Carnegie Mellon University
Research Showcase @ CMU

Computer Science Department | School of Computer Science

1-2005

Specifying Kerberos 5 Cross-Realm Authentication

Iliano Cervesato
Tulane University of Louisiana

A D. Jaggard
Tulane University of Louisiana

A Scedrov
University of Pennsylvania

C Walstad
University of Pennsylvania

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

Recommended Citation
.

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

Specifying Kerberos 5 Cross-Realm Authentication*

I. Cervesato, A.D. Jaggard
Tulane University
{iliano| adj}@math.tulane.edu

A. Scedrov, C. Walstad
University of Pennsylvania
{scedrov@cis|cwalstad@seas}.upenn.edu

ABSTRACT

Cross-realm authentication is a useful and interesting component of Kerberos aimed at enabling secure access to services astride organizational boundaries. We present a formalization of Kerberos 5 cross-realm authentication in MSR, a specification language based on multiset rewriting. We also adapt the Dolev-Yao intruder model to the cross-realm setting and prove an important property for a critical field in a cross-realm ticket. Finally, we document several failures of authentication and confidentiality in the presence of compromised intermediate realms. Although the current Kerberos specifications disclaim responsibility for these vulnerabilities, the associated security implications must be highlighted for system administrators to decide whether to adopt this technology and to aid designers with future development.

1. INTRODUCTION

Kerberos [17, 21] is a successful, widely deployed single-login protocol that is designed to strongly authenticate a client to services it may require. Kerberos has been adopted by many large companies, universities, and other organizations, and Microsoft has included a largely compatible implementation of Kerberos 5 into all its operating systems starting with Windows 2000 [13]. The core intra-realm protocol, in which clients and servers are part of the same administrative unit, or realm, has been extensively studied [2, 3, 6, 7, 18]: in particular, [2, 3, 4] verified a high-level specification of Kerberos 4 using the Isabelle theorem prover, and [7] proved confidentiality and authentication properties for a detailed formalization of the more recent Kerberos 5. While all this work has focused on intra-realm communication, Kerberos 5 also provides support for cross-realm authentication, where clients and servers straddle organizational boundaries, possibly with intermediate realms in between. While cross-realm authentication has been studied in the past [5, 14], to the best of our knowledge, it has never been formalized nor has its security been formally analyzed or verified. Moreover, we are not aware of any recent research on this topic.

In the intra-realm setting, the main objective of formal verification is to ascertain that Kerberos realizes stated confidentiality and authentication goals. An unavoidable assumption is that the administrative principals of that realm, here abstractly called the KDC (Key Distribution Center), behave honestly: a dishonest or compromised KDC trivially invalidates any authentication or confidentiality expectations. In the cross-realm case, a multitude of intermediate KDCs may be involved in the process of authenticating a client to a remote server. When configuring Kerberos for cross-realm operation, a system administrator must consider what will happen if a remote realm is compromised. What are the security implications associated with a compromised remote KDC, and how can that KDC affect clients and servers in the rest of the Kerberized network? These questions are especially important because the local system administrator has no control over other realms, cannot ensure that proper and timely maintenance is performed, and does not have a say on which realms are added to the foreign realm's trusted list. The Kerberos specification documents [17, 21] do not provide any guarantees about authentication, confidentiality, or any security properties in the case that intermediate realms are compromised. In fact, the default setting of all distributions is not to trust any foreign entity and each candidate foreign principal must be explicitly allowed by the system administrator. Therefore, the first objective of formal verification in the cross-realm case is to assess the vulnerability of a constellation of realms to the compromise of some of its members. Only then can more traditional confidentiality and authentication properties be ascertained.

This paper reports the preliminary results of a formal investigation of cross-realm authentication in Kerberos 5. We

*Cervesato was partially supported by NRL under contract N00173-00-C-2086 and by ONR under contract number N000149910150. This research was conducted while this author was employed by ITT Industries and was visiting Princeton University. Jaggard was partially supported by NSF grant DMS-0239996. Scedrov and Walstad were partially supported by the ONR CIP/SW URI "Software Quality and Infrastructure Protection for Diffuse Computing" through ONR Grant N00014-01-1-0795. Additional support for Scedrov and Walstad by the ONR CIP/SW URI "Trustworthy Infrastructure, Mechanisms, and Experimentation for Diffuse Computing" through ONR Grant N00014-04-1-0725 and by the NSF grants CCR-0098096 and CNS-0429689.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright 200X ACM X-XXXXX-XX-X/XX/XX ...$5.00.

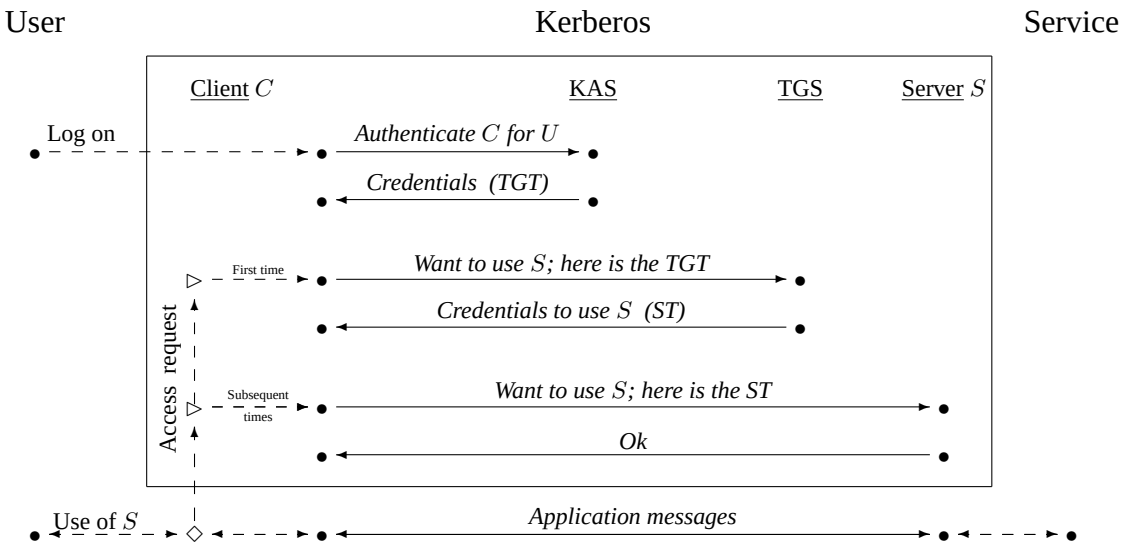

Figure 1: An Overview of Kerberos Authentication

formalize the message exchange and introduce a cross-realm threat model that extends the standard Dolev-Yao intruder with the implications of compromising intermediate KDCs. We state and prove a property that characterizes the possible ways a request for service may have been forged; as a corollary, we obtain minimum confidentiality requirements to support authentication. This same model is used to outline several attacks that a compromised remote KDC may mount. For example, an intruder getting a hold of a remote realm is capable of masquerading as any known client in order to gain access to any remote Kerberized service that this client is authorized to use. It is crucial that the extent of the damage that a compromised KDC may inflict be documented precisely, so that system administrators can make informed decisions in regard to trusting other realms. We also hope this work will sensitize the development of cross-realm authentication to making it less susceptible to compromised nodes.

We conduct our analysis using MSR [9, 12], a powerful yet flexible framework for the specification of complex cryptographic protocols, possibly structured as a collection of coordinated subprotocols. 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 short-term keys). MSR has been successfully used in the past to analyze a detailed model of the intra-realm operations of Kerberos 5 [6, 7].

We recall Kerberos 5 intra- and cross-realm authentication in Sections 2 and 3, respectively, while MSR is briefly reviewed in Section 4. The formalization of the cross-realm operations is given in Sections 5 and 6, and the corresponding attacker model is introduced in Section 7. Section 8 proves properties pertaining to cross-realm authentication in Kerberos 5, while Section 9 documents several vulnerabilities when the corresponding requirements are not met. Section 10 reports on related work. Finally, Section 11 outlines directions for future research.

2. KERBEROS 5 BASICS

Networked computer systems put a multitude of shared resources at a user's fingertips: remote hosts, file servers, printers, and many other networked services are readily available without leaving one's desk. Authentication and other security mechanisms need to be in place so that this convenience is not abused, in particular nowadays where connections to the Internet provide dangerous backdoors to one's organization. The natural solution to this, requiring users to authenticate to each service they need (for example using a password) is not only inconvenient, but also insecure in practice as humans are notoriously poor at juggling a multitude of strong passwords.

The Kerberos protocol [20] was designed to allow a legitimate user to log on to her terminal once a day (typically) and then transparently access all the networked resources she needed for the rest of that day. Each time she wanted to retrieve a file from a remote server, for example, Kerberos would securely handle the required authentication behind the scene, without any user intervention. See [13] for a gentle yet comprehensive introduction to Kerberos.

We will now review how Kerberos achieves secure authentication based on a single logon. We will concentrate on the most recent version of this protocol, Kerberos 5 [21], and for the time being on situations where all the authentications take place within the same administrative domain (or realm).

