

# Formal Analysis of Kerberos 5

Frederick Butler<sup>a</sup> Iliano Cervesato<sup>b</sup> Aaron D. Jaggard<sup>b</sup>  
Andre Scedrov<sup>c</sup> Christopher Walstad<sup>c</sup>

<sup>a</sup>*West Virginia University, Morgantown, WV 26506*  
*fbutler@math.wvu.edu*

<sup>b</sup>*Tulane University, New Orleans, LA 70118*  
*{iliano|adj}@math.tulane.edu*

<sup>c</sup>*University of Pennsylvania, Philadelphia, PA 19104*  
*{scedrov@math|cwalstad@seas}.upenn.edu*

---

## Abstract

We report on the detailed verification of a substantial portion of the Kerberos 5 protocol specification. Because it targeted a deployed protocol rather than an academic abstraction, this multi-year effort led to the development of new analysis methods in order to manage the inherent complexity. This enabled proving that Kerberos supports the expected authentication and confidentiality properties, and that it is structurally sound; these results rely on a pair of intertwined inductions. Our work also detected a number of innocuous but nonetheless unexpected behaviors, and it clearly described how vulnerable the cross-realm authentication support of Kerberos is to the compromise of remote administrative domains.

*Key words:* Security Protocol Analysis, Formal Methods, Kerberos Protocol.

---

---

\* Cervesato was partially supported by ONR under Grant N00014-99-1-0150. Jaggard is partially supported by NSF Grants DMS-0239996, CCR-0098096, and CNS-0429689, and by ONR Grant N00014-05-1-0818; this paper was written while he was a Visiting Scholar in the Mathematics Department at the University of Pennsylvania. Scedrov and Walstad are partially supported by OSD/ONR CIP/SW URI “Software Quality and Infrastructure Protection for Diffuse Computing” through ONR Grant N00014-01-1-0795 and OSD/ONR CIP/SW URI “Trustworthy Infrastructure, Mechanisms, and Experimentation for Diffuse Computing” through ONR Grant N00014-04-1-0725. Additional support from NSF Grants CCR-0098096, CNS-0429689, and CNS-0524059.

## 1 Introduction

The verification of cryptographic protocols has come a long way since supporting the Clark-Jacob Library [1] or even just the Needham-Schroeder public key protocol [2] was a mark of success. In fact, early methods and tools became impractical as soon as the protocol grew over a handful of exchanged messages. In all cases, the analysis was carried out at the symbolic Dolev-Yao level [3], a relatively tractable worst-case model, which has however been criticized for being overly simplistic. More recently, research in protocol verification has moved in two main directions: overcoming the deficiencies of Dolev-Yao analysis by taking into consideration the computational characteristics of actual cryptography [4,5,6,7], and providing Dolev-Yao assurance for the complex protocols that are deployed in practice [8] rather than the minimal academic idealizations of the previous generation. This paper summarizes a multi-year effort in this second direction. We report on the results of the most detailed formal analysis of the widespread Kerberos 5 protocol [9,10] to date, and possibly of any deployed protocol.

Kerberos [9,10] is a successful, widely deployed single-login protocol that is designed to strongly authenticate a client to all the services it may require: it allows a user to log on at the beginning of the day and transparently and securely access printers, file servers, remote hosts, etc., for the rest of that day without the inconvenience of explicitly authenticating herself to them. Initially developed at MIT, with ongoing refinement and extension under the auspices of the Internet Engineering Task Force (IETF), Kerberos has been adopted by many large companies, universities, and other organizations, and implementations are available for all major operating systems—Microsoft, which has included a largely compatible implementation of Kerberos 5 into all its operating systems starting with Windows 2000 [11], declared it will make Kerberos the center of its authentication infrastructure, hence phasing out proprietary solutions [12].

As typical of real-world systems, Kerberos 5 is not a single protocol, but a complex suite of interacting components: within the main protocol the client can request a variety of optional behaviors by setting appropriate flags and including certain sub-fields. The Kerberos key distribution center (KDC) can grant some or all of these requests, or reject them altogether by responding with an error message. The cipher to use for encrypted components is similarly subject to negotiation. Auxiliary protocols take care of administrative duties, such as changing the user password or propagating such changes to KDC replicas. Furthermore, Kerberos provides dedicated message formats to protect the communication with a given service once authentication has succeeded. This is significantly more complex than the abstract six line rendering of Kerberos whose correctness has been scrutinized for many years in the formal methods literature [13,14,15].

While not exhaustive, our work has touched several of these often ignored aspects

of a real-world protocol. In particular, our analysis has considered cipher negotiation, error messages, several optional behaviors, and cross-domain authentication, *i.e.*, the possibility of a client to authenticate to a server residing in a different administrative domain than its own. At this stage, we have not examined auxiliary protocols, application-level message formats, nor all the optional behaviors. We have also refrained from considering timestamps other than those used for authentication as this aspect has received significant attention in the literature [13,14] in the context of Kerberos 4.

This effort was made possible by substantial advances to the methodology of protocol verification. First, the inherent complexity of Kerberos 5, which is typical of real-world protocols, required a formalization language that combined an unambiguous representation with the flexibility to selectively abstract auxiliary protocol elements. We chose the language MSR [16,17,18] as our representation vehicle: MSR's semantics, based on linear logic and multiset rewriting, ensured the required rigor, while its powerful type system provided the needed abstraction support and flexibility. Second, the sheer size of Kerberos required a streamlined verification methodology that yielded manageable proofs in spite of the inherent complexity of this protocol. In this respect, we were the first to introduce distinct, but cooperating, mathematical tools in a theorem proving context to analyze the authentication and confidentiality aspects of a cryptographic protocol. This approach proved so effective as to allow us to perform the bulk of our work manually. The separation we thus pioneered has recently been formalized into a systematic methodology, which recognizes that authentication and confidentiality properties rely on radically different proof methods, based on different but interdependent logics [19]. Another technique that proved very useful is the coordinated refinement of specifications, properties and proofs: we initially established security results for a very abstract formalization of Kerberos 5; we then repeatedly added detail to this specification in such a way that we could propagate it to the properties and proofs and still maintain validity. Although this combination of proof methodologies was developed for the analysis of Kerberos 5, it is applicable to any protocol based on symmetric and/or public-key cryptography; we have not yet tried to extend it to constructs that rely on non-trivial equational theories such as Diffie-Hellman exponents. On the other hand, MSR is a generic specification language for distributed systems and has been used not only for the specification of numerous protocols [16,17,18], but also for modeling biological systems [20]. The present work on Kerberos 5 is however the largest MSR specification to date.

All in all, our analysis has shown that Kerberos 5 is a solid, well-designed protocol. Our results include theorems guaranteeing authentication, confidentiality of session keys, and structural soundness of the protocol when natural assumptions are met. We have nonetheless observed some curious behaviors—perhaps best described as “anomalies”—which do not violate the above fundamental properties of Kerberos, but which nonetheless deviate from the expected runs. Several of these anomalies appear only when taking into consideration low-level features such as

client-requested options. We also have exposed the fragility of cross-realm authentication with respect to the notion of trust: a single misbehaving Kerberos domain in a cross-realm authentication chain can create havoc.

The entire work reported here was carried out by hand. An MSR prototype implementation was started towards the end of this project and has only recently been completed [21]. It will assist us with future efforts by reducing the time needed to produce correct MSR specifications. This prototype does not yet provide automated support for the verification methodology described here, although we plan to study approaches for extending the tool in this direction.

### 1.1 Other related work

The material, summarized above and reviewed in more detail throughout the paper, directly informs our work. In addition to it, we will now review some other work related to both our methods and the Kerberos protocol.

Our approach to verification is based on theorem proving. Our proofs of security properties are inductive, showing that certain invariants (either absolute or conditional) hold at every step in all possible runs of Kerberos 5, including multiple, interleaved runs. Another example of this approach is in the work of Paulson *et al.*, with the Isabelle theorem prover. Their work included analysis of Kerberos 4 and proofs of various secrecy and authentication properties for that earlier version of Kerberos [13,14]. Our work was done by hand, but we intend to investigate the possibility of automating parts of it in the future. Indeed, following the results in [19], we suspect that the proof of our theorems can be automated to a large extent, although that technique still needs to be specialized to our setting. It may even be possible to automatically derive the authentication and confidentiality statements for specific classes of protocols.

Other analyses of Kerberos include one by Mitchell, Mitchell, and Stern [15] of a basic fragment of Kerberos 5 using the state-exploration tool Mur $\phi$ . This is an example of the model-checking (rather than inductive) approach to protocol verification; see also, *e.g.*, [22]. Mitchell *et al.*, found an attack against a restricted fragment of Kerberos 5; this attack was shown not to be achievable in the full protocol, or even in the fragment examined here. Yu, Hartman, and Raeburn [23] described a chosen-plaintext attack against Kerberos 4 and a related oracle-based attack on the original specification of Kerberos 5. These two attacks highlight a limitation of our approach, and of every approach based on a symbolic (Dolev-Yao) model of cryptography: our results here, which show that the design of the basic Kerberos 5 protocol does provide secrecy and authentication, rely on idealized cryptographic primitives, and therefore do not account for possible imperfections of implemented cryptosystems.

## 1.2 Outline of the paper

This paper has the purpose of summarizing the main contributions of this research project, in terms of both the verification methodology used and the security assurances obtained for Kerberos 5. It is organized as follows: in Section 2 we review Kerberos 5 at the most detailed level examined in our project. In Section 3 we informally present the main results of this work, concentrating on the major properties satisfied by the protocol, the proof techniques we relied upon, the anomalies we discovered and our findings on cross-realm authentication. Section 4 gives additional information on MSR and our verification methodology, and presents the details of a typical property and its proof. With the exception of this last section, which is rather technical even though it omits some of the lowest level details from theorems and proofs, the tone of this paper is intentionally kept informal, sometimes omitting minor details for the sake of clarity. The interested reader can however find full statements and complete proofs in other publications related to this project, in particular [16,24,25,26].

## 2 Kerberos 5 Basics

The Kerberos protocol [9] 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.

We will now gently review how Kerberos 5 [10], the current version of this protocol, achieves secure authentication based on a single logon. We first introduce the various principals taking part in a Kerberos exchange in Section 2.1. Then, in Section 2.2, we focus on the simple subprotocol that achieves basic authentication within a single administrative domain (or realm). In Section 2.3 we extend this description to capture how Kerberos supports authentication across different realms. We conclude this brief review in Section 2.4 by looking at some of the extended functionalities made available by the specification documents of Kerberos, and present in actual implementations.

