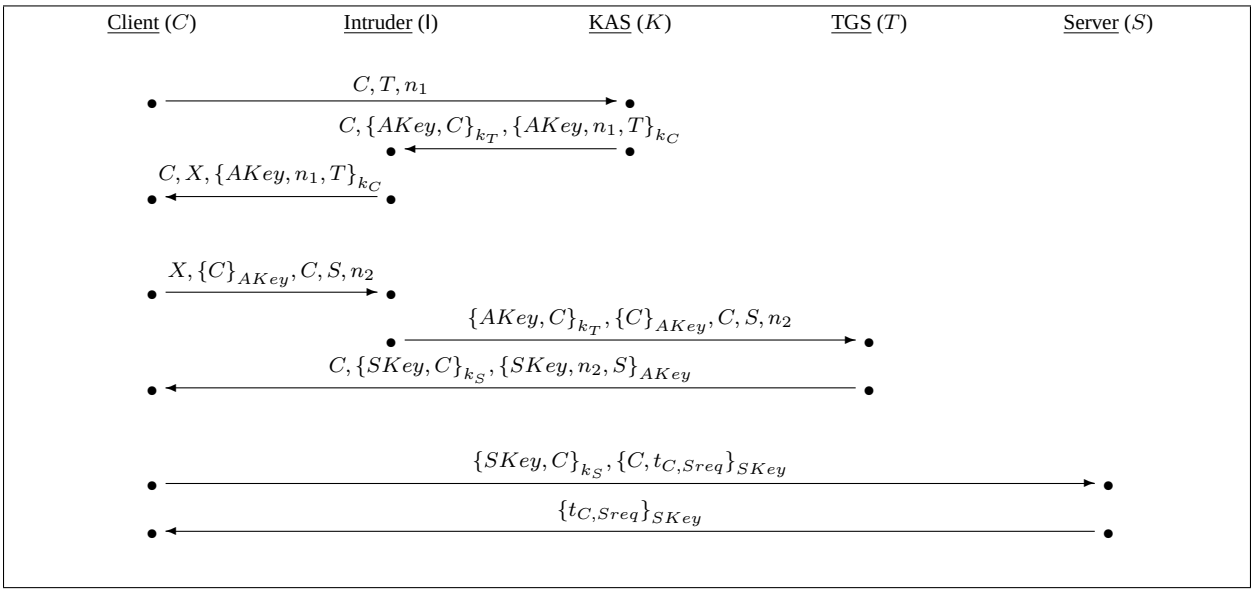

Figure 29: A level message flow in the ticket anomaly.

Here C sends her KRB_AS_REQ message as usual, but the intruder intercepts the KRB_AS_REP message from K. She replaces the ticket with a generic message X and stores the ticket in her memory; C cannot detect this because she expects to be unable to read the contents of the ticket. When C tries to send a KRB_TGS_REQ message to T (using the meaningless X instead of the ticket), I intercepts this message and replaces X with the original ticket from K and forwards the result (a well-formed KRB_TGS_REQ message) to T. T replies with a ticket for S, and the protocol continues as though the intruder had taken no action.

As a result of the actions of the intruder, T has granted a service ticket to C even though C has never sent a valid KRB_TGS_REQ message (she does *think* that she has, however). Moreover, barring additional interference by the intruder, subsequent requests by the client using the "ticket" X in her possession will fail for reasons unknown to the her. This anomaly does not appear to provide an attack against keys, but it does give a counterexample to the direct translation of a property of Kerberos 4: when Theorem 6.22 of [1] is translated to our A level formalization of Kerberos 5, it becomes the following.

Violated Property 1. *For* C : client, T : TGS, $AKey$: shK $C\,T$, k_T : dbK T, *and* S : server, *if* $\{C\}_{AKey}$ *and* $\{AKey, C\}_{k_T}$ *have appeared on the network (possibly encrypted), and* I *does not have access to* $AKey$, *then for some* n_2 : nonce, C *put the message* $\{AKey, C\}_{k_T}, \{C\}_{AKey}, C, S, n_2$ *on the network.*

As our example shows, this property does not hold in our A level formalization of Kerberos 5; it does not hold in our C level formalization either. It was possible for it to hold in Kerberos 4 because the message in that protocol sent from K to C was the equivalent of (after omitting timestamps) $\{AKey, T, TGT\}_{k_C}$, where TGT is the ticket-granting ticket. This includes the ticket for T in the encryption under k_C : dbK C, unlike the KRB_AS_REP message $C, TGT, \{AKey, n_1, T\}_{k_C}$, preventing the intruder from replacing it with any other message before it reaches C. For the same reasons, Kerberos 5 does not have the following property, the translation to our A level formalization of another theorem proved for Kerberos 4 in [2].

Violated Property 2. *For* C : client, T : TGS, Y : msg, $AKey$: shK $C\,T$, n_1 : nonce, k_C : dbK C, *and* k_T : dbK T, *if* $C, Y, \{AKey, n_1, T\}_{k_C}$ *appears on the network and* I *does not have access to* k_C, *then* $Y = \{AKey, C\}_{k_T}$ *and* T *put the message* $C, \{AKey, C\}_{k_T}, \{AKey, n_1, t\}_{k_C}$ *on the network.*

Note that the intruder may do the same thing with the KRB_TGS_REP message (instead of the KRB_AS_REP message as just described), replacing the ticket for S with an arbitrary message and then reversing the switch when C sends a KRB_AP_REQ message to S. This scenario shows that the translation of Theorem 6.23 of [1] fails for Kerberos 5, as does a corresponding theorem in [2].

The amount of practical concern raised by the ticket anomaly seems slight; here the intruder and client together function as the client is intended to [19]. Even with this anomaly, we are still able to prove in Section 9 that the tickets and authenticators originated with the proper principals.

The ticket anomaly can also be realized in the C level formalization. It is not prevented by the checksum sent by C in the KRB_TGS_REQ message, which is taken over the KRB-REQ-BODY part of the message and thus does not cover the ticket (in the C level formalization, this checksum is taken over just the fields $TOpts, C, S, n_2, e$). The anomaly appears to be fixed if the ticket-granting ticket (or what C thinks is this ticket) is also included in the above checksum, although this remains to be proved.

7.2 Anonymous ticket switch anomaly

Another anomaly involving the cutting and pasting of tickets makes use of the anonymous ticket option formalized in our C level formalization. We make no assumptions about the application specific checksums C sends in the KRB_AP_REQ messages other than that they agree with the local policy of the server S. Figure 30 shows the message flow for this anomaly (the 'anonymous ticket switch anomaly').

This scenario begins with a normal AS exchange, after which C has a ticket-granting for use with a TGS, the name T of the TGS, and the corresponding session key $AKey$. C desires two tickets, one NON-ANONYMOUS and the other ANONYMOUS , from T for a single server S and sends the appropriate KRB_TGS_REQ messages to T. T responds with the NON-ANONYMOUS service ticket ST_1 containing key $AKey_1$ and the ANONYMOUS service ticket ST_2 containing the key $AKey_2$ (along with the appropriate other components of these messages). I intercepts both messages, swaps the tickets, and forwards the resulting messages on to C, who then has incorrect beliefs about which (opaque) ticket contains her identity. She then sends S two requests for service without mutual authentication, one using each of these tickets, which the intruder intercepts. I forwards both of these messages to the server S after replacing the authenticator encrypted with $SKey_2$ with the authenticator encrypted with $SKey_1$. The server can open the ticket in each of these messages, but only the key in the NON-ANONYMOUS service ticket ST_1 will open the accompanying authenticator. S thus accepts the NON-ANONYMOUS request and generates an error message (not included here; in the C level formalization, error messages are sent only if the ticket and authenticator match) in response to the malformed

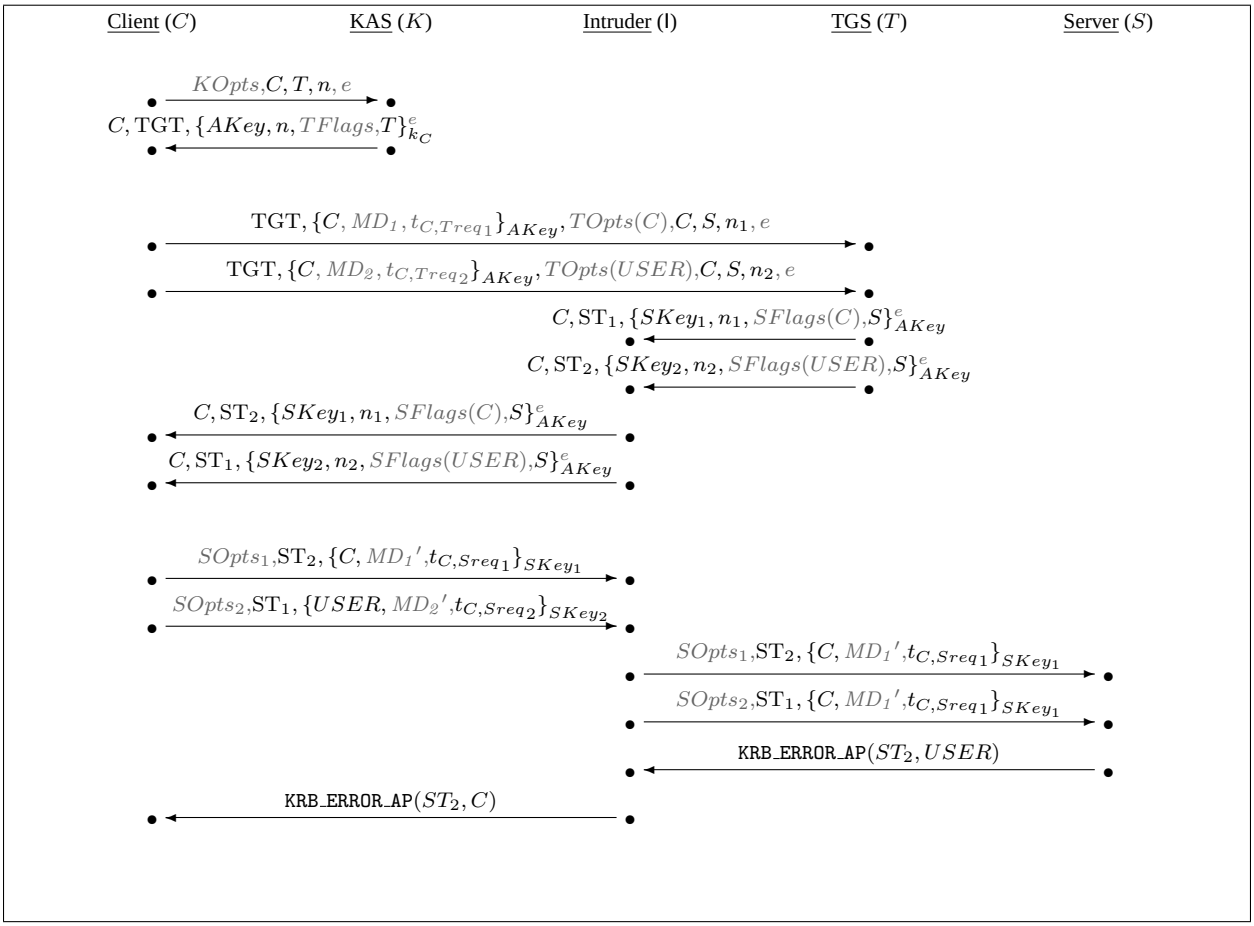

Figure 30: C level message flow in the anonymous ticket switch anomaly.

request. The generic username from the ANONYMOUS ticket is placed in this error message; I may intercept this (unencrypted) message and replace the generic name with C's name, forwarding the result to C. C then processes this message and, seeing her name, may believe that her NON-ANONYMOUS request was rejected and her ANONYMOUS request was accepted. This situation is undesirable since C believes she has completed an ANONYMOUS exchange with S and that she has not completed any exchange in which her identity has been received by S.

This anomaly violates properties proved by Bella and Paulson in [1, 2] for Kerberos 4, which are analogues for the Client/Server Authentication Exchange of Violated Properties 1 and 2 in Section 7.1. This anomaly seems to be avoided if the checksums in the KRB_AP_REQ messages are taken over the service ticket (so that the server S would be aware of any ticket switches), although we have yet to prove that such a change would prevent this anomaly. The checksum in the KRB_AP_REQ message is left as 'application specific' by the current protocol specification [18].

As in the case of the ticket anomaly, it is unclear whether the ticket switch anomaly is of practical concern. It does, however, point out some of the interactions between different parts of the protocol, namely the ANONYMOUS options and the structural change in messages made between versions 4 and 5 of the protocol. Even if something is known to have gone wrong, the client cannot pinpoint when it went wrong; unlike, *e.g.*, compilers, error messages in Kerberos do not precisely identify the first point at which the trace deviated from the expected protocol run. Here the client gets an error message from the server, even though the intruder first interfered in the protocol during the Ticket-Granting Exchange.

7.3 Encryption type anomaly

We assume that C loses her long term (database) key k_C associated with a particular encryption method e. She realizes this, but before reporting the loss of this key (or possibly as she tries to make use of a service in order to do this) she sends a KRB_AS_REQ message to K. C naturally specifies a different encryption method (e' with key k'_C) in order to avoid a response using the lost key. Since this is sent in the clear, I can modify the request to force a response using the compromised key k_C (including the construction of a new checksum using the lost key if necessary). I may then intercept and use the credentials from K's response. Thus I may not only masquerade as C using the lost key, but may also do this based upon any attempt that C makes to work around the known key loss. The message flow for this anomaly is shown in Figure 31.

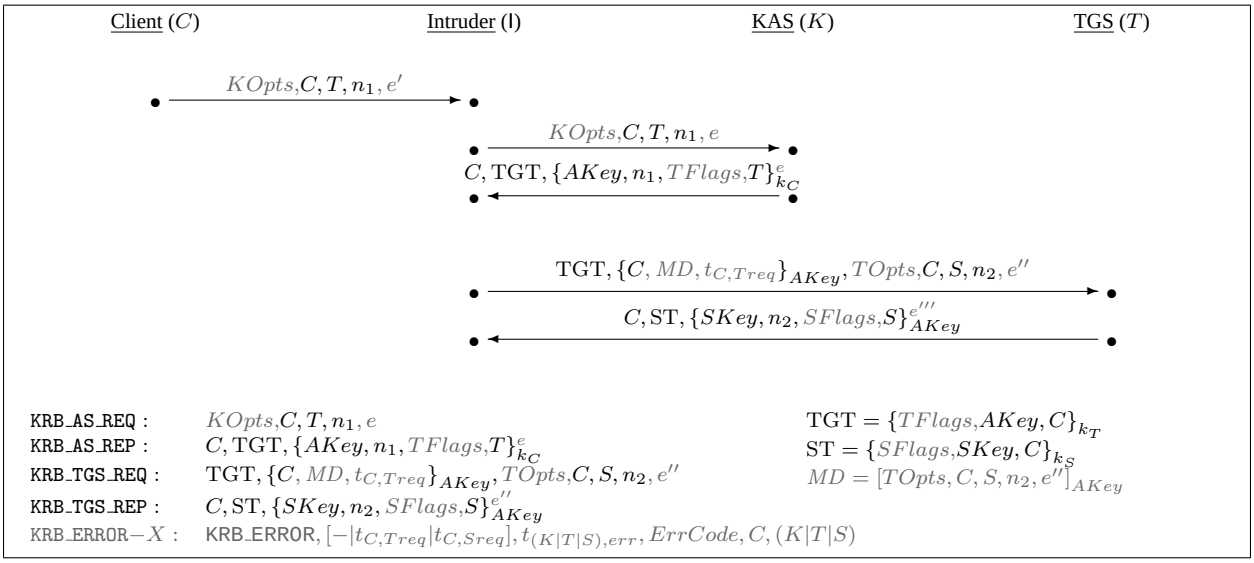

Figure 31: Message flow for encryption type anomaly

Note that this anomaly is not fixed by the checksum that C can send with the KRB_AS_REQ message (which we do not include in our formalizations, but is described in [16] as optional), keyed with a dbK C, as the following scenario shows. C puts $C, T, n, e', [C, T, n, e']^{e'}_{k'_C}$ on the network and I intervenes, replacing it with $C, T, n, e, [C, T, n, e]^e_{k_C}$

(which I can do, since the hash is public and she knows k_C and e). Then the action continues as above, with I gaining knowledge of $AKey$.

