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

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

†Department of Mathematics
University of Pennsylvania
209 South 33rd Street
Philadelphia, PA 19104–6395
{fbutler@math,adj@math,scedrov@saul.cis}.upenn.edu

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

Abstract

We formalize aspects of the Kerberos 5 authentication protocol in the Multi-Set Rewriting formalism (MSR) on two levels of detail. The more detailed formalization reflects the intricate structure of the Kerberos 5 specification, taking into account several protocol features which have not been previously considered. In the abstract formalization, we prove an authentication property about Kerberos 5. We discovered three anomalies, one of which occurs on both levels of detail, while the other two rely on the richer structure of the detailed formalization. We also discuss how the addition of checksums (some of which are in the protocol specification and some of which are not) may eliminate some of these anomalies.

1 Introduction

Kerberos [8, 10] is a widely deployed protocol, aimed at repeatedly authenticating a client to multiple application servers based on a single login. Kerberos makes use of various tickets, encrypted under a server's key unknown to the user, which when forwarded in an appropriate request authenticate the user to the desired service. A formalization of Kerberos 4, the first publicly released version of this protocol, was given in [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 [9]. This approach proved effective for finding an attack, but came short of proving positive correctness results. However, the authors of [9] note that the discovered attack is unachievable in a full implementation of Kerberos 5.

We have recently started a project, the goal of which is to use the Multi-Set Rewriting (MSR) framework to give a precise specification of Kerberos 5 at various levels of detail, ranging from the minimal account used in [9] to a detailed formalization of every behavior encompassed by this complex suite [8, 10]. Our 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 possibly proving them. 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 addition, we have started exploring forms of reasoning that best take advantage of the linguistic features of MSR.

This paper reports on preliminary results of this investigation. We provide two formalizations of Kerberos 5. Our abstract description omits most timestamps and all optional features, including only what we believe is needed to provide authentication. The second specification includes several low-level aspects of this protocol, namely options, flags, checksums and error messages, none of which have appeared in any previous study of Kerberos, but does not include temporal checks or most timestamps. A third formalization, not presented here, adds some timestamps and temporal checks to our abstract formalization. We do not consider timestamps and temporal checks in this paper due to the thorough treatment of them in [1, 2, 3], space consid-

[*]Butler and Jaggard were partially supported by ONR Grant N00014-01-1-0431. Scedrov was partially supported by the DoD University Research Initiative (URI) program administered by the Office of Naval Research under Grant N00014-01-1-0795, and by NSF Grant CCR-0098096. Cervesato was partially supported by NRL under contract N00173-00-C-2086.

erations, and the fact that our most interesting results thus far do not involve temporal properties. We are currently working to extend these formalizations, incorporating the features of each, to give additional formalizations which are even closer to the full protocol.

We have proved a sample of the expected authentication properties of our abstract formalization using the notion of rank and corank functions, inspired by [11]. It appears that this technique will carry over to proofs of other authentication properties of our abstract formalization, and also corresponding properties of our more detailed formalization; we are currently pursuing this. Kerberos is specifically claimed not to address denial of service attacks, but we found three other anomalies which may occur when using this protocol. The first, which arises in both formalizations presented, violates properties that were proved to hold for Kerberos 4 [1], highlighting the structural differences between the messages in versions 4 and 5 of the protocol. The other two anomalies, seen only in our more detailed formalization, take advantage of protocol options available at this level; the first of these is related to the anomaly seen in both formalizations while the second is completely unrelated.

In Sections 2, 3 and 4, we give an overview of the Kerberos 5 protocol, the MSR formalism, and the methods we use to analyze protocol formalizations, respectively. Our abstract formalization is discussed in Section 5, its intruder model is given in Section 6, and the anomaly at this level is discussed in Section 7. We state and prove authentication properties inspired by [12] in Section 8. Section 9 presents our detailed formalization of Kerberos 5, while the corresponding intruder is formalized in Section 10. Section 11 discusses more anomalies. Finally, Section 12 outlines directions for future work.

2 Overview of the Kerberos 5 protocol

A standard run of Kerberos 5 consists of three phases in succession outlined in Fig. 1 (please ignore the details in this figure for the moment). Suppose a client C seeks to authenticate herself to a particular application server S.

- In the first phase, C sends a request `KRB_AS_REQ` to the *Kerberos Authentication Server* (KAS) K for a *ticket granting ticket* TGT, for use with a particular *Ticket Granting Server* (TGS) T. K replies with a message `KRB_AS_REP` consisting of the ticket TGT and an encrypted component containing a fresh *authentication key* $AKey$ to be shared between C and T. TGT is encrypted using the secret key k_T of T; the accompanying message is encrypted under C's secret key k_C.

- In the second phase, C forwards TGT, along with an *authenticator* encrypted under $AKey$, to the TGS T (message `KRB_TGS_REQ`). T responds in `KRB_TGS_REP` by sending a *service ticket* ST encrypted under the secret key k_S of the application server S, and a component containing a *service key* $SKey$ to be shared between C and S, encrypted under $AKey$.

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

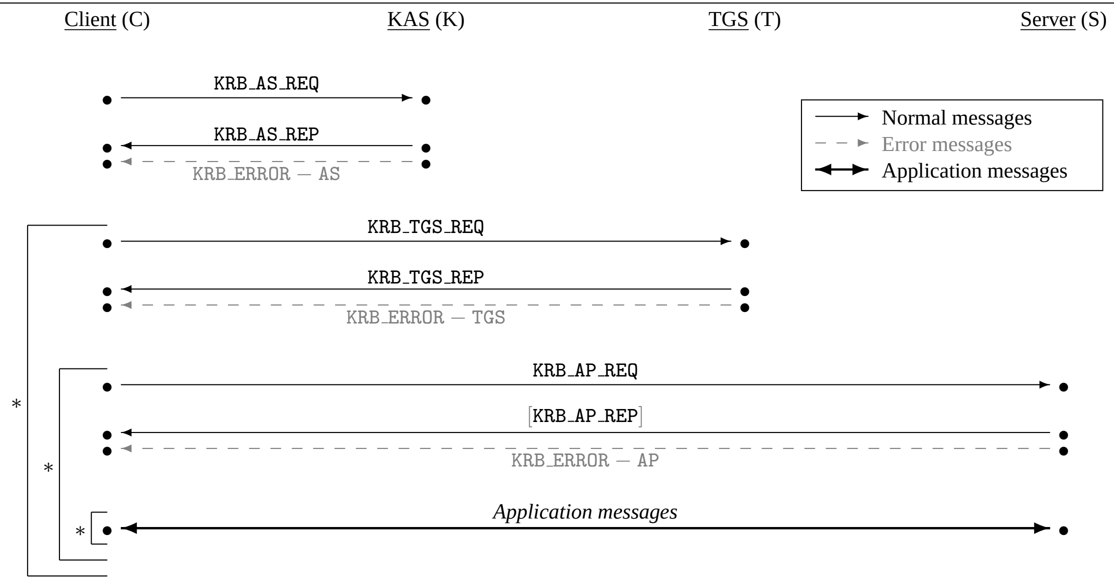
A single TGT can be used to obtain several service tickets, possibly with several application servers, while it is valid. Similarly, one ST can be used for repeated service from S before it expires. In both cases, a fresh authenticator is required for each use of the ticket.

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

We will often come back to Fig. 1 as we formalize different aspects of Kerberos 5. The messages and fields in black will constitute our abstract specification which we describe in Section 5. We will enrich it in Section 9 with the parts in gray to obtain our more detailed formalization.

3 MSR

MSR originated as a simple logic-oriented language aimed at investigating the decidability of protocol analysis under a variety of assumptions [7]. 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 [6]. It uses strongly-typed multiset rewriting rules over first-order atomic formulas to express protocol actions and relies on a form of existential quantification to symbolically model the generation of nonces and other fresh data. 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 [6], and 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.



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

Figure 1. Kerberos 5 Messages in the Abstract and Detailed Formalizations.