### 2.1 Principals

Kerberos is implemented as a number of software agents, or principals, each handling a different aspect of authentication when a human *user* at her terminal requests a networked *service* such as a remote printer. First, the *client* process accepts

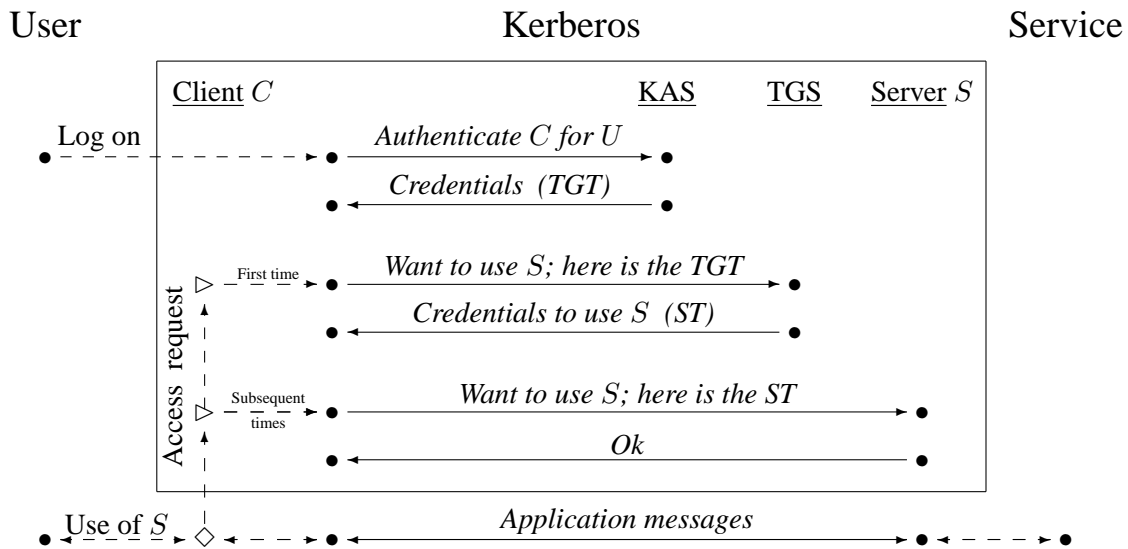


Fig. 1. An Overview of Kerberos Basic Authentication

the user's password and transparently handles the authentication aspect of each of her requests on her behalf. Dually, the service is mediated by a *server* process, for example a print server. Kerberos relies on two additional administrative agents together known as the Key Distribution Center (*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 are in general multiple clients and multiple servers. *Intra-realm authentication*, as this modality is known, has been extensively studied [13,15,24]. 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  $R_n$ ; this is part of the basic Kerberos 5

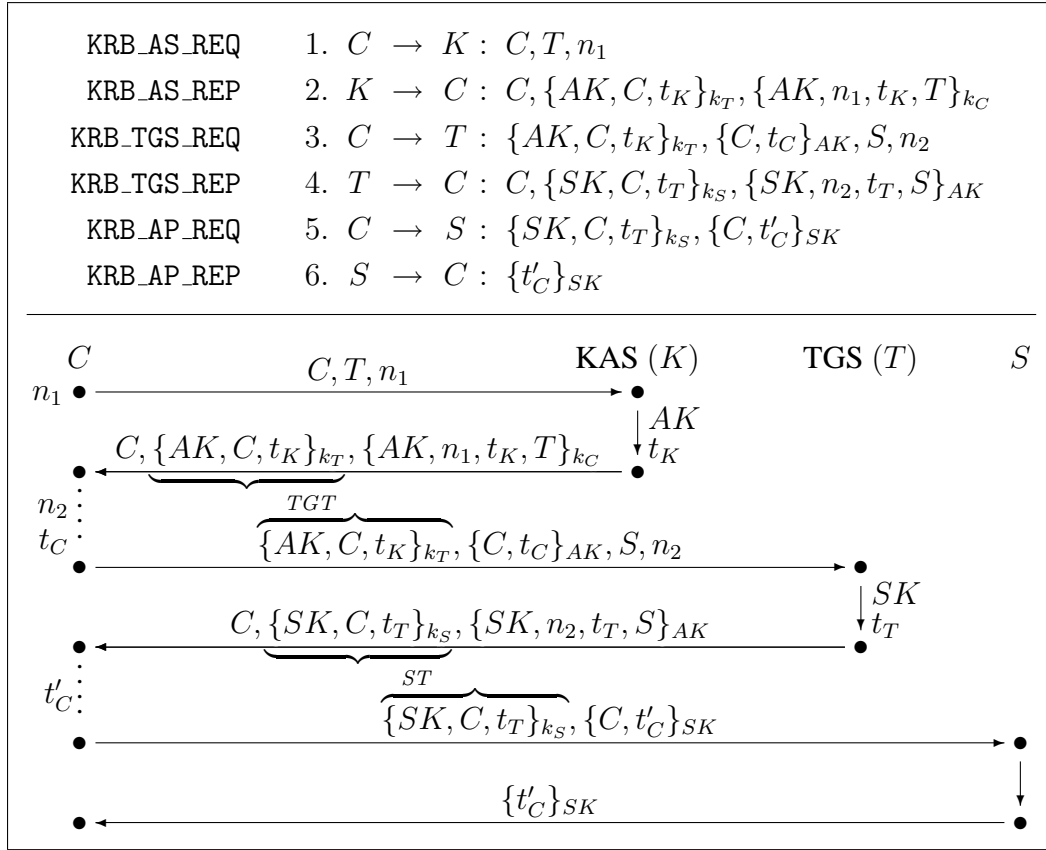


Fig. 2. Core Intra-Realm Message Exchange

specification, but to our knowledge has not been formally studied outside of this project. We now recall how the basic intra-realm protocol works and then account for cross-realm capabilities.

## 2.2 Core Authentication Exchange

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. For the moment, we will concentrate on the core authentication tasks, and therefore describe Kerberos 5 in a drastically simplified form. We will extend this basic skeleton with a variety of features of the actual protocol [10] in Section 2.4. 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. We write  $m, m'$  for the concatenation of messages  $m$  and  $m'$ , and  $\{m\}_k$  for the encryption of  $m$  with symmetric key  $k$ .

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 [10] 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 her 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 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 for the moment). 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, reducing the use of  $C$ 's long-term key  $k_C$ . (In a typical configuration,  $k_C$  is password-derived, and thus more vulnerable; at the least, it is more valuable than the shorter-term key  $AK$ .) 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 she is allowed to use  $S$ , the TGS sends a response with the same structure as the second message above except that the service ticket  $\{SK, C, t_T\}_{k_S}$  is now encrypted with the long-term key  $k_S$  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 sub-

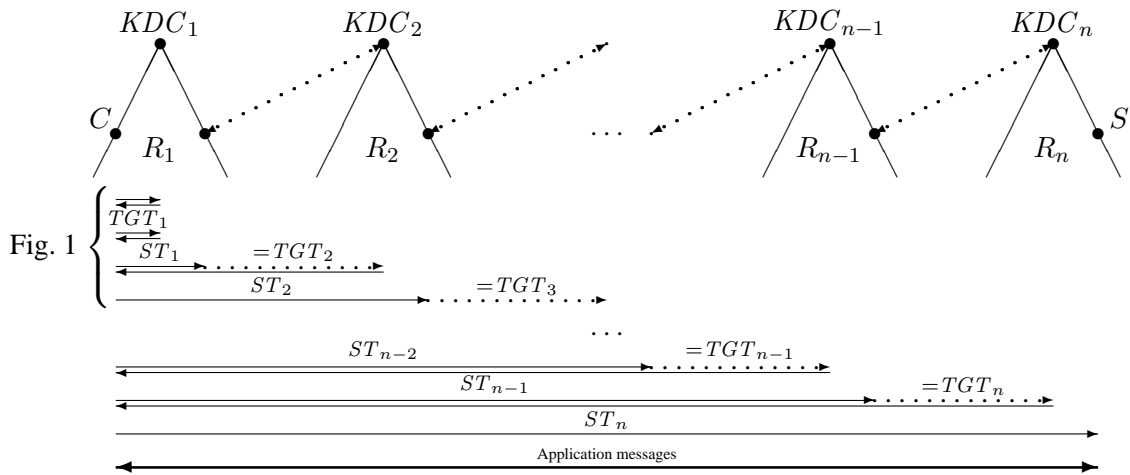


Fig. 3. Schematic Cross-Realm Authentication

sume 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.

### 2.3 Cross-Realm Support

Kerberos supports authentication across organizational boundaries by permitting clients and servers to reside on different realms [10,16]. A realm consists a group of clients, a KDC, and application servers, as seen in Section 2.2. 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 [27] 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 having the KDC of  $R_1$  register 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 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*. In Kerberos 5,  $C$ 's access to  $S$  may require traversing intermediate realms  $R_2, \dots, R_{n-1}$  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  $[R_1, \dots, R_n]$  is called the *authentication path* of  $C$ 's access to  $S$ . This high-level description is schematically represented in Figure 3. The top part of this figure shows the identification of a KDC in one realm with an application server in a neighboring realm. The bottom part of Fig. 3 gives a very abstract 'Alice and Bob' description of cross-realm authentication.  $C$  first obtains a TGT in her own realm, uses this to iteratively obtain cross-realm tickets, and then finally obtains a service ticket for the desired end server  $S$ ; the cross-realm tickets are viewed as service tickets in the realm in which they are granted, and as TGT's in the neighboring realm (whose KDC, viewed as an end server, granted the ticket).

The message structure discussed in Section 2.2 for intra-realm authentication is to a large extent adequate to support cross-realm authentication. Indeed, each communication with a new KDC in Figure 3 corresponds to a ticket granting exchange, as described in Figure 2: the service ticket returned by each TGS is forwarded to the TGS of the next realm who interprets it as a TGT. In order to support cross-realm authentication, Kerberos 5 refines the simplified message format discussed above by mandating that tickets contain a 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  $[R_1, \dots, R_{i-1}]$  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 is empty for intra-realm authentication. Therefore, the (cross-realm) service ticket created by the TGS of realm  $R_i$  has the following format:

$$ST_i = TGT_{i+1} = \{AK^i, C, t_{T_i}, \underbrace{[R_1, \dots, R_{i-1}]}_{\text{TRANSITED}}\}_{k_{R_i, T_{i+1}}}$$

where  $AK^i$  and  $t_{T_i}$  are the authentication key and timestamp generated by the TGS  $T_i$  of  $R_i$ , and  $k_{R_i, T_{i+1}}$  is the cross-realm key of the TGS  $T_{i+1}$  of  $R_{i+1}$  in  $R_i$ . The TRANSITED field is the only information available to the target KDC of a cross-realm authentication request to establish whether all previously traversed realms are trustworthy, and therefore to decide whether to grant access to the requested server. Indeed, although  $T_i$  makes a statement of trust when providing  $T_{i-1}$  with a

cross-realm key, it has no say on the partnerships that  $T_{i-1}$  forms and indeed may not trust some of these realms.

In spite of support in Kerberos as part of Windows and in other protocol suites (e.g., [28]), the literature on cross-realm authentication is scant and surprisingly old. The earliest paper in this arena appears to be [27], 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.* [29] undertook a general study of *interrealm 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 1990s. While that work was mostly concerned with designing a solid cross-realm authentication infrastructure, our effort focuses on its formal specification and analysis.

## 2.4 Real-World Features

The protocol we have described so far is a drastic simplification of Kerberos 5 as defined in the specification documents [10] and implemented in numerous systems. It does however capture the essence of what Kerberos authentication is about, and, as such, variants have been the object of formal analysis [13,14,15].<sup>1</sup> The goal of this project was however to get much closer to the actual specification of this protocol [10]. For this reason, we considered a number of features available in the actual protocol, but never before formally analyzed. The messages and exchanges examined in our most detailed work are displayed in Figure 4, where the grayed out portions identify aspects that we have not discussed so far. We will now introduce them in some detail.

**Options and Flags:** In the concrete Kerberos protocol, the client can ask any principal with whom she is communicating for some non-default behaviors by setting a number of *options* in her request. For example, she can ask for a ticket to be postdated for later use, or forwardable to another principal or renewable upon expiry. The responding server informs the client of which of these options she has granted by setting corresponding *flags* in the response (as well as in the ticket), or by directly implementing this behavior. This mechanism is generically described in Figure 4 by means of the components *KOpts* (options for the KAS), *TFlags* (flags from the KAS, appearing in the ticket for the TGS), *TOpts* (options for the TGS), *SFlags* (flags from the TGS, for the end server) and *SOpt* (options for the end server). Of the dozen or so flags/options supported by Kerberos, we have examined two:

---

<sup>1</sup> Some of this work, namely [13,14], examined Kerberos 4 [9], which has been gradually superseded by Kerberos 5.