2.1 Principals

In the above informal description, we have already encountered three of the agents, or principals, that comprise a typical Kerberos exchange: the human *user* at her terminal, the *client* process that accepts the user's password and transparently handles the authentication aspect of each of her requests on her behalf, and the requested services, or *servers* in Kerberos terminology. Kerberos relies on two additional administrative

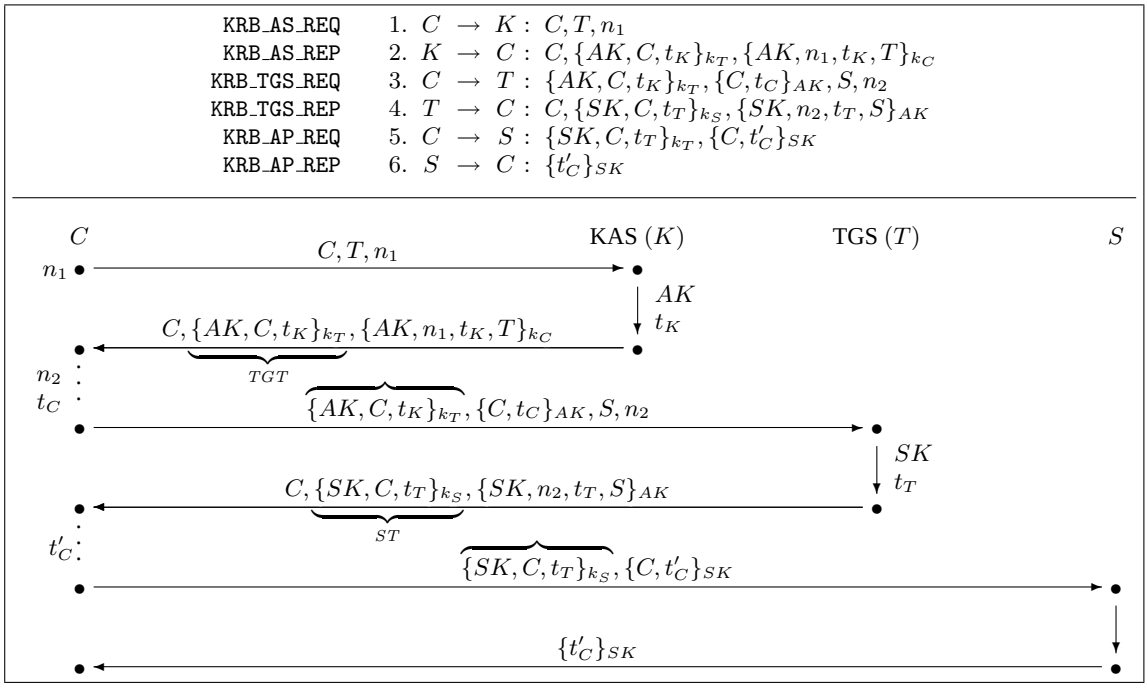

Figure 2: Intra-Realm Message Exchange

agents together known as the *KDC*: the Kerberos Authentication Server (*KAS*) who authenticates the user and provides the corresponding client with credentials to use the network for the day, and the Ticket Granting Server (*TGS*) who authenticates the client to each requested server based on those credentials.

The high-level picture is given in Figure 1. The top of the figure represents the daily authentication to Kerberos: as the user (U) logs on, the KAS authenticates the client process representing her and provides credentials to use the system for that day. These credentials from the KAS are called the Ticket Granting Ticket, abbreviated *TGT*. Whenever the user wants to use a networked service, the client on her behalf will seek authentication to the process S managing this service. This is done in two steps: the first time U attempts to access S, C presents the TGT from the KAS to the ticket granting server (TGS) who will in turn provide credentials for S. These credentials are called the Service Ticket or *ST*. Every subsequent time U wants access to this particular service, C forwards the service ticket to S, without involving the TGS. The line at the bottom of the figure represents the actual use of the desired service: this is all the user sees as her client process handles the authentication overhead.

The above mode of interaction is typical of a single organization, or *realm* in Kerberos terminology. Each realm is regulated by a single KDC (although there may be synchronized replicas for performance and fault tolerance reasons). Within a realm, there will be in general multiple clients and multiple servers. *Intra-realm authentication*, as this modality is known, is widely deployed and has been extensively studied [2, 6, 18]. Kerberos also supports *cross-realm authentication*, a scheme by which a client in a realm R_1 can access a service in a different realm R_n. We will dedicate the rest of this paper to exploring how Kerberos achieves cross-realm authentication. Before doing so, we recall how the basic intra-realm protocol works.

2.2 Intra-Realm Message Exchanges

In this section, we focus on the messages exchanged during a typical intra-realm authentication session between a client C and a server S, as sketched in the boxed part of Figure 1. We provide sufficient detail to support their formal specification in the next section. Notice however that Kerberos is far more complex than the abstract view given here: we refer the reader to [21] for a complete description, and to [6] for a formalization of an intermediate level of detail. In the following, we assume the reader familiar with the traditional concepts pertaining to security protocols, in particular the notions of nonce, shared key encryption, and timestamps.

The fleshed out version of the Kerberos 5 exchanges is given in Figure 2: the top part relies on the traditional "Alice-and-Bob" notation, with the standard name [21] for each message given on the left. The bottom part takes a clearer two-dimensional view. We will now describe each of the three roundtrips between a client (C) and the KAS (K for brevity), the TGS (T for short), and a server (S), respectively.

Authentication Service Exchange ($C \Leftrightarrow K$): This exchange takes place when the user first logs on to a Kerberized network. The client process C generates a nonce n_1 and sends it to the KAS together with his own name, C, which indirectly identifies the user, and the name of the TGS (officially "`krbtgt`", here abbreviated as T).

Upon recognizing C (and indirectly U), the KAS replies

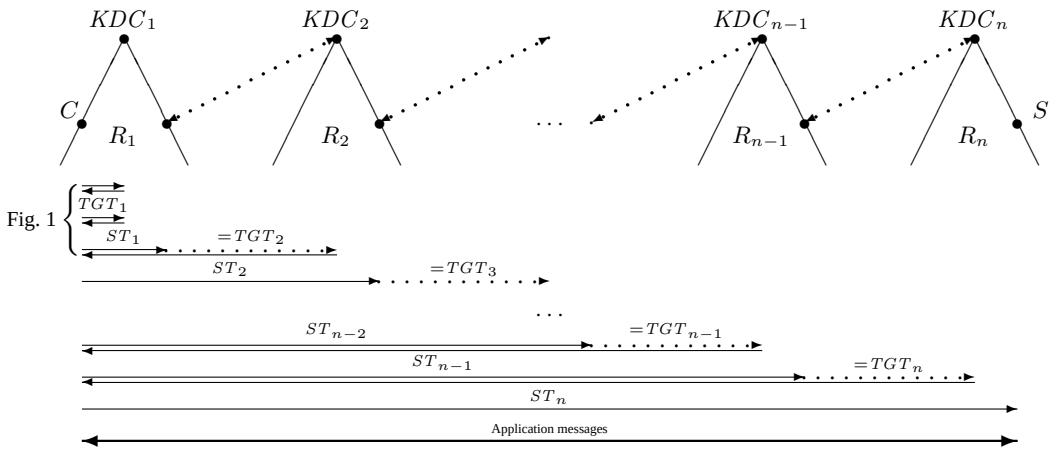

Figure 3: Schematic Cross-Realm Authentication

with a message containing two encrypted components: the ticket granting ticket (TGT) $\{AK, C, t_K\}_{k_T}$ that is cached by C and will be used to obtain service tickets for the rest of the day, and $\{AK, n_1, t_K, T\}_{k_C}$ with which the KAS informs C of the parameters of the ticket. The TGT is meant for the TGS and is encrypted with the long-term key k_T that the KAS shares with the TGS. It contains a freshly generated *authentication key* AK and a timestamp t_K in addition to C's name (and many other pieces of information, abstracted away here). The key k_C used to encrypt the second component is a long-term secret between C and the KAS derived from the user's password. AK will be used in every subsequent communication with the TGS, sparing the more vulnerable k_C. The timestamp t_K will assure the TGS and C that this ticket was issued recently, as all Kerberos principals have loosely synchronized clocks. The nonce n_1 in the second component binds this response to C's original request.

Ticket Granting Exchange ($C \Leftrightarrow T$): This exchange takes place the first time U tries to access a service S. In the outgoing message, C transmits the cached TGT and S's name together with a freshly generated nonce n_2 (again to bind this request and the subsequent response), and the *authenticator* $\{C, t_C\}_{AK}$, where t_C is a timestamp. The authenticator proves to T that C indeed knows the authentication key AK.

Upon authenticating C and verifying that he is allowed to use S, the TGS sends a response with the same structure as the second message above except the service ticket $\{SK, C, t_T\}_{k_S}$ is now encrypted with the long-term key shared between the KDC and S, and it contains a freshly generated *service key* SK, C's name, and a timestamp t_T. The other encrypted component is as in the second message above, but now encrypted with the authentication key AK. C caches the service ticket.

Client/Server Exchange ($C \Leftrightarrow S$): This exchange takes place each time the client initiates a new session with the server S. With a service ticket in hand, C simply contacts S with this ticket and an authenticator similar to the one described above.

The response from S is optional as the subsequent application exchanges may subsume it. When present, it provides assurance to C that S is alive, for example by returning the timestamp t'_C that C included in her request, encrypted with the service key.

The actual application exchanges, by which S provides the service requested by C, are not properly part of Kerberos (whose mandate is limited to authentication). However, Kerberos provides message formats to attain specified security goals. Their discussion is beyond the scope of this document since they may or may not be used by the application server.

3. CROSS-REALM AUTHENTICATION

Kerberos supports authentication across organizational boundaries by permitting clients and servers to reside on different realms. A realm consists a group of clients, a KDC, and application servers, as seen in Section 2. For example, the Graphics group in the Computer Science Department of University A may organize as an independent realm R_{Gr} with its own users, services and administrators. Similarly, the CS department may define a Kerberos realm R_{CS} to allow CS members to access common resources, and the University may in turn have a realm R_A of its own to operate university-wide resources such as printers in dormitories. Cross-realm authentication enables a student at her workstation in the Graphics lab to transparently access a file on the common CS server, and even to seamlessly print it on a printer in her dormitory. Without cross-realm authentication this student would need a separate account in each realm, log into each of them, and explicitly transfer files from account to account in order to achieve the same goals. This is impractical, not scalable, and less secure as several passwords would be needed, one for each realm. While this form of hierarchical organization [5] of realms is recommended when enabling cross-realm authentication in Kerberos, it is by no means mandated, as, for example, the Graphics realm of university A may establish a cross-realm partnership with the Graphics realm of another university with which it collaborates.