3.1 Signature

When writing a specification in MSR, one decides on a logical classification of protocol entities by laying out appropriate type declarations. The signature fragment in Figure 2 sets up the typing infrastructure in the case of Kerberos 5. The types used in this paper are summarized in the central column of the following table. 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 next column expresses the subtyping relation corresponding to these types ($\tau :: \tau'$ means that τ is a subsort of τ'). Indentation is used as a visual aid to track dependencies. The declarations shown in black support the abstract formalization of this protocol, while the grayed-out additions are necessary for the more detailed specification.

Observe that shared keys (shK _ _) can be part of a message, but database keys (dbK _) cannot. Notice also that the encryption types (needed in the detailed specification) parameterize the various keys.

The syntax of messages is given next. The first two declarations formalize concatenation and shared-key encryption (possibly using different algorithms, as specified by the encryption types). The last declaration captures message digests as an implementation of cryptographic hashing. They are declared similarly to shared-key encryption.

(Paring)	$\_,\_ : \mathsf{msg} \rightarrow \mathsf{msg} \rightarrow \mathsf{msg}.$
(Encryption)	$\{\_\}^{\_}_{\_} : \mathsf{etype} \rightarrow \mathsf{msg} \rightarrow \mathit{key} \rightarrow \mathsf{msg}.$
(Message digest)	$[\_]^{\_}_{\_} : \mathsf{etype} \rightarrow \mathsf{msg} \rightarrow \mathit{key} \rightarrow \mathsf{msg}.$

We will keep the encryption type implicit unless we are specifically discussing it (as in Section 11.2). In our present formalizations, we only use shared keys and not database keys to construct message digests.

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 issued by C to make a request to S.

	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 $\rightarrow$ type.		
	dbK : etype $\rightarrow$ *tcs* $\rightarrow$ type.	$\forall e$: etype, A : *tcs*. $\mathsf{dbK}^e\ A$ :: *key*^{e}.	$k_{\_}$
	shK : etype $\rightarrow$ client $\rightarrow$ *ts* $\rightarrow$ type.	$\forall e$: etype, C : client, A : *ts*. $\mathsf{shK}^e\ C\ A$ :: *key*^{e}.	$AKey$
		$\forall e$: etype, C : client, A : *ts*. $\mathsf{shK}^e\ C\ A$ :: 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 Abstract and Detailed Specifications of Kerberos 5

3.2 State and Roles

Intuitively, MSR represents the state of execution of a protocol as a multiset M 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 memory or role state predicates. *Memory predicates* are used to store information across several runs of a protocol, to pass data to subprotocols, and to invoke external modules. The intruder I stores intercepted information m in the predicate $\mathsf{I}(m)$. We will encounter other memory predicates as we go along. *Role state predicates*, of the form $L(\ldots)$, allow sequentializing the actions of a principal.

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

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

4 Protocol Analysis Methods

4.1 Overview of rank and corank

Inspired by Schneider's analysis of the amended Needham-Schroeder protocol [11], we define rank and corank functions on (multisets of) facts.

Our definition of k-rank relative to m_0 is intended to capture the maximum number of nested encryptions of the message m_0 under the key k which appear in a fact. We additionally require that the innermost encryption be exactly $\{m_0\}_k$. We use rank to authenticate the origin of $\{m_0\}_k$ by showing that a particular principal may create a message of k-rank 1 relative to m_0 and that no other principals, including the intruder, can increase the rank of facts.

The E-corank of relative to m_0 is intended to capture the minimum number of decryptions, using keys from E, that must be performed in order to extract m_0 from a fact. We use corank to prove the secrecy of m_0 by showing that no facts of E-corank 0 relative to m_0 can ever appear in the multisets in a trace.

The definitions of rank and corank of messages and facts are given below; we do not define rank and corank for constraints since these are not facts. We are optimistic that the notions of rank and corank and the corresponding proof techniques can be easily extended and exported to the MSR formalizations of other protocols.

$$\rho_k(m;m_0) = \begin{cases} 0, & m \text{ is an atomic message} \\ \rho_k(m_1;m_0)+1, & m = \{m_1\}_k,\ \rho_k(m_1;m_0) > 0 \\ 0, & m = \{m_1\}_k,\ \rho_k(m_1;m_0) = 0,\ m_1 \neq m_0 \\ 1, & m = \{m_0\}_k \\ \rho_k(m_1;m_0), & m = \{m_1\}_{k'},\ k' \neq k \\ \max\{\rho_k(m_1;m_0), \rho_k(m_2;m_0)\}, & m = m_1, m_2 \end{cases}.$$

Figure 3. Definition of relative k-rank on messages.

$$\hat{\rho}_E(m;m_0) = \begin{cases} \infty, & m \text{ is atomic, } m \neq m_0 \\ 0, & m = m_0 \\ \hat{\rho}_E(m_1;m_0)+1, & m = \{m_1\}_k \neq m_0,\ k \in E \\ \hat{\rho}_E(m_1;m_0), & m = \{m_1\}_k \neq m_0,\ k \notin E \\ \min\{\hat{\rho}_E(m_1;m_0), \hat{\rho}_E(m_2;m_0)\}, & m = (m_1, m_2)\ \ m_1, m_2 \neq \emptyset \end{cases}$$

Figure 4. Definition of relative E-corank on messages.

4.2 The k-rank relative to m_0

We define a function $\rho_k(m;m_0)$, the k-rank of a message m relative to a fixed message m_0, which we will use to prove properties about the origin of data. If a message m appears on the network with $\rho_k(m;m_0) = i > 0$ and the intruder is unable to increase the k-rank of messages relative to m_0, then this message originated with some other principal. Since the other principals are well-behaved (all malicious activity is done by the intruder), we may then be able to make a statement about the origin of the message m. Formally, for a fixed nonzero $m_0 : \mathsf{msg}$, any key k, and any $m : \mathsf{msg}$ we define $\rho_k(m;m_0)$ as in Fig. 3.

We extend the the definition of k-rank relative to m_0 to facts as follows. For $m, m_1, \ldots, m_j : \mathsf{msg}$, P a role state or memory predicate (other than I or N) of arity j, and k' a database key, make the following definitions.

$$\begin{aligned} \rho_k(\mathsf{N}(m);m_0) &= \rho_k(m;m_0) & (1) \\ \rho_k(\mathsf{I}(m);m_0) &= \rho_k(m;m_0) & (2) \\ \rho_k(\mathsf{I}(k');m_0) &= 0 & (3) \\ \rho_k(P(m_1,\ldots,m_j);m_0) &= \max_{1\leq i\leq j} \rho_k(m_1;m_0) & (4) \end{aligned}$$

We then define the k-rank relative to m_0 of a multiset A of finitely many distinct facts as

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

A rule r which has the multiset A on its left hand side and the multiset B on its right hand side is said to increase (resp. preserve, decrease) k-rank relative to m_0 if $\rho_k(B;m_0) > \rho_k(A;m_0)$ (resp. $=$, $<$). We use weakly increase, etc., in the usual manner.

4.3 The E-corank relative to m_0

While the k-rank relative to m_0 will be useful in proving properties about the origin of data, we turn to an intuitively dual notion, that of E-corank relative to a message m_0, to prove things about secrecy. We require that m_0 cannot be written as m_1, m_2 for two nonempty messages m_1, m_2. Formally, we define the E-corank of m relative to m_0 as in Fig. 4.

We extend corank to facts and multisets of facts in a slightly different way than we do for rank. Since corank is intended to capture the minimum work, with respect to a set E of keys, needed to obtain a secret m_0, we need to take care when considering decryption by principals. If a principal decrypts a message to obtain the secret m_0 and stores m_0 in a predicate *but never sends m_0 over the network, encrypted or otherwise,* this action by itself should not mean we view m_0 as accessible without further decryption using keys from E. For a predicate P of arity j, we thus do not want to define the E-corank relative to m_0 of $P(m_1,\ldots,m_j)$ to be the minimum of the relative E-coranks of P's arguments, but rather the minimum over those which might ever go back onto the network. For the moment, we make the following definitions using this idea as a guide and do not give a formal definition of the relative E-corank of a general predicate P.