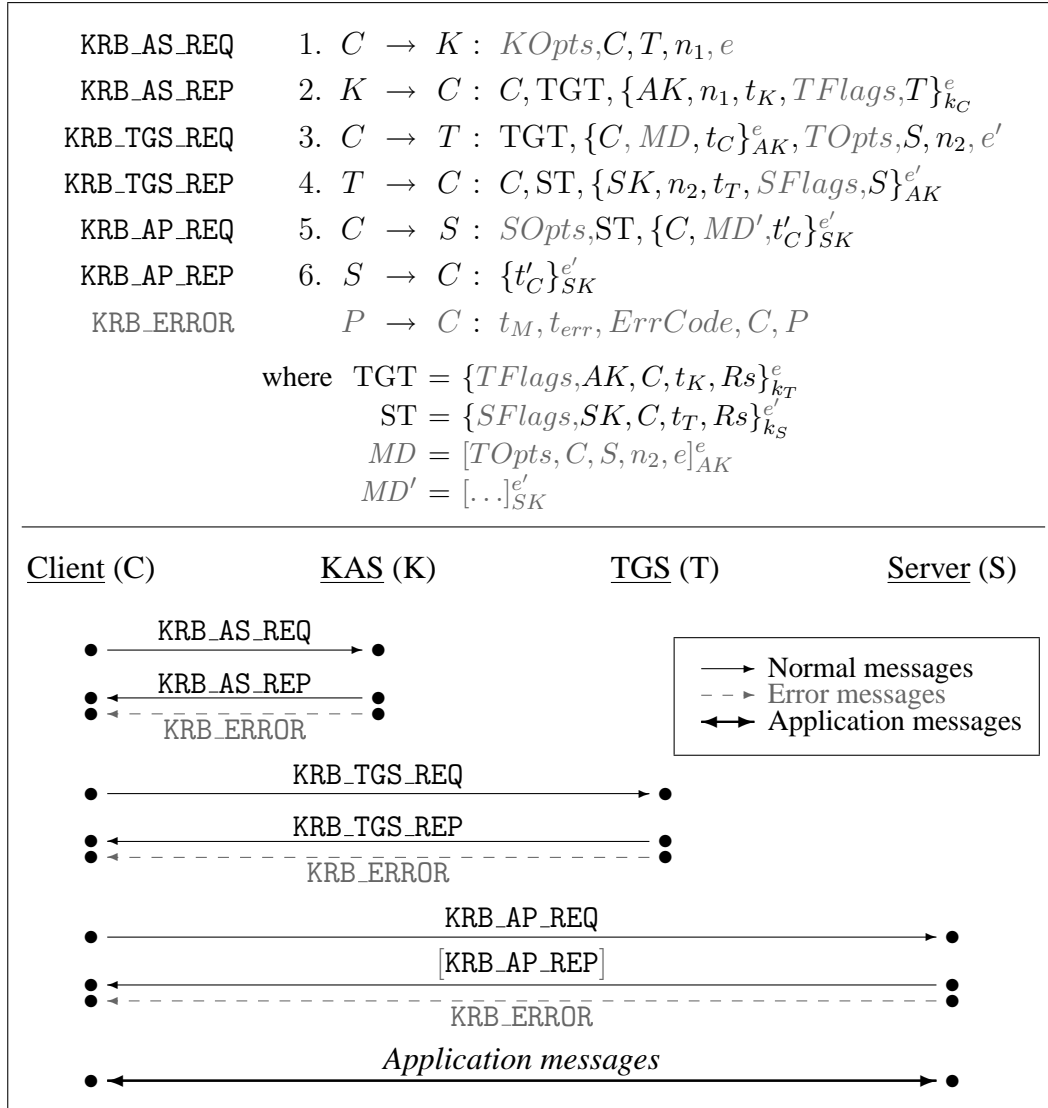


Fig. 4. Examined Features of the Kerberos 5 Specification

- **MUTUAL-REQUIRED:** with this  $SOpt$  option, the client requests that the end-server responds with the otherwise optional sixth message of Kerberos (shown as KRB\_AP\_REP in Figure 2).
- **ANONYMOUS:** this option [30] is used to allow asking the TGS not to include the client's name when constructing a service ticket but the generic string "USER". This option was removed from recent specifications of Kerberos, but it has just come again under consideration as a possible extension to the basic protocol [31].

**Error Messages:** Error messages are almost invariably ignored in academic abstractions of real world protocols. They however play an important role throughout the lifetime of a protocol: they are useful debugging tools during development and configuration, they can be tell-tale signs of attempted intrusion or other network problem, they also support the more mundane task of informing a principal that the combination of optional behaviors she has requested are unavailable

so that she may try the request again with different parameters. In Kerberos, error messages can be issued only in response to a client request, as described in Figure 4. They have the form

$$t_M, t_{err}, ErrCode, C, P$$

where  $C$  is the client,  $P$  is the issuing server,  $t_M$  is a timestamp copied from the request that caused the error,  $t_{err}$  is a timestamp generated by  $P$ , and  $ErrCode$  is one of a large number of error codes meant to identify what went wrong. Note that error messages are never encrypted.

**Encryption Types:** As often done in protocol verification, we have represented the encryption of a term  $m$  with a key  $k$  as  $\{m\}_k$ . In the real world, this is achieved by using some encryption algorithm. In contrast to earlier versions, Kerberos 5 supports multiple ciphers—it is indeed extensible as new ciphers can be accommodated if they satisfy some basic criteria [32]. Because various principals may implement different encryption algorithms, Kerberos provides some support for cipher negotiation. When requesting a ticket, the client can indeed ask for a particular encryption method to be used by including an *encryption type*, written  $e$  in Figure 4. If the KAS or TGS supports this cipher, it will use it in the reply: we express the fact that  $m$  is encrypted with key  $k$  using the algorithm identified by  $e$  as  $\{m\}_k^e$ , but omit  $e$  when unimportant or easily inferable from the context. If this server does not know about this cipher, it will return an error message which may lead the client to retry the request with a different encryption type.

**Message Digests:** The Kerberos specification [10] allows the KAS and TGS to return a keyed message digest to the client in order to ensure the integrity of the response. They are indicated as  $MD$  and  $MD'$  in Figure 4. Note that while  $MD$  has a fixed format, the contents of  $MD'$  depends of the particular service the client has requested. We use  $[m]_k$  for the message digest of  $m$ , keyed with  $k$ . We do not assume that this operation obscures  $m$ , but we do assume that  $[m]_k$  can only be constructed by a principal who knows both  $m$  and  $k$ .

While our analysis took these features of the concrete specification of Kerberos 5 into account, time and resource limitations kept us from examining other aspects of this protocol. In particular, we did not exhaustively scrutinize all the supported flags and options. We similarly left aside the optional pre-authentication phase, by which the KAS expects the client to partially encrypt the initial request. We ignored all temporal information unless it played a role in the authentication process: this includes ticket validity and expiration dates, for example. We did not consider the support for subkeys, by which client can specify the key with which she would like to communicate with an end server. Although the client and server applications are ultimately responsible for the messages exchanged after authentication has gone through, Kerberos provides a few message formats that ensure properties such as integrity and confidentiality: we did not attempt to model them. Finally, we did not consider the various auxiliary protocols of the Kerberos suite, which allow for example a user to changes her password with the KDC (and therefore the key  $k_C$

that the corresponding client extracts from it) or to propagate changes to replicas of the KDC. It should be emphasized that none of these omissions is due to limitations in our methodology, but only to the time and other resources we had available.

### 3 Results

Our analysis of Kerberos 5 has produced formal proofs that the protocol has a number of desirable (positive) properties, as well as some examples of protocol behavior which does not violate these properties but which is nonetheless curious. Among our positive results, the primary one—because Kerberos is an authentication protocol—is that the client and server are authenticated to each other under natural assumptions; we state these as *data origin authentication* properties [33] (*e.g.*, if a message appears on the network in some state of a trace, then earlier in the trace a specified principal put that message on the network). Our results also include confidentiality and structural soundness (*i.e.*, that a later round cannot take place unless the exchanges that precede it have occurred) properties for Kerberos 5. These are important in their own right, especially as some of the session keys may be used in future communications between protocol participants.

In order to show these properties for our formalization (in the MSR language), we needed to develop new proof techniques. We inductively defined two classes of functions that we then used to carry out a pair of intertwined inductions (arising from the interplay between secrecy and authentication); these functions were initially defined in [24] and used in our other work, but here we give a more complete perspective on their use. Our techniques are discussed in Section 3.2, and the positive properties we proved using them are outlined in Section 3.1.

While the positive properties we have proved provide minimum guarantees for Kerberos, the curious behaviors—‘anomalies’—described in Section 3.3 illustrate ways in which these guarantees cannot be extended. It is important to note both that these anomalies do not violate authentication and confidentiality of Kerberos and that it may be unwise to assume the protocol provides more than it claims.

Our positive results hold in the presence of a Dolev-Yao intruder, and we assume the powers of such an intruder in exhibiting the anomalies. This intruder has complete control of the network and may intercept messages, decompose them (in particular perform decryption if she knows the relevant key), copy messages, create new messages, and send her own messages to other protocol participants. Note that we do not intentionally leak keys to the intruder as was done in [13,14,34]. An MSR specification of the intruder can be found in Section 4.2, with additional details in in [24,26].

Finally, we state most of our results using a fairly abstract view of Kerberos. In a

related technical report [26] we give three formalizations of Kerberos, one which essentially corresponds to the presentation here and two that refine it in different ways. We also gave formal proofs at both an abstract and a detailed level for some of the properties we discuss here; it is notable that the properties and proofs for our more detailed formalization can be obtained by adding detail to the statements of properties and proofs for our abstract formalization. While we restrict our discussion to an abstract view of Kerberos, here we state our properties in the context of cross-realm Kerberos while [26] covered only intra-realm Kerberos. This extends the work in [16], which proved Property 8 but did not treat intra-realm authentication and confidentiality properties within the formalization that covered intra- and cross-realm Kerberos.

### 3.1 *Secrecy, Authentication, and Structural Soundness*

We begin with the positive properties that we have proved about Kerberos 5. As Kerberos is intended to provide authentication, it is important that we are able to prove that the client is authenticated to the TGS and application servers involved in a protocol run. We are also able to prove that the KAS, TGS, and server are authenticated to the client. We also prove secrecy, which is important for two reasons: first, it guarantees that the client and server share a fresh secret key, which may be used in later communication that we do not model here; secondly, secrecy properties are needed to prove authentication (the reverse is also true). Finally, we prove “structural soundness” of the protocol—*i.e.*, that if a protocol exchange takes place, then the various earlier protocol exchanges took place and did so in the expected order.

As just alluded to, the properties that we prove are interdependent. In the basic intra-realm case, authentication of the client to the TGS depends on the secrecy of the session keys generated by the KAS, and her authentication to a server depends on the secrecy of the session keys generated by TGS. We state these authentication properties informally as Properties 4, 5, and 7, and the secrecy properties as Properties 2, 3, and 6. The secrecy of session keys generated by the KAS depends on the secrecy of long-term keys, which we assume (these are set up out-of-band and are never sent as network messages). However, the secrecy of session keys generated by the TGS depends on authenticating the source of the key  $AK$  that the TGS shares with the client (which the TGS uses to encrypt the freshly generated key).