A lost long term key is quite serious, as it allows the intruder to obtain and use credentials in the name of the client whose key has been compromised. Raeburn [19] has noted that when this happens the key database must be updated to prevent the lost key from being used. We have not yet formalized the database update mechanism(s); the effect of a compromised key on these is unclear.

7.4 Ticket replay anomaly

We now look at another anomaly whose effects resemble those of the anonymous ticket switch anomaly; the actions of the intruder are different, but again make use of the ability to cut tickets out of messages. The intruder uses a replay to unpack the timestamp encrypted in the authenticator by inducing the server to return it in an (unencrypted) error message. She could also guess this timestamp using rule TA, but we see here that she does not need this rule (which may be unreasonably strong) in order for this anomaly to be realized. Figure 32 shows the message flow for this anomaly, which proceeds as follows.

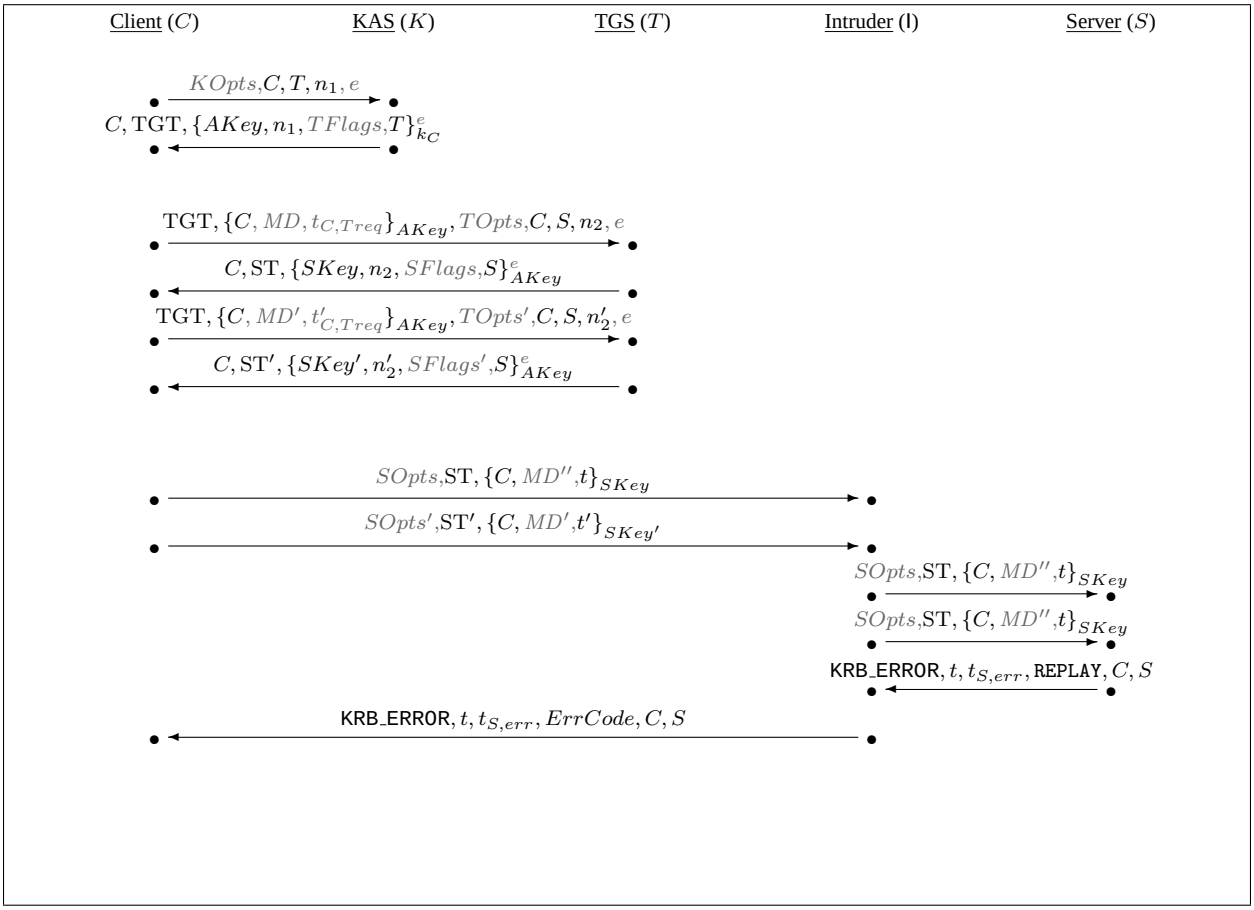

Figure 32: Message flow in the ticket replay anomaly

A client initiates and completes the Authentication Service Exchange with an authentication server, obtaining a ticket granting ticket TGT for a TGS T. She then uses TGT to make two requests for service tickets for a single server, 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 T are different for these two tickets. Recall that

T sends a copy of the granted options along with the new session key (both encrypted under the session key shared by the client and T), 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 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, 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).

As for the ticket switch anomaly, 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. We also note that since our formalizations do not include explicit checks for replayed authenticators, this anomaly may not be realizable in these formalizations.

7.5 Possible replays

The abstraction of the A level formalization, in particular the omission of nonce and timestamp checks, precludes the detection of replayed messages. The KRB_AS_REQ, KRB_TGS_REQ, and KRB_AP_REQ messages may be intercepted by the intruder, copied, and then forwarded to the intended server with the intruder maintaining a copy. The intruder may then, at a later time, replay the copied messages. If the original messages were accepted by the server then the replays may be as well, in which case the servers would generate fresh credentials based upon the replayed requests. These possible replays differ from the ticket replay anomaly in that they would be used to force the creation of fresh credentials.

In order to prevent replayed authenticators, TGSes and servers should save the included timestamps for the length of the allowable clock skew. For the Client-Server Exchange, Version 10 of the protocol revisions (Section 3.2.3 of [16]) makes the following note.

> Unless the application server provides its own suitable means to protect against replay (for example, a challenge-response sequence initiated by the server after authentication, or use of a server-generated encryption subkey), the server must utilize a replay cache to remember any authenticator presented within the allowable clock skew.

Jeffrey [12] has observed that this may place an unreasonable burden on application servers, and that (at least some of) these servers do not in practice make use of a replay cache.

8 Rank and Corank Functions

We now define the two classes of functions—rank and corank—which we use to prove results about our MSR formalizations of the Kerberos 5 protocol. These are inspired by work of Schneider [20] in CSP; related ideas have been discussed in the context of strand spaces [21]. Rank functions are generally used to prove results about data origin authentication, while corank functions are used to prove confidentiality results. Intuitively, the former class captures the amount of work done to produce a certain message, while the latter class captures the amount of work needed to extract a certain (hopefully secret) message. We shall see that because the abilities to encrypt and extract messages are not perfectly symmetric, these classes of functions differ in important ways.

In order to use these types of functions in the MSR formalization of a protocol, we need to define their values on facts. Just as facts are built up from atomic terms in the language of the protocol, we inductively define rank and corank functions starting with their values on atomic terms and then defining the effects on these values of the operations used to build non-atomic terms. The extension of these definitions from terms to facts requires some care in the case of

corank functions; we note some general principles which appear to be applicable to this process and then use these to define this class of functions for our formalizations of Kerberos 5.

8.1 Rank

The k-*rank relative to* m_0 is intended to capture the amount of work done using the key k to encrypt exactly the message m_0. We start with the definition of rank for terms. Let k be a key, t, t_1, t_2 terms, and m_0 a msg. Then we define the k-rank of t relative to m_0, denoted by $\rho_k(t; m_0)$, by

$$\rho_k(t; m_0) = \begin{cases} 0, & t \text{ is an atomic term} \\ \rho_k(m_1; m_0) + 1, & t = \{m_1\}_k,\ \rho_k(m_1; m_0) > 0 \\ 0, & t = \{m_1\}_k,\ \rho_k(m_1; m_0) = 0,\ m_1 \neq m_0 \\ 1, & t = \{m_0\}_k \\ \rho_k(m_1; m_0), & t = \{m_1\}_{k'},\ k' \neq k \\ \rho_k(m_1; m_0) + 1, & t = [m_1]_k,\ \rho_k(m_1; m_0) > 0 \\ 0, & t = [m_1]_k,\ \rho_k(m_1; m_0) = 0,\ m_1 \neq m_0 \\ 1, & t = [m_0]_k \\ \rho_k(m_1; m_0), & t = [m_1]_{k'},\ k' \neq k \\ \max\{\rho_k(t_1; m_0), \rho_k(t_2; m_0)\}, & t = t_1, t_2 \end{cases} . \tag{1}$$

If t is atomic, then no work has been done to encrypt the message m_0 and we set the rank equal to 0. If t is exactly the message $\{m_0\}_k$ we set the rank equal to 1. Encrypting any message of positive k-rank with the key k increases the rank by 1 as additional work has been done using k, while encryption with $k' \neq k$ has no effect on k-rank. Keyed checksums have the same effects, as these also represent cryptographic work done using k. The rank of the concatenation of two messages equals the larger of the ranks of the constituent messages. We will be concerned primarily with whether or not the k-rank relative to m_0 of a message equals 0, *i.e.*, whether or not $\{m_0\}_k$ is contained within the message.

The extension of rank from terms to facts is straightforward; intuitively, the number of nested encryptions of m_0 using k which must have occurred to produce a certain predicate equals the maximum number of such encryptions which were needed to produce one of the arguments of the predicate. Formally, for k a key, m_0 and m of type msg, and t, t_i terms, and P any predicate in the protocol signature, we define the k-rank of a fact F relative to m_0 ($\rho_k(F; m_0)$) by

$$\rho_k(P(t_1, \ldots, t_j); m_0) = \max_{1 \leq i \leq j} \rho_k(t_i; m_0). \tag{2}$$

In particular, we have

$$\rho_k(\mathsf{N}(m); m_0) = \rho_k(m; m_0) \tag{3}$$

$$\rho_k(\mathsf{I}(t); m_0) = \rho_k(t; m_0). \tag{4}$$

For a multiset A of finitely many distinct facts, we define the k-rank of A relative to m_0 by

$$\rho_k(A; m_0) = \max_{F \in A} \rho_k(F; m_0) \tag{5}$$

if $A \neq \emptyset$, and let $\rho_k(\emptyset; m_0) = 0$.

Given a rule

$$F_1, \ldots, F_i \rightarrow \exists x_1 \ldots \exists x_n G_1, \ldots, G_j,$$

we say that this rule increases (preserves, weakly decreases, *etc.*) k-rank relative to m_0 if

$$\rho_k(\{F_1, \ldots, F_i\}; m_0) < \rho_k(\{G_1, \ldots, G_j\}; m_0)$$

($=$, $\geq$, *etc.*). If, in an MSR trace the k-rank of a multiset M_{i+1} is greater (less than) the previous multiset M_i, then the rule used to obtain M_{i+1} must increase (decrease), possibly weakly, relative k-rank; it is clear that the converse does not hold in general.

Any reasonable formulation of the intruder should be such that the intruder cannot do cryptographic work using the key k (as measured by relative k-rank) without possessing the key k. Formally, we expect the intruder rules to satisfy the following property.

Property 1. *If an intruder rule R can increase k-rank relative to m_0, then the left hand side of R contains $\mathsf{I}(k)$.*

As expected, this property is true for our formalizations of the Dolev-Yao intruder. Before proving this, we state the assumption made in Section 4.4.2 about the intruder rule MG as an axiom involving rank functions.

Axiom 1. *If a multiset M_{i+1} is obtained from a multiset M_i by an application of rule MG and $\mathsf{I}(X)$ is the unique fact in $M_{i+1} \setminus M_i$ (i.e., $X : \mathsf{msg}$ is the message freshly generated by the intruder using MG), then for every $k : \mathsf{key}$ and $m_0 : \mathsf{msg}$, $\rho_k(X; m_0) = 0$.*

We may now prove that Property 1 in the formalizations of Kerberos 5 that we have analyzed.

Lemma 1. *Property 1 holds in our A level formalization of Kerberos 5, i.e., for any $k : \mathsf{key}$ and $m_0 : \mathsf{msg}$, any A level intruder rule which increases k-rank relative to m_0 contains the fact $\mathsf{I}(k)$ on its left hand side.*

Proof. Inspection of A level intruder rules shows that of the network, pairing, and encryption rules, only SEC' and DEC' could increase relative k-rank. If either of these rules increases k_0-rank relative to m_0, then the key k mentioned by each of these rules must equal k_0. Among the data generation rules, MG is the only one the relative k-rank of whose right hand side is not obviously equal to 0, but this holds by Axiom 1. Finally, the right hand side of each data access rule is $\mathsf{I}(t)$ for some atomic term t, so none of these can increase relative k-rank. □

Lemma 2. *Property 1 holds in our C level formalization of Kerberos 5, i.e., for any $k : \mathsf{key}$ and $m_0 : \mathsf{msg}$, any C level intruder rule which increases k-rank relative to m_0 contains $\mathsf{I}(k)$ on its left hand side.*

Proof. The addition of encryption types does not change the arguments given in the proof of Lemma 1. Among the rules specific to our C level formalization, EA, OA, and FA create atomic messages (with relative k-rank equal to 0). If an application of MD increases k-rank relative to m_0, then the key used by the must be k; we see that the left hand side of this rule then contains the fact $\mathsf{I}(k)$. □

Our approach to data origin authentication is outlined by the following theorem, which might be viewed as a loose analogue of Schneider's rank function theorem for our rank functions (recall that it is our corank functions which more closely parallel Schneider's rank functions).

Theorem 1. *If $\rho_k(F; m_0) = 0$ for every fact F in the initial state of a trace and no intruder rule can increase k-rank relative to m_0 then the existence of a fact F with $\rho_k(F; m_0) > 0$ in some non-initial state of the trace implies that some honest principal fired a rule which produced a fact built up from $\{m_0\}_k$ or $[m_0]_k$.*

Proof. If no intruder rule can increase k-rank relative to m_0, some honest participant must have fired a rule which increased this rank from 0 to some positive value. A fact of positive k-rank relative to m_0 must contain (as an argument to the predicate) a term of positive k-rank relative to m_0. By induction on the structure of terms, this term must be built up from at least one of the two terms $\{m_0\}_k$ and $[m_0]_k$. □

We then authenticate the origin of $\{m_0\}_k$ (assuming this was not present at the beginning of the trace) by ensuring the confidentiality of k, invoking Property 1, and then determining which honest principal(s) could create $\{m_0\}_k$.

8.2 Corank

The *E-corank relative to m_0* is intended to capture the minimum amount of work, using keys from the set E, needed to obtain the atomic message m_0. As for rank, we start by inductively defining corank on terms and then extending the definition to facts.

Let E be a set of keys, m_0 an atomic term of type msg, and t, t_1, and t_2 terms. Then we define the E-corank of t relative to m_0, denoted by $\hat{\rho}_E(t; m_0)$, as

$$\hat{\rho}_E(t; m_0) = \begin{cases} \infty, & t \text{ is atomic, } t \neq m_0 \\ 0, & t \text{ is atomic, } t = m_0 \\ \hat{\rho}_E(m_1; m_0) + 1, & t = \{m_1\}_k,\ k \in E \\ \hat{\rho}_E(m_1; m_0), & t = \{m_1\}_k,\ k \notin E \\ \infty, & t = [m_1]_k,\ k \text{ any key} \\ \min\{\hat{\rho}_E(t_1; m_0), \hat{\rho}_E(t_2; m_0)\}, & t = t_1, t_2 \end{cases} \tag{6}$$

If t is atomic then no work using keys from E is required to obtain m_0 if $t = m_0$, while no amount of such work can extract m_0 from $t \neq m_0$. The number of decryptions using keys from E needed to obtain m_0 from $\{m\}_k$ is the same as or 1 more than the number needed to obtain m_0 from m, depending on whether $k \notin E$ or $k \in E$. Since we assume that message digestion is one-way, no amount of decryption can extract m_0 from $[m]_k$, regardless of whether or not $k \in E$; this appears in gray since message digests appear in the signature of our C level protocol formalization but not our A level formalization. A message m_0 can be extracted from the concatenation of two terms by extracting it from one of these two terms (since we are assuming that m_0 is atomic), whence the final case.