For a fixed $m_0 : \mathsf{msg}$ and $m, m_i : \mathsf{msg}$, L a role state predicate of arity j, k' a database key, define the E-corank of facts relative to m_0 as follows.

$$\begin{aligned} \hat{\rho}_E(\mathsf{N}(m);m_0) &= \hat{\rho}_E(m;m_0) \\ \hat{\rho}_E(\mathsf{I}(m);m_0) &= \hat{\rho}_E(m;m_0) \end{aligned}$$

$$\begin{aligned}
\hat{\rho}_E(Auth_C(m_1, m_2, m_3, m_4); m_0) &= \hat{\rho}_E(m_1; m_0)\\
\hat{\rho}_E(Service_C(m_1, m_2, m_3, m_4); m_0) &= \hat{\rho}_E(m_1; m_0)\\
\hat{\rho}_E(L(m_1, \ldots, m_j); m_0) &= \infty\\
\hat{\rho}_E(DoneMut_C(m_1, m_2); m_0) &= \infty\\
\hat{\rho}_E(Mem_S(m_1, m_2, m_3); m_0) &= \infty\\
\hat{\rho}_E(\mathsf{I}(k'); m_0) &= \infty
\end{aligned}$$

We may then define the E-corank relative to m_0 of a multiset A of finitely many distinct facts as

$$\hat{\rho}_E(A; m_0) = \min_{F \in A} \hat{\rho}_k(F; m_0). \qquad (6)$$

5 Abstract Level Protocol Formalization

In this section we give our abstract formalization of the protocol. The MSR rules corresponding to each exchange here are depicted in Figs. 5—10. Portions of these figures are grayed out; the dark type represents our abstract level rules, and the grayed portions will be ignored for now. When we refer to an abstract level rule by name, we shall use an α with an appropriate subscript, given in the figures in dark type over each arrow.

We believe that this formalization contains the minimum amount of detail needed to capture the Kerberos 5 protocol. Although we do not directly use the nonces in the lemmas and proofs given here, omitting them would remove the connection between the KRB_AS_REQ and KRB_AS_REP messages.

5.1 Authentication Service Exchange

Recall that the Authentication Service Exchange allows a client C to obtain from a KAS K credentials to be used in the Ticket Granting Exchange with a TGS T. C's actions in this exchange are formalized in Fig. 5, K's in Fig. 6.

C asks K for credentials for the server T using rule $\alpha_{1.1}$, sending a KRB_AS_REQ message with her name C, the name T of the TGS for which she wishes to obtain credentials, and a fresh nonce n_1, and storing these in the role state predicate L. Rule $\alpha_{1.2}$ allows C to process the KRB_AS_REP message which K sends in response to her initial request. This message is expected to contain C's name, an opaque ticket X to be passed on to T, and, encrypted under C's long-term key k_C, a session key $AKey$ for use with T, the nonce n_1 from the original request, and the name T of the TGS. If the message is of this form and if C, T, and n_1, match the data from the original request (stored in L), C removes the KRB_AS_REP message from the network, deletes the role state predicate L, and stores the relevant data in the persistent memory predicate $Auth_C$. In the abstract formalization, C does not process any other (i.e., error) messages which K may return as defined in [10].

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

5.2 The Ticket-Granting Exchange

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

The client's actions in this exchange are formalized in Fig. 7. If, as indicated by the memory predicate $Auth_C(X, T, AKey)$, the client C has completed the authentication service exchange to get credentials for the TGS T, rule $\alpha_{3.1}$ allows her to initiate an exchange with T to obtain credentials for the application server S. In doing so, she chooses a new nonce n_2 and sends a KRB_TGS_REQ message to T consisting of the previously obtained ticket X, an authenticator (C encrypted under the session key $AKey$), her name C, the name S of the server for which C wishes to obtain credentials, and the new nonce n_2. C stores the information about this request in the role state predicate L, and retains the memory predicate $Auth_C$ for use in future exchanges with T.

The client's second rule, $\alpha_{3.2}$, allows her to read from the network a KRB_TGS_REP message that matches her request to T. This message consists of C's name, an opaque ticket Y to be passed to the application server, and, encrypted under the session key $AKey$, a session key $SKey$ for use by C and the application server, the nonce n_2, and the application server's name S. C, S, and n_2 must match the stored information about the original request to T in order for C to process the KRB_TGS_REP message. If C does process the message using $\alpha_{3.2}$, she stores the ticket Y, server name S, and session key $SKey$ in the memory predicate $Service_C$.

Fig. 8 shows the actions of the TGS T in the Ticket Granting Exchange. The rule $\alpha_{4.1}$ allows T to process a valid KRB_TGS_REQ message and send a KRB_TGS_REP reply containing credentials for C to use with S. The request on the network must contain a ticket ($AKey$ and C encrypted under T's secret key k_T), an authenticator with C's name encrypted under the key from the ticket, the client's name C, the server's name S, and a nonce n_2. If the message is of the correct form and is valid (as determined by the external process $Valid$), then T may reply with credentials for

$$\left(\begin{array}{llcl}
\multicolumn{4}{l}{\exists L : \mathsf{client} \times \mathsf{KOpt} \times \mathsf{TGS} \times \mathsf{nonce} \times \mathsf{etype}.} \\
\begin{array}{l}\forall T : \mathsf{TGS}\ .\\ \forall K : \mathsf{KAS}\ .\\ \forall KOpts : \mathsf{KOpt}.\\ \forall e : \mathsf{etype}\ .\end{array} & & \xrightarrow{\alpha\delta_{1.1}} & \begin{array}{l}\exists n_1 : \mathsf{nonce}\\ \mathsf{N}(KOpts, C, T, n_1, e)\\ L(C, KOpts, T, n_1, e)\end{array} \\
\begin{array}{l}\forall \ldots\ .\\ \forall k_C : \mathsf{dbK}\ C\ .\\ \forall AKey : \mathsf{shK}\ C\ T.\\ \forall X : \mathsf{msg}\ .\\ \forall n_1 : \mathsf{nonce}\ .\\ \forall TFlags : \mathsf{TFlag}\ .\end{array} & \begin{array}{l}\mathsf{N}(C, X, \{AKey,\\ \quad n_1, TFlags, T\}_{k_C})\\ L(C, KOpts, T, n_1, e)\end{array} & \xrightarrow{\alpha\delta_{1.2}} & \begin{array}{l}Auth_C(X, TFlags,\\ \quad T, AKey)\end{array} \\
\begin{array}{l}\forall \ldots\ .\\ \forall ErrorCode : \mathsf{msg}.\\ \forall t_{K,err} : \mathsf{time}\ .\end{array} & \begin{array}{l}\mathsf{N}(\mathtt{KRB\_ERROR}, t_{K,err},\\ \quad ErrorCode, C, K)\\ L(C, KOpts, T, n_1, e)\end{array} & \xrightarrow{\delta_{1.2'}} & \begin{array}{l}ASError_C(\mathtt{KRB\_ERROR},\\ \quad t_{K,err}, ErrorCode, K)\end{array}
\end{array}\right)^{\forall C:\mathsf{client}}$$