We may also authenticate the sources of some of the data that the client receives; one example of this is in Property 1, and others follow from the confidentiality properties that we show. Property 8 shows that the ticket presented to an application server contains accurate information about the realms involved in the authentication of the client to the server. Finally, Property 9 gives a structural soundness property of Kerberos; this covers the entire protocol, but we may also prove similar

guarantees that hold after initial segments of a protocol run.

In this section we present the English-language statements of the properties that we have proved for Kerberos 5. Some of these are stated more formally, in terms of our MSR formalization, in Section 4.4; the correspondence between the properties here and the theorems there is noted below. A more complete list of properties, the lemmas they depend on and their proofs can be found in [26].

Finally, it is important to note that the ticket-authentication properties implicitly assume that the tickets being authenticated did not exist at the beginning of the protocol run. (This is reasonable, as there are no expected circumstances in which the network would start in a state containing a ticket.) This assumption is explicitly noted in the formal theorems in Section 4.4.

### *3.1.1 Authentication Service (AS) Exchange*

We start with the authentication of the KAS to the client in the AS exchange; Property 1 is more formally stated as Theorem 3 in Section 4.4.

**Property 1 (Authentication of KAS to client).** If the client sees what appears to be a valid reply from the KAS and if her long-term key is secret, then the KAS in her realm generated a reply to a request that named the client.

Note that the client is not authenticated to the KAS in this exchange because the client's request is sent in the clear. However, as is shown below, if the client is able to create an authenticator that matches the ticket generated by the KAS then the client can be authenticated to the TGS.

### *3.1.2 Ticket Granting (TG) Exchange*

As the TG exchange is closer to the beginning of a protocol run than is the CS exchange, the properties in this section are simpler than the ones in the next section.

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

Property 2 covers the confidentiality of the intra-realm keys generated in the AS

exchange and used in the TG exchange; this is formally stated as Theorem 4 in Section 4.4.

**Property 2 (Confidentiality of intra-realm  $AK$ ).** If the intruder does not know the long-term secret keys ( $k_C$  and  $k_T$ ) used to encrypt the intra-realm session key  $AK$  generated by the authentication server  $K$  for use by  $C$  and  $T$ , then the intruder cannot learn  $AK$ .

Property 3 covers the confidentiality of the cross-realm keys generated in the TGS exchange for use in other TGS exchanges; this is formally stated as Theorem 5 in Section 4.4.

**Property 3 (Confidentiality of cross-realm  $AK$ ).** If the intruder does not know the cross-realm key  $k_{T,T'}$  or the key  $AK$ , each used by  $T$  to encrypt the cross-realm session key  $AK'$  for use by  $C$  and  $T'$ , then the intruder cannot learn  $AK'$ .

Property 4 shows that we may authenticate the origin of the ticket and authenticator used to request a service ticket. This is similar to an authentication theorem for purely intra-realm Kerberos, although in this case the TGT on which the ST is based may be either intra- or cross-realm.

**Property 4 (Authentication of request for ST).** If a TGS processes a request for a service ticket and if neither the long-term key encrypting the ticket nor any key shared between the client and the KAS/TGS which apparently produced the ticket is known to the intruder, then the ticket in the request is a valid intra- or cross-realm ticket and was generated by a KAS or TGS with whom the TGS shares a long-term key. Furthermore, the authenticator included in the request was generated by the client named in the ticket.

Finally, Property 5 shows that we may authenticate the origin of the ticket and authenticator in a request for a cross-realm ticket; this is formally stated as Theorem 6 in Section 4.4.

**Property 5 (Authentication of request for TGT).** If a TGS processes a request for a cross-realm ticket and if neither the key encrypting the ticket nor any key shared between the client and the KAS/TGS which apparently produced the ticket is known to the intruder, then the ticket in the request is a valid cross-realm ticket and was generated by a KAS/TGS with whom the TGS shares a long-term key. Furthermore, the authenticator included in the request was generated by the client named in the ticket.

### 3.1.3 Client/Server (CS) Exchange

We now move to properties of the CS Exchange; as this exchange parallels the TG Exchange, its properties parallel the properties we have proved for that exchange.

These properties are built on those stated above and may be viewed as our main positive results.

The first property for the CS Exchange gives conditions under which the session key shared by the client and server is not known to the intruder. This parallels Theorem 6.19 of [13] for Kerberos 4.

**Property 6 (Confidentiality of  $SK$ ).** If the intruder knows neither the long-term secret key  $k_{R_T,S}$  used by a TGS to encrypt the service ticket containing a new session key  $SK$  for a client to use with a server nor the session key used by the client to request the service ticket, then the intruder cannot learn  $SK$ .

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

**Property 7 (Authentication of request to server).** If the intruder does not know the long term key used to encrypt a service ticket for a client  $C$  to present to a server  $S$  then if  $S$  processes a request, ostensibly from  $C$ , containing this service ticket and the session key  $SK$ , then some Ticket Granting Server generated the session key  $SK$  for  $C$  to use with  $S$  and also created the service ticket. Furthermore, if the intruder never learns the session key which the Ticket Granting Server used to encrypt  $SK$  when sending the service ticket to  $C$ , then  $C$  created the authenticator.

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 8 (Correctness of transited realm data).** 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, and if neither keys between realms in  $\mathcal{R}$  nor  $S$ 's long-term key have been compromised, then some sequence of TGSes from realms in  $\mathcal{R}$ , starting with the TGS in  $C$ 's realm, authenticated  $C$  to  $S$  and the TGS of each realm in  $\mathcal{R}$  took part in this authentication.

### 3.1.4 Structural soundness of Kerberos 5

We close our report on the positive results with a property about the *structural soundness* of Kerberos 5, *i.e.*, that if the final exchange happens then the first two exchanges also happened and did so in the expected order. Because of the complexity of this property, we omit its formal statement and proof.

**Property 9 (Structural soundness).** If an application server  $S$  processes a request for service from a client  $C$ , and if the long-term keys for  $C$ , the KAS's in her realm, and between the realms listed in the ST presented to  $S$  are all secret, then  $C$  requested a TGT from the KAS in her realm and then later obtained an ST for  $S$ . If  $C$  and  $S$  are in the same realm, this ST was based directly on the TGT, while if  $C$  and  $S$  are in different realms then  $C$  used the TGT to obtain a sequence of TGTs, the last of which was used to obtain the ST for  $S$ .

## 3.2 Innovation

Results such as the ones informally reported in the previous section are routinely assessed of the protocol abstractions typically examined in the formal verification literature. Many of the methods in use would however be overwhelmed by the size and complexity of a real-world protocol specification. These considerations led to the design a streamlined methodology that proved robust enough to allow us to manually construct formal proofs for all the above results, a sample of which is reported in Section 4.4. We will now present an overview of the main aspects of this methodology, with selected details described more thoroughly in Section 4. The complete set of properties and their proofs can be found in other publications related to this project, such as [16,24,25,26]. Section 3.2.1 presents the case for the use of the language MSR as the basis of our formalization. Section 3.2.2 discusses the structure of our proofs and the mathematical tools that support it. Section 3.2.3 describes the use of refinement to incrementally produce a formalization and associated results for detailed specifications.

### 3.2.1 Formalization Language

A first task in the verification process consisted in translating the hefty IETF specification documents of Kerberos 5 [10] into a form more conducive to formal ver-

ification than English text. The inherent complexity of the Kerberos specification, typical of most real-world protocols, required a formalization language that was not only unambiguous, but also provided the flexibility to selectively abstract auxiliary protocol elements as well as to annotate formalized objects with important semantic information. We settled on the language MSR [16,17,18], although it had previously been used only on very simple protocols. This proved to be a valid choice as the linguistic features it provided not only permitted an easy formalization of the Kerberos specification, but also integrated quite well in our overall verification methodology. It should be noted however that although we expressed the protocol in MSR, our proof techniques are not bound to this formalism and can easily be ported to many other languages, provided they are sufficiently expressive. By the same token, MSR is a specification language, and is methodology-independent.

MSR is a flexible framework for specifying distributed systems, in particular cryptographic protocols. It uses strongly-typed multiset rewriting rules over first-order atomic formulas to express protocol actions and relies on a form of existential quantification to symbolically model the generation of nonces and other fresh data. Dependent types and subsorting provide powerful means to incorporate semantic annotations in a specification. Because it is based on multiset rewriting, MSR specifications are executable and a prototype implementation has recently been developed [21], although most of the Kerberos project predates it. MSR is also closely related to linear logic.

The mentioned prototype [21] is built on top of the Maude rewriting system [35,36]. It currently provides a simple environment that assists a user in the process of writing and debugging an MSR specification. It implements a full type checker for MSR, which allows verifying the syntactic correctness of a specification and therefore to gain confidence in it. Type reconstruction significantly simplifies writing MSR rules as the system automatically fills in the majority of typing information present in a rule. Finally, this prototype supports the controlled execution of rules, which falls short of model checking as no termination guarantee is provided. The MSR prototype was developed after most of the work discussed in this paper had been carried out (manually). In fact, parts of this work were used to debug the implementation. In the future, we will use this prototype as our prime tool to develop MSR specifications. Furthermore, we plan to have the prototype more closely cooperate with the numerous tools that have been developed on top of Maude over the years. Of particular interest are an interactive theorem prover, which we could automate aspects of our technique, but also several model checkers that could perform state exploration on the basis of an appropriately transformed MSR specification. These developments are however left for future work.

### 3.2.2 *Proof Methodology*

Our approach to protocol verification lies in the general area of theorem proving,

and in this respect it is reminiscent of Paulson’s inductive method [14,34]. It relies on MSR for providing a precise specification of both the protocol and the attacker’s capabilities in the form of a finite and fixed set of state transition rules (see Section 4 for details). Given an initial state, these rules implicitly define all the possible traces that are anchored at this state. Our theorems have one of two forms: (1) given certain conditions on the initial state, all reachable states are subject to a given invariant (this is the form of a typical confidentiality result); (2) given certain conditions on the initial state, if a reachable state in any trace has a given characteristic, then some specific event must have occurred earlier on that trace that caused it (this conditional form is typical of authentication results). The proofs of these results are inductive in the length of a trace and use two functions, *rank* and *corank*, to guide the induction. Each case of the induction corresponds to a protocol or attacker rule. We will now give a general overview of the way these functions are used, leaving a more detailed discussion to Section 4.3.

Both types of functions are defined inductively on terms and predicates in the MSR formalization of a protocol, but they may be easily defined for use in other formalisms. Our rank functions capture the amount of cryptographic work done, using a key  $k$ , starting with a specified message  $m_0$  (this is the *k-rank relative to  $m_0$* ). We use rank functions to prove our authentication results (although these often assume confidentiality results proved with corank functions). In general, we assume that no predicate (MSR fact) of positive  $k$ -rank relative to  $m_0$  was present at the start of a trace; if such a fact appears later in the trace, then some action must have increased the value of this function. The intruder cannot increase  $k$ -rank without knowing the key  $k$ . Thus, if we can prove that  $k$  is secret, then we know that the term  $\{m_0\}_k$  was produced by some honest principal. If there is a unique honest principal who could have produced this fact, then we have authenticated the origin of this datum.