In the simplest case, the cross-realm authentication of a client C in realm R_1 to a server S in R_n is accomplished by registering the KDC of R_n as a special server in R_1 and using a variant of the intra-realm protocol to first acquire a TGT for C in R_1 (as always) and then a service ticket for R_n's KDC seen as a local service in R_1. This service ticket has the same

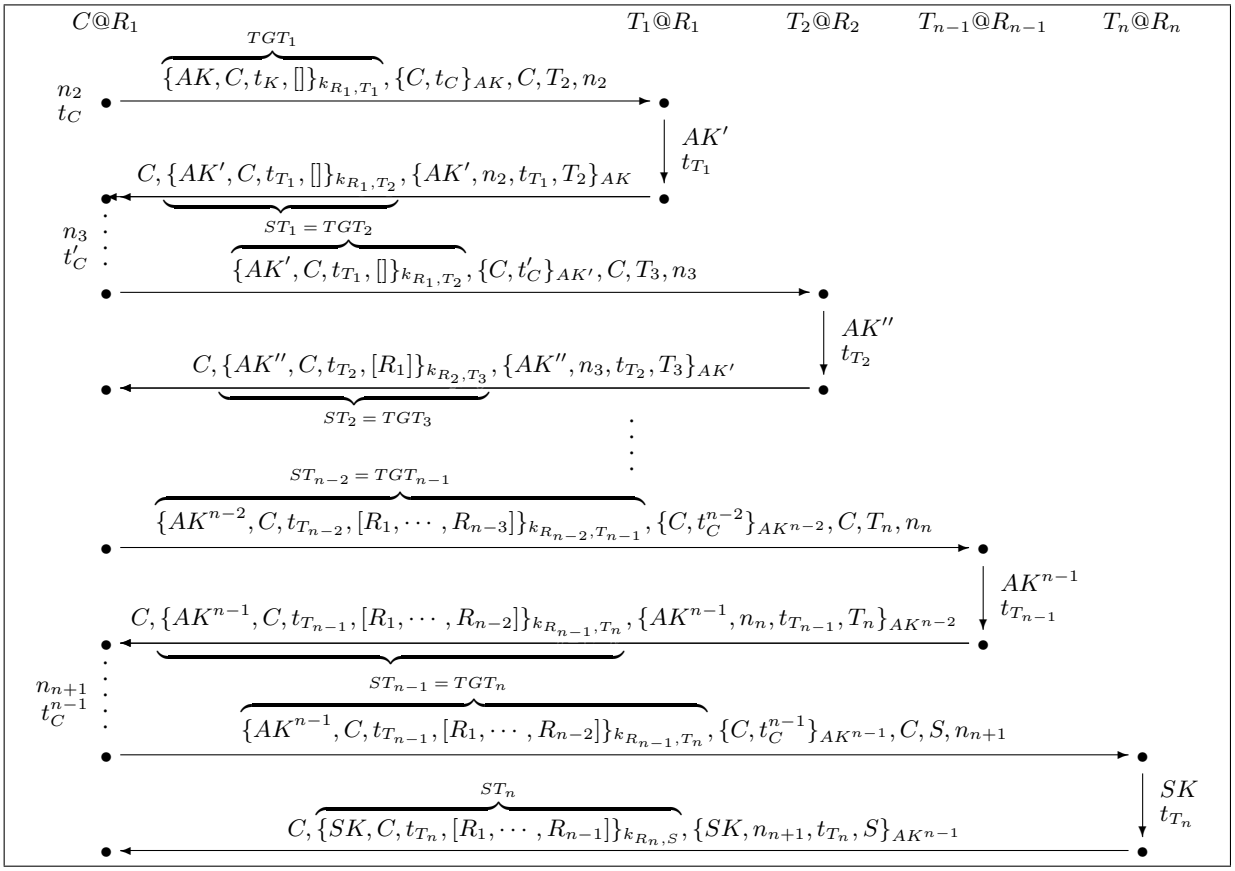

Figure 4: Messages in Cross-Realm Authentication

format as a TGT for C in R_n, and as such it is handed to the KDC of R_n to obtain a service ticket for accessing S. The key used by R_1's KDC to encrypt the ticket for the special service corresponding to R_n's KDC is called a *cross-realm key*. This is all Kerberos 4 allows. In Kerberos 5, C's access to S may require traversing intermediate realms $R_2, \ldots, R_{n-1}$ if there is no cross-realm key between R_1 and R_n, but R_1 has such a partnership with R_2, R_2 with R_3, etc. up to R_n. C then needs to obtain a TGT for each of these realms in succession before accessing S. The list of traversed KDC's $[R_1, \ldots, R_n]$ is called the *authentication path* of C's access to S. This high-level description is schematically represented in Figure 3.

A message-oriented view of the cross-realm authentication exchange highlighted in Figure 3 is given in Figure 4. Each of the four roundtrips in this figure corresponds to a ticket granting exchange, as described in Figure 2: the service ticket returned by each TGS is forwarded to the next TGS who interprets it as a TGT. The messages in Figure 4 differ in two points from the intra-realm abstraction presented in Figure 2. First, we explicitly annotate each database key with the realm it is defined in. Therefore, the local key between the KAS and the TGS of realm R_1 is written k_{R_1,T_1} rather than k_{T_1}, but more importantly the cross-realm key of the TGS of R_{i+1} in realm R_i can be written as $k_{R_i,T_{i+1}}$. Second, our formalization of the ticket is extended with an additional field known as `TRANSITED`. This field implements the partial authentication path of C's request on its way to S: whenever a ticket leaves realm R_i, it lists all the realms $[R_1, \ldots, R_{i-1}]$ that have been previously traversed; R_i itself will be added by the KDC of R_{i+1} (in this way, R_i's KDC cannot hide the fact that R_i appeared on the authentication path—although the KDC of R_{i+1} may). The `TRANSITED` field will play an important role in our analysis of Kerberos cross-realm authentication in Section 8 as it is the only information available the target KDC to establish whether all previously traversed realms are trustworthy, and therefore to decide whether to grant access to the requested server.

The authentication process described in Figure 4 assumes that C knows the authentication path he wants to follow since C specifies the (TGS of the) "next" realm with each successive request. This is only one of the options available in Kerberos, and not the most common. In general, C will ask her local TGS for a service that could be the special server corresponding to the "next" realm, could be the TGS of S's realm, or could be the TGS of any realm in between. Whenever R_1 does not have a cross-realm key with the realm R_i specified by C, his local TGS will plot a route to this realm. This can be accomplished through configuration files, or using a DNS server

in case the hierarchy of Kerberos realms mirrors the standard DNS hierarchy. See [14] for a discussion of the underlying algorithm. In this case, C's TGS will return a ticket for the TGS of a realm R_2, "closer" to the desired realm, with which it does share a cross-realm key. C will then contact this TGS with this ticket and her original request, until he reaches R_i.

4. OVERVIEW OF MSR

MSR is a formal language for describing security protocols [9, 12]. More precisely, it is a powerful yet flexible framework for the specification of complex cryptographic protocols, possibly structured as a collection of coordinated subprotocols. 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 short-term keys). We will now describe its basic syntactic features, demonstrate them with the specification of Kerberos intra-realm authentication, and outline its execution model. This excerpt will be completed and extended in Sections 5 and 6.

MSR represents network messages and their components as first-order terms (for simplicity, we use the same syntax as in the informal descriptions above). Therefore, the TGT $\{AK, C, t_K\}_{k_T}$ above is seen as a term obtained by applying the binary encryption symbol $\{\_\}\_$ to the constant k_T and the subterm (AK, C, t_K). In turn, this subterm results from applying the binary message concatenation symbol (here the infix "$\_,\_$") twice.

Terms are classified by types, which describe their intended meaning and restrict the set of terms that can be legally constructed. For example, $\{\_\}\_$ accepts a key (type key) and a message (type msg), producing a msg: using a nonce as the key yields an ill-formed message. These objects are declared as:

$$\begin{array}{l}\mathsf{msg} : \mathsf{type}.\\ \mathsf{key} : \mathsf{type}.\\ \{\_\}\_ : \mathsf{key} \rightarrow \mathsf{msg} \rightarrow \mathsf{msg}.\end{array}$$

Nonces, principals, etc., often appear within messages. MSR handles this by relying on the notion of a subsort (written $<:$). The following declarations define the type of nonces (nonce) and the fact that every nonce can be used where a message is expected:

$$\mathsf{nonce} : \mathsf{type}. \qquad \mathsf{nonce} <: \mathsf{msg}.$$

While simple types such as msg adequately describe simple categories, MSR provides means to capture more structured information. For example, it can express the fact that sk_{37} is a short term key shared between a particular client c_3 and a particular server s_7 directly within sk_{37}'s type. This is achieved through the notion of dependent type. The following declarations formalize this example, and define the short-term keys (shK) of any principal as a particular type of key:

$$\begin{array}{l}\mathsf{shK} : \mathsf{principal} \rightarrow \mathsf{principal} \rightarrow \mathsf{type}.\\ \quad \forall A, B : \mathsf{principal}.\, \mathsf{shK}\ A\ B <: \mathsf{key}.\\ c_3 : \mathsf{principal}. \qquad s_7 : \mathsf{principal}. \qquad sk_{37} : \mathsf{shK}\ c_3\ s_7.\end{array}$$

We will come back to these declarations in Section 5 to show how they need to be updated to express cross-realm authentication.

The *state* of a protocol execution is determined by the network messages in transit, the local knowledge of each principal, and other similar data. MSR formalizes individual bits of information in a state by means of *facts* consisting of *predicate name* and one or more terms. For example, the network fact $\mathsf{N}(\{AK, C, t_K\}_{k_T})$ indicates that ticket $\{AK, C, t_K\}_{k_T}$ is present on the network. The network predicate N is declared as:

$$\mathsf{N} : \mathsf{msg} \rightarrow \mathsf{state}.$$

We will encounter many other predicates while formalizing Kerberos.

A protocol consists of a sequence of actions that transform the state. In MSR, this is modeled by the notion of *rule*: a description of the facts that an action removes from the current state and of the facts it replaces them with to produce the next state. For example, the rule describing the functionalities of the KAS in Figure 2 follows:

$$\begin{array}{l}\forall K : \mathsf{KAS} \quad \forall C : \mathsf{client} \quad \forall T : \mathsf{TGS} \quad \forall n_1 : \mathsf{nonce}\\ \forall k_C : \mathsf{dbK}\ C \quad \forall k_T : \mathsf{dbK}\ T \quad \forall t_K : \mathsf{time}.\\ \qquad \exists AK : \mathsf{shK}\ C\ T\\ \quad \Rightarrow \quad \left[\begin{array}{l}\mathsf{N}(C, \{AK, C, t_K\}_{k_T},\\ \qquad \{AK, n_1, t_K, T\}_{k_C})\end{array}\right]\\ \mathit{if} \quad \mathsf{Valid}_K(C, T), \mathsf{clock}_K(t_K)\end{array}$$

Rules are parametric, as evidenced by the leading string of typed universal quantifiers (here, all types are mnemonic, except perhaps dbK A which stands for the long-term key shared between principal A and the KDC): actual values need to be supplied before applying the rule. The middle portion describes the transformation performed by the rule: it replaces states containing a network fact of the form $\mathsf{N}(C, T, n_1)$ with states that contain $\mathsf{N}(C, \{AK, C, t_K\}_{k_T}, \{AK, n_1, t_K, T\}_{k_C})$ but are otherwise identical. The existential marker "$\exists AK$: shK $C\ T$" requires AK to be replaced with a newly generated symbol of type shK $C\ T$: this is how freshness requirements are modeled in MSR. The line starting with *if* lists additional facts that must be present in the state for the rule to be applicable: predicate Valid models a validity check performed by the KAS and clock looks up the local clock. Differently from the facts in the antecedent, they are consulted but not removed as a result of applying the rule. Note that facts are simply state elements manipulated by the rules — in particular predicates do not have a semantics of their own. For example, the policy specified by Valid may be described extensionally in the state. Instead, a traditional "tick rule" or the scheme in [16] can be used to update the value held in clock (although a simulator may rely on an external routine to update the clock value).