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

$$\left(\begin{array}{llcl}
\begin{array}{l}\forall C : \mathsf{client}\ .\\ \forall T : \mathsf{TGS}\ .\\ \forall n_1 : \mathsf{nonce}\ .\\ \forall k_C : \mathsf{dbK}\ C\ .\\ \forall k_T : \mathsf{dbK}\ T\ .\\ \forall AKey : \mathsf{shK}\ C\ T.\\ \forall KOpts : \mathsf{KOpt}\ .\\ \forall e : \mathsf{etype}\ .\\ \forall TFlags : \mathsf{TFlag}\ .\end{array} & \begin{array}{l}\mathsf{N}(KOpts, C, T, n_1, e)\\ Valid(KOpts, C, T, n_1, e)\\ SetAuthFlags(KOpts, TFlags)\end{array} & \xrightarrow{\alpha\delta_{2.1}} & \begin{array}{l}\exists AKey : \mathsf{shK}\ C\ T\\ \mathsf{N}(C, \{TFlags, AKey, C\}_{k_T},\\ \quad \{AKey, n_1, TFlags, T\}_{k_C})\end{array} \\
\begin{array}{l}\forall \ldots\ .\\ \forall ErrorCode : \mathsf{msg}.\\ \forall t_{K,err} : \mathsf{time}\ .\end{array} & \begin{array}{l}\mathsf{N}(KOpts, C, T, n_1, e)\\ Invalid(KOpts, C, T, n_1, e)\\ Clock_K(t_{K,err})\end{array} & \xrightarrow{\delta_{2.1'}} & \begin{array}{l}\mathsf{N}(\mathtt{KRB\_ERROR}, t_{K,err},\\ \quad ErrorCode, C, K)\end{array}
\end{array}\right)^{\forall K:\mathsf{KAS}}$$

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

C to use with S. To do so, T generates a new session key for use by C and S and constructs a ticket for S containing this key and C's name and encrypted under S's secret key k_S. T then sends a network message consisting of C's name, the ticket for S, and, encrypted under the session key $AKey$ that T shares with C, the session key $SKey$ for use by C and S, the nonce from C's request, and the name of the server S. T does not send any error messages in this level of abstraction.

5.3 The Client/Server Authentication Exchange

The fifth and sixth messages in dark in Fig. 1 form the Client/Server Authentication Exchange. Here the client C presents credentials, previously obtained from a TGS, to the application server S. In the abstract level formalization we assume that mutual authentication is requested by C; thus the sixth message (of type KRB_AP_REP) is required for the protocol to finish.

Fig. 9 shows the role of the client C in the exchange. If she has previously obtained credentials (a ticket Y and session key $SKey$, stored in the memory predicate $Service_C$) for use with S she may fire rule $\alpha_{5.1}$. This places a KRB_AP_REQ message containing the ticket and an authenticator (obtained by encrypting C and the current time $t_{C,Sreq}$ on C's system, given by the external process $Clock_C$, under the session key $SKey$) on the network and stores the relevant information about this request in the role state predicate L. C's second rule, $\alpha_{5.2}$, may be fired when the network contains a KRB_AP_REP message consisting of $t_{C,Sreq}$ encrypted under $SKey$. This rule reads the message from the network and stores the server name S and the session key $SKey$ in the the $DoneMut_C$ predicate. Although not modeled in the abstract level formalization, this information is intended to be used in additional communications with S.

The server's role in this exchange is given in Fig. 10. If the network contains a valid (as determined by the external process $Valid$) KRB_AP_REQ message consisting of a ticket $\{SKey, C\}_{k_S}$ encrypted under S's secret key k_S and a matching authenticator $\{C, t_{C,Sreq}\}_{SKey}$, S may process the request by firing rule $\alpha_{6.1}$. This reads the message from the network and replies with the timestamp $t_{C,Sreq}$ encrypted under the session key $SKey$ from the ticket. Note that in this formalization, we are essentially using $t_{C,Sreq}$

$$\left(\begin{array}{llcl}
\multicolumn{4}{l}{\exists L : \mathsf{client}\times\mathsf{TOpt}\times\mathsf{server}\times\mathsf{TGS}\times\mathsf{nonce}\times\mathsf{time}.} \\[1ex]
\begin{array}{l}\forall T:\mathsf{TGS}.\\ \forall S:\mathsf{server}.\\ \forall AKey:\mathsf{shK}\;C\;T.\\ \forall X:\mathsf{msg}.\\ \forall t_{C,Treq}:\mathsf{time}.\\ \forall TFlags:\mathsf{TFlag}.\\ \forall TOpts:\mathsf{TOpt}.\\ \forall e:\mathsf{etype}.\end{array} &
\begin{array}{l} Auth_C(X, TFlags, T, AKey)\\ Clock_C(t_{C,Treq})\end{array} &
\xrightarrow{\alpha\delta_{3.1}} &
\begin{array}{l}\exists n_2:\mathsf{nonce}\\ \mathsf{N}(X, \{C, [TOpts, C, S, n_2, e]_{AKey},\\ \quad t_{C,Treq}\}_{AKey}, TOpts, C, S, n_2, e)\\ Auth_C(X, TFlags, T, AKey)\\ L(C, TOpts, S, T, n_2, t_{C,Treq}, e)\end{array} \\[1ex]
\begin{array}{l}\forall\ldots.\\ \forall SKey:\mathsf{shK}\;C\;S.\\ \forall Y:\mathsf{msg}.\\ \forall n_2:\mathsf{nonce}.\\ \forall SFlags:\mathsf{SFlag}.\end{array} &
\begin{array}{l}\mathsf{N}(C, Y,\\ \quad \{SKey, n_2, SFlags, S\}_{AKey})\\ L(C, TOpts, S, T, n_2, t_{C,Treq}, e)\end{array} &
\xrightarrow{\alpha\delta_{3.2}} &
\begin{array}{l} Service_C(Y, SFlags,\\ \quad S, SKey)\end{array} \\[1ex]
\begin{array}{l}\forall\ldots.\\ \forall ErrorCode:\mathsf{msg}.\\ \forall t_{T,err}:\mathsf{time}.\end{array} &
\begin{array}{l}\mathsf{N}(\mathtt{KRB\_ERROR}, t_{C,Treq}, t_{T,err},\\ \quad ErrorCode, C, T)\\ L(C, TOpts, S, T, n_2, t_{C,Treq}, e)\end{array} &
\xrightarrow{\delta_{3.2'}} &
TGSError_C(T, t_{T,err}, ErrorCode)
\end{array}\right)^{\forall C:\mathsf{client}}$$

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

$$\left(\begin{array}{llcl}
\begin{array}{l}\forall C:\mathsf{client}.\\ \forall S:\mathsf{server}.\\ \forall AKey:\mathsf{shK}\;C\;T.\\ \forall k_T:\mathsf{dbK}\;T.\\ \forall k_S:\mathsf{dbK}\;S.\\ \forall n_2:\mathsf{nonce}.\\ \forall t_{C,Treq}:\mathsf{time}.\\ \forall TOpts:\mathsf{TOpt}.\\ \forall e:\mathsf{etype}.\\ \forall SFlags:\mathsf{SFlag}.\\ \forall ck:\mathsf{msg}.\\ \forall TFlags:\mathsf{TFlag}.\end{array} &
\begin{array}{l}\mathsf{N}(\{TFlags, AKey, C\}_{k_T},\\ \quad \{C, ck, t_{C,Treq}\}_{AKey},\\ \quad TOpts, C, S, n_2, e)\\ Valid(TOpts, C, S, n_2, e, t_{C,Treq})\\ SetServFlags(TOpts, SFlags)\\ ck = [TOpts, C, S, n_2, e]_{AKey}\end{array} &
\xrightarrow{\alpha\delta_{4.1}} &
\begin{array}{l}\exists SKey:\mathsf{shK}\;C\;S\\ \mathsf{N}(C, \{SFlags, SKey, C\}_{k_S},\\ \quad \{SKey, n_2, SFlags, S\}_{AKey})\end{array} \\[1ex]
\begin{array}{l}\forall\ldots.\\ \forall ErrorCode:\mathsf{msg}.\\ \forall t_{T,err}:\mathsf{time}.\end{array} &
\begin{array}{l}\mathsf{N}(\{TFlags, AKey, C\}_{k_T},\\ \quad \{C, ck, t_{C,Treq}\}_{AKey},\\ \quad TOpts, C, S, n_2, e)\\ Invalid(TOpts, C, S, n_2,\\ \quad e, t_{C,Treq}, ck)\\ Clock_T(t_{T,err})\end{array} &
\xrightarrow{\delta_{4.1'}} &
\begin{array}{l}\mathsf{N}(\mathtt{KRB\_ERROR}, t_{C,Treq}, t_{T,err},\\ \quad ErrorCode, C, T)\end{array}
\end{array}\right)^{\forall T:\mathsf{TGS}}$$

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

as a nonce; no temporal checks are performed on it, but this reply is intended to authenticate S to C. The server S also remembers the relevant data from the request for future communication with C and to guard against replay attacks, although neither of these uses is modeled in this formalization.