Corank functions capture the minimum amount of cryptographic work that must be done, using keys from a set  $E$ , to extract a specified message  $m_0$  from some other message (this is the *E-corank relative to  $m_0$* ). We use corank functions to prove our confidentiality results, although these may depend on authentication results and rank functions. The MSR fact corresponding to the intruder’s knowledge of  $m_0$  has  $E$ -corank equal to 0 relative to  $m_0$  for every set  $E$  of keys. Thus, to prove the secrecy of  $m_0$ , we just need to find some set  $E$  of keys such that no fact in the trace has  $E$ -corank 0 relative to  $m_0$ . Assuming no such fact was present at the start of a trace, we simply need to show that no possible action will decrease this corank to 0. The intruder cannot decrease  $E$ -corank without knowing at least one of the keys in  $E$ ; if all of these keys are secret, then confidentiality becomes a matter of considering the actions of honest principals and their effects on  $E$ -corank relative to  $m_0$ .

As noted above, authentication of a message constructed using a key  $k$  may depend on the confidentiality of  $k$ . It is perhaps less obvious that confidentiality may depend on authentication as well. This arises when a freshly generated key is en-

encrypted using another key that has also been generated during the protocol run—if the encrypting key is not known to originate with a trusted participant, then the encrypted key need not be secret (as was discussed for Kerberos in Section 3.1). Thus, inductive arguments using rank and corank functions become intertwined in the course of analyzing a complex protocol such as Kerberos.

As suggested by their names, our rank and corank functions are inspired in part by Schneider’s work in CSP [37]; related ideas have been discussed in the context of strand spaces [38]. However, the CSP and MSR (co)rank functions are less alike than their names suggest. In CSP, rank functions are used to partition messages into those which an intruder might know and those which the intruder cannot know [39]; the question then becomes whether or not a suitable (CSP) rank function exists. In contrast, our rank and corank functions are defined inductively on (MSR) terms regardless of whether secrecy or authentication properties hold. Given a desired authentication or confidentiality property, it is straightforward to determine which (co)rank function is of immediate interest; the necessary work is then the determination of conditions that guarantee that this corank is never 0 (for confidentiality) or that only a specified principal could increase this rank (for authentication).

We may also draw connections between rank and corank functions and natural deduction-style message derivation (*e.g.*, as in [22]). In this view,  $k$ -rank relative to  $m_0$  captures the maximum number of encryption rules (using the key  $k$ ) that appear on a branch of a normalized derivation tree above a message  $m$  and below a use of  $k$  to encrypt exactly the message  $m_0$ . Similarly,  $E$ -corank relative to  $m_0$  captures the minimum possible number of decryption rules (using keys from  $E$ ) below a message  $m$  and above an instance of  $m_0$  in a derivation tree. Because of the inductive definition of terms in MSR, rank and corank functions are the natural measures to use for our work here.

The verification of any security protocol based on symmetric and/or asymmetric cryptography can be decomposed into alternating confidentiality and authentication analysis phased [19], the most interesting cases being key distribution protocols such as Kerberos, since their purpose is to transmit newly created keys to requesting parties over supposedly secure channels. Therefore, for all of these protocols, the notions of rank and corank are relevant and can be advantageously used in the proofs of authentication and confidentiality, respectively. As we will see in Section 4.3, protocol-specific definitions of rank functions can be derived automatically, while some aspects of the definition of the corresponding corank function still needs manual intervention.

### 3.2.3 Proof Refinement

Even when using rank and corank functions, the direct construction of a proof at the level of detail considered in this paper is a substantial and error-prone task. Since

the start of this project, we relied on the concept of refinement to make it more manageable [24].

We wrote an initial MSR specification of Kerberos at a very high level of abstraction, akin to the description presented in Section 2.2: it contained just enough elements to support basic authentication. Lacking the complexity of our detailed target, we could state its security properties and derive proofs for them with relative ease. We then refined this formalization to support additional features, which were propagated to the statements of properties and woven into their proofs. In particular, the more abstract proofs provided the structure of the more concrete proof, which allowed us to focus on integrating the additions in the proof.

Space considerations prevent us from demonstrating this technique in this paper. The interested reader is referred to [24,26] where several refinements are used.

We have not attempted to automate the process of proof refinement just discussed, but other authors have taken steps in this direction. Refinement is indeed a basic construct of the derivation methods based on CSP [37]. Closer to our setting, the protocol derivation framework of Pavlovic *et al.* embeds sophisticated rules to maintain provability through refinement [19,40].

### 3.3 Anomalies

In this section we describe some anomalous protocol behaviors that we have noted as we were analyzing Kerberos 5. These anomalies do not appear to pose any fundamental threat to the security of the protocol — the positive properties discussed in Section 3.1 indeed prove that Kerberos provides the expected strong confidentiality and authentication assurance — so the behaviors described in this section may be viewed as ‘interesting curiosities.’ Many of these anomalies were discovered when attempting to translate known theorems about Kerberos 4 [13,14] into results for Kerberos 5: anytime a proof broke down, an anomaly emerged. A detailed description of this process can be found in [26]. Some authors have devised methods to automatically propagate attacks and other anomalous behaviors through refinement [40]

The primary structural difference between versions 4 and 5 of Kerberos is the manner the KAS and the TGS transmit tickets. In Kerberos 4, the client receives tickets as part of the data encrypted under either her long term key ( $t_C$ ) or a session key ( $AK$ ). We saw that version 5 sends the tickets as a separate component without additional encryption. An intruder can exploit this message structure to tamper with the unprotected ticket in various ways (although she seems unable to cause serious damage by doing so). This structural difference between versions 4 and 5 is the root cause of most of the anomalies we note below.

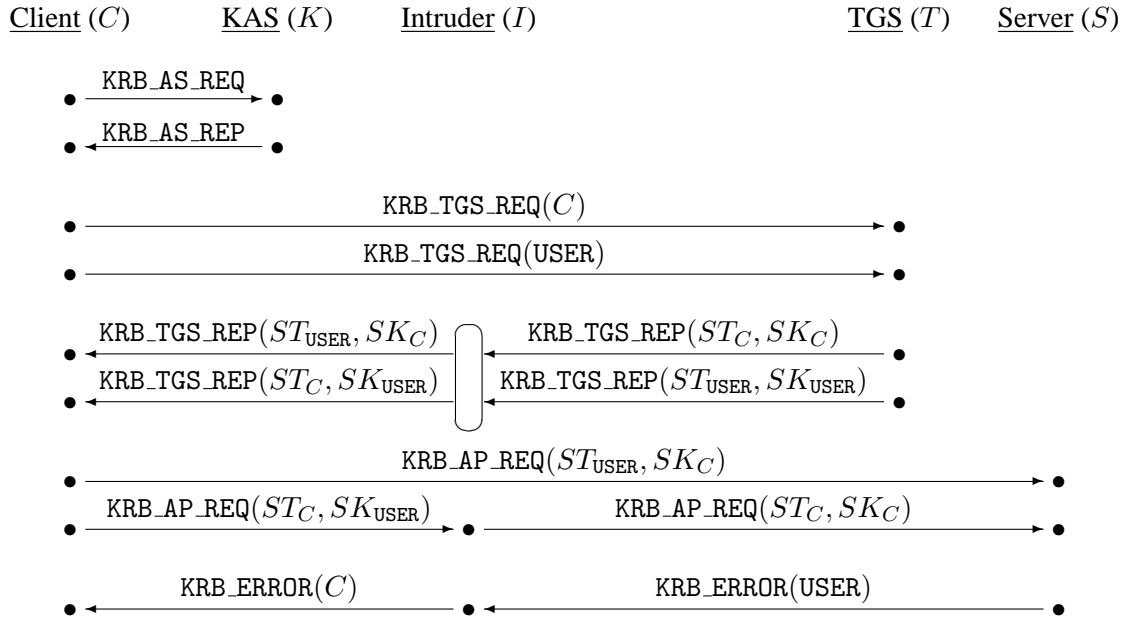


Fig. 5. Message Flow in the Anonymous Ticket Switch Anomaly.

### 3.3.1 Ticket Anomaly

In our first scenario,  $C$  sends her request message as usual, but the intruder  $I$  intercepts the reply from  $K$ .  $I$  replaces the ticket with a generic bitstring  $X$  ( $C$  does not expect to be able to read the ticket because it is encrypted) and stores the ticket in her memory. When  $C$  tries to send a request message to  $T$  (using the meaningless  $X$  instead of the ticket),  $I$  intercepts this message and replaces  $X$  with the original ticket from  $K$  and forwards the result (a well-formed request for a TGT) to  $T$ . A similar demeanor can take place when asking the TGT for a service ticket.

These anomalies do not appear to compromise any keys used in the protocol, but they are notable because they are counterexamples to the direct translation of properties established for Kerberos 4 [13, Theorems 6.22 and 6.23]. These anomalies are not prevented by the message digests  $MD$  and  $MD'$  introduced in Section 2, because they do not cover the tickets. It appears that these anomalies could be prevented if the digests were modified to also cover the ticket.

### 3.3.2 Anonymous Ticket Switch Anomaly

Another anomaly involving tampering with tickets makes use of the ANONYMOUS ticket option. This scenario, schematically reported in Figure 5, begins with a normal AS exchange. Afterward we suppose  $C$  desires two tickets from  $T$  for a given server  $S$ , one regular ticket and the other anonymous.  $T$  responds with the regular service ticket  $ST_C$  containing key  $AK_C$ , and the anonymous service ticket  $ST_{USER}$  containing the key  $AK_{USER}$  (along with the appropriate other components of these messages).  $I$  intercepts these messages and switches the tickets, causing  $C$  to have

incorrect beliefs about which (opaque) ticket contains her identity. When  $C$  sends requests using these tickets, both requests will be rejected by  $S$  (since the tickets do not match the service keys) unless  $I$  intervenes.  $I$  then replaces the authenticator encrypted with  $SK_{\text{USER}}$  with the authenticator encrypted with  $SK_C$ . The server can open the ticket in each of these messages, but only the key in the regular service ticket  $ST_C$  will open the accompanying authenticator. An error message is sent by  $S$ , in which  $I$  can replace the generic name USER with  $C$ 's name (since error messages are sent unencrypted).  $C$  may then believe that her anonymous request was accepted and her regular request rejected, when in fact the opposite is true.

This anomaly, a specific instance of the ticket anomaly above, seems to be avoided if the message digest in the KRB\_AP\_REQ messages are taken over the service ticket (so that the server  $S$  would be aware of any ticket switches).

After completing our initial work [24] on Kerberos, we brought the anomalies discussed here to the attention of the IETF Kerberos working group. At that time the protocol draft no longer specified anonymous tickets, but it was thought that they might be included again at a later date [41]. This possibility is now under active consideration in the IETF working group [31], and our work may inform this discussion [42].

While the use of the ANONYMOUS ticket option has historical significance, this anomaly can be enacted relative to most options that can appear in a ticket. The intruder can also rearrange the KRB\_AP\_REQ as to cause the end-server to issue a replay error message, which can be manipulated at will. This is the substance of the replay anomaly observed in [26].

### 3.3.3 Encryption Type Anomaly

Assume that the client's long term key  $k_C$  associated with a particular encryption type  $e$  has been compromised. While the loss of a long term key is quite serious, this example illustrates that this prospect is especially difficult to recover from in Kerberos 5. If  $C$  realizes the key associated to encryption type  $e$  has been compromised, she can send a KRB\_AS\_REQ message to  $K$  specifying a different encryption method ( $e'$  with key  $k'_C$ ). Since this request is sent in the clear,  $I$  can modify it to force a response using the compromised  $k_C$ .  $I$  may then intercept and use the credentials from  $K$ 's response.