The extension of the definition of corank from terms to facts requires more care than the parallel extension of rank. For a memory predicate P with j arguments, a natural first definition of the E-corank of $P(t_1, \ldots, t_j)$ relative to m_0 would be $\min_{1 \leq i \leq j} \hat{\rho}_E(t_i; m_0)$. However, we wish to have principals store messages in predicates without necessarily compromising the confidentiality of these messages (*e.g.*, an honest principal storing an unencrypted session key in memory does not correspond to the intruder knowing this key). If a certain argument to a predicate P will never be placed on the network, we will ignore the term it contains when determining the E-corank of P. We thus modify the initial definition given above to instead take the minimum to be over those i for which t_i might be placed on the network (to state this imprecisely). We leave a general approach to this problem for future work; for the moment, we use this intuition to guide our extension of corank to facts as follows.

Let E be a set of keys, m_0 an atomic term of type msg, m of type msg, and t, t_i be terms. Then, for L any role state predicate and considering all predicates which appear in our formalizations of Kerberos 5 (with gray type indicating things present only in our C level formalization), we may define the E-corank of a fact F relative to m_0 as follows.

$$\begin{aligned}
\hat{\rho}_E(\mathsf{N}(m); m_0) &= \hat{\rho}_E(m; m_0) \\
\hat{\rho}_E(\mathsf{I}(t); m_0) &= \hat{\rho}_E(t; m_0) \\
\hat{\rho}_E(Auth_C(m_1, m_2, m_3, m_4); m_0) &= \hat{\rho}_E(m_1; m_0) \\
\hat{\rho}_E(Service_C(m_1, m_2, m_3, m_4); m_0) &= \hat{\rho}_E(m_1; m_0) \\
\hat{\rho}_E(L(m_1, \ldots, m_j); m_0) &= \infty \\
\hat{\rho}_E(DoneMut_C(m_1, m_2); m_0) &= \infty \\
\hat{\rho}_E(DoneNoMut_C(m_1, m_2); m_0) &= \infty \\
\hat{\rho}_E(Mem_S(m_1, m_2, m_3); m_0) &= \infty \\
\hat{\rho}_E(ASError_C(m_1, m_2, m_3, m_4); m_0) &= \infty \\
\hat{\rho}_E(TGSError_C(m_1, m_2, m_3); m_0) &= \infty \\
\hat{\rho}_E(APError_C(m_1, m_2, m_3); m_0) &= \infty
\end{aligned}$$

For a multiset A of facts, we define the E-corank of A relative to m_0 by

$$\hat{\rho}_E(A; m_0) = \min_{F \in A} \hat{\rho}_E(F; m_0) \tag{7}$$

if $A \neq \emptyset$, and let $\hat{\rho}_E(\emptyset; m_0) = \infty$.

We identify the confidentiality of m_0 with the fact $\mathsf{I}(m_0)$ being prohibited from appearing in a trace. As an immediate consequence of the definition of the corank of facts, we see that corank relative to m_0 is directly connected to the confidentiality of m_0 as follows.

Lemma 3. *Let m_0 : msg be atomic. If there is any set E of keys such that no fact F with $\hat{\rho}_E(F; m_0) = 0$ appears in a trace, then that trace does not contain $\mathsf{I}(m_0)$.*

Proof. For every set E of keys, $\hat{\rho}_E(\mathsf{I}(m_0); m_0) = \hat{\rho}_E(m_0; m_0) = 0$. □

We expect that in any reasonable intruder formulation, if an intruder decreases the amount of decryption with keys in the set E needed to learn a message, then she either knows some key protecting that message or she creates that message herself. We formalize this as the following property.

Property 2. *If an intruder rule R can decrease E-corank relative to m_0, where I does not have access to m_0 simply by virtue of the type of m_0, then the left hand side of R contains $\mathsf{I}(k)$ for some $k \in E$ or R freshly generates m_0.*

As expected, this holds for both of our formalizations of the Dolev-Yao intruder.

Lemma 4. *Property 2 holds for our A level formalization, i.e., if m_0 is not a* principal *name,* time*, or key of one of the types* dbK I, shK I A *for A : TS, or* shK C I *for C :* client*, then any A level intruder rule which decreases E-corank relative to m_0 either contains $\mathsf{I}(k)$ in its left hand side for some $k \in E$ or freshly generates m_0.*

Proof. The only network, pairing, and encryption rules which can decrease relative corank are SDC' and DDC'; if one of these rules does indeed decrease E-corank, then the key k mentioned in each rule must belong to the set E. If any data generation rule decreases E-corank relative to m_0, by inspection we see that its right hand side must freshly generate m_0. The right hand side of each data access rule is $\mathsf{I}(t)$ for a term t whose type is assumed not to be the type of m_0, so the lemma is trivially true for these rules. □

Lemma 5. *Property 2 holds for our C level formalization, i.e., if m_0 is not a* principal *name,* time, etype, Flag, Opt*, or key of one of the types* dbK I, shK I A *for A : TS, or* shK C I *for C :* client*, then any C level intruder rule which decreases E-corank relative to m_0 either contains $\mathsf{I}(k)$ in its left hand side for some $k \in E$ or freshly generates m_0.*

Proof. The addition of encryption types does not change any of the arguments used to prove the A level version (Lemma 4). The new rule MD cannot decrease relative corank (the right hand side is $\mathsf{I}(m)$ for non-atomic m). The new data access rules are covered by the data types listed in the statement of the lemma. □

We prove confidentiality using the following result; like Theorem 1, this may be viewed as some type of analogue of Schneider's rank function theorem.

Theorem 2. *If $\hat{\rho}_E(F; m_0) > 0$ for every fact in the initial state of a trace, no intruder rule can decrease E-corank relative to m_0, and no honest principal creates a fact F with $\hat{\rho}_E(F; m_0) = 0$, then m_0 is secret throughout the trace.*

Proof. We identify the secrecy of m_0 throughout a trace with the MSR fact $\mathsf{I}(m_0)$ never appearing in trace. Because $\hat{\rho}_E(\mathsf{I}(m_0); m_0) = 0$ for every set E of keys, if the conditions of the theorem are satisfied, m_0 is secret throughout the trace in question. □

We may thus show that m_0 is confidential by finding some set E of keys, each of which is confidential (which may require additional corank arguments) and which satisfies the conditions of this theorem.

9 Properties of Kerberos 5

In our work to date, we have established two types of properties for Kerberos 5. Since Kerberos is intended to provide authentication, it is important to see what sort of authentication properties the protocol has. In proving authentication properties of the protocol, we have also established confidentiality properties for various session keys which are established during a protocol run. These properties are important in their own right, since some of the session keys may be used in future communications between protocol participants.

We have established confidentiality and authentication properties connected to both the Ticket Granting Exchange and the Client/Server Exchange. Since these exchanges have similar structure, it is not surprising that the properties are expressed and indeed proved in very similar ways. Table 1 shows the parallel relationships between the properties that we have established thus far. The confidentiality properties discussed here state that an intruder never learns certain

	Confidentiality	Authentication
TG Exchange	Property 3	Property 4
C/S Exchange	Property 5	Property 6

Table 1: Properties established for Kerberos 5

information. The authentication properties that we have established are *data origin authentication* properties [11]. These show that if certain messages are ever seen on the network, then they must have been originally sent by a specified protocol participant. 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 [1, 2, 3]

The properties related to the Ticket Granting Exchange have been established in both our A and C level formalizations. Properties 3 and 4 for our A level formalization were included, albeit in somewhat different form, in [6]. Their extension to our C level formalization is a new result here. Properties 5 and 6, for the Client/Server Exchange, have so far been proved only for the A level formalization; these results are also new since [6].

The precise statements and proofs of these properties are related in much the same way that the formalizations themselves are related—removing some information from the detailed version gives the more abstract version. As a result, we expect that we will soon be able to extend the properties of the Client/Server Exchange to our C level formalization. In this section we outline the proofs of the theorems stated using gray text to indicate those parts of the outline which are specific to the C level version of the property. The full proofs, which involve numerous minor lemmas about individual MSR rules, are given in Appendix A

9.1 The Ticket-Granting Exchange

We start with the properties that we have established for the Ticket Granting Exchange. As this exchange is closer to the beginning of the standard protocol run, these properties are slightly simpler than for the Client/Server Exchange below.

Because the communications between the client and TGS use the shared key generated by the KAS which created the ticket granting ticket, we want to ensure that this key remains confidential. In this exchange, the ticket granting server produces credentials (a service ticket) in response to a request which contains a ticket granting ticket and an authenticator. We thus also wish to authenticate the origin of these objects; in the case of the authenticator, which is encrypted using the key shared between C and T, we make use of the confidentiality result for this exchange.

9.1.1 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. This parallels Theorem 6.18 of [1] for Kerberos 4.

Property 3. *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 $AKey$.*

We formalize this property for our A and C level formalizations as the following two theorems.

Theorem 3. *For C : client, T : TGS, $C, T \neq \mathsf{I}$, k_C : dbKC, k_T : dbKT, $AKey$: shKCT, and n : nonce, if the initial state of a finite trace does not contain $\mathsf{I}(k_C)$ or $\mathsf{I}(k_T)$ and some K : KAS fires rule $\alpha_{2.1}$, freshly generating $AKey$ and creating the fact $\mathsf{N}(C, \{AKey, C\}_{k_T}, \{AKey, n, T\}_{k_C})$, then no state of the trace contains the fact $\mathsf{I}(AKey)$.*

Theorem 4. *For C : client, T : TGS, $C, T \neq \mathsf{I}$, k_C : dbK C, k_T : dbK T, $AKey$: shK $C\,T$, $TFlags$: TFlag, and n : nonce, if the initial state of a finite trace does not contain $\mathsf{I}(k_C)$ or $\mathsf{I}(k_T)$ and some K : KAS fires rule $\gamma_{2.1}$, freshly generating $AKey$ and creating the fact $\mathsf{N}(C, \{TFlags, AKey, C\}_{k_T}, \{AKey, n, TFlags, T\}_{k_C})$, then no state of the trace contains the fact $\mathsf{I}(AKey)$.*

Proof. (Sketch) We show that no fact whose $\{k_C, k_T\}$-corank equals 0 relative to $AKey$ ever appears in the trace. (We use this set of keys because one of them encrypts $AKey$ whenever it is transmitted over the network.) K decreases this corank when it freshly generates $AKey$, but not below 1; no KAS may otherwise decrease this corank. No client, TGS, or server can ever decrease this corank, nor can the intruder. As this relative corank in question must have been infinite for every fact in the trace before K freshly generated $AKey$, no fact whose $\{k_C, k_T\}$-corank equals 0 relative to $AKey$ can every appear in the trace. □

9.1.2 Authentication of 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 4. *If the intruder does not know the long term key used to encrypt a ticket-granting ticket and this ticket did not exist at the beginning of the trace, then if the* TGS *processes a request, ostensibly from a client C, containing the ticket-granting ticket and the session key $AKey$, then some Authentication Server created the session key $AKey$ for C to use with the* TGS *and also generated this ticket-granting ticket. Furthermore, if the intruder does not know the long term key that the authentication server used to send $AKey$ to C, then the authenticator was created by C.*

We formalize this property for our abstract and detailed formalizations as the following two theorems.

Theorem 5. *For C : client, T : TGS, $C, T \neq$ I, S : server, $AKey$: shK $C\ T$, k_T : dbK T, and n : nonce, if the beginning state of a finite trace does not contain $\mathsf{I}(k_T)$ or any fact F with $\rho_{k_T}(F; AKey, C) > 0$, and at some point in the trace T fires rule $\alpha_{4.1}$, consuming the fact $\mathsf{N}(\{AKey, C\}_{k_T}, \{C\}_{AKey}, C, S, n)$, then some K : KAS previously fired rule $\alpha_{2.1}$, freshly generating $AKey$ and producing the fact $\mathsf{N}(C, \{AKey, C\}_{k_T}, \{AKey, n', T\}_{k'})$ for some n' : nonce, and k' : dbK C. Furthermore, if $\mathsf{I}(k')$ did not appear in the initial state of the trace, then after K fired rule $\alpha_{2.1}$ and before T fired rule $\alpha_{4.1}$, C fired rule $\alpha_{3.1}$, creating the fact $\mathsf{N}(X, \{C\}_{AKey}, C, S', n'')$ for some X : msg, S' : server, and n'' : nonce.*

Theorem 6. *For C : client, T : TGS, $C, T \neq$ I, S : server, $AKey$: shK $C\,T$, k_T : dbK T, $TFlags$: TFlag, ck : msg, $t_{C,Treq}$: time, $TOpts$: TOpt, e : etype, and n : nonce, if the beginning state of a finite trace does not contain $\mathsf{I}(k_T)$ or any fact F with $\rho_{k_T}(F; TFlags, AKey, C) > 0$, and at some point in the trace T fires rule $\gamma_{4.1}$, consuming the fact $\mathsf{N}(\{TFlags, AKey, C\}_{k_T}, \{C, ck, t_{C,Treq}\}_{AKey}, TOpts, C, S, n, e)$, then some K : KAS previously fired rule $\gamma_{2.1}$, freshly generating $AKey$ and producing the fact $\mathsf{N}(C, \{TFlags, AKey, C\}_{k_T}, \{AKey, n', TFlags, T\}^{e'}_{k'})$ for some n' : nonce, e' : etype, and k' : dbK$^{e'}$ C. Furthermore, if $\mathsf{I}(k')$ did not appear in the initial state of the trace, then after K fired rule $\gamma_{2.1}$ and before T fired rule $\gamma_{4.1}$, C fired rule $\gamma_{3.1}$, creating the fact $\mathsf{N}(X, \{C, [TOpts', C, S', n'', e'']_{AKey}, t_{C,Treq}\}_{AKey}, TOpts', C, S', n'', e'')$ for some X : msg, $TOpts'$: TOpt, e'' : etype, S' : server, and n'' : nonce.*

Proof. (Sketch) We first consider k_T-rank relative to $TFlags, AKey, C$. No client, server, or TGS can increase this rank, and I cannot increase it without knowing k_T. Some K : KAS must have increased this rank; we see that rule $\alpha\gamma_{2.1}$ was fired by K and the other claims of the first part of the theorem follow.

The assumption that $\mathsf{I}(k')$ is not in the initial state of the trace allows us to apply Property 6, which shows that I does not learn $AKey$. Thus I cannot increase $AKey$-rank relative to $C, [TOpts', C, S', n'', e'']_{AKey}, t_{C,Treq}$. No KAS, TGS, or server will do so, and C is the only client who will; inspection of the client rules shows that she must do so in the manner claimed. □

9.2 The Client/Server Exchange

We now move to properties of the Client/Server Exchange; as this exchange parallels the Ticket Granting Exchange, its 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.

9.2.1 Confidentiality of $SKey$

The first property for the Client/Server Exchange gives conditions under which the session key shared by the client and server is not known to the intruder. This parallels Theorem 6.19 of [1] for Kerberos 4.

Property 5. *If the intruder knows neither the long term secret key used by a* TGS *to encrypt the service ticket 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 $SKey$.*

We formalize this property for our abstract formalization as the following theorem. We have not yet proved Property 5 for our detailed formalization, but expect to do so soon.

Theorem 7. *For C : client, T : TGS, S : server, k_T : dbK T, k_S : dbK S, $SKey$: shK $C\,S$, $AKey$: shK $C\,T$, and n : nonce, if T fires rule $\alpha_{4.1}$, consuming the fact $\mathsf{N}(\{AKey, C\}_{k_T}, \{C\}_{AKey}, C, S, n)$, freshly generating $SKey$, and creating the fact $\mathsf{N}(C, \{SKey, C\}_{k_S}, \{SKey, n, S\}_{AKey})$, and if the initial state of the trace does not contain $\mathsf{I}(k_S)$ and no state of the trace contains $\mathsf{I}(AKey)$, then no state of the trace contains $\mathsf{I}(SKey)$.*

*Proof. (Sketch)*We show that no fact with $\{AKey, k_S\}$-corank relative to $SKey$ equal to 0 appears in the trace. The only way that a TGS can decrease this corank is through T's rule firing as in the theorem statement; the resulting multiset has $\{AKey, k_S\}$-corank relative to $SKey$ equal to 1, and this corank was infinite for every previous state in the trace. No KAS, client, or server decreases this corank. The intruder cannot freshly generate $SKey$ or decrease this corank through other means, finishing the proof. □

We may explicitly give conditions guaranteeing that $\mathsf{I}(AKey)$ doe not appear in the trace in order to obtain the following corollary.