While this rule completely describes the behavior of the KAS, more than one rule may be needed to model the actions of a generic principal. This is the case of the client, even when focusing just on his exchange with the KAS. Coordinated rules describing the behavior of a principal are collected in a *role*. A role is just a sequence of rules, parameterized by the principal executing them (the *owner* of the role). The 2-rule role describing the client's view of the authentication

service exchange is as follows:

$$\forall C : \mathsf{client}$$
$$\left[\begin{array}{l} \forall T : \mathsf{TGS} \\ \qquad \cdot \;\Rightarrow \exists n_1 : \mathsf{nonce}. \begin{bmatrix} \mathsf{N}(C,T,n_1) \\ L(C,T,n_1) \end{bmatrix} \\ \\ \forall T : \mathsf{TGS} \quad \forall k_C : \mathsf{dbK}\ C \quad \forall AK : \mathsf{shK}\ C\ T \\ \forall X : \mathsf{msg} \quad \forall n_1 : \mathsf{nonce} \quad \forall t_K : \mathsf{time}. \\ \begin{bmatrix} L(C,T,n_1) \\ \mathsf{N}(C,X,\{AK,n_1,t_K,T\}_{k_C}) \end{bmatrix} \Rightarrow \mathsf{Auth}_C(X,T,AK) \end{array}\right]$$

In the first rule, we write "$\cdot$" for an antecedent containing no predicates. The predicate L ensures that the second rule can execute only after the first has fired. Finally, the client uses the predicate Auth to cache the ticket (here modeled as the inscrutable message X) and its parameters for subsequent ticket granting exchanges.

The rule shown earlier for the KAS is easily turned into a 1-rule role by putting the quantification "$\forall K : \mathsf{KAS}$" in the owner position. The complete specification of Kerberos 5 intra-realm specification at this level of detail requires four additional roles: one for the TGS, one for the end-server, and two for modeling the client's exchanges with each of these principals. For space reasons, we do not show these roles here: they follow the above pattern; moreover cross-realm variants will be presented in Section 7. The interested reader is referred to [6, 7] for a full specification and analysis of Kerberos 5.

5. TYPING

In this section, we complete the MSR declarations needed to specify Kerberos 5 and update them to cope with cross-realm authentication. We will now comment on the resulting signature, which is displayed in Figure 5.

When modeling intra-realm authentication in [6, 7] (partially reproduced in Section 4), we did not need to give a representation to administrative domains since everything happened within one realm. Clearly this needs to be upgraded when dealing with cross-realm authentication. We start by adding the type realm to represent a realm. Our next change is to associate each principal, be it a client, a server, a KAS or a TGS, with the realm it lives in. In MSR, the most convenient way to achieve this effect is to parameterize each principal type with a realm. Therefore, while the type of a generic principal was "principal" in the intra-realm model, we upgrade it to "principal R" for an appropriate realm R in the cross-realm setting.

Kerberos relies on a number of specialized principals, which we formalize as different subsorts of principal: clients, servers, TGSs and KASs are modeled by the type families client, server, TGS, and KAS, respectively, each parameterized by a realm. For convenience, we introduce two auxiliary types: *ts* represents either a TGS or a server, and *tcs* additionally encompasses clients. These relations are realized using subsorting.

The next segment in Figure 5 formalizes the types of keys used in Kerberos: "$\mathsf{dbK}^R\ A$" represents a long-term key that the KDC of realm R shares with principal A, which can be a client, a server, or the TGS of R. The type "$\mathsf{shK}\ C\ A$" instead represents the short-term authentication and service keys that client C receives to communicate with a TGS or a server. Both are subsorts of the generic type key of all keys, but shK must also be a subsort of msg since it is transmitted in the network.

We indirectly model the novel notion of cross-realm keys by defining the type rTGS of remote TGSs. A principal T declared of type rTGS R_1 R_2 is seen as a server in R_1 and as a TGS in R_2. This is achieved through the subsorting relation:

$$\forall R_1, R_2 : \mathsf{realm}.\ \mathsf{rTGS}\ R_1\ R_2 <: \mathsf{server}\ R_1.$$
$$\forall R_1, R_2 : \mathsf{realm}.\ \mathsf{rTGS}\ R_1\ R_2 <: \mathsf{TGS}\ R_2.$$

In order to represent the fact that the long-term key of T seen a server in R_1 is the same as the long-term key of T seen as a TGS in R_2, we need the following additional subsorting relation:

$$\forall T : \mathsf{rTGS}\ R_1\ R_2.\ \mathsf{dbK}^{R_1}\ T <: \mathsf{dbK}^{R_2}\ T$$

Note that this relation holds only for remote TGSs: it does not imply that the long-term key of any principal in R_1 is also a long-term key for this principal in R_2. Observe also that this relation is oriented: T sees a service ticket in R_1 as a TGT in R_2, but not vice versa.

We introduce Rset to model the `TRANSITED` field; Rset is a collection of realm names. We express this field as a list with $(\_\hat{}\_)$ as its constructor (infix for convenience) and Rnil as the empty list:

$$\mathsf{Rnil} : \mathsf{Rset}.$$
$$(\_\hat{}\_) : \mathsf{realm} \rightarrow \mathsf{Rset} \rightarrow \mathsf{Rset}.$$

Note that the Kerberos specification documents [17, 21] do not assume that `TRANSITED` is implemented as a list. Therefore, none of the MSR predicates that make decisions based on the value of this field shall rely on this structure.

6. FORMALIZATION

We now present a formalization of Kerberos 5 intra- and cross-realm authentication. The rules described in the following sections are sufficient for modeling both modes of operation and include the minimum level of detail we believe necessary to prove properties on authentication, confidentiality, and the effect of compromised realms. At the time of writing, this specification is being validated using the MSR implementation in [23], where the entire code for this case-study can be found. For the convenience of the reader, Appendix A lists the predicates used in this formalization that are not formally part of MSR.

6.1 Authentication Service Exchange

Figures 6 and 7 show the MSR rules used by the client and KAS, respectively, during the initial authentication service exchange. The actions of the client and KAS are the same in both the intra-realm and cross-realm case. Section 2.2 gives a description of these rules; the interested reader is referred to [7] for more details. Upon receiving a valid reply from the KAS, C stores the TGT, the name of the TGS T, and the shared key AK for future use using the $Auth_C$ predicate. The $Auth_C$ predicate is used to store a postulated ticket, *ts* and shK tuple, and models storing a ticket-granting ticket or a service-ticket along with associated parameters for future use.

Note that in general, MSR relies on a form of existential quantification to distinguish between various instances of L

	Types	Subsorting	Names
(Messages)	msg : type.		m, X, Y
(Realm)	realm : type.		R
(Principals)	principal : realm $\rightarrow$ type. KAS : realm $\rightarrow$ type. *tcs* : realm $\rightarrow$ type *ts* : realm $\rightarrow$ type TGS : realm $\rightarrow$ type. server : realm $\rightarrow$ type. client : realm $\rightarrow$ type.	$\forall R$: realm. principal $R <:$ msg. $\forall R_1, R_2$: realm. KAS $R_1 <:$ principal R_2. $\forall R_1, R_2$: realm. *tcs* $R_1 <:$ principal R_2. $\forall R_1, R_2$: realm. *ts* $R_1 <:$ *tcs* R_2. $\forall R_1, R_2$: realm. TGS $R_1 <:$ *ts* R_2. $\forall R_1, R_2$: realm. server $R_1 <:$ *ts* R_2. $\forall R_1, R_2$: realm. client $R_1 <:$ *tcs* R_2.	K T S C
(Keys)	*key* : type. dbK : realm $\rightarrow$ *tcs* $R \rightarrow$ type. shK : client $R_1 \rightarrow$ *ts* $R_2 \rightarrow$ type.	$\forall R$: realm, B : *tcs* R_2. dbK^R $B <:$ *key*. $\forall C$: client R_1, A : *ts* R_2. shK C $A <:$ *key*. $\forall C$: client R_1, A : *ts* R_2. shK C $A <:$ msg.	$k_{\_,\_}$ AK SK
(Remote TGS)	rTGS : realm $\rightarrow$ realm $\rightarrow$ type.	$\forall R_1, R_2$: realm. rTGS R_1 $R_2 <:$ server R_1. $\forall R_1, R_2$: realm. rTGS R_1 $R_2 <:$ TGS R_2. $\forall T$: rTGS R_1 R_2. dbK^{R_1} $T <:$ dbK^{R_2} T	T
(Nonces)	nonce : type.	nonce $<:$ msg.	n
(Timestamps)	time : type.	time $<:$ msg.	$t_{\_,\_}$
(Transited)	Rset : type.	Rset $<:$ msg	Rs

Figure 5: An MSR Signature for Kerberos 5 with Cross-Realm Capabilities

$$\forall C : \mathsf{client}\ R_C$$
$$\left[\begin{array}{l} \forall T : \mathsf{TGS}\ R_C \\ \qquad \cdot \ \stackrel{\alpha_{1.1}}{\Longrightarrow} \ \begin{bmatrix} \mathsf{N}(C, T, n_1) \\ L(C, T, n_1) \end{bmatrix} \\ \\ \forall T : \mathsf{TGS}\ R_C \quad \forall k_{R_C,C} : \mathsf{dbK}^{R_C}\ C \quad \forall AK : \mathsf{shK}\ C\ T \\ \forall X : \mathsf{msg} \quad \forall n_1 : \mathsf{nonce} \quad \forall t_K : \mathsf{time}. \\ \begin{bmatrix} L(C, T, n_1) \\ \mathsf{N}(C, X, \{AK, n_1, t_K, T\}_{k_{R_C,C}}) \end{bmatrix} \ \stackrel{\alpha_{1.2}}{\Longrightarrow} \ \mathsf{Auth}_C(X, T, AK) \end{array}\right]$$

Figure 6: The Client's role in the Authentication Service Exchange

predicates in different roles. However, this is not required in the formalization of Kerberos 5 because nonces are used for the same purpose. Thus, for simplicity, this feature of MSR has been left out of the formalization. Nevertheless, note that there are protocols, such as Kerberos 4, that do require existential quantification of L predicates.