This anomaly cannot be fixed by introducing a message digest, keyed with  $C$ 's long-term key in the KRB\_AS\_REQ message. Indeed,  $I$  could compute  $[C, T, n, e]_{k_C}^e$  using the compromised key  $k_C$  (since the hash is public), making the modified request  $C, T, n, e, [C, T, n, e]_{k_C}^e$  appear legitimate to  $K$ .

### 3.4 Cross-Realm Trust

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 (the KDC) behave honestly: a dishonest or compromised KDC trivially invalidates any authentication or confidentiality expectations. In the cross-realm case, a multitude of intermediate KDC's may be involved in the process of authenticating a client to a remote server, and it can no longer be assumed that every KDC will behave honestly. 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 [10] 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.

Our analysis of cross-realm authentication [16] has established some basic properties for Kerberos 5 (see Section 3.1). It also exposed several serious failures of authentication and confidentiality in the presence of compromised intermediate realms. In spite of the disclaimer in [10], this is important from a practical point of view as it enables an administrator to make informed decision about the consequence of supporting cross-realm authentication.

#### 3.4.1 TGSes Learn $AK$ and $SK$

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, each TGS is capable of learning all of the temporary keys shared between the client and the other TGSes from that point on in the authentication path. 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  $T_{i+1}$ . Since  $T_i$  generates  $AK^i$  it knows it and can store it in memory. The exchange of a new key  $AK^{i+1}$  between  $T_{i+1}$  and  $C$  is encrypted with  $AK^i$ , so  $T_i$  can learn  $AK^{i+1}$ .  $T_i$  can repeat this process for all of the TGSes 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 TGSes 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.

### 3.4.2 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 does not know  $C$  in the first place. Note that the rogue TGS is capable of fabricating the `TRANSITED` field, however Property 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.

### 3.4.3 Routing Vulnerability

With no intruder present, a client is capable of determining the authentication path by keeping track of the TGSes that she uses. When a client  $C$  receives a message of the form  $C, \{AK^i, C, t_{T_i}, [R_1, \dots, R_{i-1}]\}_{k_{R_i, T_{i+1}}} \{AK^i, n_{i+1}, t_{T_i}, T_{i+1}\}_{AK^{i-1}}$  from a TGS  $T_i$  (see Section 2.3), the client believes that the next TGS on the authentication path is  $T_{i+1}$ . 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 3.4.1 describes how every TGS on the authentication path is capable of learning  $SK$ , and in the process learning all of the session keys from that point on in the authentication path. The intruder can use the appropriate key to change the name of the TGS listed in the portion of the message encrypted with  $AK^{i-1}$ . Thus, if there is an intruder TGS  $T_I$ , then  $\forall i \geq I$ ,  $T_i$  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.

## 4 Methodology

We start with a brief review of the MSR specification language, which we have used to formalize the Kerberos protocol, and then discuss parts of one of our formalizations of Kerberos. Here we present a formalization that extends a previous abstract specification to allow for cross-realm authentication. As discussed above, we have studied even more detailed formalizations in [26], which focuses on intra-realm authentication, and which treats messages at the level of detail discussed in connection with the anonymous ticket switch anomaly given in Section 3.3.2. Because

the more detailed formalization is quite complex, although structurally similar—it adds error messages, but the usual message flow does not change—we omit it here but refer the interested reader to [26].

We continue in Section 4.2 with a brief discussion of the attacker model on which this work is based. In particular, we give excerpts of the detailed MSR formalization of the Dolev-Yao intruder as used in the proofs of our results.

We then present a formal definition of our rank and corank functions in Section 4.3. We conclude with some sample theorems—the statements using MSR of some properties from Section 3.1—and proofs from our positive results. These are stated at a level of detail corresponding to the formalization reviewed in Section 4.1; however, one of the notable results of [26] is that we proved the same English-language properties for our more detailed formalization by adding more detail to the corresponding theorems and proofs. (*I.e.*, for a given property and its corresponding theorem/proof in our abstract formalization, we could give a corresponding theorem/proof in our detailed formalization simply by adding details to the abstract theorem/proof.) The theorems and proofs presented here have not previously appeared; they parallel results at multiple levels of detail for purely intra-realm authentication [26], but our preliminary report on cross-realm authentication [16] focused on a different property, summarized here as Property 8.

While it would be straightforward to give additional detail for any single one of these sections, we prefer to use the same level of detail for each of them; giving the additional detail in all of Sections 4.1–4.4 would make this part of the paper unwieldy. The additional detail is presented in [26]. Finally, note that some of the symbols used in the formalization for timestamps, etc., differ from those used in the protocol overview in Section 2; this does not affect the formalization or results.

#### 4.1 MSR

MSR [16,17,18] is a flexible framework for specifying complex cryptographic protocols, including those structured as a collection of coordinated subprotocols. We use MSR 2.0 [17] for our formalization of Kerberos. This version of 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); this includes dependent types, discussed below. A recently completed MSR prototype [21] supporting type reconstruction, proof checking, and controlled execution will assist in the development of future specifications. We plan to integrate it with existing theorem proving tools to automate aspects of our methodology.

### 4.1.1 Terms and types

MSR represents network messages and their components as first-order terms. Thus the TGT sent from  $K$  to  $C$  is modeled as the term obtained by applying the binary encryption symbol  $\{-\}_-$  to the constant  $k_T$  and the subterm  $(AK, C, t_K, certPath)$ . This subterm is built using atomic terms and three applications of the binary concatenation symbol  $(-, -)$ . 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:

$$\begin{aligned} \text{msg} &: \text{type}. \\ \text{key} &: \text{type}. \\ \{-\}_- &: \text{key} \rightarrow \text{msg} \rightarrow \text{msg}. \end{aligned}$$

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

$$\text{nonce} : \text{type}. \quad \text{nonce} <: \text{msg}.$$

While simple types such as `msg` adequately describe simple categories, MSR provides means to capture more structured information [17]. For example, it can express the fact that a client  $c_3$  belongs to realm  $R_6$  directly within  $c_3$ 's type. This is achieved through the notion of dependent type, used as follows in defining the type of principals:

$$\text{principal} : \text{realm} \rightarrow \text{type}.$$

where `realm` is the type of realms. This has the effect of parameterizing the type of principals by a particular realm. Thus, if  $R_6$  is declared as a realm (by means of the assertion  $R_6 : \text{realm}$ ), then the type of  $c_3$  above is `principal  $R_6$`  (declared as  $c_3 : \text{principal } R_6$ ).

Kerberos relies on a number of specialized principals, which we formalize as different subsorts of `principal`: clients, servers, TGS's and KAS's are modeled by the type families `client`, `server`, `TGS`, and `KAS`, respectively, each parameterized by a realm. For example, the assertions

$$\text{KAS} : \text{realm} \rightarrow \text{type}. \quad \forall R : \text{realm}. \text{KAS } R <: \text{principal } R.$$

declares the type of KAS's and makes every KAS a principal within the same realm. For readability, we often omit the realm information when restricting attention to intra-realm authentication in a single realm [24,25,26]. 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.

Dependent types also support abstracting away the binding between keys and the

principals they were issued for. By parameterizing the type of a symmetric key by the names of the principals who share it, as in the following declaration,

$$\begin{aligned} \text{shK} &: \text{principal } R \rightarrow \text{principal } R \rightarrow \text{type.} \\ \forall R. \forall A, B &: \text{principal } R. \text{shK } A B <: \text{key.} \end{aligned}$$

we can express the fact that  $sk_{37}$  is a short-term key shared between a particular client  $c_3$  and a particular server  $s_7$  by means of the declarations  $c_3 : \text{client } R_6$ ,  $s_7 : \text{server } R_6$ ,  $sk_{37} : \text{shK } c_3 s_7$  (subsorthing reconciles the types of  $c_3$  and  $s_7$  with the types expected by shK).

While shK will tag the keys issued by the KAS and the TGS, we use dbK  $R A$  for the long-term keys that the KDC of realm  $R$  shares with principal  $A$ . Appropriate subsorthing declarations allow keys of the first type to be embedded in a network message for distribution, but prohibit the transmission of a long-term key (see [16,26] for details). While all keys used in the fragment of Kerberos specified here are atomic, composite keys could be handled by declaring appropriate types and operators.

We use the type Rset for the transited field; this is constructed from the empty field Rnil : Rset and the operator  $(\hat{\_}) : \text{realm} \rightarrow \text{Rset} \rightarrow \text{Rset}$ .

A comprehensive discussion of the typing declarations used in our analysis of Kerberos 5 can be found in [16,26]. While we found this typing layout convenient, it was by no means the only possibility. Indeed, MSR does not impose any restriction on what types are declared and for what purpose (however, type checking requires that they are used consistently). In particular, we could have simulated a completely untyped setting by defining a single type, say msg, and declaring every object in our formalization of this type. Doing so would have forced us to maintain more book-keeping information as explicit facts within rules (see next section) rather than segregating it within the more structured types we used. As observed in [17,43], this would have made our specification harder to get right since errors now automatically caught by the type checker would then be subject to visual inspection only. Furthermore, the specification task itself would have been more complex as MSR supports (and the MSR prototype [21] implements) a sophisticated form of type reconstruction that allows omitting the vast majority of typing annotations, in practice. In order to maintain conceptual simplicity, all declarations reported in this paper are fully type-reconstructed.

In this work, we did not look for type flaw attacks, mostly because this would have meant conducting our investigation at a much lower level than we were prepared to do: only a specification of the full ASN.1 syntax of Kerberos 5 [10] would have allowed us to draw any meaningful conclusion. Furthermore, because of the redundancy built into this syntax, we doubt the required extensive low-level specification would have yielded any interesting result. It should be noted however that the very typing infrastructure of MSR provides one of the best approaches available for the

discovery of type flaw attacks [44]. Indeed, subsorting offers a convenient method to declare what combination of fields could reasonably be confused with each other (*e.g.*, a nonce and a key) and which certainly could not (*e.g.*, a nonce and a ticket). This contrasts with other strongly typed languages, which typically disallow any type confusion, and all untyped formalisms, for which any two entities can be confused.

#### 4.1.2 States, rules, roles, and the formalization of Kerberos 5

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  $N(\{AK, C, t_K\}_{k_T})$  indicates that ticket  $\{AK, C, t_K\}_{k_T}$  is present on the network. The network predicate  $N$  is declared as  $N : \text{msg} \rightarrow \text{state}$ .

A protocol consists 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 the facts it replaces them with to produce the next state. For example, Fig. 6 describes the actions of the KAS. Ignoring for the moment the leading  $\forall K :$  KAS  $R_C$  and the outermost brackets ( $[\dots]$ ) leaves us with a single MSR rule—rule  $\alpha_{2.1}$ , as indicated above the arrow—that we will use to illustrate characteristics of MSR rules in general.

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

Fig. 6. The KAS's Role in the Authentication Service Exchange

Rules are parametric, as evidenced by the leading string of typed universal quantifiers: actual values need to be supplied before applying the rule. The middle portion ( $[\dots] \Rightarrow [\dots]$ ) describes the transformation performed by the rule: it replaces states containing a fact of the form  $N(C, T, n_1)$  with states that contain a fact of the form  $N(C, \{AK, C, t_K, \text{Rnil}\}_{k_{R_C, T}}, \{AK, n_1, t_K, T\}_{k_{R_C, C}})$  but which are otherwise identical. The existential marker “ $\exists AK : \text{shKCT}$ ” requires  $AK$  to be replaced with a newly generated symbol of type  $\text{shK } C T$ ; this is how freshness requirements are modeled in MSR. The *if* ... line lists additional constraints, used as in [16], that

must be satisfied by a state if the rule is to be applied to it:  $\text{Valid}_K$  models a validity check (implementing local policy) performed by  $K$ , and  $\text{clock}_K$  looks up  $K$ 's local time. Unlike the facts on the left-hand side of the rule, these constraints are consulted but neither added to nor removed from the state as a result of applying the rule. Section 4.1.3 describes the formal assumptions we make about the constraints we use here.