6 Abstract Level Intruder Formalization

In this section, we present the rules specifying the Dolev-Yao intruder model for Kerberos 5. We ask the reader to ignore for the moment the grayed-out text as it describes additions needed for the more detailed intruder. We will come back to them in Section 10.

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

- the fairly standard operations of interception/transmission of a message over the network, decomposition/composition of a pair, and decryption/encryption of a message given a known key (Section 6.1);
- the often overlooked action of generating new data (Section 6.2);
- and the use of accessible data (Section 6.3).

6.1 Network, Pairing and Encryption Rules

We have the following rules describing how the Dolev-Yao intruder can intercept/transmit messages, decom-

$$\left(\begin{array}{llcl}
\multicolumn{4}{l}{\exists L : \mathsf{client}^{(C)} \times \mathsf{SOpt}\times\mathsf{server}^{(S)} \times \mathsf{shK}\ C\ S \times \mathsf{time} \times \mathsf{msg}.}\\
\begin{array}{l}\forall S : \mathsf{server}\ .\\ \forall SKey : \mathsf{shK}\ C\ S.\\ \forall t_{C,Sreq} : \mathsf{time}\ .\\ \forall Y : \mathsf{msg}\ .\\ \forall SFlags : \mathsf{SFlag}\ .\\ \forall SOpts : \mathsf{SOpt}\ .\end{array} & \begin{array}{l} Service_C(Y, SFlags, S, SKey)\\ Mutual(SOpts)\\ Clock_C(t_{C,Sreq})\end{array} & \xrightarrow{\alpha\delta_{5.1}} & \begin{array}{l}\mathsf{N}(SOpts, Y, \{C, [\ldots]_{SKey}, t_{C,Sreq}\}_{SKey})\\ Service_C(Y, SFlags, S, SKey)\\ L(C, SOpts, S, SKey, t_{C,Sreq}, Y)\end{array}\\
\forall \ldots & \begin{array}{l}\mathsf{N}(\{t_{C,Sreq}\}_{SKey})\\ L(C, SOpts, S, SKey, t_{C,Sreq}, Y)\end{array} & \xrightarrow{\alpha\delta_{5.2}} & DoneMut_C(S, SKey)\\
\begin{array}{l}\forall\ldots\ .\\ \forall ErrorCode : \mathsf{msg}.\\ \forall t_{S,err} : \mathsf{time}\ .\end{array} & \begin{array}{l}\mathsf{N}(\mathtt{KRB\_ERROR}, t_{C,Sreq},\\ \quad t_{S,err}, ErrorCode, C, S)\\ L(C, SOpts, S, SKey, t_{C,Sreq}, Y)\end{array} & \xrightarrow{\delta_{5.2'}} & APError_C(S, t_{S,err}, ErrorCode)
\end{array}\right)^{\forall C:\mathsf{client}}$$

Figure 9. The client's role in the Client/Server Exchange with mutual authentication.

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

Figure 10. The end server's role in the Client/Server Exchange with mutual authentication.

pose/compose pairs, and decrypt/encrypt messages under the various types of known keys. We have also some administrative rules that permit the duplication and deletion of deleted data.

The following two rules model interception (INT) and transmission (TRN):

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

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

The following two rules model decomposition (DMC) and composition (CMP):

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

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

The following two rules model decryption (SDC′) and encryption (SEC′) with a shared key:

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

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

The following two rules model decryption (DDC′) and encryption (DEC′) with a long term (database) key:

$$\left(\begin{array}{l}\forall A : \mathit{tcs}\ .\\ \forall e : \mathsf{etype}\ .\\ \forall k : \mathsf{dbK}^e\ A.\\ \forall m : \mathsf{msg}\ .\end{array}\quad \begin{array}{l}\mathsf{I}(\{m\}_k^e)\\ \mathsf{I}(k)\end{array} \xrightarrow{\text{DDC'}} \mathsf{I}(m)\right)^{\mathsf{I}}$$

$$\left(\begin{array}{l}\forall A : \mathit{tcs}\ .\\ \forall e : \mathsf{etype}\ .\\ \forall k : \mathsf{dbK}^e\ A.\\ \forall m : \mathsf{msg}\ .\end{array}\quad \begin{array}{l}\mathsf{I}(m)\\ \mathsf{I}(k)\end{array} \xrightarrow{\text{DEC'}} \mathsf{I}(\{m\}_k^e)\right)^{\mathsf{I}}$$

We conclude with the some administrative rules that allow the intruder to duplicate known data (DPM and DPD) and to forget information (DLM and DLD), for both messages and database keys. The deletion form can safely be omitted from the specification.

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

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

$$\left(\begin{matrix}\forall A : \mathit{tcs} & .\\ \forall e : \mathsf{etype} & .\\ \forall k_A : \mathsf{dbK}^e\ A. & \end{matrix}\quad \mathsf{I}(k_A) \quad \xrightarrow{\text{DPD}} \quad \begin{matrix}\mathsf{I}(k_A)\\ \mathsf{I}(k_A)\end{matrix}\right)^{\mathsf{I}}$$

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

6.2 Data Generation Rules

As a rule, the intruder should be able to generate everything an honest principal can generate, in our case nonces and session keys, and no more. In the case of Kerberos, we must admit an exception to this rule: since principals forward uninterpreted data, we must allow the intruder to create garbage, modeled as objects of the generic type msg.

The rules for generating nonces (NG) and session keys (KG′) are as follows:

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

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

We have the following message generation rule:

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

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

6.3 Data Access Rules

The intruder is entitled to look up the same data as any other principal. It can however store intercepted data in the $\mathsf{I}(\_)$ predicate to mount attacks. The intruder therefore has access to the name of any principal, to its own keys (both long- and short-term), and to timestamps.

Access to a generic principal name (possibly a client, a server, a TGS or a KAS) is handled by the following rule:

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

The next rule below gives the intruder access to any defined timestamp. This makes objects in this syntactic category guessable and in this regard different from nonces.

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

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

$$\left(\begin{matrix}\forall A : \mathit{ts} & .\\ \forall e : \mathsf{etype} & .\\ \forall k : \mathsf{shK}^e\ \mathsf{I}\ A. & \end{matrix}\quad \cdot \quad \xrightarrow{\text{SA1'}} \quad \mathsf{I}(k)\right)^{\mathsf{I}}$$

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

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

Finally, we have the following rule describing how the intruder access his long-term keys:

$$\left(\begin{matrix}\forall e : \mathsf{etype} & .\\ \forall k : \mathsf{dbK}^e\ \mathsf{I}. & \end{matrix}\quad \cdot \quad \xrightarrow{\text{DA'}} \quad \mathsf{I}(k)\right)^{\mathsf{I}}$$

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

7 Abstract Level Anomalous Behavior

One of the most notorious differences between versions 4 and 5 of Kerberos is the manner the KAS and the TGS transmit the ticket-granting and service tickets, respectively. In Kerberos 4, the client receives these tickets as part of the data encrypted under either her dbK or a shK she knows. We saw that version 5 sends the tickets as a separate component. Thus it seems possible for the intruder to take advantage of this new message structure and tamper with the unprotected ticket. A specific example of such a scenario is as follows.

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 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 does not know 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). He 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.

As a result, T has fired rule $\alpha_{4.1}$ based on the network message $\{AKey, C\}_{k_T}, \{C\}_{AKey}, C, S, n_2$ even though C never sent this message. This 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 this formalization of the new version of Kerberos, it becomes the following.