6.2 Ticket Granting Exchange

After C has obtained a TGT X she may use X to obtain TGTs for remote realms or service tickets for application servers. Figure 8 shows the MSR rules that formalize these actions. At this point C has a TGT for use with the ticket granting server T. There are two modes of operation in which $\alpha_{1.3}$ may be used. The first mode is the standard intra-realm ticket granting exchange. In this case, T and S are in the same realm (*i.e.*, $R_T = R_S$). The $DesiredHop$ constraint is used by C to pick the TGS that she should use to obtain the service or ticket granting ticket (see Appendix A). R_n is the realm that will be the next hop. In the intra-realm case, $R_n = R_T$ and $T_n = S$. Thus C sends the supposed TGT X, the authenticator (C's name and a timestamp encrypted under AK), C's name, T_n's name (*i.e.*, the server's name), and a freshly chosen nonce to T. C stores these values using the L predicate and retains the $Auth_C$ predicate as a ticket-granting ticket may be reused. The second mode of operation is a cross-realm ticket granting ticket request and is identical to the intra-realm case except $T_n \neq S$ and S is not in the same realm as T. There are two methods by which the authentication path may be constructed in a cross-realm request. C may allow the TGS to determine the authentication path by simply always requesting a TGT for use with the destination realm, or C may know the authentication path she wishes to use and explicitly request tickets for each of these ticket granting servers. These methods can actually work together, for example when C has only partial knowledge of the nodes on a path to S. T_n and the $DesiredHop$ constraint are intended to capture both of these methods of operation. T_n is the name of the TGS that C wishes to use next on the authentication path. C sends the same message over the network and stores the same information as in the intra-realm request.

Rule $\alpha_{1.4}$ receives a TGS response and results in a TGT or service ticket for use with T_j. Note that T_j is of type *ts* and thus may be a TGS or server. If T_j is a TGS, note that T_j is not necessarily equal to T_n since T may have not shared a cross-realm key with T_n. C expects T's response to consist of C's name, a TGT or service ticket Z, and some data encrypted

$$
\begin{array}{l}
\forall K : \mathsf{KAS}\ R_C \\
\left[\begin{array}{l}
\forall C : \mathsf{client}\ R_C \quad \forall T : \mathsf{TGS}\ R_C \quad \forall n_1 : \mathsf{nonce} \\
\forall k_{R_C,C} : \mathsf{dbK}^{R_C}\ C \quad \forall k_{R_C,T} : \mathsf{dbK}^{R_C}\ T \quad \forall t_K : \mathsf{time}. \\
\qquad\qquad\qquad\qquad \exists AK : \mathsf{shK}\ C\ T \\
\mathsf{N}(C,T,n_1) \quad \overset{\alpha_{2.1}}{\Longrightarrow} \quad \left[\begin{array}{l} \mathsf{N}(C, \{AK, C, t_K, \mathsf{Rnil}\}_{k_{R_C,T}}, \\ \qquad \{AK, n_1, t_K, T\}_{k_{R_C,C}}) \end{array}\right] \\
\textit{if} \quad \mathsf{Valid}_K(C,T), \mathsf{clock}_K(t_K)
\end{array}\right]
\end{array}
$$

Figure 7: The KAS's Role in the Authentication Service Exchange

$$
\begin{array}{l}
\forall C : \mathsf{client}\ R_C \\
\left[\begin{array}{lcl}
\forall X : \mathsf{msg} \quad . & & \\
\forall T : \mathsf{TGS}\ R_T \quad . & & \exists n_2 : \mathsf{nonce} \\
\forall T_n : \mathit{ts}\ R_n \quad . & \overset{\alpha_{1.3}}{\Longrightarrow} & \mathsf{N}(X, \{C, t_C\}_{AK}, C, T_n, n_2) \\
\forall S : \mathsf{server}\ R_S \quad . & & L'(C, T_n, T, AK, n_2) \\
\forall AK : \mathsf{shK}\ C\ T. & & \\
\forall t_C : \mathsf{time} \quad . & & \\
\textit{if} \quad \mathsf{Auth}_C(X,T,AK),\ \mathsf{DesiredHop}(C,S,R_T,R_n),\ \mathsf{clock}_C(t_C) & & \\
 & & \\
\forall T_j : \mathsf{ts}\ R_J \quad . & & \\
\forall Z : \mathsf{msg} \quad . & \mathsf{N}(C, Z, \{AK', n_2, t_T, T_j\}_{AK}) & \\
\forall AK' : \mathsf{shK}\ C\ T_j. & L'(C, T_n, T, AK, n_2) \quad \overset{\alpha_{1.4}}{\Longrightarrow} & Auth_C(Z, T_j, AK') \\
\forall n_2 : \mathsf{nonce} \quad . & & \\
\forall t_T : \mathsf{time} \quad . & &
\end{array}\right]
\end{array}
$$

Figure 8: The client's role in TGS exchange.

$$
\begin{array}{l}
\forall T_i : \mathsf{TGS}\ R_i \\
\left[\begin{array}{lcl}
\forall R_i : \mathsf{realm} \quad . & & \\
\forall C : \mathsf{client}\ R_C \quad . & & \\
\forall S : \mathsf{server}\ R_i \quad . & & \\
\forall AK : \mathsf{shK}\ C\ T_i \quad . & \mathsf{N}(\{AK, C, t, Rs\}_{k_{R_i,T_i}}, & \exists SK : \mathsf{shK}\ C\ S \\
\forall k_{R_i,T_i} : \mathsf{dbK}^{R_i}\ T_i. & \{C, t_C\}_{AK}, C, S, n_2) \quad \overset{\alpha_{3.1}}{\Longrightarrow} & \mathsf{N}(C, \{SK, C, t_{T_i}, (R_i\hat{\ }Rs)\}_{k_{R_i,S}}, \\
\forall k_{R_i,S} : \mathsf{dbK}^{R_i}\ S \quad . & & \{SK, n_2, t_{T_i}, S\}_{AK}) \\
\forall n_2 : \mathsf{nonce} \quad . & & \\
\forall Rs : \mathsf{Rset} \quad . & & \\
\forall t_{T_i}, t_C, t : \mathsf{time} \quad . & & \\
\textit{if} \quad \mathsf{Valid}_{T_i}(C, S, t_C, (R_k\hat{\ }Rs)),\ \mathsf{clock}_{T_i}(t_{T_i}) & &
\end{array}\right]
\end{array}
$$

$$
\begin{array}{l}
\forall T_i : \mathsf{TGS}\ R_i \\
\left[\begin{array}{lcl}
\forall R_{i-1}, R_{i+1}, R_n : \mathsf{realm}. & & \\
\forall C : \mathsf{client}\ R_C \quad . & & \\
\forall AK : \mathsf{shK}\ C\ T_i \quad . & & \\
\forall T_n : \mathsf{TGS}\ R_n \quad . & & \\
\forall T_{i+1} : \mathsf{rTGS}\ R_i\ R_{i+1} \quad . & \mathsf{N}(\{AK, C, t, Rs\}_{k_{R_{i-1},T_i}}, & \exists AK' : \mathsf{shK}\ C\ T_{i+1} \\
\forall k_{R_{i-1},T_i} : \mathsf{dbK}^{R_{i-1}}\ T_i \quad . & \{C, t_C\}_{AK}, C, T_n, n_2) \quad \overset{\alpha_{3.2}}{\Longrightarrow} & \mathsf{N}(C, \{AK', C, t_{T_i}, (R_{i-1}\hat{\ }Rs)\}_{k_{R_i,T_{i+1}}}, \\
\forall k_{R_i,T_{i+1}} : \mathsf{dbK}^{R_i}\ T_{i+1} \quad . & & \{AK', n_2, t_{T_i}, T_{i+1}\}_{AK}) \\
\forall n_2 : \mathsf{nonce} \quad . & & \\
\forall Rs : \mathsf{Rset} \quad . & & \\
\forall t_{T_i}, t_C, t : \mathsf{time} \quad . & & \\
\textit{if} \quad \mathsf{Valid}_{T_i}(C, T_n, t_C, (R_{i-1}\hat{\ }Rs)),\ \mathsf{CloserRealm}(T_n, T_{i+1}),\ \mathsf{clock}_{T_i}(t_{T_i}) & &
\end{array}\right]
\end{array}
$$

Figure 9: TGS response to a cross-realm TGT request or an intra-realm service ticket request.

with AK. This data consists of a new shared key AK' that is to be shared by C and T_j, n_2, a timestamp, and T_j's name. C uses the $Auth_C$ predicate to model storing the ticket, key, and T_j's name.

Figure 9 shows the TGS's role in the ticket granting exchange. Rule $\alpha_{3.1}$ is used to create service tickets for use with the server S and $\alpha_{3.2}$ is used to create cross-realm ticket granting tickets for the TGS T_{i+1}. In $\alpha_{3.1}$ T_i receives C's request for a service ticket and checks that it is valid according to the local policy using the $Valid_{T_i}$ predicate. A ticket is considered valid if C is allowed to access S, the timestamp t_C is within the allowed clock skew and the `TRANSITED` field (Rs) is acceptable. If the request is valid, T_i chooses a fresh key SK to be shared by C and S. T_i then sends a reply that consists of C's name, the service ticket, and some data for C encrypted with AK. Note that T_i adds the name of the previously transited realm to the transited list. It knows this realm because it shares a key with the KAS or TGS that produced the TGT. The data for C consists of SK, n_2, a timestamp, and the name of the server.