Rule  $\alpha_{2.1}$  completely describes the behavior of the KAS; in general, multiple rules may be needed, as when modeling the actions of the client in the AS exchange. 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 “ $\forall K : \text{KAS}$ ” above the brackets in Figure 6.

A 2-rule role describing the client's actions in the AS exchange is shown in Figure 7. Rule  $\alpha_{1.1}$  describes the client's sending of her request; rule  $\alpha_{1.2}$  describes her processing of the reply. The predicate  $\text{MemASE}$  is used to coordinate the two rules: firing rule  $\alpha_{1.1}$  creates the predicate, which saves data about the request; firing rule  $\alpha_{1.2}$  consumes this predicate, ensuring that the processed reply corresponds to an *earlier* request. The predicate  $\text{Auth}_C$ , created when the client processes the KAS's reply, caches the ticket and key  $AK$  for use in the TG exchange. (In particular, this predicate appears on the left-hand side of rule  $\alpha_{3.1}$ , given in Appendix A, by which  $C$  sends a request message to  $T$ .)

$$\begin{array}{c}
 \forall C : \text{client } R_C \\
 \left[ \begin{array}{l}
 \forall T : \text{TGS } R_C \\
 \quad \exists n_1 : \text{nonce} \\
 \quad \cdot \xrightarrow{\alpha_{1.1}} \left[ \begin{array}{l}
 \text{N}(C, T, n_1) \\
 \text{MemASE}_C(T, n_1)
 \end{array} \right] \\
 \\
 \forall T : \text{TGS } R_C \quad \forall k_{R_C, C} : \text{dbK } R_C C \quad \forall AK : \text{shK } C T \quad \forall X : \text{msg} \\
 \forall n_1 : \text{nonce} \quad \forall t_K : \text{time.} \\
 \left[ \begin{array}{l}
 \text{MemASE}_C(T, n_1) \\
 \text{N}(C, X, \{AK, n_1, t_K, T\}_{k_C})
 \end{array} \right] \xrightarrow{\alpha_{1.2}} \text{Auth}_C(X, T, AK)
 \end{array} \right]
 \end{array}$$

Fig. 7. The Client's role in the Authentication Service Exchange

The remaining roles in this formalization are presented in Appendix A.

An MSR specification may appear to have little in common with the Alice&Bob notation traditionally used to discuss cryptographic protocols. Yet, the skeleton of a set of MSR roles is readily obtained from this informal notation: there is one role for each involved principal and every sequence of a receive followed by a send gives rise to a rule. The MSR specification is completed by the choice of an appropriate

typing infrastructure (the recent MSR prototype [21] will automatically infer the type of most variables present in a rule) and memory predicates as needed. However, there is no push-button way to derive an MSR specification from a protocol described using the Alice&Bob notation in general since the latter is ambiguous, which is one of the reasons why MSR was introduced in the first place [16,17,18].

### 4.1.3 Constraint details

We now summarize the constraints that are used in our formalization of Kerberos; some of these appear in the rules that are given in Appendix A.

The constraints  $\text{Valid}_P$ , for various principals  $P$ , implement the local policies of KAS's, TGS's, and application servers. In particular, these model whatever conditions these various servers impose (beyond those inherent in the protocol) on the client's requests.

The constraints  $\text{clock}_P$ , for various  $P$ , check the local clock of the principal  $P$ ; if  $\text{clock}_P(t)$  holds, then the time on  $P$ 's local clock is  $t$ .

The constraint  $\text{DesiredHop}$  implements a client's local policy in choosing the next realm for which it will request a cross-realm ticket. In particular, if  $\text{DesiredHop}(C, S, R_T, R_n)$  holds then  $C$  will attempt to obtain a ticket for realm  $R_n$  if she already has a TGT for realm  $R_T$  and her goal is to obtain a service ticket for the server  $S$ .

The constraint  $\text{CloserRealm}$  implements a TGS's selection of a realm that closer to the destination realm requested by the client than the TGS's own realm. In particular,  $\text{CloserRealm}(T_i, T_n, T_{i+1})$  holds if, when  $T_i$  is asked to provide a ticket for  $T_n$ , the best (*i.e.*, closest to  $T_n$ ) realm for which it can provide a TGT is  $T_{i+1} \neq T_i$ . In general, if  $T_i$  shares a key with  $T_n$  then  $T_{i+1}$  will be  $T_n$ ; otherwise,  $T_i$  will use a hierarchical approach or a hardcoded path to determine  $T_{i+1}$ . We assume that realms are organized so that the iterated use of  $\text{CloserRealm}(T_i, T_n, T_{i+1})$  by TGS's in different realms never produces a cycle in the authentication path, and that if  $T_n \neq T_i$  then  $T_{i+1} \neq T_i$ .

### 4.1.4 Execution Semantics

We will now expand on the execution model of MSR by concentrating on a high-level definition of execution step and the critical notion of a trace. The interested reader is invited to consult [45] for a detailed definition of execution, and is referred to [7] for a discussion of how these two levels relates to each other.

Simplifying somewhat from [7], the execution semantics of MSR operates by transforming *configurations* of the form  $\langle S \rangle_\Sigma^R$ , where the *state*  $S$  is a multiset of ground predicates (as introduced in the previous section), the *signature*  $\Sigma$  keeps track of the

symbols in use (with their type), and the *active role set*  $R = (\rho_1^{a_1}, \dots, \rho_n^{a_n})$  records the remaining actions of the currently executing roles ( $\rho_i$ ), and who is executing them ( $a_i$ ).

The basic execution step is expressed by judgments of the form  $\mathcal{P} \triangleright C \longrightarrow C'$  where  $\mathcal{P}$  is the protocol specification, and  $C$  and  $C'$  are consecutive configurations. This judgment is defined by the following two rules (simplified, see [45]):

$$\frac{}{(\mathcal{P}, \rho) \triangleright \langle S \rangle_{\Sigma}^R \longrightarrow \langle S \rangle_{\Sigma}^{R, \rho^a}} \text{ inst}$$

$$\frac{\mathcal{P} \triangleright \langle S, [\theta] lhs \rangle_{\Sigma}^{R, ((lhs \rightarrow \exists \vec{x}. rhs), \rho)^a}}{\mathcal{P} \triangleright \langle S, [\theta, \vec{c}/\vec{x}] rhs \rangle_{\Sigma, \vec{c}}^{R, \rho^a}} \text{ rw}$$

The first rule prepares a role for execution by inserting it in the active role set: it associates it with the principal  $a$  that will be executing it. Note that the same role can be loaded arbitrarily many times by any principal (subject to some typing limitations), which provides support for the concurrent execution of multiple sessions and therefore multi-session attacks. The second inference describes the application of a rule  $r = lhs \rightarrow rhs$ : if an instance  $[\theta]lhs$  of its left-hand side appears in the state, it is replaced by the corresponding instantiation of the right-hand side after instantiating the existential variables  $\vec{x}$  with new constants  $\vec{c}$ . The rule  $r$  is removed from the active role set, and  $\vec{c}$  is added to the signature.

An *abstract execution step* is a quadruple  $C \xrightarrow{r, \iota} C'$ , where  $C$  and  $C'$  are consecutive configurations,  $r$  identifies the rule from  $\mathcal{P}$  being executed, and  $\iota$  stands for the overall substitution: it comprises  $\theta$  above, which deals with the universally quantified variables of  $r$ , and  $\vec{c}/\vec{x}$  which handles its existential variables. An abstract execution step is just a compact yet precise way to denote rule application. It is reasonable to think about it as a partial function from  $C$ ,  $r$  and  $\iota$  to  $C'$ . We say that  $r$  is *applicable* in  $C$  if there is a substitution  $\iota$  and a configuration  $C'$  such that  $C \xrightarrow{r, \iota} C'$  is defined.

A *trace*  $\mathcal{T}$  is then a sequence of applications

$$C_0 \xrightarrow{r_1, \iota_1} C_1 \xrightarrow{r_2, \iota_2} \dots \xrightarrow{r_n, \iota_n} C_{n+1}$$

Here,  $C_0$  is the *initial configuration* of  $\mathcal{T}$ . In the context of Kerberos, the state component  $S_0$  of the initial configuration contains only the predicates used in constraints and the intruder's knowledge (see next section); in particular, it does not contain any network message or memory predicate. The initial signature  $\Sigma_0$  within  $C_0$  contains the name of all principals and their type (client, server, KAS, *etc.*), their keys, realms, *etc.* The initial active role set  $\rho_0$  is empty. Because execution in any (networked) computer system proceeds by discrete steps and started at a specific moment in time (in the last 60 years), we only need to consider traces that are finite. This also means that a generic trace will contain a finite, although a priori unbounded, number of applications of inference *inst* for the same role  $\rho$ .

While we rely on the notion of sequence in describing traces, this definition could be generalized to a lattice with minimum  $C_0$  and maximum  $C_n$  to account for action independence. We will however stick to sequences for simplicity.

## 4.2 Intruder Model

Our analysis of Kerberos 5 is based on a variant of the classic Dolev-Yao intruder model [2,3]. This attacker can traditionally intercept and originate network traffic, encrypt and decrypt captured messages as long as he knows the key, concatenate and split messages at will, generate certain types of messages (*e.g.*, keys and nonces) but not others (*e.g.*, principals), and access public data such as principal names. Because we are considering cross-realm authentication, we need to extend this list with a few additional capabilities, in particular compromising a realm (which gives the intruder access to any dbK in that realm) and creating fake clients and servers as well as their long-term keys in compromised realms.

This set of capabilities is given a precise specification in MSR. For this purpose, the knowledge of the intruder is modeled as a collection of facts  $I(m)$  (“the intruder knows message  $m$ ”) distributed in the state. The intruder himself is represented as the distinguished principal  $I$  by means of the declaration  $I : \text{principal}$ . Each capability is expressed by means of a one-rule role that can be executed only by  $I$ . For example, the specifications of network message interception, message splitting and nonce generation have the following form:

$$\begin{array}{c} I \\ \left[ \begin{array}{l} \forall m : \text{msg.} \\ N(m) \xrightarrow{INT} I(m) \end{array} \right] \quad \begin{array}{c} I \\ \left[ \begin{array}{l} \forall m_1, m_2 : \text{msg.} \\ \left[ \begin{array}{l} I(m_1) \\ I(m_2) \end{array} \right] \xrightarrow{CPM} I(m_1, m_2) \end{array} \right] \end{array} \quad \begin{array}{c} I \\ \left[ \begin{array}{l} \cdot \xrightarrow{NG} \exists n : \text{nonce. } I(n) \end{array} \right] \end{array}
 \end{array}$$

Notice that these roles identify  $I$  as their owner (above the left brace) rather than a generic principal (introduced by a universal quantifier in the previous section).