Violated Property 1. *For* C : client, T : TGS, $AKey$: shK C T, k_T : dbK T, S : server, *and* n_2 : nonce, *if* $\{C\}_{AKey}$ *and* $\{AKey, C\}_{k_T}$ *have appeared on the network (possibly encrypted), and* I *does not have access to* $AKey$*, then* 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 abstract formalization of Kerberos 5; as noted below in Sec. 11, it does not hold for our detailed formalization either. It was possible for it to hold for Kerberos 4 because the message in that protocol sent from K to C was the equivalent of (after dropping timestamps) $\{AKey, T, TGT\}_{k_C}$. 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. The above comments show that this anomaly also violates the translation 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 described above), replacing the ticket for S with an arbitrary bit-string and then reversing the switch when C sends a `KRB_AP_REQ` message to S. Such a switch shows that the translation of Theorem 6.23 of [1] also fails for Kerberos 5 for similar structural reasons, as does a corresponding theorem in [2].

We believe that the attack found by Mitchell, Mitchell, and Stern [9] on 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 $AKey$.

8 Abstract Level Protocol Verification using Rank and Corank

8.1 Authentication in the TGS Exchange

In this section we state and outline the proof of an authentication theorem that we were able to prove using the ideas of rank and corank. Although we have not directly formulated it, there is some measure of confidentiality implicit in the proof of this theorem as we show that the intruder does not know $AKey$. Note that keys are not intentionally leaked to the intruder after some period of time as they are in [1, 2, 3].

The authentication theorem which we include is intended to capture authentication of the client C to the TGS T, in the form of *data origin authentication*. The theorem states that, given a few assumptions, if T grants credentials to C for use with S on the basis of some ticket and authenticator, then the ticket originated with some K : KAS and was generated by K's firing of a certain MSR rule to grant C credentials for use with T. Furthermore, the authenticator originated with C and was generated by C's firing of another specified MSR rule to request credentials for some S' : server from T, and this was fired after K fired its rule. The assumptions required for this conclusion are that the intruder does not have access to the database keys of C and T which are used in the messages under consideration and that the authenticator and ticket did not exist in the initial state of the trace. This theorem appears to be the appropriate weakening of the property of Kerberos 4 which is violated by the anomaly discussed above; in particular, the theorem does not state that C sent the ticket and authenticator together. Formally, we have the following.

Theorem 1. *For* C : client, T : TGS, $C, T \neq$ I, S : server, k_C : dbK C, k_T : dbK T, $AKey$: shK C T, *and* n_2 : nonce, *if the beginning state of a finite trace does not contain* $\mathsf{I}(k_C)$, $\mathsf{I}(k_T)$, *or any fact* F *with* $\rho_{k_T}(F; AKey, C) > 0$ *or* $\rho_{AKey}(F; C) > 0$, *and at some point in the trace* T *fires rule* $\alpha_{4.1}$, *consuming the fact* $\mathsf{N}(\{AKey, C\}_{k_T}, \{C\}_{AKey}, C, S, n_2)$, *then earlier in the trace, some* K : KAS *fired rule* $\alpha_{2.1}$, *existentially 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. *Also, after* K *fired this rule and before* T *fired the rule in the hypothesis,* C *fired rule* $\alpha_{3.1}$ *to create the fact* $\mathsf{N}(X, \{C\}_{AKey}, C, S', n')$ *for some* X : msg, S' : server, *and* n' : nonce.

In order to show that the ticket $\{AKey, C\}_{k_T}$ originates with K, we consider the k_T-rank of facts relative to $AKey, C$. By hypothesis this rank is not positive for any fact at the beginning of the trace. A few straightforward lemmas, proved by inspection of the MSR rules and the hypothesis that the intruder does not know k_T at the beginning of the trace, show that K may create the ticket and that neither any other honest principal nor the intruder could have created it. In order to show that the authenticator $\{C\}_{AKey}$ originates with C, we consider the $AKey$-rank of facts relative to C. A number of lemmas show that the authenticator cannot originate with an honest principal other than C. Further lemmas and an argument based on $\{k_C, k_T\}$-corank relative to $AKey$ show show that the intruder cannot generate the authenticator. A lemma connecting existential generation of keys to rank shows that C generated the authenticator after K generated the ticket.

We expect that this theorem should lift to the more detailed formalization fairly easily and that similar techniques can be used to prove other authentication properties.

9 Detailed Level Protocol Formalization

Our more detailed formalization is closer to the version of Kerberos 5 specified in [10] than is the abstract level. We have formalized the option (in the field of type TOpt) which allows the client to request an `ANONYMOUS` ticket from T (Sec. 9.2) and the message field (of type etype) which allows the client to request a particular encryption method, and noted anomalies involving both of these details. Message digests now appear as specified in the protocol; we are investigating their utility in avoiding the anomalies we discuss here. We have added the message field (of type SOpt) which allows C to specify whether a `KRB_AP_REP` response from S is requested (Sec. 9.3) and have also incorporated error messages. These will allow investigation of protocol runs which would not appear in the previous analyses of Kerberos. The authenticators sent to T and S now include the timestamps $t_{C,Treq}$ and $t_{C,Sreq}$ so that error messages may be associated with their corresponding requests, although we have not added any temporal checks. 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.

When we refer to Figs. 5—10, we now mean the entire figure, including the grayed-out portions. Figs. 11 and 12 are grayed-out entirely to represent the fact that they do not appear in the abstract level formalization at all, but do appear here. When we refer to a detailed level rule by name we will now use δ with an appropriate subscript as given in the various figures; this will mean the entire rule depicted in the figure, including the grayed-out portions. While fields specifying encryption type appear in several messages in this level, and should technically appear for every encrypted message that occurs (according to Sec. 3), we make the convention that we will omit the etype in this capacity unless we are explicitly discussing it (as in Sec. 11.2).

9.1 Authentication Service Exchange

The client's actions in this exchange are formalized in Fig. 5. As in the abstract formalization, in rule $\delta_{1.1}$ C asks K for credentials for the server T. She may additionally request the issued ticket to support options $KOpts$ and the KAS to use an encryption method e.

Rule $\delta_{1.2}$ allows C to process the `KRB_AS_REP` message which K sends in response to her initial request. The expected form of this reply is exactly like that in the abstract formalization, with the addition of the $TFlags$ field in the part encrypted under k_C. This field contains the $KOpts$ which were both requested by C and granted by K; we do not explicitly consider any specific $TFlags$ in our formalization.

Rule $\delta_{1.2'}$ shows C's error processing of a generic error message, formalized by C storing relevant information in the memory predicate $ASError_C$. The $ErrorCode$ describes the reason why the `KRB_AS_REQ` failed; see Sec. 8.3 of [10] for a complete and current list of possible $ErrorCode$s.

The actions of the KAS K are formalized in Fig. 6. Rule $\delta_{2.1}$ is similar to rule $\alpha_{2.1}$, except K also checks the validity of $KOpts$ and e. We represent the process of determining which $KOpts$ are to be granted and set in $TFlags$ by the $SetAuthFlags$ external process.

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

9.2 The Ticket-Granting Exchange

The client's actions in this exchange are formalized in Fig. 7. Rule $\delta_{3.1}$ fires like rule $\alpha_{3.1}$, but C also includes the current time $t_{C,Treq}$ (obtained via the external process $Clock_C$), a message digest, keyed by $AKey$, of the unencrypted portion of the KRB_TGS_REQ, along with $TOpts$ and the etype e. Recall that the only one of many possible TOpts that we will explicitly consider is a request for an ANONYMOUS service ticket.

The client's second rule, $\delta_{3.2}$, allows her to read from the network a KRB_TGS_REP message that matches her request to T. It is exactly analogous to rule $\delta_{1.2}$. Note that if ANONYMOUS is one of the TFlags sent by T, then instead of the name C the dummy identifier USER will appear in the ticket Y in rule $\delta_{3.2}$.