Rule $\alpha_{3.2}$ is very similar to $\alpha_{3.1}$ except that $\alpha_{3.2}$ is used to grant cross-realm ticket granting tickets. T_i will invoke $\alpha_{3.2}$ if it receives a valid cross-realm TGT request from C. Checking the validity of the ticket is the same as the intra-realm case except T_n is a TGS rather than an application server. The main difference in $\alpha_{3.2}$ is that T_i must search its local database to find a cross-realm key k_{R_i,T_n} that is shares with T_n. If this key does not exist, T_i searches for a cross-realm key $k_{T_i,T_{i+1}}$ shared with a closer TGS T_{i+1}. This process is modeled with the $CloserRealm$ predicate, which is satisfied if $T_{i+1} = T_n$ or if T_{i+1} is the next TGS on the standard hierarchical path or a preconfigured authentication path. T_i adds the name of the previous TGS's realm to the `TRANSITED` field. T_i's response is the same as the intra-realm case except $T_{i+1} : \mathsf{rTGS} R_i R_{i+1}$ is substituted for $S : \mathsf{server}$.

6.3 Client/Server Exchange

Once C has obtained a service ticket, C may execute rule $\alpha_{1.5}$ to connect to an application server. Note that in $\alpha_{1.5}$ S is of type server instead of ts. This typing allows C to distinguish between $\alpha_{1.3}$ and $\alpha_{1.5}$. C sends the service ticket along with her name and a timestamp encrypted under SK to S.

When S receives C's request S uses $\alpha_{4.1}$ to validate it. Validation is modeled using the $Valid_S$ predicate. $Valid_S$ is true if C is allowed to access S, the timestamp is within the allowed clock skew, and the `TRANSITED` field is valid. Note that C and S's realms are added to the `TRANSITED` field before checking that it is valid. This is because those fields are required to be on the authentication path since C must have obtained a TGT for her realm first and the service ticket for S may only by generated by the TGS in S's realm. If the request is valid, S sends the timestamp encrypted with SK to C.

7. CROSS-REALM INTRUDER

Formal specification and analysis work on Kerberos in the literature [2, 6, 18] has concentrated on intra-realm authentication. All these efforts relied on the traditional Dolev-Yao model of the intruder [11, 19], which gives an attacker complete control over the network, short of guessing random values (such as keys and nonces) and performing cryptographic operations without the required keys. In MSR, the knowledge of the intruder is modeled as a collection of facts $\mathsf{I}(m)$ ("the intruder knows message m") distributed in the state. Two typical intruder rules follow (see [6] for the complete set):

$$\forall m : \mathsf{msg}.\ \mathsf{N}(m) \ \Rightarrow \mathsf{I}(m)$$
$$\forall m : \mathsf{msg}.\ \forall k : \mathsf{key}.\ \mathsf{I}(\{m\}_k), \mathsf{I}(k) \ \Rightarrow \mathsf{I}(m)$$

The first describes the interception of a network message m, while the second shows how the intruder can decrypt a ciphertext $\{m\}_k$ if he knows the key k. Dishonest clients and servers are easily modeled by having them share their long-term keys with such an intruder: for each such key k, the state contains $\mathsf{I}(k)$. In the intra-realm case, considering the KDC to be dishonest trivially invalidates any authentication result.

The cross-realm setting naturally suggests a more elaborate intruder model. Indeed, a realm often resides within the boundaries of a local network, partially segregated from the Internet by means of NATs, firewalls and other mechanisms. Consequently, it makes sense to consider local attackers who may form coalitions with each other or with an intruder overseeing the outside network. Finally, KDC's and therefore entire realms could be compromised and still allow some authentications to be secure. While all these aspects are worth considering and could easily be modeled in MSR, we will focus on the functionalities that are specific to the Kerberos cross-realm setting. In particular, we refrain from modeling segregated networks since they can be studied independently from Kerberos. We instead assume a worst-case scenario in which all principals communicate on the same network, independently of the realm they reside on. Similarly, we assume a single intruder rather than a distributed coalition as it has been shown in [10] that any alliance of Dolev-Yao intruders can always be simulated by a single such attacker.

This leaves us with a Dolev-Yao threat model in which the global intruder can impersonate not only clients and end-servers but also KDC's. As mentioned earlier, a principal that is dishonest or compromised is modeled by giving all of its keys to the intruder. In the case of the KDC of a realm R, this means making the whole key database of R available to the intruder, including all the cross-realm keys it mentions. Note that there is no distinction between compromising a KAS or a TGS since both have access to the same keys. This is easily accomplished in our setting by extending the traditional Dolev-Yao model with the following MSR rule:

$$\begin{bmatrix} \forall R : \mathsf{realm} \quad \forall P : \mathsf{tcs}\ R \quad \forall k : \mathsf{dbK}^R\ P \\ \cdot \ \Rightarrow \mathsf{I}(k) \\ \mathit{if} \quad \mathsf{compromised}(R) \end{bmatrix}$$

Here, the fact $\mathsf{compromised}(R)$ marks a realm as compromised by the intruder. This simple rule will be sufficient to enable all the anomalous behaviors outlined in the remaining sections of this paper.

Additional cross-realm capabilities are conceivable, although we are not aware of any realistic attack that relies on them. First, the intruder can clearly create fake clients and end-servers

$$\forall C : \mathsf{client}\ R_C \quad \left[\begin{array}{l} \forall S : \mathsf{server}\ R_S\ . \\ \forall SK : \mathsf{shK}\ C\ S.\ . \\ \forall t_{C,Sreq} : \mathsf{time}\ . \\ \forall Y : \mathsf{msg}\ . \\ \textit{if}\quad \mathsf{Auth}_C(Y,S,SK),\ \mathsf{clock}_C(t_{C,S_{req}}) \end{array} \quad \xrightarrow{\alpha_{1.5}} \quad \mathsf{N}(Y,\{C,t_{C,Sreq}\}_{SK}) \right]$$

Figure 10: The client's role in the application server exchange.

$$\forall S : \mathsf{server}\ R_S \quad \left[\begin{array}{l} \forall C : \mathsf{client}\ R_C\ . \\ \forall T_S : \mathsf{TGS}\ R_S\ . \\ \forall SK : \mathsf{shK}\ C\ S\ . \\ \forall t_{C,Sreq}, t_{T_S} : \mathsf{time}. \\ \forall k_{R_S,S} : \mathsf{dbK}^{R_S}\ S\ . \\ \forall Rs : \mathsf{Rset}\ . \\ \textit{if}\quad \mathsf{Valid}_S(C,t_{C,Sreq},(R_C\hat{}R_S\hat{}Rs)) \end{array} \quad \begin{array}{l} \mathsf{N}(\{SK,C,t_{T_S},Rs\}_{k_{R_S,S}} \\ \{C,t_{C,Sreq}\}_{SK}) \end{array} \quad \xrightarrow{\alpha_{4.1}} \quad \begin{array}{l} \mathsf{N}(\{t_{C,Sreq}\}_{SK}) \\ Mem_S(C,SK,t_{C,Sreq},Rs) \end{array} \right]$$

Figure 11: The server's role in the service ticket exchange.

in a compromised realm R:

$$\left[\begin{array}{l} \forall R : \mathsf{realm} \\ \cdot\ \Rightarrow \exists P : \mathsf{client}\ R.\ \exists k : \mathsf{dbK}^R\ P.\ \mathsf{I}(P), \mathsf{I}(k) \\ \textit{if}\quad \mathsf{compromised}(R) \end{array}\right]$$

(and similarly for a server). Second, the global intruder could create fake realms, and populate them using the above method. With the following rule, the intruder makes up a realm R' and sets up a cross-realm key from some other realm R it has compromised and R':

$$\left[\begin{array}{l} \forall R : \mathsf{realm} \\ \cdot\ \Rightarrow \exists R' : \mathsf{realm}.\ \exists P : \mathsf{rTGS}\ R\ R'.\ \exists k : \mathsf{dbK}^R\ P. \\ \qquad \mathsf{I}(P), \mathsf{I}(k), \mathsf{compromised}(R') \\ \textit{if}\quad \mathsf{compromised}(R) \end{array}\right]$$

8. PROPERTIES

This section formally describes the necessary conditions for cross-realm authentication to be possible in Kerberos 5. In future work, we expect to build on this result by extending the traditional intra-realm confidentiality and authentication properties proved for Kerberos in [6, 7] to the cross-realm setting.

Kerberos's only defense against compromised or untrusted intermediate realms is the use of the transited field to enable a server to determine if authentication should be performed. The following property proves that if there are any compromised realms involved in the authentication of a client then at least one of them will appear in the transited field. This is a critical property because it allows servers to make informed authentication decisions. In addition, this property shows that under expected network conditions (*i.e.* when there are no compromised ticket granting servers) the transited field of a service ticket contains exactly the realms involved in authenticating the client. While not following from this property, we expect that under the assumptions that the client's long-term key has not been compromised and the set of transited realms contains only non-compromised realms, then the client named in the message did in fact request this authentication.

PROPERTY 1. *If a server S processes a request from a client C and if $\mathcal{R}$ is the set of realms encoded by the transited field of the request, together with the realm of the TGS that created the ticket in the request and C's realm, if no tickets were present in the initial state of the trace, and if neither keys between realms in $\mathcal{R}$ nor S's long-term key have been compromised, then some sequence of TGSs from realms in $\mathcal{R}$, starting with a TGS in C's realm, authenticated C to S and the TGS of each realm in $\mathcal{R}$ took part in this authentication.*

We formalize this as Corollary 2 to Theorem 1; this theorem also implies that if the intended authentication did not take place, then at least one of a specified set of unintended behaviors occurred in the trace.

Our formal proofs make use of rank and corank functions; these are inspired by Schneider's rank functions [22] for analysis of protocols in CSP and were defined in [6, 7] in the context of MSR and Kerberos. 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 invite the reader to consult [6, 7] for additional details.