Corollary 8. *For C : client, T : TGS, S : server, k_T : dbK T, k_S : dbK S, $SKey$: shK $C\,S$, $AKey$: shK $C\,T$, and n : nonce, if T fires rule $\alpha_{4.1}$, consuming the fact $\mathsf{N}(\{AKey, C\}_{k_T}, \{C\}_{AKey}, C, S, n)$, freshly generating $SKey$, and creating the fact $\mathsf{N}(C, \{SKey, C\}_{k_S}, \{SKey, n, S\}_{AKey})$, and if the initial state of the trace did not contain $\mathsf{I}(k_T)$, $\mathsf{I}(k_S)$, $\mathsf{I}(k_C)$ for every k_C : dbK C, or any fact F with $\rho_{k_T}(F; AKey, C)$, then no state of the trace contains $\mathsf{I}(SKey)$.*

9.2.2 Authentication of ST and authenticator

The second property for the Client/Server Exchange is our main result for this exchange and captures authentication of the client C to the server S, again in the form of data origin authentication. It states conditions which guarantee that if S receives a certain message (consisting of a service ticket and an authenticator), apparently sent by C, then the service ticket originated with some T : TGS and the authenticator originated with C. The assumptions needed for the theorem to hold are that the ticket did not already exist at the beginning of the trace, and that the intruder I does not have access to the long term key of the server S or the key shared between C and the TGS T who generated the ticket.

Property 6. *If the intruder does not know the long term key used to encrypt a service ticket for a* client *C to present to a* server *S and this ticket did not exist at the beginning of the trace, then if S processes a request, ostensibly from C, containing this service ticket and the session key $SKey$, then some Ticket Granting Server generated the session key $SKey$ for C to use with S and also created the service ticket. Furthermore, if the intruder never learns the session key which the Ticket Granting Server used to encrypt $SKey$ when sending the service ticket to C, then C created the authenticator.*

We formalize this property for our abstract formalization as the following theorem. We have not yet proved Property 6 for our detailed formalization, but expect to do so soon.

Theorem 9. *For C : client, S : server, k_S : dbK S, $SKey$: shK $C\,S$, and $t_{C,Sreq}$: time, if the beginning state of a finite trace does not contain $\mathsf{I}(k_S)$ or any fact F with $\rho_{k_S}(F; SKey, C) > 0$, and at some point in the trace S fires rule $\alpha_{6.1}$ consuming the fact $\mathsf{N}(\{SKey, C\}_{k_S}, \{C, t_{C,Sreq}\}_{SKey})$, then some T : TGS previously fired rule $\alpha_{4.1}$, freshly generating $SKey$ and producing the fact $\mathsf{N}(C, \{SKey, C\}_{k_S}, \{SKey, n, S\}_k)$ for some n : nonce and k : shK $C\,T$. Furthermore, if the fact $\mathsf{I}(k)$ has not yet appeared in the trace, then after T fired rule $\alpha_{4.1}$ and before S fired the rule $\alpha_{6.1}$, C fired rule $\alpha_{5.1}$ to create the fact $\mathsf{N}(Y, \{C, t_{C,Sreq}\}_{SKey})$ for some Y : msg.*

Proof. (Sketch) We first consider k_S-rank relative to $SKey, C$; this was 0 for all facts in the initial state of the trace and S's rule firing consumes a fact of positive k_S-rank relative to $SKey, C$. No client, KAS, or server can increase this rank, nor can the intruder. The only TGS that could do so is T and in the manner claimed.

S's rule firing also consumes a fact of positive $SKey$-rank relative to $C, t_{C,Sreq}$; this rank must have been increased during the protocol trace because $SKey$ was freshly generated during the trace. We may invoke Property 3 to show that I could not increase this rank; by inspection, we see that no KAS, TGS, or server could either. The only client who could do so was C, and she must have done so in the manner claimed. □

We may explicitly add hypotheses which will guarantee that $\mathsf{I}(k)$ does not appear in the trace; this gives us the following corollary.

Corollary 10. *For C : client, S : server, k_S : dbK S, $SKey$: shK C S, and $t_{C,Sreq}$: time, if the beginning state of a finite trace does not contain $\mathsf{I}(k_S)$ or any fact F with $\rho_{k_S}(F; SKey, C) > 0$, and at some point in the trace S fires rule $\alpha_{6.1}$ consuming the fact $\mathsf{N}(\{SKey, C\}_{k_S}, \{C, t_{C,Sreq}\}_{SKey})$, then some T : TGS previously fired rule $\alpha_{4.1}$, freshly generating $SKey$ and producing the fact $\mathsf{N}(C, \{SKey, C\}_{k_S}, \{SKey, n, S\}_k)$ for some n : nonce and k : shK C T. Furthermore, if the initial state of the trace did not contain $\mathsf{I}(k_C)$ for any k_C : dbK C, or, for any k_T : dbK T, and if $T \neq \mathsf{I}$, $\mathsf{I}(k_T)$ or any fact F with $\rho_{k_T}(F; k, C) > 0$, then after T fired rule $\alpha_{4.1}$ and before S fired the rule $\alpha_{6.1}$, C fired rule $\alpha_{5.1}$ to create the fact $\mathsf{N}(Y, \{C, t_{C,Sreq}\}_{SKey})$ for some Y : msg.*

Part IV
Conclusions and References

10 Conclusions and Future Work

10.1 Conclusions

In this paper, we gave three formalizations of Kerberos 5 in the Multi-Set Rewriting (MSR) framework. The A level formalization included just enough detail to prove authentication and confidentiality results for the protocol; due to structural changes in the messages from Kerberos 4, these properties were slightly weaker than those proved for that version of the protocol[1, 2, 3, 4]. The C level formalization was closer to the full protocol as given in [13, 16], adding error messages, checksums, and a number of options to the A level formalization. Many of these details are new to version 5 of Kerberos. We extended our analysis of the A level case to the C level, observing that the structure of the proofs is preserved in doing this and again proving authentication and confidentiality properties of the protocol. The B level formalization extended the A level in a different direction by adding timestamps and temporal checks. We did not extensively analyze this formalization as these details are not significantly changed from Kerberos 4.

We noted four possible instances of curious protocol behavior, although none of these compromises the security of the protocol. Three of these arose because tickets are not bound to the rest of the messages containing them (as they were in Kerberos 4); one of these three was seen in both the A and C levels, while the other two made use of the options in the C level formalization. The fourth anomaly was related to the encryption type option which was included in the C level. It appeared that some of these anomalies may be prevented though the use of cryptographic checksums beyond those specified in the protocol, but we have not yet formally proved this. We did not notice any new anomalies in our informal analysis of the B level formalization.

The proofs of protocol properties made use of rank and corank functions, inspired by the work of Schneider [20]. Our analysis gave insight into approaches to reasoning about the MSR specifications of protocols. Throughout this work, MSR proved to be an adequate language for formalizing and analyzing a real-world protocol.

10.2 Future work

We close with an outline of logical extensions of the work described in this paper.

Kerberos 5 is a complex protocol suite, with numerous details remaining to be formalized and analyzed. One natural continuation of this work is the formalization and analysis of the common refinement of our B and C level formalizations; this might be further extended to include even more timestamps and temporal checks, explicit consideration of all options specified in [18] (in particular renewable and postdatable tickets), and the formalization of the other available subprotocols (such as for client-server communication after authentication has been achieved).

We have seen parallels between the analyses of the A and C level formalizations. The relationships between different formalizations of Kerberos 5 and the corresponding relationships between their security properties (and the proofs of these properties) should be investigated in a precise manner. Analysis including timestamps should be done, either using the B level formalization or some refinement of it; additional work may be merited on anomalous behavior in more detailed formalizations, how it might be prevented (including through modification of existing checksums in the protocol), and whether such preventative measures would be worth implementing.

Acknowledgments

We are grateful to Alan Jeffrey, John Mitchell, Clifford Neuman, and Ken Raeburn for a number of helpful comments on the shorter version of this work.

References

[1] G. Bella, *Inductive Verification of Cryptographic Protocols*, Ph.D. thesis, University of Cambridge, March 2000, <http://www.cl.cam.ac.uk/~gb221/papers/bella14.ps.gz>.

[2] G. Bella and L. C. Paulson, *Using Isabelle to Prove Properties of the Kerberos Authentication System*, Proc. of DIMACS'97, Workshop on Design and Formal Verification of Security Protocols (CD-ROM) (H. Orman and C. Meadows, eds.), 1997, <http://www.cl.cam.ac.uk/~gb221/papers/bella4.ps.gz>.

[3] _____, *Kerberos Version IV: Inductive Analysis of the Secrecy Goals*, Proc. of ESORICS '98, Fifth European Symposium on Research in Computer Science, Lecture Notes in Computer Science, no. 1485, Springer-Verlag, 1998, <http://www.cl.cam.ac.uk/~gb221/papers/bella6.ps.gz>, pp. 361–375.

[4] _____, *Mechanising BAN Kerberos by the Inductive Method*, Proc. of CAV98 – Tenth International Conference on Computer Aided Verification, 1998, <http://www.cl.cam.ac.uk/~gb221/papers/bella5.ps.gz>.

[5] G. Bella and E. Riccobene, *Formal Analysis of the Kerberos Authentication System*, J. Universal Comp. Sci. **3** (1997), no. 12, 1337–1381.

[6] F. Butler, I. Cervesato, A. D. Jaggard, and A. Scedrov, *An Analysis of Some Properties of Kerberos 5 Using MSR*, Proceedings of the 15th Computer Security Foundations Workshop, 2002.

[7] I. Cervesato, *The Logical Meeting Point of Multiset Rewriting and Process Algebra*, Unpublished manuscript. Available at <http://theory.stanford.edu/~iliano/forthcoming>.

[8] _____, *Typed MSR: Syntax and Examples*, 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.

[9] I. Cervesato, N. A. Durgin, P.D. Lincoln, J.C. Mitchell, and A. Scedrov, *A Meta-notation for Protocol Analysis*, Proc. of the Twelfth IEEE Computer Security Foundations Workshop, 1999, pp. 55–69.

[10] N. A. Durgin, P. D. Lincoln, J. C. Mitchell, and A. Scedrov, *Multiset Rewriting and the Complexity of Bounded Security Protocols*, Journal of Computer Security, To appear. Preliminary version available at <ftp://ftp.cis.upenn.edu/pub/papers/scedrov/msr+long.[pdf,ps,ps.gz]>.

[11] D. Gollmann, *Authentication–Myths and Misconceptions*, Progress in Computer Science and Applied Logic **20** (2001), 203–225.

[12] A. Jeffrey, Personal communication.

[13] J. Kohl and C. Neuman, *The Kerberos Network Authentication Service (V5)*, September 1993, Network Working Group Request for Comments: 1510. <ftp://ftp.isi.edu/in-notes/rfc1510.txt>.

[14] J. C. Mitchell, M. Mitchell, and U. Stern, *Automated Analysis of Cryptographic Protocols Using Murφ*, Proc. of the IEEE Symposium on Security and Privacy, IEEE Computer Society Press, 1997, pp. 141–153.

[15] C. Neuman, June 2002, Personal communication.

[16] C. Neuman, J. Kohl, T. Ts'o, Ken Raeburn, and Tom Yu, *The Kerberos Network Authentication Service (V5)*, November 20 2001, Internet draft, expires 20 May 2002. `<http://www.ietf.org/internet-drafts/draft-ietf-cat-kerberos-revisions-10.txt>`.

[17] C. Neuman and T. Ts'o, *Kerberos: An Authentication Service for Computer Networks*, IEEE Communications **32** (1994), no. 9, 33–38.

[18] C. Neuman, T. Yu, S. Hartman, and K. Raeburn, *The Kerberos Network Authentication Service (V5)*, February 15 2004, Internet draft, expires 15 August 2004. `<http://www.ietf.org/internet-drafts/draft-ietf-krb-wg-kerberos-clarifications-05.txt>`.

[19] K. Raeburn, June 2002, Personal communication.

[20] S. Schneider, *Verifying Authentication Protocols in CSP*, IEEE Transactions on Software Engineering **24** (1998), no. 9, 741–758.

[21] F. J. Thayer Fábrega, J. Herzog, and J. D. Guttman, *Honest ideals on strand spaces*, Proceedings, 1998 Computer Security Foundations Workshop, 1998.

Part V
Appendices

A Proofs of Protocol Properties

The structure of this appendix parallels the structure of Section 9, where the theorems proved here are originally stated.

A.1 The Ticket-Granting Exchange

A.1.1 Confidentiality of $AKey$

Theorem 3. *For $C : \mathsf{client}$, $T : \mathsf{TGS}$, $C, T \neq \mathsf{I}$, $k_C : \mathsf{dbK}\,C$, $k_T : \mathsf{dbK}\,T$, $AKey : \mathsf{shK}\,CT$, and $n : \mathsf{nonce}$, if the initial state of a finite trace does not contain $\mathsf{I}(k_C)$ or $\mathsf{I}(k_T)$ and some $K : \mathsf{KAS}$ fires rule $\alpha_{2.1}$, freshly generating $AKey$ and creating the fact $\mathsf{N}(C, \{AKey, C\}_{k_T}, \{AKey, n, T\}_{k_C})$, then no state of the trace contains the fact $\mathsf{I}(AKey)$.*

Proof. We claim that no fact with $\{k_C, k_T\}$-corank relative to $AKey$ equal to 0 appears in the trace.

By Lemma 8, if any KAS fires a rule which decreases $\{k_C, k_T\}$-corank relative to $AKey$, then that rule freshly generates $AKey$ and, if the newly created fact in the resulting multiset is $\mathsf{N}(C, \{AKey, C\}_{k_T}, \{AKey, n, T\}_{k_C})$, the $\{k_C, k_T\}$-corank relative to $AKey$ of this multiset equals 1. By Lemma 9, no previous multiset in the trace contained a fact with finite $\{k_C, k_T\}$-corank relative to $AKey$, nor can any KAS later fire a rule which decreases $\{k_C, k_T\}$-corank relative to $AKey$.

By Lemmas 10, 11, and 12, no client, TGS, or server decreases $\{k_C, k_T\}$-corank relative to $AKey$.

By hypothesis and Lemma 6, the facts $\mathsf{I}(k_C)$ and $\mathsf{I}(k_T)$ never appear in the trace under consideration. By hypothesis, K freshly generates $AKey$, so by Lemma 9 I cannot freshly generate $AKey$. Thus, by Lemma 13, I does not fire any rule which decreases $\{k_C, k_T\}$-corank relative to $AKey$.

As a result, no fact of $\{k_C, k_T\}$-corank 0 relative to $AKey$, in particular $\mathsf{I}(AKey)$, occurs in any multiset of the trace. □

Theorem 4. *For C : client, T : TGS, $C, T \neq \mathsf{I}$, k_C : dbK C, k_T : dbK T, $AKey$: shK C T, $TFlags$: TFlag, and n : nonce, if the initial state of a finite trace does not contain $\mathsf{I}(k_C)$ or $\mathsf{I}(k_T)$ and some K : KAS fires rule $\gamma_{2.1}$, freshly generating $AKey$ and creating the fact $\mathsf{N}(C, \{TFlags, AKey, C\}_{k_T}, \{AKey, n, TFlags, T\}_{k_C})$, then no state of the trace contains the fact $\mathsf{I}(AKey)$.*

Proof. We claim that no fact with $\{k_C, k_T\}$-corank relative to $AKey$ equal to 0 appears in the trace.

By Lemma 14, if any KAS fires a rule which decreases $\{k_C, k_T\}$-corank relative to $AKey$, then that rule freshly generates $AKey$ and, if the newly created fact in the resulting multiset is $\mathsf{N}(C, \{AKey, C\}_{k_T}, \{AKey, n, T\}_{k_C})$, the $\{k_C, k_T\}$-corank relative to $AKey$ of this multiset equals 1. By Lemma 15, no previous multiset in the trace contained a fact with finite $\{k_C, k_T\}$-corank relative to $AKey$, nor can any KAS later fire a rule which decreases $\{k_C, k_T\}$-corank relative to $AKey$.

By Lemmas 16, 17, and 18, no client, TGS, or server decreases $\{k_C, k_T\}$-corank relative to $AKey$.

By hypothesis and Lemma 7, the facts $\mathsf{I}(k_C)$ and $\mathsf{I}(k_T)$ never appear in the trace under consideration. By hypothesis, K freshly generates $AKey$, so by Lemma 15 I cannot freshly generate $AKey$. Thus, by Lemma 19, I does not fire any rule which decreases $\{k_C, k_T\}$-corank relative to $AKey$.