Rule $\delta_{3.2'}$ allows for C's processing of error messages in exactly the same manner as rule $\delta_{1.2'}$. Note that the names C and T in the error message must match those in the role state predicate L.

Fig. 8 shows the actions of the TGS T in the ticket granting exchange. If C's request is valid (as determined by the $Valid$ external process) and the checksum ck sent by C is of the correct form, then T uses the $SetServFlags$ external process to determine which SFlags will be set, and fires rule $\delta_{4.1}$. We again note that if ANONYMOUS is requested by C in the $TOpts$ field and granted by T (that is, it appears in the $SFlags$ field sent in the KRB_TGS_REP) then T will send the generic name USER in place of C in the ticket.

Rule $\delta_{4.1'}$ exactly parallels K's sending of error messages in Sec. 9.1. Note that the $Invalid$ external process encompasses the possibility that the checksum ck sent in the authenticator is not the expected message digest. Of course, the $Invalid$ predicate can indicate rejection of a request from C for a number of other reasons.

9.3 The Client/Server Authentication Exchange

Since the message flow in this exchange depends upon whether the MUTUAL_REQUIRED or MUTUAL_NON_REQUIRED SOpt is requested in the KRB_AP_REQ, we have subdivided this exchange into these two cases. The MUTUAL_REQUIRED /MUTUAL_NON_REQUIRED option is the only one of the possible SOpts we explicitly consider in this exchange.

9.3.1 Mutual Authentication Required

Fig. 9 shows the role of the client C in this exchange. Rule $\delta_{5.1}$ is the obvious extension of rule $\alpha_{5.1}$; here we use the constraint $Mutual$ to indicate that the MUTUAL_REQUIRED option is being requested by C. Note that if the ticket Y stored in the $Service_C(-)$ predicate is ANONYMOUS (indicated by the presence of ANONYMOUS in the $SFlags$ field, also stored in $Service_C(-)$), then C will send the generic identifier USER in place of her name in the authenticator sent in rule $\delta_{5.1}$. We denote the message digest here by $[\ldots]_{SKey}$ because [10] specifies that this optional checksum is "application specific."

Rule $\delta_{5.2}$ is virtually identical to rule $\alpha_{5.2}$, the only difference being the presence of $SOpts$ in the role state predicate L.

Rule $\delta_{5.2'}$ models C's handling of error messages in exactly the same way as in the previous exchanges with KAS and TGS, with C storing relevant information sent in the error message in the $APError$ memory predicate.

The server's role in this exchange is given in Fig. 10. Rule $\delta_{6.1}$ formalizes S's reply (because mutual authentication is requested by C, as indicated again by the $Mutual$ constraint) to a valid KRB_AP_REQ. Here S checks that the message digest ck sent by C is the appropriate application specific message digest, just as T does in the Ticket Granting Exchange.

Rule $\delta_{6.1'}$ models S sending error messages just as in the Ticket Granting Exchange.

9.3.2 Mutual Authentication Not Required

This exchange (shown in Figs. 11 and 12) is very similar to the previous one, except C does not require a response from S (denoted by the $NoMutual(SOpts)$ constraint). Upon sending the initial application request to S, C stores $S, SKey$ in the $DoneNoMut$ predicate, in rule $\delta_{5'.1}$. The remarks above about C's use of an ANONYMOUS service ticket hold here as well.

When S receives the request he sends no response, provided the request is valid (as determined by the external process $Valid$). In this case S fires rule $\delta_{6'.1}$, storing $C, SKey, t_{C,Sreq}$ in the Mem_S predicate.

The handling of error messages for both C and S (rules $\delta_{5'.2'}$ and $\delta_{6'.1'}$) is virtually identical to the Mutual Authentication Required case.

10 Detailed Level Intruder Formalization

The admissible Dolev-Yao intruder actions are updated to reflect the above changes 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. There is no disassembling rule for message digests since (cryptographic) hashing does not permit recovering a message. However, the intruder can construct

$$\left(\begin{array}{llcl}
\multicolumn{4}{l}{\exists L : \mathsf{client}^{(C)} \times \mathsf{SOpt} \times \mathsf{server}^{(S)} \times \mathsf{shK}\ C\ S \times \mathsf{time} \times \mathsf{msg}.} \\
\begin{array}{l} \forall S : \mathsf{server}\ . \\ \forall SKey : \mathsf{shK}\ C\ S. \\ \forall t_{C,Sreq} : \mathsf{time}\ . \\ \forall Y : \mathsf{msg}\ . \\ \forall SFlags : \mathsf{SFlag}\ . \\ \forall SOpts : \mathsf{SOpt}\ . \end{array} & \begin{array}{l} Service_C(Y, SFlags, S, SKey) \\ NoMutual(SOpts) \\ Clock_C(t_{C,Sreq}) \end{array} & \xrightarrow{\delta_{5'.1}} & \begin{array}{l} \mathsf{N}(SOpts, Y, \{C, [\ldots]_{SKey}, t_{C,Sreq}\}_{SKey}) \\ Service_C(Y, SFlags, S, SKey,) \\ L(C, SOpts, S, SKey, t_{C,Sreq}, Y) \\ DoneNoMut_C(S, SKey) \end{array} \\
\begin{array}{l} \forall \ldots\ . \\ \forall ErrorCode : \mathsf{msg}. \\ \forall t_{S,err} : \mathsf{time}\ . \end{array} & \begin{array}{l} \mathsf{N}(\mathtt{KRB\_ERROR}, t_{C,Sreq}, \\ \quad t_{S,err}, ErrorCode, C, S) \\ L(C, SOpts, S, SKey, t_{C,Sreq}, Y) \end{array} & \xrightarrow{\delta_{5'.2'}} & \begin{array}{l} APError_C(S, t_{S,err}, \\ \quad ErrorCode) \end{array}
\end{array}\right)^{\forall C:\mathsf{client}}$$

Figure 11. The client's role in the Client/Server Exchange without mutual authentication.

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

Figure 12. The end server's role in the Client/Server Exchange without mutual authentication.

a message digest as long as he knows the proper key.

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

The updates to the generation rules are limited to allowing the intruder to choose the encryption type of any session key he may generate. None of the new data types introduced at this level of detail can be generated by the intruder (or any other principal). Therefore there are no additional data generation rules with respect to what we presented in Section 6.2.

Data access rules are subject to similar changes. However, the new data types, encryption types, options and flags, shall be treated 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.

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

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

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

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 abstract specification of the intruder remains inaccessible.

11 Detailed Level Anomalous Behavior

We now discuss two anomalies that we have discovered in our detailed formalization of Kerberos 5 which are not present in the abstract level. Both take advantage of additional fields not included in our abstract formalization. The first anomaly we discuss in an analogue of that in Sec. 7, making use of new options available here. The second anomaly we discuss is completely unrelated to the others, as it exploits the encryption type field e send by C in the KRB_AS_REQ.

We note that the anomaly discussed in Sec. 7 also appears in this formalization. It is not prevented by the checksum sent by C in the KRB_TGS_REQ message as we have formalized it (taken over $TOpts, C, S, n_2, e$). The anomaly appears to be fixed if the ticket-granting ticket (or what C

thinks is the TGT) is also included in the above checksum, although this remains to be proved.

11.1 Anonymous Ticket Switch Anomaly

In this scenario we make no explicit assumptions about the application specific checksums C sends in the KRB_AP_REQ messages, only that they agree with the local policy of the server S. It begins with a normal AS exchange, after which C has a ticket X for use with a TGS, the name T of the TGS, and the corresponding shK $C\ T\ AKey$ all stored in the memory predicate $Auth_C$.