THEOREM 1. *For all S : $\mathsf{server}\ R_S$, C : $\mathsf{client}\ R_C$, T_S : $\mathsf{TGS}\ R_S$, Rs : Rset, SK : $\mathsf{shK}\ C\ S$, t_1, t_2 : time, and $k_{R_S,S}$: $\mathsf{dbK}^{R_S}\ S$, if S fires rule $\alpha_{4.1}$, consuming the fact $\mathsf{N}(\{SK, C, t_1, Rs\}_{k_{R_S,S}}, \{C, t_2\}_{SK})$, if no fact of positive rank appeared in the initial state of the trace, and if $\mathcal{R} = R_C\hat{}R_S\hat{}Rs$, then at least one of the following holds.*

1. *The intruder created the fact $\mathsf{I}(\{SK, C, t_1, Rs\}_{k_{R_S,S}})$. Furthermore, the fact $\mathsf{I}(k_{R_S,S})$ appeared in some previous state of the trace.*
2. *The intruder created the fact $\{AK, C, t, Rs'\}_{k_{R',T''}}$, for some AK : $\mathsf{shK}\ C\ T''$, t : time, Rs : Rset, $R', R'' \in \mathcal{R}$, T'' : $\mathsf{TGS}\ R''$, and $k_{R',T''}$: $\mathsf{dbK}^{R'}\ T''$. Furthermore, the fact $\mathsf{I}(k_{R',T''})$ appeared in some previous state of the trace.*

3. *The intruder created the fact* $\{AK, C, t, Rs\}_{k_{R_C T_C}}$ *for some* AK : shK $C\ T_C$, t : time, Rs : Rset, T_C : TGS R_C, *and* k_{R_C,T_C} : dbK^{R_C} T_C. *Furthermore, the fact* $\mathsf{I}(k_{R_C,T_C})$ *appeared in some previous state of the trace.*

4. *For some sequence of realms* $R_C = R_1, \ldots, R_i = R_S$ *from* $\mathcal{R}$: T_i : TGS R_i *fired rule* $\alpha_{3.1}$, *consuming the fact* $\mathsf{N}(\{AK_i, C, t_i, Rs_i\}_{k_i}, \{C, t_i'\}_{AK_i}, C, S, n_i)$ *for some* AK_i : shK $C\ T_i$, t_i, t_i' : time, Rs_i : Rset, k_i : dbK^R T_i, *where* R : realm *is* R_{i-1} *if* $i > 1$ *and* R_i *otherwise, and* n_i : nonce; *if* $i > 1$, *then for* $1 \leq j < i$, *some* T_j : TGS R_j *fired rule* $\alpha_{3.2}$, *consuming the fact* $\mathsf{N}(\{AK_j, C, t_j, Rs_j\}_{k_j}$, $\{C, t_j'\}_{AK_j}, C, S, n_j)$ *for some* AK_j : shKCT_j, t_j, t_j' : time, Rs_j : Rset, k_j : dbK$^{R_{j-1}}$ T_j, *where* $R_0 = R_1$, *and* n_j : nonce.

COROLLARY 2. *For all* S : serverR_S, C : clientR_C, T_S : TGSR_S, Rs : Rset, SK : shKCS, t_1, t_2 : time, *and* $k_{R_S,S}$: dbK^{R_S} S, *if* S *fires rule* $\alpha_{4.1}$, *consuming the fact* $\mathsf{N}(\{SK, C, t_1, Rs\}_{k_{R_S,S}}, \{C, t_2\}_{SK})$, *if no fact of positive rank appeared in the initial state of the trace, if* $\mathcal{R} = \hat{}R_C\hat{}R_S Rs$, *and if the intruder has never learned* $k_{R_S,S}$, *any keys between realms in* $\mathcal{R}$, *or the long-term key of any* T_C : TGS R_C, *then for some sequence of realms* $R_C = R_1, \ldots, R_i = R_S$ *from* $\mathcal{R}$: T_i : TGSR_i *fired rule* $\alpha_{3.1}$, *consuming the fact* $\mathsf{N}(\{AK_i, C, t_i, Rs_i\}_{k_i}$, $\{C, t_i'\}_{AK_i}, C, S, n_i)$ *for some* AK_i : shK $C\ T_i$, t_i, t_i' : time, Rs_i : Rset, k_i : dbK^R T_i, *where* R : realm *is* R_{i-1} *if* $i > 1$ *and* R_i *otherwise, and* n_i : nonce; *if* $i > 1$, *then for* $1 \leq j < i$, *some* T_j : TGS R_j *fired rule* $\alpha_{3.2}$, *consuming the fact* $\mathsf{N}(\{AK_j, C, t_j, Rs_j\}_{k_j}$, $\{C, t_j'\}_{AK_j}, C, S, n_j)$ *for some* AK_j : shK $C\ T_j$, t_j, t_j' : time, Rs_j : Rset, k_j : dbK$^{R_{j-1}}$ T_j, *where* $R_0 = R_1$, *and* n_j : nonce.

PROOF. (Of Theorem 1) S's firing of rule $\alpha_{4.1}$ consumes a fact of positive $k_{R_S,S}$-rank relative to SK, C, t_1, Rs. As no such fact existed in the initial state of the trace, some rule fired earlier in the trace increased this rank. If the intruder fired a rule that increased this rank, then she created the fact $\{SK, C, t_1, Rs\}_{k_{R_S,S}}$ and the fact $\mathsf{I}(k_{R_S,S})$ must have appeared in the trace; case 1 of the theorem then holds. If the intruder did not increase this rank, then some principal rule must have been fired to do so. By inspection of the various rules, the only possibility for this is T_S firing rule $\alpha_{3.1}$, creating the fact $\mathsf{N}(C, \{SK, C, t_1, Rs\}_{k_{R_S,S}}, \{SK, n, S\}_{AK})$ and consuming the fact $\mathsf{N}(\{AK, C, t_2, Rs'\}_{k_{R',T_S}}, \{C, t_3\}_{AK}, C, S, n)$ for some t_2, t_3 : time, n : nonce, R' : realm, k_{R',T_S} : dbK$^{R'}$ T_S, AK : shK $C\ T_S$, and Rs' : Rset such that $R'\hat{}Rs' = Rs$. The fact consumed by this rule firing has positive k_{R',T_S}-rank relative to AK, C, t_2, Rs'; as no such fact existed in the initial state of the trace, some rule firing must have increased this rank. If the intruder increased this rank, then she created the fact $\{AK, C, t_2, Rs'\}_{k_{R',T_S}}$ and the fact $\mathsf{I}(k_{R',T_S})$ has previously appeared in the trace; this falls into case 2 of the theorem.

If the intruder did not increase this rank and $R' \neq R_C$, then inspection of the principal rules shows that some T' : TGS R' must have fired rule $\alpha_{3.2}$ to increase this rank, in the process creating the fact $\mathsf{N}(C, \{AK, C, t_2, Rs'\}_{k_{R',T_S}}$, $\{AK, n', t_2, T_S\}_{AK''})$ and consuming the fact $\mathsf{N}(\{AK'', C, t_4, Rs''\}_{k_{R'',T'}}, \{C, t_5\}_{AK''}, C, T''', n')$ for some t_4, t_5 : time, n' : nonce, AK'' : shK $C\ T'$, R'', R''' : realm, $k_{R'',T'}$: dbK$^{R''}T', T'''$: TGSR''', and Rs'' : Rset such that $R''\hat{}Rs'' = Rs'$. Note that this fact has positive $k_{R'',T'}$-rank relative to AK'', C, t_4, Rs''; as no such fact existed in the initial state of the trace, some rule firing must have increased this rank. If the intruder increased this rank, then she created the fact $\{AK'', C, t_4, Rs''\}_{k_{R'',T'}}$ and the fact $\mathsf{I}(k_{R'',T'})$ previously appeared in the trace; this falls into case 2 of the theorem. If the intruder did not increase this rank, then we consider whether or not $R'' = R_C$.

Either in the initial $R' = R_C$ case or after some finite number of iterations of this argument, if we do not fall into case 2 of the theorem, then we have some R_0 : realm in which T_0 : TGSR_0 fires rule $\alpha_{3.2}$ to consume the fact $\mathsf{N}(\{AK_0, C, t_0, Rs_0\}_k$, $\{C, t_0'\}_{AK_0}, C, T_z, n_0)$, where k : dbK$^{R_0}T_0$, AK_0 : shKCT_0, t_0, t_0' : time, R_z : realm, T_z : TGS R_z, Rs_0 : Rset, and n_0 : nonce. (Otherwise, we could apply the argument again and see that the initial state of the trace must have contained a fact of positive rank, contradicting the hypothesis of the theorem.) This consumes a fact of positive k-rank relative to AK_0, C, t_0, Rs_0; as no such fact appeared in the initial state of the trace, some rule firing must have increased this rank. If the intruder fired a rule to do this, then she created the fact $\{AK_0, C, t_0, Rs_0\}_k$ and the fact $\mathsf{I}(k)$ must have previously appeared in the trace; this falls under case 3 of the theorem. If the intruder did not increase this rank, by inspection of the principal rules we see that some K : KAS R_0 must have done so, firing rule $\alpha_{2.1}$ to produce fact $\mathsf{N}(C, \{AK_0, C, t_0, Rs_0\}_k$, $\{AK_0, n_0', t_0, T_0\}_{k'})$, with $Rs_0 =$ Rnil, and consume the fact $\mathsf{N}(C, T_0, n_0')$ for C : R_0, n_0' : nonce, k' : dbK^{R_0} C. This implies that $R_0 = R_C$, so T_0 : TGS R_C and case 4 of the theorem holds. □

We note that if the `TRANSITED` field preserved the order of the transited realms, then we could show that the last compromised realm of the authentication path, as well as all (uncompromised) realms subsequently traversed (except for the endpoints), appears in the `TRANSITED` field. However, Sec. 3.3.3.1 of [21] notes that the `TRANSITED` field of the ticket-granting ticket "is treated as an unordered set of realm names[.]"

9. VULNERABILITIES

This section documents some attacks on Kerberos 5 cross-realm authentication that may be executed by the cross realm intruder presented in Section 7. It is important to note that this behavior does not conflict with guarantees described in [21]. However, when an administrator is deciding whether to trust another realm, the amount of damage that can be done on the local realm by a compromised foreign realm must be considered. These sections document some of these attacks but are not necessarily complete.