As a result, no fact of $\{k_C, k_T\}$-corank 0 relative to $AKey$, in particular $\mathsf{I}(AKey)$, occurs in any multiset of the trace. □

A.1.2 Authentication of TGT and authenticator

Theorem 5. *For C : client, T : TGS, $C, T \neq \mathsf{I}$, S : server, $AKey$: shK C T, k_T : dbK T, and n : nonce, if the beginning state of a finite trace does not contain $\mathsf{I}(k_T)$ or any fact F with $\rho_{k_T}(F; AKey, C) > 0$, and at some point in the trace T fires rule $\alpha_{4.1}$, consuming the fact $\mathsf{N}(\{AKey, C\}_{k_T}, \{C\}_{AKey}, C, S, n)$, then some K : KAS previously fired rule $\alpha_{2.1}$, freshly generating $AKey$ and producing the fact $\mathsf{N}(C, \{AKey, C\}_{k_T}, \{AKey, n', T\}_{k'})$ for some n' : nonce, and k' : dbK C. Furthermore, if $\mathsf{I}(k')$ did not appear in the initial state of the trace, then after K fired rule $\alpha_{2.1}$ and before T fired rule $\alpha_{4.1}$, C fired rule $\alpha_{3.1}$, creating the fact $\mathsf{N}(X, \{C\}_{AKey}, C, S', n'')$ for some X : msg, S' : server, and n'' : nonce.*

Proof. T's firing of rule $\alpha_{4.1}$ consumes $\mathsf{N}(\{AKey, C\}_{k_T}, \{C\}_{AKey}, C, S, n_2)$, a fact of k_T-rank 1 relative to $AKey, C$. As the initial state of the trace did not contain any fact F with $\rho_{k_T}(F; AKey, C) > 0$, some rule must have been fired which increased k_T-rank relative to $AKey, C$.

By Lemmas 20, 21, and 22, no client, server, or TGS can fire a rule which increases k_T-rank relative to $AKey, C$. By Lemmas 6 and 23, if the intruder fires a rule which increases k_T-rank relative to $AKey, C$ then the initial state of the trace contains k_T, a contradiction. Thus some K : KAS must have fired a rule which increased k_T-rank relative to $AKey, C$. By Lemma 24, the firing of that rule freshly generated $AKey$ and created the fact $\mathsf{N}(C, \{AKey, C\}_{k_T}, \{AKey, n, T\}_{k'})$ for some n : nonce and k' : dbK C.

T's firing of rule $\alpha_{4.1}$ consumes a fact of $AKey$-rank 1 relative to C. Because $AKey$ was freshly generated by some rule fired in the trace, by Lemma 25 no fact in the initial state of the trace had positive $AKey$-rank relative to C; thus some protocol participant must have fired a rule which increased this rank.

As $\mathsf{I}(k')$ was not in the initial state of the trace, the conditions of Theorem 3 are satisfied (by hypothesis and the first part of the theorem) and no state of the trace contains $\mathsf{I}(AKey)$. By Lemma 23, I cannot fire a rule which increases $AKey$-rank relative to C. By Lemmas 26–28, no KAS, TGS, or server can fire a rule which increases $AKey$-rank relative to C. Some C' : client must have fired a rule which increased $AKey$-rank relative to C; by Lemma 29, this was C firing rule $\alpha_{3.1}$ and creating the fact $\mathsf{N}(X, \{C\}_{AKey}, C, S', n')$ for some X : msg, S' : server, and n' : nonce. Finally, Lemma 25 implies that K's firing of rule $\alpha_{2.1}$ (freshly generating $AKey$) preceded C's firing of rule $\alpha_{3.1}$. □

Theorem 6. *For C : client, T : TGS, $C, T \neq \mathsf{I}$, S : server, $AKey$: shK C T, k_T : dbK T, $TFlags$: TFlag, ck : msg, $t_{C,Treq}$: time, $TOpts$: TOpt, e : etype, and n : nonce, if the beginning state of a finite trace does not contain $\mathsf{I}(k_T)$ or any fact F with $\rho_{k_T}(F; TFlags, AKey, C) > 0$, and at some point in the trace T fires rule $\gamma_{4.1}$, consuming the fact $\mathsf{N}(\{TFlags, AKey, C\}_{k_T}, \{C, ck, t_{C,Treq}\}_{AKey}, TOpts, C, S, n, e)$, then some K : KAS previously fired rule $\gamma_{2.1}$, freshly generating $AKey$ and producing the fact $\mathsf{N}(C, \{TFlags, AKey, C\}_{k_T}, \{AKey, n', TFlags, T\}^{e'}_{k'})$ for some n' : nonce, e' : etype, and k' : dbK$^{e'}$ C. Furthermore, if $\mathsf{I}(k')$ did not appear in the initial state of the trace, then after K fired rule $\gamma_{2.1}$ and before T fired rule $\gamma_{4.1}$, C fired rule $\gamma_{3.1}$, creating the fact $\mathsf{N}(X, \{C, [TOpts', C, S', n'', e'']_{AKey}, t_{C,Treq}\}_{AKey}, TOpts', C, S', n'', e'')$ for some X : msg, $TOpts'$: TOpt, e'' : etype, S' : server, and n'' : nonce.*

Proof. T's firing of rule $\gamma_{4.1}$ consumes $\mathsf{N}(\{TFlags, AKey, C\}_{k_T}, \{C, ck, t_{C,Treq}\}_{AKey}, TOpts, C, S, n, e)$, a fact of k_T-rank 1 relative to $TFlags, AKey, C$. As the initial state of the trace did not contain any fact F with $\rho_{k_T}(F; TFlags, AKey, C) > 0$, some rule must have been fired which increased k_T-rank relative to $TFlags, AKey, C$.

By Lemmas 30, 31, and 32, no client, server, or TGS can fire a rule which increases k_T-rank relative to $TFlags, AKey, C$. By Lemmas 7 and 33, if the intruder fires a rule which increases k_T-rank relative to $TFlags, AKey, C$ then the initial state of the trace contains k_T, a contradiction. Thus some K : KAS must have fired a rule which increased k_T-rank relative to $TFlags, AKey, C$. By Lemma 34, the firing of that rule freshly generated $AKey$ and created the fact $\mathsf{N}(C, \{TFlags, AKey, C\}_{k_T}, \{AKey, n_1, TFlags, T\}^{e'}_{k_C})$ for some n_1 : nonce, e' : etype, and k_C : $\mathsf{dbK}^{e'}\ C$.

T's firing of rule $\gamma_{4.1}$ consumes a fact of $AKey$-rank 1 relative to $C, ck, t_{C,Treq}$ for some ck : msg and $t_{C,Treq}$: time. Because $AKey$ was freshly generated by some rule fired in the trace, by Lemma 35 no fact in the initial state of the trace had positive $AKey$-rank relative to $C, ck, t_{C,Treq}$; thus some protocol participant must have fired a rule which increased this rank.

As $\mathsf{I}(k')$ was not in the initial state of the trace, the conditions of Theorem 4 are satisfied (by hypothesis and the first part of the theorem) and no state of the trace contains $\mathsf{I}(AKey)$. By Lemma 33, I cannot fire a rule which increases $AKey$-rank relative to $C, ck, t_{C,Treq}$. By Lemmas 36–38, no KAS, TGS, or server can fire a rule which increases $AKey$-rank relative to $C, ck, t_{C,Treq}$. Some C' : client must have fired a rule which increased $AKey$-rank relative to C; by Lemma 39, this was C firing rule $\gamma_{3.1}$ and creating the fact $\mathsf{N}(X, \{C, ck', t_{C,Treq}\}_{AKey}, TOpts', C, S', n'_2, e'')$ with $ck' = [TOpts', C, S', n'_2, e'']_{AKey}$, for some X : msg, $TOpts'$: TOpt, S' : server, n'_2 : nonce, and e'' : etype. Finally, Lemma 35 implies that K's firing of rule $\gamma_{2.1}$ (freshly generating $AKey$) preceded C's firing of rule $\gamma_{3.1}$. □

A.2 The Client/Server Exchange

A.2.1 Confidentiality of $SKey$

Theorem 7. *For C : client, T : TGS, S : server, k_T : dbK T, k_S : dbK S, $SKey$: shK $C\ S$, $AKey$: shK $C\ T$, and n : nonce, if T fires rule $\alpha_{4.1}$, consuming the fact $\mathsf{N}(\{AKey, C\}_{k_T}, \{C\}_{AKey}, C, S, n)$, freshly generating $SKey$, and creating the fact $\mathsf{N}(C, \{SKey, C\}_{k_S}, \{SKey, n, S\}_{AKey})$, and if the initial state of the trace does not contain $\mathsf{I}(k_S)$ and no state of the trace contains $\mathsf{I}(AKey)$, then no state of the trace contains $\mathsf{I}(SKey)$.*

Proof. We claim that no fact with $\{AKey, k_S\}$-corank relative to $SKey$ equal to 0 appears in the trace.

By Lemma 40, if any TGS fires a rule which decreases $\{AKey, k_S\}$-corank relative to $SKey$, then that rule freshly generates $SKey$ and, if the newly created fact in the resulting multiset is $\mathsf{N}(C, \{SKey, C\}_{k_S}, \{SKey, n, S\}_{AKey})$, the $\{AKey, k_S\}$-corank relative to $SKey$ of this multiset equals 1. By Lemma 9, no previous multiset in the trace contained a fact with finite $\{AKey, k_S\}$-corank relative to $SKey$, nor can any TGS later fire a rule which decreases $\{AKey, k_S\}$-corank relative to $SKey$.

By Lemmas 41, 42, and 43, no KAS, client, or server decreases $\{AKey, k_S\}$-corank relative to $SKey$.

By hypothesis, the fact $\mathsf{I}(AKey)$ never appears in the trace under consideration; by hypothesis and Lemma 6, the fact $\mathsf{I}(k_S)$ never appears in this trace. As T freshly generates $SKey$, by Lemma 9 I cannot freshly generate $SKey$. Thus, by Lemma 13, I does not fire any rule which decreases $\{AKey, k_S\}$-corank relative to $SKey$.

As a result, no fact of $\{AKey, k_S\}$-corank 0 relative to $SKey$, in particular $\mathsf{I}(SKey)$, occurs in any multiset of the trace. □

Corollary 8. *For C : client, T : TGS, S : server, k_T : dbK T, k_S : dbK S, $SKey$: shK $C\ S$, $AKey$: shK $C\ T$, and n : nonce, if T fires rule $\alpha_{4.1}$, consuming the fact $\mathsf{N}(\{AKey, C\}_{k_T}, \{C\}_{AKey}, C, S, n)$, freshly generating $SKey$, and creating the fact $\mathsf{N}(C, \{SKey, C\}_{k_S}, \{SKey, n, S\}_{AKey})$, and if the initial state of the trace did not contain $\mathsf{I}(k_T)$, $\mathsf{I}(k_S)$, $\mathsf{I}(k_C)$ for every k_C : dbK C, or any fact F with $\rho_{k_T}(F; AKey, C)$, then no state of the trace contains $\mathsf{I}(SKey)$.*

Proof. We simply need to show that $\mathsf{I}(AKey)$ never appears in the trace; we may then apply Theorem 7 to see that $\mathsf{I}(SKey)$ does not appear in any state of the trace. By Theorem 5, we see that some K : KAS fired rule $\alpha_{2.1}$ as specified by that theorem. By hypothesis, for every k' : dbK C, $\mathsf{I}(k')$ does not appear in the trace, including its initial state. Applying Theorem 3, we see that $\mathsf{I}(AKey)$ does not appear in the trace. □

A.2.2 Authentication of ST and authenticator

Theorem 9. *For C : client, S : server, k_S : dbK S, $SKey$: shK C S, and $t_{C,Sreq}$: time, if the beginning state of a finite trace does not contain $\mathsf{I}(k_S)$ or any fact F with $\rho_{k_S}(F; SKey, C) > 0$, and at some point in the trace S fires rule $\alpha_{6.1}$ consuming the fact $\mathsf{N}(\{SKey, C\}_{k_S}, \{C, t_{C,Sreq}\}_{SKey})$, then some T : TGS previously fired rule $\alpha_{4.1}$, freshly generating $SKey$ and producing the fact $\mathsf{N}(C, \{SKey, C\}_{k_S}, \{SKey, n, S\}_k)$ for some n : nonce and k : shK C T. Furthermore, if the fact $\mathsf{I}(k)$ has not yet appeared in the trace, then after T fired rule $\alpha_{4.1}$ and before S fired the rule $\alpha_{6.1}$, C fired rule $\alpha_{5.1}$ to create the fact $\mathsf{N}(Y, \{C, t_{C,Sreq}\}_{SKey})$ for some Y : msg.*

Proof. S's firing of rule $\alpha_{6.1}$ consumes $\mathsf{N}(\{SKey, C\}_{k_S}, \{C, t_{C,Sreq}\}_{SKey})$, a fact of k_S-rank 1 relative to $SKey, C$. As the initial state of the trace did not contain any fact F with $\rho_{k_S}(F; SKey, C) > 0$, some rule must have been fired which increased k_S-rank relative to $SKey, C$.

By Lemmas 44, 45, and 46, no client, KAS, or server can fire a rule which increases k_S-rank relative to $SKey, C$. By Lemmas 6 and 23, if the intruder fires a rule which increases k_S-rank relative to $SKey, C$ then the initial state of the trace contains k_S, a contradiction. Thus some T : TGS must have fired a rule which increased k_S-rank relative to $SKey, C$. By Lemma 47, this rule was $\alpha_{4.1}$ and T's firing of it consumed some fact $\mathsf{N}(\{k, C\}_{k_T}, \{C\}_k, C, S, n)$, freshly generated $SKey$, and created the fact $\mathsf{N}(C, \{SKey, C\}_{k_S}, \{SKey, n', S\}_k)$ for some k : shKCT, k_T : dbKT, and n, n' : nonce.

S's firing of rule $\alpha_{6.1}$ consumes a fact of $SKey$-rank 1 relative to $C, t_{C,Sreq}$. Because $SKey$ was freshly generated by some rule fired in the trace, by Lemma 25 no fact in the initial state of the trace had positive $SKey$-rank relative to $C, t_{C,Sreq}$; thus some protocol participant must have fired a rule which increased this rank.

As $\mathsf{I}(k)$ has not appeared in the trace, the conditions of Theorem 7 are satisfied (by hypothesis and the first part of the theorem) and no state of the trace contains $\mathsf{I}(SKey)$. By Lemma 23, I cannot have fired a rule which increases $SKey$-rank relative to $C, t_{C,Sreq}$. By Lemmas 48–50, no KAS, TGS, or server can fire a rule which increases $SKey$-rank relative to $C, t_{C,Sreq}$. Thus some C' : client must have fired a rule which increased $SKey$-rank relative to $C, t_{C,Sreq}$; by Lemma 51, this was C firing rule $\alpha_{5.1}$ and creating the fact $\mathsf{N}(Y, \{C, t_{C,Sreq}\}_{SKey})$ for some Y : msg. Finally, Lemma 25 implies that T's firing of rule $\alpha_{4.1}$ (freshly generating $SKey$) preceded C's firing of rule $\alpha_{5.1}$. □

Corollary 10. *For C : client, S : server, k_S : dbK S, $SKey$: shK C S, and $t_{C,Sreq}$: time, if the beginning state of a finite trace does not contain $\mathsf{I}(k_S)$ or any fact F with $\rho_{k_S}(F; SKey, C) > 0$, and at some point in the trace S fires rule $\alpha_{6.1}$ consuming the fact $\mathsf{N}(\{SKey, C\}_{k_S}, \{C, t_{C,Sreq}\}_{SKey})$, then some T : TGS previously fired rule $\alpha_{4.1}$, freshly generating $SKey$ and producing the fact $\mathsf{N}(C, \{SKey, C\}_{k_S}, \{SKey, n, S\}_k)$ for some n : nonce and k : shK C T. Furthermore, if the initial state of the trace did not contain $\mathsf{I}(k_C)$ for any k_C : dbK C, or, for any k_T : dbK T, and if $T \neq \mathsf{I}$, $\mathsf{I}(k_T)$ or any fact F with $\rho_{k_T}(F; k, C) > 0$, then after T fired rule $\alpha_{4.1}$ and before S fired the rule $\alpha_{6.1}$, C fired rule $\alpha_{5.1}$ to create the fact $\mathsf{N}(Y, \{C, t_{C,Sreq}\}_{SKey})$ for some Y : msg.*

Proof. We simply need to guarantee that $\mathsf{I}(k)$ has not yet appeared in the trace; we may then apply Theorem 9 to get the claimed result.