Now C desires two tickets from the TGS T for the same server S, one a NON-ANONYMOUS and the other an ANONYMOUS ticket. She fires rule $\delta_{3.1}$ twice and T responds as expected by firing rule $\delta_{4.1}$ twice, sending C, Y_1, Z_1 and C, Y_2, Z_2 (where Y_1 is the non-anonymous ticket $\{SKey_1, C\}_{k_s}$, $Z_1 = \{SKey_1, n_1, S\}_{AKey}$, Y_2 is the anonymous ticket $\{\texttt{ANONYMOUS}, SKey_2, \texttt{USER}\}_{k_S}$, and $Z_2 = \{SKey_2, n_2, \texttt{ANONYMOUS}, S\}_{AKey}$). At this point the intruder I intervenes (firing rules INT, INT, DCM, DCM, CMP, CMP, TRN, and TRN) and switches the tickets in the two KRB_TGS_REP messages, putting C, Y_1, Z_2 and C, Y_2, Z_1 on the network. C has no reason to suspect that anything is wrong with these two altered messages, so she fires rule $\delta_{3.2}$ twice, storing $Y_1, \texttt{ANONYMOUS}, S, SKey_2$ in a memory predicate $Service_{C,1}$ and $Y_2, S, SKey_1$ in $Service_{C,2}$. At this point C wrongly thinks that Y_1 is an ANONYMOUS ticket and Y_2 in an ordinary ticket, although she does correctly consider $SKey_2$ to be an "ANONYMOUS key", and $SKey_1$ a "NON-ANONYMOUS key".

C then contacts S using the tickets obtained in the Ticket Granting Exchange without requiring mutual authentication from S. Thus she fires rule $\delta_{5'.1}$ twice, putting the messages MUTUAL_NON_REQUIRED, $Y_1, Auth_2$ and MUTUAL_NON_REQUIRED, $Y_2, Auth_1$ on the network (where $Auth_1 = \{C, ck_{S,2}, t_{C,Sreq2}\}_{SKey_1}$ and $Auth_2 = \{C, ck_{S,1}, t_{C,Sreq1}\}_{SKey_2}$). These requests will not be processed unless the intruder I intervenes again since the key inside of Y_1 will not decrypt the authenticator $Auth_2$, and the key inside Y_2 will not decrypt $Auth_1$. Thus I fires some rules (namely INT, INT, DCM, DCM, CMP, CMP, TRN, and TRN again), and takes the pair of messages MUTUAL_NON_REQUIRED, $Y_1, Auth_2$ and MUTUAL_NON_REQUIRED, $Y_2, Auth_1$ from the network and replaces them with MUTUAL_NON_REQUIRED, $Y_1, Auth_1$ and MUTUAL_NON_REQUIRED, $Y_2, Auth_1$; note that both use $Auth_1$. Then S receives the first of these messages, decrypts Y_1 and finds $SKey_1, C$ inside, uses this key to decrypt $Auth_1$ and finds $C, ck_{S,2}, t_{C,Sreq2}$ inside, then fires rule $\delta_{6'.1}$ and stores $C, SKey_1, t_{C,Sreq2}$ in the memory predicate Mem_S. S then decrypts Y_2 and finds ANONYMOUS, $SKey_2$, USER inside and attempts to decrypt $Auth_1$ using $SKey_2$, but is unable to do so. In our present formalization, S does not do anything in response to an unreadable authenticator. However, a natural analogue of rule $\delta_{6'.1'}$ allows S to put the error message KRB_ERROR, $t_{S,err}, ErrorCode$, USER, S on the network (omitting the unreadable authenticator timestamp $t_{C,Sreq2}$). I intercepts this message (using INT), replaces USER with C (using DMC and CMP), and transmits this modified error message to C (using TRN). C processes this error and thus believes 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 Sec. 7. This anomaly seems to be avoided if the checksums in the KRB_AP_REQ messages are also taken over the service ticket (so that the server S would be aware of any ticket switches), but this is also yet to be proved.

11.2 Encryption Type Anomaly

This anomaly involves the following scenario. Since different encryption methods can require different keys which are not interchangeable, suppose that C has a database key $k_{C,1}$ which is used for encryption of type $e1$ and which has been compromised in some way, and a second database key $k_{C,2}$ for encryption of type $e2$ which is still secure. Further suppose that the loss of $k_{C,1}$ has not yet been reported to the KAS. Now C wants to obtain a ticket granting ticket for use with a TGS T, and she wants the response encrypted using $e2$ so that the uncompromised key $k_{C,2}$ will be used. Thus she fires rule $\delta_{1.1}$, putting $C, T, n, e2$ on the network (here we assume that $KOpts$ is empty and thus ignore it). I intervenes, firing rules INT, DMC, and CMP, and replaces the message C sent with $C, T, n, e1$. K sees this altered request as a legitimate message and fires rule $\delta_{2.1}$, placing $C, X, \{AKey, n, T\}^{e1}_{k_{C,1}}$ on the network, where X is the ticket granting ticket $\{AKey, C\}_{k_T}$ for use with T. I intercepts the KRB_AS_REP with rule INT, storing it in a memory predicate. Finally, I can fire rule DCC' to decrypt the contents of $\{AKey, n, T\}^{e1}_{k_{C,1}}$, obtaining $AKey$. This allows the intruder to masquerade as C not only using the compromised key $k_{C,1}$, but also when C attempts to use the uncompromised key $k_{C,2}$ by requesting the "safe" encryption type $e2$.

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 [10] as optional), keyed with a dbK C, as the following scenario shows. C puts $C, T, n, e2, [C, T, n, e2]^{e2}_{k_{C,2}}$ on the network and I intervenes, replacing it with $C, T, n, e1, [C, T, n, e1]^{e1}_{k_{C,1}}$ (which I can do, since the hash is public and she knows $k_{C,1}$ and $e1$). Then the action continues as above, with I gaining knowledge of $AKey$.

12 Conclusions and Future Work

In this paper, we have specified Kerberos 5 at two levels of detail using the MultiSet Rewriting (MSR) framework. The abstract formalization exhibits anomalous behavior not seen in Kerberos 4, and we prove an appropriately weakened authentication property that still holds. We do this using notions of rank and corank which were inspired by the work of Schneider [11]. The detailed formalization is closer to the full protocol as given in [8, 10] and exhibits both the anomaly seen at the abstract level and additional anomalous behavior related to the added detail. It appears that some of these anomalies may be resolved through the use of cryptographic checksums, but we did not yet prove this formally.

MSR proved to be an adequate language in this effort and only minor adaptations were needed. This work also gave some preliminary insight into possible approaches to reasoning on an MSR specification.

This work is part of a larger project aimed at formalizing of Kerberos 5 at various levels of details, with the ultimate goal of capturing all the facets of this complex protocol suite [10]. Future additions include more timestamps and temporal checks, explicit consideration of all options specified in [8] (in particular renewable or postdatable tickets), and the formalization of the other available subprotocols. As we proceed, we intend to state and prove security properties in the spirit of [11, 12], in particular additional authentication properties and confidentiality properties which are implicit in our proofs of authentication. We also hope to consider, and precisely state in our eventual full report on this project, the connections between our different formalizations of Kerberos 5; one connection we are investigating is the reuse of proofs from the abstract formalization as skeletons for proofs in the more detailed formalizations.

References

[1] G. Bella, *Inductive Verification of Cryptographic Protocols*, Ph.D. thesis, University of Cambridge, March 2000.

[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.

[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, pp. 361–375.

[4] ______, *Mechanising BAN Kerberos by the Inductive Method*, Proc. of CAV98 – Tenth International Conference on Computer Aided Verification, 1998.

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

[6] I. Cervesato, *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.

[7] 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.

[8] J. Kohl and C. Neuman, *The Kerberos Network Authentication Service (V5)*, September 1993.

[9] 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.

[10] C. Neuman, J. Kohl, T. Ts'o, K. Raeburn, and T. Yu, *The Kerberos Network Authentication Service (V5)*, November 20 2001.

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

[12] T. Y. C. Woo and S. S. Lam, *A Semantic Model for Authentication Protocols*, Proc. of the IEEE Symposium on Security and Privacy, 1993, pp. 178–194.