9.1 TGSs learn AKey and SKey

During cross-realm authentication all of the ticket granting servers on the authentication path are capable of learning SK

(*i.e.* the key shared between the server and the client). In addition, given an authentication path $R_1, R_2, \ldots R_n, T_i : \mathsf{TGS} R_i$ is capable of learning $AK^j : \mathsf{shK}\ C\ T_{j+1}\ \forall j \geq i$, where AK^j is the temporary session key shared by C and T_{j+1} when authenticating C along the specified authentication path [14]. For any TGS T_i along the authentication path, T_i generates a new key AK^i that is used to encrypt communication with the next TGS T_{i+1} using rule $\alpha_{3.2}$. Since T_i generates AK^i it knows it and can store it in memory. Since the exchange of a new key AK^{i+1} between T_{i+1} and C is encrypted with AK^i (also rule $\alpha_{3.2}$), T_i can learn AK^{i+1}. T_i can repeat this process for all of the TGSs on the rest of the authentication path and is thus capable of learning SK. This means that all of the ticket granting servers on the authentication path can learn SK and thus use it to spoof the identify of S or C or to eavesdrop on the communication between S and C. Whether this is a security threat depends on whether all of the TGSs on the authentication path can be trusted. Note that in the intra-realm case the same behavior described above occurs, however the authentication path consists of just one TGS.

9.2 Remote TGS can Impersonate C

Any rogue TGS can impersonate a client C anywhere outside of C's realm, even if C has never contacted this TGS. This dishonest TGS T_I can make up a ticket for a "next hop" and spoof C's sending a request using this ticket. C does not need to be involved, and not even to exist if the end-server doesn't know C in the first place. To do this T_I creates the facts on the right hand side of $\alpha_{3.2}$. T_I can do this since it shares $k_{R_i,T_{i+1}}$ with TGS T_{i+1}. Note that the rogue TGS is capable of fabricating the `TRANSITED` field, however the property proved in Section 8 still holds. The compromised TGS is not able to masquerade as C in C's realm because local authentications are not supposed to use the cross-realm mechanism.

9.3 Routing Attack

With no intruder present, a client is capable of determining the authentication path by keeping track of the TGSs that she uses. When a client receives a message as in $\alpha_{1.4}$, the client believes that the next TGS on the authentication path is T_j. However, if there is an intruder TGS on the authentication path then this field may be fabricated and the client will be tricked into believing she is following a false authentication path. Section 9.1 describes how every TGS on the authentication path is capable of learning SK, and in the process learning all of the session keys (AK) from that point on in the authentication path. This implies that in $\alpha_{3.2}$ if $x < i{+}1$ then T_x is capable of reading and modifying the fact $N(\{AK', n_2, t_T, T_{i+1}\}_{AK})$. Thus, if there is an intruder TGS T_I, then $\forall i \geq I$, T_i may or may not be on the authentication path. Note that an intruder TGS can trick C into thinking that it is not on the authentication path and is capable of manipulating the rest of the authentication path in any way it wants without C's knowledge.

10. RELATED WORK

In spite of support in Kerberos, in particular as part of the popular Windows operating system, and in other protocol suites (*e.g.*, [1]), the literature on cross-realm authentication is scant and surprisingly old. The earliest paper in this arena appears to be [5] which proposes organizing authentication servers into a hierarchy for efficient authentication across administrative boundaries, a design still recognizable in Kerberos. More recently, Gligor *et al.* [14] undertook a general study of *inter-realm authentication* (as they call it) with particular focus on defining local trust policies that mitigate global security exposure. In particular, they studied an algorithm for finding authentication paths that is very close to Kerberos's own. The original interest in research in cross-realm authentication seems to have ended in the early 1990's.

While that work was mostly concerned with designing a solid cross-realm authentication infrastructure, our effort focuses on its formal specification and analysis. To this end, we have defined a precise cross-realm intruder model, proposed a machine-checkable formalization of Kerberos 5, and proved an important property of cross-realm trust. Differently from previous work, we point out specific threats that are likely to be more useful to a system administrator than the generic language in the Kerberos documents [17, 21].

11. CONCLUSIONS

In this paper, we formalized the support available in Kerberos 5 for cross-realm authentication. We also extended the Dolev-Yao intruder model to account for threats specific to this mode of operation. We characterized minimum requirements in view of assessing confidentiality and authentication properties, and documented a range of harmful behaviors in the presence of compromised or untrusted realms.

This preliminary investigation lends itself to extensions in numerous directions. First, we can embark in proving that cross-realm operation does satisfy similar confidentiality and authentication properties already established in the intra-realm case [6, 7]. Second, we want to extend our analysis to mechanisms that have been proposed to mitigate the harm that a compromised KDC sitting on an authentication path can inflict. One promising approach is the public-key extension of Kerberos through the PKINIT and especially PKCROSS subprotocols [8, 15]. Another involves a formal analysis of relevant aspects of SESAME [1]. In both cases, we believe that our approach can contribute to the active discussion within the Kerberos working group.

Acknowledgments

We are grateful to Sam Hartman and the Kerberos Working Group for interesting and informative discussions. We are also indebted to the anonymous reviewers for their useful comments.

12. REFERENCES

[1] *A Secure European System for Applications in a Multi-vendor Environment*, URL ON THE FOLLOWING PAGE.

[2] G. Bella, *Inductive Verification of Cryptographic Protocols*, Ph.D. thesis, University of Cambridge, March 2000, URL ON THE NEXT PAGE.

[3] G. Bella and L. C. Paulson, *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, URL BELOW, pp. 361–375.

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

[5] Andrew Birrell, Butler W. Lampson, Roger M. Needham, and Michael D. Schroeder, *A Global Authentication Service without Global Trust*, IEEE Symposium on Security and Privacy, 1986, pp. 223–230.

[6] F. Butler, I. Cervesato, A. Jaggard, and A. Scedrov, *A Formal Analysis of Some Properties of Kerberos 5 using MSR*, 15th IEEE Computer Security Foundations Workshop — CSFW-02, IEEE Computer Society Press, 24-26 June 2002, pp. 175–190.

[7] Frederick Butler, Iliano Cervesato, Aaron D. Jaggard, and Andre Scedrov, *A Formal Analysis of Some Properties of Kerberos 5 Using MSR*, Tech. Report MS-CIS-04-04, University of Pennsylvania Department of Computer and Information Science, April 2004, URL BELOW.

[8] USC CCSS, *The Kerberos Network Authentication Service*, URL BELOW.

[9] 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.

[10] Iliano Cervesato, Catherine Meadows, and Paul Syverson, *Dolev-Yao is no better than Machiavelli*, First Workshop on Issues in the Theory of Security — WITS'00 (Geneva, Switzerland) (P. Degano, ed.), 2000.

[11] D. Dolev and A. C. Yao, *On the Security of Public-Key Protocols*, IEEE Trans. on Information Theory **2** (1983), no. 29, 198–208.

[12] N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov, *Multiset Rewriting and the Complexity of Bounded Security Protocols*, Journal of Computer Security **12** (2004), 247–311.

[13] Jason Garman, *Kerberos: The Definitive Guide*, O'Reilly, 2003.

[14] Virgil D. Gligor, Shyh-Wei Luan, and Joe Pato, *On Inter-Realm Authentication in Large Distributed Systems*, Journal of Computer Security **2** (1993), no. 2-3, 137–158.

[15] Sam Hartman, Personal communications, September 2004.

[16] M.I. Kanovich, M. Okada, and A. Scedrov, *Specifying Real-Time Finite-State Systems in Linear Logic*, Proc. COTIC '98 (Nice, France), Electronic Notes in Theoretical Computer Science 16(1), 1998.

[17] J. Kohl and C. Neuman, *The Kerberos Network Authentication Service (V5)*, September 1993, Network Working Group Request for Comments: 1510, URL BELOW.

[18] 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.

[19] R. M. Needham and M. D. Schroeder, *Using Encryption for Authentication in Large Networks of Computers*, Communications of the ACM **21** (1978), no. 12, 993–999.

[20] B. Clifford Neuman and Theodore Ts'o, *Kerberos: An Authentication Service for Computer Networks*, IEEE Communications **32** (1994), no. 9, 33–38.

[21] Clifford Neuman, John Kohl, Theodore Ts'o, Tom Yu, Sam Hartman, and Ken Raeburn, *The Kerberos Network Authentication Service (V5)*, September 7 2004, Internet draft, expires 7 March 2005, URL BELOW.

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

[23] Mark-Oliver Stehr, Iliano Cervesato, and Stefan Reich, *An execution environment for the MSR cryptoprotocol specification language*, URL BELOW.

APPENDIX

A. PREDICATES

This appendix lists the protocol-specific predicates used in our formalization of Kerberos 5 cross-realm authentication. We give an intuitive explanation of what we intend to capture with them.

- $Auth_C(X, S, AK)$
  The $Auth_C$ memory predicate is used to model C obtaining a TGT or service ticket X for use with a TGS or server S. AK is the key associated with this ticket.
- $Valid_K(C, T)$
  This predicate is used to model the fact that the client C and TGS T are valid in K's realm.
- $DesiredHop(C, S, R_T, R_n)$
  $DesiredHop$ is the machinery used to model R_n as the desired next hop used in authenticating C to S if we are currently in realm R_T.

URLs:

[1]: `https://www.cosic.esat.kuleuven.ac.be/sesame/`
[2]: `http://www.cl.cam.ac.uk/~gb221/papers/bella14.ps.gz`
[3]: `http://www.cl.cam.ac.uk/~gb221/papers/bella6.ps.gz`
[7]: `ftp://ftp.cis.upenn.edu/pub/papers/scedrov/ms-cis-04-04.pdf`
[8]: `http://www.kerberos.isi.edu/`
[17]: `ftp://ftp.isi.edu/in-notes/rfc1510.txt`
[21]: `http://www.ietf.org/internet-drafts/draft-ietf-krb-wg-kerberos-clarifications-07.txt`
[23]: `http://formal.cs.uiuc.edu/stehr/msr_eng.html`

- $Valid_{ts}(C, S, t, Rs)$

  This predicate is used to model a valid service ticket or cross-realm ticket granting ticker request. ts is the TGS or server that validates the ticket, C is the client requesting the ticket, S is the TGS or server requested, t_C is a timestamp, and Rs is the `TRANSITED` field.

- $CloserRealm(T_n, T_j)$

  The $CloserRealm$ predicate is used to model T_i searching for a cross-realm key. T_n is the TGS that C requested and T_j is the TGS that is actually returned. If T shares a cross-realm key with the requested TGS then it will return a TGT for use with it. Otherwise, T will use the standard hierarchical path or a hard-coded authentication path to determine a closer TGS.

- $Mem_S(C, SK, t, Rs)$

  An application server S uses the Mem_S memory predicate to store information relevant to an authenticated client. C is the client's name and SK is the key they share. The timestamp of the request and the `TRANSITED` field are also stored for possible future use.