As T's firing of rule $\alpha_{4.1}$ produced the fact $\mathsf{N}(C, \{SKey, C\}_{k_S}, \{SKey, n, S\}_k)$ for some n : nonce and k : shK C T, it must have consumed a fact of the form $\mathsf{N}(\{k, C\}_{k'_T}, \{C\}_k, C, S, n)$ for some k'_T : dbK T. We may thus apply Theorem 5 to see that some K : KAS fired rule $\alpha_{2.1}$, freshly generating k and producing the fact $\mathsf{N}(C, \{k, C\}_{k'_T}, \{k, n', T\}_{k'_C})$ for some n' : nonce and k'_C : dbK C. By hypothesis, neither $\mathsf{I}(k'_C)$ nor $\mathsf{I}(k'_T)$ appeared in the initial state of the trace, so we may apply Theorem 3 to see that $\mathsf{I}(k)$ never appears in the trace. □

B Lemmas for Authentication Properties

B.1 General lemmas

B.1.1 Lemmas for A level analysis

Lemma 6. *For every* principal $P \neq \mathsf{I}$ *and every k : dbK P, if $\mathsf{I}(k)$ is in a state then $\mathsf{I}(k)$ was in the initial state of the trace.*

Proof. If $\mathsf{I}(k)$ appears on the right hand side of a rule for k : dbK P, then either $\mathsf{I}(k)$ appears on the left hand side of the rule (DPD) or $P = \mathsf{I}$ (DA'). □

B.1.2 Lemmas for C level analysis

Lemma 7. *For every* principal $P \neq \mathsf{I}$ *and every* $k : \mathsf{dbK}\ P$*, if* $\mathsf{I}(k)$ *is in a state then* $\mathsf{I}(k)$ *was in the initial state of the trace.*

Proof. If $\mathsf{I}(k)$ appears on the right hand side of any rule for $e : \mathsf{etype}$ and $k : \mathsf{dbK}^e\ P$ then either $\mathsf{I}(k)$ appears on the left hand side of the rule (DPD) or $P = \mathsf{I}$ (DA'). □

B.2 Lemmas for Ticket-Granting Exchange

B.2.1 Lemmas for Theorem 3

Lemma 8. *For every* $T : \mathsf{TGS}$*,* $C : \mathsf{client}$*,* $k : \mathsf{shK}\ C\ T$*, set* E *of keys, and* $K : \mathsf{KAS}$*, if* K *fires an A level rule which decreases* E*-corank relative to* k*, then the rule is rule* $\alpha_{2.1}$ *and its firing freshly generates* k*. Furthermore, the only fact in the resulting multiset which is not in the previous multiset is* $\mathsf{N}(C, \{k, C\}_{k_T}, \{k, n, T\}_{k_C})$ *for some* $k_C : \mathsf{dbK}\ C$*,* $k_T : \mathsf{dbK}\ T$*, and* $n : \mathsf{nonce}$*, and the* $\{k_C, k_T\}$*-corank relative to* k *of the resulting multiset equals* 1*.*

Proof. The only A level rule that an honest $K : \mathsf{KAS}$ may fire is $\alpha_{2.1}$. The only $k : \mathsf{shK}\ C\ T$ relative to which this rule may decrease some E-corank is the key freshly generated by this rule. For some $k_C : \mathsf{dbK}\ C$, $k_T : \mathsf{dbK}\ T$, and $n : \mathsf{nonce}$ this rule firing produces the fact $\mathsf{N}(C, \{k, C\}_{k_T}, \{k, n, T\}_{k_C})$, which has $\{k_C, k_T\}$-corank of 1 relative to k. This is the only fact on the right-hand side of rule $\alpha_{2.1}$, and thus the only fact in the multiset resulting from this rule firing that was not in the previous multiset of the trace. As k is freshly generated by this rule firing, by Lemma 9 no fact appearing earlier in the trace had finite $\{k_C, k_T\}$-corank relative to k, so the $\{k_C, k_T\}$-corank relative to k of the multiset resulting from this rule firing equals the $\{k_C, k_T\}$-corank relative to k of the new network fact. □

Lemma 9. *For every* $m_0 : \mathsf{msg}$ *and set* E *of keys, if a fact* F *such that* $\hat{\rho}_E(F; m_0) < \infty$ *occurs in a multiset of a trace, then no rule fired later in the trace freshly generates* m_0*.*

Proof. If $\hat{\rho}_E(F; m_0) < \infty$, then at least one of the arguments to the predicate forming F must be a term built up from m_0 using symmetric encryption and concatenation. By the definition of freshness, if m_0 is freshly generated by some rule firing, no fact in any multiset earlier in the trace may be built up from m_0. □

Lemma 10. *For every* $C, C' : \mathsf{client}$*,* $T : \mathsf{TGS}$*, set* E *of* keys*, and* $k : \mathsf{shK}\ C\ T$*, no A level rule that* C' *fires decreases* E*-corank relative to* k*.*

Proof. Inspection of rules $\alpha_{1.1}$, $\alpha_{1.2}$, $\alpha_{3.1}$, $\alpha_{3.2}$, $\alpha_{5.1}$, and $\alpha_{5.2}$. □

Lemma 11. *For every* $C : \mathsf{client}$*,* $T, T' : \mathsf{TGS}$*, set* E *of* keys*, and* $k : \mathsf{shK}\ C\ T$*, no A level rule that* T' *fires decreases* E*-corank relative to* k*.*

Proof. Inspection of rule $\alpha_{4.1}$. □

Lemma 12. *For every* $C : \mathsf{client}$*,* $T : \mathsf{TGS}$*, set* E *of* keys*,* $k : \mathsf{shK}\ C\ T$*, and* $S : \mathsf{server}$*, no A level rule that* S *fires decreases* E*-corank relative to* k*.*

Proof. Inspection of rule $\alpha_{6.1}$. □

Lemma 13. *For any nonempty set* E *of keys,* $C : \mathsf{client}$*,* $T : \mathsf{TGS}$*,* $C, T \neq \mathsf{I}$*, key* $k' : \mathsf{shK}\ C\ T$*, and A level intruder rule* R*, if* R *decreases* E*-corank relative to* k' *then the left hand side of* R *includes* $\mathsf{I}(k)$ *for some* $k \in E$ *or* R *freshly generates* k'*.*

Proof. Inspection of A level intruder rules. □

B.2.2 Lemmas for Theorem 4

Lemma 14. *For every T : TGS, C : client, k : shK $C\ T$, set E of keys, and K : KAS, if K fires a C level rule which decreases E-corank relative to k, then the rule is rule $\gamma_{2.1}$ and its firing freshly generates k. Furthermore, the only fact in the resulting multiset which is not in the previous multiset is* $\mathsf{N}(C, \{TFlags, k, C\}_{k_T}, \{k, n, TFlags, T\}_{k_C})$ *for some k_C : dbK C, k_T : dbK T, $TFlags$: TFlag, and n : nonce, and the $\{k_C, k_T\}$-corank relative to k of the resulting multiset equals* 1.

Proof. The only C level rules that an honest K : KAS may fire are $\gamma_{2.1}$ and $\gamma_{2.1'}$; the latter cannot decrease E-corank relative to any key. The only k : shK $C\ T$ relative to which $\gamma_{2.1}$ may decrease some E-corank is the key freshly generated by this rule. For some k_C : dbK C, k_T : dbK T, $TFlags$: TFlag, and n : nonce this rule firing produces the fact $\mathsf{N}(C, \{TFlags, k, C\}_{k_T}, \{k, n, TFlags, T\}_{k_C})$, which has $\{k_C, k_T\}$-corank of 1 relative to k. This is the only fact on the right-hand side of rule $\gamma_{2.1}$, and thus the only fact in the multiset resulting from this rule firing that was not in the previous multiset of the trace. As k is freshly generated by this rule firing, by Lemma 15 no fact appearing earlier in the trace had finite $\{k_C, k_T\}$-corank relative to k, so the $\{k_C, k_T\}$-corank relative to k of the multiset resulting from this rule firing equals the $\{k_C, k_T\}$-corank relative to k of the new network fact. □

Lemma 15. *For every m_0 : msg and set E of keys, if a fact F such that $\hat{\rho}_E(F; m_0) < \infty$ occurs in a multiset of a trace, then no rule fired later in the trace freshly generates m_0.*

Proof. If $\hat{\rho}_E(F; m_0) < \infty$, then at least one of the arguments to the predicate forming F must be a term built up from m_0 using symmetric encryption and concatenation. By the definition of freshness, if m_0 is freshly generated by some rule firing, no fact in any multiset earlier in the trace may be built up from m_0. □

Lemma 16. *For every C, C' : client, T : TGS, set E of keys, and k : shK $C\ T$, no C level rule that C' fires decreases E-corank relative to k.*

Proof. Inspection of rules $\gamma_{1.1}$, $\gamma_{1.2}$, $\gamma_{1.2'}$, $\gamma_{3.1}$, $\gamma_{3.2}$, $\gamma_{3.2'}$, $\gamma_{5.1}$, $\gamma_{5.2}$, $\gamma_{5.2'}$, $\gamma_{5'.1}$, and $\gamma_{5'.2'}$. □

Lemma 17. *For every C : client, T, T' : TGS, set E of keys, and k : shK $C\ T$, no C level rule that T' fires decreases E-corank relative to k.*

Proof. Inspection of rules $\gamma_{4.1}$ and $\gamma_{4.1'}$. □

Lemma 18. *For every C : client, T : TGS, set E of keys, k : shK $C\ T$, and S : server, no C level rule that S fires decreases E-corank relative to k.*

Proof. Inspection of rules $\gamma_{6.1}$, $\gamma_{6.1'}$, $\gamma_{6'.1}$, and $\gamma_{6'.1'}$. □

Lemma 19. *For any nonempty set E of keys, C : client, T : TGS, $C, T \neq \mathsf{I}$, key k' : shK $C\ T$, and C level intruder rule R, if R decreases E-corank relative to k' then the left hand side of R includes $\mathsf{I}(k)$ for some $k \in E$ or R freshly generates k'.*

Proof. Inspection of C level intruder rules. □

B.2.3 Lemmas for Theorem 5

Lemma 20. *For every T : TGS, k : dbK T, nonempty m_0 : msg, and A level rule R which may be fired by C : client, R does not increase k-rank relative to m_0.*

Proof. Inspection of rules $\alpha_{1.1}$, $\alpha_{1.2}$, $\alpha_{3.1}$, $\alpha_{3.2}$, $\alpha_{5.1}$, and $\alpha_{5.2}$. □

Lemma 21. *For every T : TGS, k : dbK T, nonempty m_0 : msg, and A level rule R which may be fired by S : server, R does not increase k-rank relative to m_0.*

Proof. Inspection of rule $\alpha_{6.1}$. □

Lemma 22. *For every T : TGS k : dbK T, nonempty m_0 : msg and A level rule R which may be fired by T : TGS, R does not increase k-rank relative to m_0.*

Proof. Inspection of rule $\alpha_{4.1}$. □

Lemma 23. *For any key k, message m_0, and A level intruder rule R, if R increases k-rank relative to m_0, then the left hand side of R includes $\mathsf{I}(k)$.*

Proof. Inspection of A level intruder rules (and Axiom 1 in the case of MG). □

Lemma 24. *For C : client, T : TGS, k_1 : shK $C\,T$, and k_2 : dbK T, if K : KAS fires an A level rule which increases k_2-rank relative to k_1, C, then that rule firing freshly generates k_1 and creates the fact $\mathsf{N}(C, \{k_1, C\}_{k_2}, \{k_1, n, T\}_{k_3})$ for some n : nonce and k_3 : dbK C.*

Proof. Inspection of rule $\alpha_{2.1}$. □

Lemma 25. *For every m_0 : msg and key k, if a fact F occurs in a trace and $\rho_k(F; m_0) > 0$, then no A level rule fired later in the trace freshly generates k.*

Proof. If k is freshly generated, then k does not appear in any previous multiset of the trace. Any fact F with positive k-rank relative to some m_0 must have as some argument a term constructed using encryption by k. □

Lemma 26. *For C : client, T : TGS, k : shK $C\,T$, m_0 : msg, and K : KAS, K cannot fire an A level rule which increases k-rank relative to m_0.*

Proof. Inspection of rule $\alpha_{2.1}$. □

Lemma 27. *For C : client, T, T' : TGS, k : shK $C\,T$, m_0 : msg, and K : KAS, if T' fires an A level rule which increases k-rank relative to m_0, then $m_0 = k_{CS}, n, S$ for some S : server, k_{CS} : shK $C\,S$, and n : nonce.*

Proof. Inspection of rule $\alpha_{4.1}$. □

Lemma 28. *For C : client, T : TGS, k : shK $C\,T$, m_0 : msg, and S : server, S cannot fire an A level rule which increases k-rank relative to m_0.*

Proof. Inspection of rule $\alpha_{6.1}$. □

Lemma 29. *For C, C' : client, T : TGS, k : shK $C\,T$, and m_0 : msg, if C' fires a A level rule R which increases k-rank relative to m_0, then $C' = C$, $m_0 = C$, R is $\alpha_{3.1}$ and creates the fact $\mathsf{N}(X, \{C\}_{AKey}, C, S, n_2)$ for some X : msg, $TOpts$: TOpt, S : server, and n_2 : nonce.*

Proof. Rules $\alpha_{1.1}$, $\alpha_{1.2}$, $\alpha_{3.2}$, $\alpha_{5.1}$, and $\alpha_{5.2}$ can never increase k-rank relative to m_0 for k : shK $C\,T$.

Rule $\alpha_{3.1}$, fired by C', produces the fact $\mathsf{N}(X, \{C'\}_{AKey}, C', S, n_2)$ for some X : msg, S : server, n_2 : nonce, $AKey$: shK $C\,T$, and T : TGS. This has $AKey$-rank of 1 relative to C. The only other term which might have positive $AKey$-rank relative to some m_0 is X, but this contributes to the relative rank of the left side as well. □

B.2.4 Lemmas for Theorem 6

Lemma 30. *For every T : TGS, k : dbK T, nonempty m_0 : msg, and C level rule R which may be fired by C : client, R does not increase k-rank relative to m_0.*

Proof. Inspection of rules $\gamma_{1.1}$, $\gamma_{1.2}$, $\gamma_{1.2'}$, $\gamma_{3.1}$, $\gamma_{3.2}$, $\gamma_{3.2'}$, $\gamma_{5.1}$, $\gamma_{5.2}$, $\gamma_{5.2'}$, $\gamma_{5'.1}$, and $\gamma_{5'.2'}$. □

Lemma 31. *For every T : TGS, k : dbK T, nonempty m_0 : msg, and C level rule R which may be fired by S : server, R does not increase k-rank relative to m_0.*

Proof. Inspection of rules $\gamma_{6.1}$, $\gamma_{6.1'}$, $\gamma_{6'.1}$, and $\gamma_{6'.1'}$. □

Lemma 32. *For every T : TGS k : dbK T, nonempty m_0 : msg and C level rule R which may be fired by T : TGS, R does not increase k-rank relative to m_0.*

Proof. Inspection of rules $\gamma_{4.1}$ and $\gamma_{4.1'}$. □

Lemma 33. *For any key k, message m_0, and C level intruder rule R, if R increases k-rank relative to m_0, then the left hand side of R includes $\mathsf{I}(k)$.*

Proof. Inspection of C level intruder rules. □

Lemma 34. *For C : client, T : TGS, k_1 : shK $C\ T$, k_2 : dbK T, and $TFlags$: TFlag, if K : KAS fires a C level rule which increases k_2-rank relative to $TFlags, k_1, C$, then that rule firing freshly generates k_1 and creates the fact $\mathsf{N}(C, \{TFlags, k_1, C\}_{k_2}, \{k_1, n, TFlags, T\}_{k_3})$ for some n : nonce and k_3 : dbK C.*

Proof. Inspection of rules $\gamma_{2.1}$ and $\gamma_{2.1'}$. □

Lemma 35. *For every m_0 : msg and key k, if a fact F occurs in a trace and $\rho_k(F; m_0) > 0$, then no C level rule fired later in the trace freshly generates k.*

Proof. If k is freshly generated, then k does not appear in any previous multiset of the trace. □

Lemma 36. *For C : client, T : TGS, k : shK $C\ T$, m_0 : msg, and K : KAS, K cannot fire a C level rule which increases k-rank relative to m_0.*

Proof. Inspection of rules $\gamma_{2.1}$ and $\gamma_{2.1'}$. □

Lemma 37. *For C : client, T, T' : TGS, k : shK $C\ T$, m_0 : msg, and K : KAS, if T' fires a C level rule which increases k-rank relative to m_0 then $m_0 = k_{CS}, n, SFlags, S$ for some S : server, k_{CS} : shK $C\ S$, n : nonce, and $SFlags$: SFlag.*

Proof. Inspection of rules $\gamma_{4.1}$ and $\gamma_{4.1'}$. □

Lemma 38. *For C : client, T : TGS, k : shK $C\ T$, m_0 : msg, and S : server, S cannot fire a C level rule which increases k-rank relative to m_0.*

Proof. Inspection of rules $\gamma_{6.1}$, $\gamma_{6.1'}$, $\gamma_{6'.1}$, and $\gamma_{6'.1'}$. □

Lemma 39. *For C, C' : client, T : TGS, k : shK $C\ T$, and m_0 : msg, if C' fires a C level rule R which increases k-rank relative to m_0, then $C' = C$, m_0 is either $TOpts', C, S', n_2', e''$ or $C, [TOpts', C, S', n_2', e'']_{AKey}, t_{C,Treq}$, R is $\gamma_{3.1}$ and creates the fact $\mathsf{N}(X, \{C, [TOpts, C, S, n_2, e]_{AKey}, t_{C,Treq}\}_{AKey}, TOpts, C, S, n_2, e)$ for some X : msg, $TOpts$: TOpt, S : server, n_2 : nonce, and e : etype.*

Proof. Rules $\gamma_{1.1}$, $\gamma_{1.2}$, $\gamma_{1.2'}$, $\gamma_{3.2}$, $\gamma_{3.2'}$, $\gamma_{5.1}$, $\gamma_{5.2}$, $\gamma_{5.2'}$, $\gamma_{5'.1}$, and $\gamma_{5'.2'}$ can never increase k-rank relative to m_0 for k : shK $C\ T$.

Rule $\gamma_{3.1}$, fired by C', produces the fact $\mathsf{N}(X, \{C', [TOpts, C', S, n_2, e]_{AKey}, t_{C,Treq}\}_{AKey}, TOpts, C', S, n_2, e)$ for some X : msg, $TOpts$: TOpt, S : server, n_2 : nonce, e : etype, $AKey$: shK $C\ T$, T : TGS, and $t_{C,Treq}$: time. This has $AKey$-rank of 1 relative to each of the messages $TOpts, C', S, n_2, e$ and $C', [TOpts, C', S, n_2, e]_{AKey}, t_{C,Treq}$. The only other term which might have positive $AKey$-rank relative to some m_0 is X, but this contributes to the relative rank of the left side as well. □

B.3 Lemmas for Client/Server Exchange

B.3.1 Lemmas for Theorem 7

Lemma 40. *For every C : client, S : server, k : shK $C\ S$, set E of keys, and T : TGS, if T fires an A level rule which decreases E-corank relative to k, then the rule is rule $\alpha_{4.1}$ and its firing freshly generates k. Furthermore, the only fact in the resulting multiset which is not in the previous multiset is $\mathsf{N}(C, \{k, C\}_{k_S}, \{k, n, S\}_{AKey})$ for some k_S : dbK S, $AKey$: shK $C\ T$, and n : nonce, and the $\{AKey, k_S\}$-corank relative to k of the resulting multiset equals 1.*

Proof. The only A level rule that an honest T : TGS may fire is $\alpha_{4.1}$. The only k : shK C S relative to which this rule may decrease some E-corank is the key freshly generated by this rule. For some k_S : dbK S, $AKey$: shK C T, and n : nonce, this rule firing produces the fact $\mathsf{N}(C, \{k, C\}_{k_S}, \{k, n, S\}_{AKey})$, which has $\{AKey, k_S\}$-corank of 1 relative to k. This is the only fact on the right-hand side of rule $\alpha_{4.1}$, and thus the only fact in the multiset resulting from this rule firing that was not in the previous multiset of the trace. As k is freshly generated by this rule firing, by Lemma 9 no fact appearing earlier in the trace had finite $\{AKey, k_S\}$-corank relative to k, so the $\{AKey, k_S\}$-corank relative to k of the multiset resulting from this rule firing equals the $\{AKey, k_S\}$-corank relative to k of the new network fact. □

Lemma 41. *For every K : KAS, C : client, S : server, set E of keys, and k : shK C S, no A level rule that K fires decreases E-corank relative to k.*

Proof. Inspection of rule $\alpha_{2.1}$. □

Lemma 42. *For every C, C' : client, S : server, set E of keys, and k : shK C S, no A level rule that C' fires decreases E-corank relative to k.*

Proof. Inspection of rules $\alpha_{1.1}$, $\alpha_{1.2}$, $\alpha_{3.1}$, $\alpha_{3.2}$, $\alpha_{5.1}$, and $\alpha_{5.2}$. □

Lemma 43. *For every C : client, set E of keys, S, S' : server, and k : shK C S, no A level rule that S' fires decreases E-corank relative to k.*

Proof. Inspection of rule $\alpha_{6.1}$. □

B.3.2 Lemmas for Theorem 9

Lemma 44. *For every S : server, k : dbK S, nonempty m_0 : msg, no A level rule that C : client fires increases k-rank relative to m_0.*

Proof. Inspection of rules $\alpha_{1.1}$, $\alpha_{1.2}$, $\alpha_{3.1}$, $\alpha_{3.2}$, $\alpha_{5.1}$, and $\alpha_{5.2}$. □

Lemma 45. *For every S : server, k : dbK S, nonempty m_0 : msg, no A level rule that K : KAS fires increases k-rank relative to m_0.*

Proof. Inspection of rule $\alpha_{2.1}$. □

Lemma 46. *For every S : server, k : dbK S, nonempty m_0 : msg, no A level rule that S' : server fires increases k-rank relative to m_0.*

Proof. Inspection of rule $\alpha_{6.1}$. □

Lemma 47. *For C : client, S : server, k_1 : shKCS, k_2 : dbKS, and T : TGS, if T fires an A level rule which increases k_2-rank relative to k_1, C, then that rule is $\alpha_{4.1}$, and its firing consumes the fact $\mathsf{N}(\{k_3, C\}_{k_T}, \{C\}_{k_3}, C, S, n)$, freshly generates k_1, and creates the fact $\mathsf{N}(C, \{k_1, C\}_{k_2}, \{k_1, n', S\}_{k_3})$ for some k_3 : shK C T, k_T : dbK T, and n, n' : nonce.*

Proof. Rule $\alpha_{4.1}$ is the only A level rule that an honest TGS may fire; the rest of the lemma follows by inspection of this rule. □

Lemma 48. *For C : client, S : server, k : shK C S, m_0 : msg, and K : KAS, K cannot fire an A level rule which increases k-rank relative to m_0.*

Proof. Inspection of rule $\alpha_{2.1}$. □

Lemma 49. *For C : client, S : server, k : shK C S, m_0 : msg, and T : TGS, T cannot fire an A level rule which increases k-rank relative to m_0.*

Proof. Inspection of rule $\alpha_{4.1}$. □

Lemma 50. *For C : client, S, S' : server, k : shK C S, and m_0 : msg, if S' fires an A level rule which increases k-rank relative to m_0 then $S' = S$ and $m_0 = t$ for some t : time.*

Proof. Inspection of rule $\alpha_{6.1}$. □

Lemma 51. *For C, C' : client, S : server, k : shK C S, and m_0 : msg, if C' fires an A level rule which increases k-rank relative to m_0, then $C' = C$, $m_0 = C, t$ for some t : time, the rule is $\alpha_{3.1}$, and its firing creates the fact $\mathsf{N}(Y, \{C, t\}_k)$ for some Y : msg.*

Proof. Rules $\alpha_{1.1}$, $\alpha_{1.2}$, $\alpha_{3.1}$, $\alpha_{3.2}$, and $\alpha_{5.2}$ can never increase k-rank relative to any m_0 for k : shK C S.

Rule $\alpha_{5.1}$, fired by C', produces the fact $\mathsf{N}(Y, \{C', t\}_{k'})$ for some Y : msg, t : time, S' : server, and k' : shK C' S'. As Y also appears on the left hand side of this rule, whose firing increases k-rank relative to m_0, it must be that the k-rank of $\{C', t\}_{k'}$ relative to m_0 is greater than the k-rank relative to m_0 of the left hand side of this rule. This term has positive k-rank relative to m_0 if and only if $m_0 = C', t$ and $k = k'$; considering the type of $k = k'$, we see that $C = C'$ and $S = S'$. □

C Anomalous Traces

In this appendix we give detailed traces for some of the anomalies discussed in Section 7. These include discussion of the MSR rules which are fired in each trace; the message flows are as shown in Section 7.

C.1 A level trace of ticket anomaly

		$C, \alpha_{1.1}$ $\longrightarrow$	$\mathsf{N}(C, T, n_1)$ $L(C, T, n_1)$
$L(C, T, n_1)$	$\mathsf{N}(C, T, n_1)$	$K, \alpha_{2.1}$ $\longrightarrow$	$\mathsf{N}(C, \{AKey, C\}_{k_T}, \{AKey, n_1, T\}_{k_C})$
$L(C, T, n_1)$	$\mathsf{N}(C, \{AKey, C\}_{k_T}, \{AKey, n_1, T\}_{k_C})$	I $\longrightarrow$	$\exists X : \mathsf{msg}$ $\mathsf{I}(C, X, \{AKey, n_1, T\}_{k_C})$ $\mathsf{I}(\{AKey, C\}_{k_T})$
$L(C, T, n)$ $\mathsf{I}(\{AKey, C\}_{k_T})$	$\mathsf{I}(C, X, \{AKey, n_1, T\}_{k_C})$	I $\longrightarrow$	$\mathsf{N}(C, X, \{AKey, n_1, T\}_{k_C})$
$\mathsf{I}(\{AKey, C\}_{k_T})$	$\mathsf{N}(C, X, \{AKey, n_1, T\}_{k_C})$ $L(C, T, n_1)$	$C, \alpha_{1.2}$ $\longrightarrow$	$Auth_C(X, n_1, T, AKey)$
$\mathsf{I}(\{AKey, C\}_{k_T})$	$Auth_C(X, n_1, T, AKey)$	$C, \alpha_{3.1}$ $\longrightarrow$	$\mathsf{N}(X, \{C\}_{AKey}, C, S, n_2)$ $L(C, S, T, n_2)$ $Auth_C(X, n_1, T, AKey)$
$Auth_C(X, n_1, T, AKey)$ $L(C, S, T, n_2)$ $\mathsf{I}(\{AKey, C\}_{k_T})$	$\mathsf{N}(X, \{C\}_{AKey}, C, S, n_2)$	I $\longrightarrow$	$\mathsf{I}(X, \{C\}_{AKey}, C, S, n_2)$
$Auth_C(X, n_1, T, AKey)$ $L(C, S, T, n_2)$	$\mathsf{I}(X, \{C\}_{AKey}, C, S, n_2)$ $\mathsf{I}(\{AKey, C\}_{k_T})$	I $\longrightarrow$	$\mathsf{N}(\{AKey, C\}_{k_T}, \{C\}_{AKey}, C, S, n_2)$
$Auth_C(X, n_1, T, AKey)$ $L(C, S, T, n_2)$ $\mathsf{I}(C, X, \{AKey, C\}_{k_T})$	$\mathsf{N}(\{AKey, C\}_{k_T}, \{C\}_{AKey}, C, S, n_2)$ $Valid(C, S, n_2)$	$T, \alpha_{4.1}$ $\longrightarrow$	$\mathsf{N}(C, \{SKey, C\}_{k_S}, \{SKey, n_2, S\}_{AKey})$

Figure 33: Producing anomalous behavior in the abstract formalization.

Figure 33 shows a sequence of rule firings which realize the ticket anomaly of Section 7.1. Arrows indicate the firing of rules, with the labels above each arrow indicating the principal firing the rule and the rule being fired. In each row, the rule being used rewrites the facts in the second column as those in the fourth; the facts in the first column remain untouched by the rule in question.

C sends a request for credentials to K using the rule $\alpha_{1.1}$. K sees the network message C, T, n_1 and replies using rule $\alpha_{2.1}$, sending the network message $C, TGT, \{AKey, n_1, T\}_{k_C}$ where $TGT = \{AKey, C\}_{k_T}$ is the ticket granting ticket. The intruder I reads this message from the network using rule INT and creates a new message X using MG. I then creates (using DMC, DMC, CMP, CMP) the message $C, X, \{AKey, n_1, T\}_{k_C}$, *i.e.*, K's message with the ticket TGT for T replaced by the freshly generated message X, and puts it on the network using TRN. C now sees the network message $C, X, \{AKey, n_1, T\}_{k_C}$, which is of the form she expects (she does not expect to be able to read the ticket, and so cannot tell that it has been replaced by X). She thus completes the Authentication Service Exchange by firing rule $\alpha_{1.2}$, storing X, n_1, T, and $AKey$ in the memory predicate $Auth_C$.

Believing she has obtained credentials for T, C now initiates the Ticket Granting Exchange with T. She uses the $Auth_C$ memory predicate to fire rule $\alpha_{3.1}$ and send the network message $X, \{C\}_{AKey}, S, n_2$. When I sees this message on the network he removes the message from the network (INT). She then generates a new message $TGT, \{C\}_{AKey}, S, n_2$ by replacing X with the original ticket TGT (DMC, CMP). I then puts this message onto

the network (TRN). Finally, T sees the network message $TGT, \{C\}_{AKey}, S, n_2$ and uses this to fire the rule $\alpha_{4.1}$, granting C's apparent request for credentials for use with S.

C.2 C level trace of encryption type anomaly

Here we give a trace which realizes the encryption type anomaly sketched in Section 7.3. Recall that C knows that the key $k_C : \mathsf{dbK}^e\ C$ has been compromised and attempts to request a ticket-granting ticket from $K : \mathsf{KAS}$ using some other database key $k'_C : \mathsf{dbK}^{e'}\ C$. C thus fires rule $\gamma_{1.1}$, putting the message $KOpts, C, T, n_1, e'$ on the network. I already has possession of k_C and e as indicated by the predicates $\mathsf{I}(k_C)$ and $\mathsf{I}(e)$. Using rules INT, DMC, CMP, and TRN, she intercepts C's message and constructs the message $KOpts, C, T, n_1, e$ and then put this message on the network. K sees this altered request as a legitimate $\mathsf{KRB\_AS\_REQ}$ message and fires rule $\gamma_{2.1}$, placing $C, X, \{AKey, n_1, TFlags, T\}^e_{k_C}$ on the network, where X is the ticket granting ticket $\{TFlags, AKey, C\}_{k_T}$ for C to present to T. I intercepts this message using INT and then obtains $AKey$ using DMC, SDC', and DMC; an additional application of DMC allows her to obtain the ticket-granting ticket. At this point, I has the ticket-granting ticket and the session key $AKey$ needed to use it, so she may impersonate C to the TGS T.

$\mathsf{I}(k_C)$ $\mathsf{I}(e)$	$C, \gamma_{1.1}$ $\longrightarrow$	$\mathsf{N}(KOpts, C, T, n_1, e')$ $L(C, KOpts, T, n_1, e')$
$\mathsf{N}(KOpts, C, T, n_1, e')$	I, INT $\longrightarrow$	$\mathsf{I}(KOpts, C, T, n_1, e')$
$\mathsf{I}(KOpts, C, T, n_1, e')$	I, DMC $\longrightarrow$	$\mathsf{I}(KOpts, C, T, n_1)$ $\mathsf{I}(e')$
$\mathsf{I}(KOpts, C, T, n_1)$ $\mathsf{I}(e)$	I, CMP $\longrightarrow$	$\mathsf{I}(KOpts, C, T, n_1, e)$
$\mathsf{I}(KOpts, C, T, n_1, e)$	I, TRN $\longrightarrow$	$\mathsf{N}(KOpts, C, T, n_1, e)$
$\mathsf{N}(KOpts, C, T, n_1, e)$ $Valid_K(KOpts, C, T, n_1, e)$ $SetAuthFlags_K(KOpts, C, T, TFlags)$ $SetETypes_K(C, e, e, T, e'')$	$K, \gamma_{2.1}$ $\longrightarrow$	$\mathsf{N}(C, \{TFlags, AKey, C\}^{e''}_{k_T},$ $\{AKey, n_1, TFlags, T\}^e_{k_C})$
$\mathsf{N}(C, \{TFlags, AKey, C\}^{e''}_{k_T}, \{AKey, n_1, TFlags, T\}^e_{k_C})$	I, INT $\longrightarrow$	$\mathsf{I}(C, \{TFlags, AKey, C\}^{e''}_{k_T},$ $\{AKey, n_1, TFlags, T\}^e_{k_C})$
$\mathsf{I}(C, \{TFlags, AKey, C\}^{e''}_{k_T}, \{AKey, n_1, TFlags, T\}^e_{k_C})$	I, DMC $\longrightarrow$	$\mathsf{I}(C, \{TFlags, AKey, C\}^{e''}_{k_T})$ $\mathsf{I}(\{AKey, n_1, TFlags, T\}^e_{k_C})$
$\mathsf{I}(\{AKey, n_1, TFlags, T\}^e_{k_C})$ $\mathsf{I}(k_C)$	I, SDC' $\longrightarrow$	$\mathsf{I}(AKey, n_1, TFlags, T)$
$\mathsf{I}(AKey, n_1, TFlags, T)$	I, DMC $\longrightarrow$	$\mathsf{I}(AKey)$

Figure 34: Encryption Type Anomaly

D Message Fields

This appendix consists of tables describing the correspondence between the message structures defined in [16] and the network messages in our formalizations of Kerberos 5. Each table lists the field names (in `typewriter type`) as given in Section 5 of [16], with subfields indented to indicate the level of nesting, and the corresponding names used for the fields included in our formalizations. For those fields with subfields, the entry for the field shows the included subfields as well as any encryption that is used; this provides some overlap with the entries for the subfields.

The KRB_AS_REQ message type is a message of type KRB_KDC_REQ, which is defined in Section 5.4.1 of [16]. The fields of this message are listed in Table 2.

Field Name	A Level	B Level	C Level
pvno	(omitted) 5	(omitted) 5	(omitted) 5
msg-type	(omitted) KRB_AS_REQ	(omitted) KRB_AS_REQ	(omitted) KRB_AS_REQ
padata	(omitted)	(omitted)	(omitted)
req-body	C, T, n_1	C, T, n_1	$KOpts, C, T, n_1, e$
kdc-options	(omitted)	(omitted)	$KOpts$
cname	C	C	C
sname	T	T	T
from	(omitted)	(omitted)	(omitted)
till	(omitted)	(omitted)	(omitted)
rtime	(omitted)	(omitted)	(omitted)
nonce	n_1	n_1	n_1
etype	(omitted)	(omitted)	e
addresses	(omitted)	(omitted)	(omitted)
enc-authorization-data	(omitted)	(omitted)	(omitted)
additional-tickets	(omitted)	(omitted)	(omitted)

Table 2: Fields in the KRB_AS_REQ message.

The KRB_AS_REP message type is a message of type KRB_KDC_REP, which is defined in Section 5.4.2 of [16]. Note that the structure of the `ticket` field is defined in Section 5.3.1 of [16]. The fields of this message are listed in Table 3.

The KRB_TGS_REQ message type is a message of type KRB_KDC_REQ, which is defined in Section 5.4.1 of [16]. The fields of this message are listed in Table 4.

As noted in the description of `padata` under Section 5.4.1 of [16], "[r]equests for additional tickets (KRB_TGS_REQ) must contain a `padata` of PA_TGS_REQ." The description of PA-DATA in Section 5.2.7 of [16] seems to suggest that, at least in this message, the checksum should be present and keyed. Section 5.2.7.1 of [16] notes that "[t]he checksum in the authenticator (which must be collision-proof) is to be computed over the KDC-REQ-BODY encoding." Thus we take the `cksum` field to be a keyed checksum over the `req-body` field. *It is important to note that this field does not include the* `ticket`*, so the checksum will not be able to detect tampering with the ticket.*

The `padata` field contains the authentication header, which is of type KRB_AP_REQ (as noted in the second paragraph under section 3.3 and the first paragraph of 5.5.1 in [16]). The subfields of this message type are listed directly as subfields of `padata`. These include the `ticket`, whose constituent subfields are not listed (see Table 3) since it is unreadable by the client, and the freshly constructed `authenticator`, whose subfields are listed. A full description of authenticators is given in section 5.3.2 of [16]. The `cksum` field of the authenticator "contains a checksum of the application data that accompanies the KRB_AP_REQ" (under the description of `cksum` in 5.3.2 of [16]), *i.e.*, that accompanies the authentication header.

The KRB_TGS_REP message type is a message of type KRB_KDC_REP, which is defined in Section 5.4.2 of [16], and as such parallels the structure of the KRB_AS_REP message given above. The fields of this message are listed in Table 3. We do not show the effects of an anonymous ticket (in which the ANONYMOUS flag in the `ticket` is set); this would change the `cname` from C to a generic client name.

The fields of the KRB_AP_REQ message are shown in Table 6. This message has the same structure as the authentication header of the KRB_TGS_REQ message above. The `cksum` field in the authenticator is described as optional

Field Name	A Level	B Level	C Level
pvno	(omitted) 5	(omitted) 5	(omitted) 5
msg-type	(omitted) KRB_AS_REP	(omitted) KRB_AS_REP	(omitted) KRB_AS_REP
padata	(omitted)	(omitted)	(omitted)
crealm	(omitted)	(omitted)	(omitted)
cname	C	C	C
ticket	$\{AKey, C\}_{k_T}$	$\{AKey, C, t_{K,auth}, t_{K,end}\}_{k_T}$	$\{TFlags, AKey, C\}_{k_T}$
tkt-vno	(omitted) 5	(omitted) 5	(omitted) 5
realm	(omitted)	(omitted)	(omitted)
sname	(omitted)	(omitted)	(omitted)
enc-part	$\{AKey, C\}_{k_T}$	$\{AKey, C, t_{K,auth}, t_{K,end}\}_{k_T}$	$\{TFlags, AKey, C\}_{k_T}$
flags	(omitted)	(omitted)	$TFlags$
key	$AKey$	$AKey$	$AKey$
crealm	(omitted)	(omitted)	(omitted)
cname	C	C	C
transited	(omitted)	(omitted)	(omitted)
authtime	(omitted)	$t_{K,auth}$	(omitted)
starttime	(omitted)	(omitted)	(omitted)
endtime	(omitted)	$t_{K,end}$	(omitted)
renew-till	(omitted)	(omitted)	(omitted)
caddr	(omitted)	(omitted)	(omitted)
authorization-data	(omitted)	(omitted)	(omitted)
enc-part	$\{AKey, n_1, T\}_{k_C}$	$\{AKey, n_1, t_{K,auth}, t_{K,end}, T\}_{k_C}$	$\{AKey, n_1, TFlags, T\}_{k_C}$
key	$AKey$	$AKey$	$AKey$
last-req	(omitted)	(omitted)	(omitted)
nonce	n_1	n_1	n_1
key-expiration	(omitted)	(omitted)	(omitted)
flags	(omitted)	(omitted)	$TFlags$
authtime	(omitted)	$t_{K,auth}$	(omitted)
starttime	(omitted)	(omitted)	(omitted)
endtime	(omitted)	$t_{K,end}$	(omitted)
renew-till	(omitted)	(omitted)	(omitted)
srealm	(omitted)	(omitted)	(omitted)
sname	T	T	T
caddr	(omitted)	(omitted)	(omitted)

Table 3: Included fields for the KRB_AS_REP message.

Field Name	A Level	B Level	C Level
pvno	(omitted) 5	(omitted) 5	(omitted) 5
msg-type	(omitted) KRB_TGS_REQ	(omitted) KRB_TGS_REQ	(omitted) KRB_TGS_REQ
padata	$\{AKey, C\}_{k_T}$ $\{C\}_{AKey}$	$\{AKey, C,$ $t_{K,auth}, t_{K,end}\}_{k_T}$ $\{C, t_{C,Treq}\}_{AKey}$	$\{TFlags, AKey, C\}_{k_T}$ $\{C, [\texttt{req-body}]_{AKey},$ $t_{C,Treq}\}_{AKey}$
pvno	(omitted)	(omitted)	(omitted)
msg-type	(omitted)	(omitted)	(omitted)
ap-options	(omitted)	(omitted)	(omitted)
ticket	$\{AKey, C\}_{k_T}$	$\{AKey, C,$ $t_{K,auth}, t_{K,end}\}_{k_T}$	$\{TFlags, AKey, C\}_{k_T}$
authenticator	$\{C\}_{AKey}$	$\{C, t_{C,Treq}\}_{AKey}$	$\{C, [\texttt{req-body}]_{AKey},$ $t_{C,Treq}\}_{AKey}$
authenticator-vno	(omitted)	(omitted)	(omitted)
crealm	(omitted)	(omitted)	(omitted)
cname	C	C	C
cksum	(omitted)	(omitted)	$H(\texttt{req-body})$
cusec	(omitted)	(omitted)	(omitted)
ctime	(omitted) $t_{C,Treq}$	$t_{C,Treq}$	$t_{C,Treq}$
subkey	(omitted)	(omitted)	(omitted)
seq-number	(omitted)	(omitted)	(omitted)
authorization-data	(omitted)	(omitted)	(omitted)
req-body	C, S, n_2	C, S, n_2	$TOpts, C, S, n_2, e$
kdc-options	(omitted)	(omitted)	$TOpts$
cname	C	C	C
sname	S	S	S
from	(omitted)	(omitted)	(omitted)
till	(omitted)	(omitted)	(omitted)
rtime	(omitted)	(omitted)	(omitted)
nonce	n_2	n_2	n_2
etype	(omitted)	(omitted)	e
addresses	(omitted)	(omitted)	(omitted)
enc-authorization-data	(omitted)	(omitted)	(omitted)
additional-tickets	(omitted)	(omitted)	(omitted)

Table 4: Fields in the KRB_TGS_REQ message.

Field Name	A Level	B Level	C Level
pvno	(omitted) 5	(omitted) 5	(omitted) 5
msg-type	(omitted) KRB_TGS_REP	(omitted) KRB_TGS_REP	(omitted) KRB_TGS_REP
padata	(omitted)	(omitted)	(omitted)
crealm	(omitted)	(omitted)	(omitted)
cname	C	C	C
ticket	$\{SKey, C\}_{k_S}$	$\{SKey, C, t_{T,auth}, t_{T,end}\}_{k_S}$	$\{SFlags, SKey, C\}_{k_S}$
tkt-vno	(omitted)	(omitted)	(omitted)
realm	(omitted)	(omitted)	(omitted)
sname	(omitted)	(omitted)	(omitted)
enc-part	$\{SKey, C\}_{k_S}$	$\{SKey, C, t_{T,auth}, t_{T,end}\}_{k_S}$	$\{SFlags, SKey, C\}_{k_S}$
flags	(omitted)	(omitted)	$SFlags$
key	$SKey$	$SKey$	$SKey$
crealm	(omitted)	(omitted)	(omitted)
cname	C	C	C
transited	(omitted)	(omitted)	(omitted)
authtime	(omitted)	$t_{T,auth}$	(omitted)
starttime	(omitted)	(omitted)	(omitted)
endtime	(omitted)	$t_{T,end}$	(omitted)
renew-till	(omitted)	(omitted)	(omitted)
caddr	(omitted)	(omitted)	(omitted)
authorization-data	(omitted)	(omitted)	(omitted)
enc-part	$\{SKey, n_2, S\}_{AKey}$	$\{SKey, n_2, t_{T,auth}, t_{T,end}, S\}_{AKey}$	$\{SKey, n_2, SFlags, S\}_{AKey}$
key	$SKey$	$SKey$	$SKey$
last-req	(omitted)	(omitted)	(omitted)
nonce	n_2	n_2	n_2
key-expiration	(omitted)	(omitted)	(omitted)
flags	(omitted)	(omitted)	$SFlags$
authtime	(omitted)	$t_{T,auth}$	(omitted)
starttime	(omitted)	(omitted)	(omitted)
endtime	(omitted)	$t_{T,end}$	(omitted)
renew-till	(omitted)	(omitted)	(omitted)
srealm	(omitted)	(omitted)	(omitted)
sname	S	S	S
caddr	(omitted)	(omitted)	(omitted)

Table 5: Included fields for the KRB_TGS_REP message.

and application specific in the first paragraph under Section 3.2.2 of [16].

Field Name	A Level	B Level	C Level
pvno	(omitted) 5	(omitted) 5	(omitted) 5
msg-type	(omitted) KRB_AP_REQ	(omitted) KRB_AP_REQ	(omitted) KRB_AP_REQ
ap-options	(omitted)	(omitted)	$SOpts$
reserved	(omitted)	(omitted)	(omitted) $0\|1$
use-session-key	(omitted)	(omitted)	(omitted) $0\|1$
mutual-required	(omitted)	$0\|1$	$0\|1$
reserved	(omitted)	(omitted)	(omitted) $0\|\ldots\|2^{29}-1$
ticket	$\{SKey, C\}_{k_S}$	$\{SKey, C, t_{T,auth}, t_{T,end}\}_{k_S}$	$\{SFlags, SKey, C\}_{k_S}$
authenticator	$\{C, t_{C,Sreq}\}_{SKey}$	$\{C, t_{C,Sreq}\}_{SKey}$	$\{C, []_{SKey}, t_{C,Sreq}\}_{SKey}$
authenticator-vno	(omitted)	(omitted)	(omitted)
crealm	(omitted)	(omitted)	(omitted)
cname	C	C	C
cksum	(omitted)	(omitted)	$H(\cdots)$
cusec	(omitted)	(omitted)	(omitted)
ctime	$t_{C,Sreq}$	$t_{C,Sreq}$	$t_{C,Sreq}$
subkey	(omitted)	(omitted)	(omitted)
seq-number	(omitted)	(omitted)	(omitted)
authorization-data	(omitted)	(omitted)	(omitted)

Table 6: Fields in the KRB_AP_REQ message.

The structure of the KRB_AP_REP message, which S sends to C when mutual authentication has been requested, is shown in Table 7.

Field Name	A Level	B Level	C Level
pvno	(omitted) 5	(omitted) 5	(omitted) 5
msg-type	(omitted) KRB_AP_REP	(omitted) KRB_AP_REP	(omitted) KRB_AP_REP
enc-part	$\{t_{C,Sreq}\}_{SKey}$	$\{t_{C,Sreq}\}_{SKey}$	$\{t_{C,Sreq}\}_{SKey}$
ctime	$t_{C,Sreq}$	$t_{C,Sreq}$	$t_{C,Sreq}$
cusec	(omitted)	(omitted)	(omitted)
subkey	(omitted)	(omitted)	(omitted)
seq-number	(omitted)	(omitted)	(omitted)

Table 7: Fields in the KRB_AP_REP message.

Finally, the structure of the KRB_ERROR messages is shown in Table 8. Error messages are not implemented in the A level formalization.

Field Name	A Level	B Level	C Level
`pvno`	(omitted) 5	(omitted) 5	(omitted) 5
`msg-type`	(omitted) KRB_ERROR	KRB_ERROR	KRB_ERROR
`ctime`	(omitted) $--\|t_{C,Treq}\|t_{C,Sreq}$	$--\|t_{C,Treq}\|t_{C,Sreq}$	$--\|t_{C,Treq}\|t_{C,Sreq}$
`cusec`	(omitted)	(omitted)	(omitted)
`stime`	(omitted) $t_{(K\|T\|S),err}$	$t_{(K\|T\|S),err}$	$t_{(K\|T\|S),err}$
`susec`	(omitted)	(omitted)	(omitted)
`error-code`	(omitted) $ErrorCode$	$ErrorCode$	$ErrorCode$
`crealm`	(omitted)	(omitted)	(omitted)
`cname`	(omitted) C	(omitted) C	(omitted) C
`realm`	(omitted)	(omitted)	(omitted)
`sname`	(omitted) $K\|T\|S$	$K\|T\|S$	$K\|T\|S$
`e-text`	(omitted)	(omitted)	(omitted)
`e-data`	(omitted)	(omitted)	(omitted)

Table 8: Fields in the KRB_ERR message.