Because our specification of Kerberos distinguishes among different types of keys (short-term shK keys and long-term dbK keys), the generic capability “the intruder can decrypt an encrypted message assuming he has the encryption key” must be split among two rules:

$$\begin{array}{c} I \\ \left[ \begin{array}{l} \forall R : \text{realm} \quad \forall C : \text{client } R \quad \forall A : \text{ts } R \\ \forall k : \text{shK } R C A \quad \forall m : \text{msg} \\ \left[ \begin{array}{l} I(\{m\}_k) \\ I(k) \end{array} \right] \xrightarrow{SDC} I(m) \end{array} \right] \quad \begin{array}{c} I \\ \left[ \begin{array}{l} \forall R : \text{realm} \quad \forall A : \text{tcs } R \\ \forall k : \text{dbK } R A \quad \forall m : \text{msg} \\ \left[ \begin{array}{l} I(\{m\}_k) \\ I(k) \end{array} \right] \xrightarrow{NG} I(m) \end{array} \right] \end{array}
 \end{array}$$

Recall that  $ts$  stands for either a TGS or a server, and that  $tcs$  is either a  $ts$  or a client. The rules for encryptions are similar.

The rules implementing the other traditional Dolev-Yao intruder capabilities are defined similarly. A complete list can be found in [26]. Modeling cross-realm authentication requires an additional set of rules to express capturing a realm, populating it with fictitious clients and servers, *etc.* For example, realm capture is modeled as follows:

$$\left[ \begin{array}{l} \forall R : \text{realm} \quad \forall A : tcs\ R \quad \forall k : dbK\ R\ A \\ \cdot \Rightarrow I(k) \\ \text{if } \text{compromised}(R) \end{array} \right]$$

Here, the fact  $\text{compromised}(R)$  marks a realm as compromised by the intruder. This rule states that the intruder has access to all long-term keys held by the KDC of a compromised realm. This simple rule is at the basis of our analysis of cross-realm authentication in Kerberos 5 [7]. Additional cross-realm intruder rules can be found in [7].

The intruder rule excerpts given above are a simple transliteration of the natural-language Dolev-Yao capabilities, with typing adding minimal complication. This format is appropriate to the verification technique described in this paper, which is based on theorem proving. Clearly, rules of this form would be totally inadequate for a methodology rooted on state exploration as they allow the intruder to generate an infinite set of messages (most of which are of no use to a Kerberos principal) or even to run in place by repeatedly performing actions and undoing them later. Protocol-specific intruder rules that do not suffer from this deficiency can be devised and expressed in MSR, as this language is open-ended and methodology-independent. We have done so for small protocols in unpublished work. Such rules would be appropriate for a model-checking approach.

### 4.3 Rank and Corank

We now turn to the definition of the rank and corank functions discussed in Section 3.2. Recall that they are used to guide the inductive proofs of authentication and confidentiality, respectively, with each protocol and intruder rule corresponding to a case in the induction. In order to use them, we need to define their values on the antecedent and consequents of a rule. Just as multisets collect facts and facts are built up from atomic terms in the language of the protocol, we inductively define rank and corank functions starting with their values on atomic terms and then defining the effects on these values of the operations used to build non-atomic terms. The extension of the definition of corank from terms to facts requires some

human consideration of the predicates in the protocol signature; we note some general principles which appear to be applicable to this process.

### 4.3.1 Rank

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

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

If  $t$  is atomic, then no work has been done to encrypt the message  $m_0$  and we set the rank equal to 0. If  $t$  is exactly the message  $\{m_0\}_k$  we set the rank equal to 1. Encrypting any message of positive  $k$ -rank with the key  $k$  increases the rank by 1 as additional work has been done using  $k$ , while encryption with  $k' \neq k$  has no effect on  $k$ -rank. Similarly, encrypting a message (other than exactly  $m_0$ ) that has  $k$ -rank 0 with the key  $k$  leaves the  $k$ -rank at 0. The rank of the concatenation of two messages equals the larger of the ranks of the constituent messages. The  $k$ -rank of message digests, which play a role in the detailed formalization in [26] but not here, is defined as for encryptions, namely a message digest function keyed with  $k$  has the same effect as encryption using the key  $k$ . We will be concerned primarily with whether or not the  $k$ -rank relative to  $m_0$  of a message is positive, *i.e.*, whether or not  $\{m_0\}_k$  is contained within the message.

The extension of rank from terms to facts is straightforward; intuitively, the number of nested encryptions of  $m_0$  using  $k$  which must have occurred to produce a certain predicate equals the maximum number of such encryptions which were needed to produce one of the arguments of the predicate. Formally, for  $k$  a key,  $m_0$  and  $m$  of type msg, and  $t, t_i$  terms, and  $P$  any predicate in the protocol signature, we define the  $k$ -rank of a fact  $F$  relative to  $m_0$  ( $\rho_k(F; m_0)$ ) by  $\rho_k(P(t_1, \dots, t_j); m_0) = \max_{1 \leq i \leq j} \rho_k(t_i; m_0)$ . In particular, for network facts we have  $\rho_k(\mathbf{N}(m); m_0) = \rho_k(m; m_0)$  and for terms in the intruder's memory we have  $\rho_k(I(t); m_0) = \rho_k(t; m_0)$ . For a multiset  $A$  of finitely many distinct facts, we define the  $k$ -rank of  $A$  relative to  $m_0$  by  $\rho_k(A; m_0) = \max_{F \in A} \rho_k(F; m_0)$  if  $A \neq \emptyset$ , and let  $\rho_k(\emptyset; m_0) = 0$ .

Given a rule

$$[F_1, \dots, F_i] \implies \exists x_1 \dots \exists x_n [G_1, \dots, G_j],$$

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

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

(=,  $\geq$ , *etc.*, respectively).

We now outline how the notion of rank is used to prove data origin authentication for a protocol. This task is divided in two parts: we first show that the intruder specification is unable to encrypt a message  $m_0$  of interest with a key  $k$  unless he possesses  $k$ , and then we show that if  $\{m_0\}_k$  is ever created, then an honest principal must have done so as part of the protocol. This is expressed by the following property.

Any reasonable formulation of the intruder should be such that the intruder cannot do cryptographic work using the key  $k$  (as measured by relative  $k$ -rank) without possessing the key  $k$ .

**Property 10.** If an intruder rule  $r_I$  can increase  $k$ -rank relative to  $m_0$ , then the left hand side of  $r_I$  contains  $I(k)$ .

The proof proceeds by cases on the rules defining the intruder capabilities. As expected, this holds for our formalizations of the Dolev-Yao intruder in Section 4.2. This proof can be found in [26].

Our approach to data origin authentication is outlined by the following theorem, which might be viewed as a loose analog of Schneider's rank function theorem [37] for our rank functions; as with rank in general, this is straightforward to extend to the case of message digests and other cryptographic operators [26].

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

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

We then authenticate the origin of  $\{m_0\}_k$  (assuming this was not present at the beginning of the trace—neither as intruder knowledge or in a message in transit in particular) by ensuring the confidentiality of  $k$ , invoking Property 10, and then determining which honest principal(s) could create  $\{m_0\}_k$ . One might also consider traces in which a certain rank is at most  $i$  in the initial state and then a fact of rank  $j > i$  appears in a later state; however, we do not need this here.

### 4.3.2 Corank

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

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

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

If  $t$  is atomic then no work using keys from  $E$  is required to obtain  $m_0$  if  $t = m_0$ , while no amount of such work can extract  $m_0$  from  $t \neq m_0$ . The number of decryptions using keys from  $E$  needed to obtain  $m_0$  from  $\{m\}_k$  is the same as or 1 more than the number needed to obtain  $m_0$  from  $m$ , depending on whether  $k \notin E$  or  $k \in E$ . A message  $m_0$  can be extracted from the concatenation of two terms by extracting it from one of these two terms (since we are assuming that  $m_0$  is atomic), whence the final case. When message digests are used, as in our more detailed formalization, a message digest function keyed with  $k \in E$  will produce a term of infinite  $E$ -corank relative to  $m_0$  as no amount of work can extract  $m_0$  from the resulting message, assuming that  $m_0$  is not equal to the result of the message digest operation.

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

Let  $E$  be a set of keys,  $m_0$  an atomic term of type `msg`,  $m$  of type `msg`, and  $t, t_i$  be terms. We may define the  $E$  corank relative to  $m_0$  of facts built from the predicates  $N()$ ,  $I()$ ,  $Auth_C$ , and  $DoneMut_C$  as follows; the definitions for other predicates

appearing in our formalizations similarly follow the intuition given above.

$$\begin{aligned}\hat{\rho}_E(\mathbf{N}(m); m_0) &= \hat{\rho}_E(m; m_0) \\ \hat{\rho}_E(I(t); m_0) &= \hat{\rho}_E(t; m_0) \\ \hat{\rho}_E(\text{Auth}_C(t_1, t_2, t_3); m_0) &= \hat{\rho}_E(t_1; m_0)\end{aligned}$$

For a multiset  $A$  of facts, we define the  $E$ -corank of  $A$  relative to  $m_0$  by  $\hat{\rho}_E(A; m_0) = \min_{F \in A} \hat{\rho}_E(F; m_0)$  if  $A \neq \emptyset$ , and let  $\hat{\rho}_E(\emptyset; m_0) = \infty$ .

We identify the confidentiality of  $m_0$  with the fact  $I(m_0)$  being prohibited from appearing in a generic trace  $\mathcal{T}$ . As an immediate consequence of the definition of the corank of facts, we see that if there is some set  $E$  of keys such that no fact of  $E$ -corank 0 relative to  $m_0$  appears in  $\mathcal{T}$ , then  $m_0$  remains confidential throughout  $\mathcal{T}$ .

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

**Property 11.** If an intruder rule  $r_I$  can decrease  $E$ -corank relative to  $m_0$ , where  $I$  does not have access to  $m_0$  simply by virtue of the type of  $m_0$ , then the left hand side of  $r_I$  contains  $I(k)$  for some  $k \in E$  or  $r_I$  freshly generates  $m_0$ .

Again, it is proved by cases on the rules defining the intruder capabilities. As expected, this property holds for our formalizations of the Dolev-Yao intruder in Section 4.2, as proved in [26].

We prove confidentiality using the following result; like Theorem 1, this may be viewed as roughly analogous to Schneider's rank function theorem.

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

*Proof.* (Sketch) The MSR fact that corresponds to the intruder knowing a message  $m_0$  has  $E$ -corank equal to 0 relative to  $m_0$  for every set  $E$  of keys. By the conditions of the theorem, this fact can appear in no state of the trace.  $\square$

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

#### 4.4 Verification excerpts

We now present a selection of formal theorems corresponding to the protocol properties discussed in Section 3.1; these give the flavor of how our formal results are stated and proved and illustrate the interconnected rank and corank arguments. Theorem 3, formalizing Property 1, is an example of an authentication guarantee that the client has when she sees a certain message on the network. Theorems 4 and 5 formalize secrecy results, stated informally as Properties 2 and 3, the former for an intra-realm key and the latter for a cross-realm key; as these concern secrecy, they rely upon corank arguments. Theorem 6 corresponds to Property 5 and gives an authentication guarantee to a TGS about the origin of a cross-realm ticket; this rank argument relies on the secrecy property captured in Theorem 5 and the corresponding corank arguments. (Secrecy properties in the CS exchange rely on this authentication property, completing the intertwining of rank/authentication and corank/secrecy.)

Each of these results refers to a generic finite trace, which as discussed in Section 4.1.4 may comprise multiple concurrent executions of Kerberos. We do not put a priori constraints on the initial configuration of this trace except those noted explicitly in the statements. For the most part, these are requirements that the intruder does not know the long term keys of relevant principals.

We include somewhat detailed proof sketches for the theorems in this section. The full proofs, which add slightly more detail, rely on tens of lemmas that capture the effects of the various MSR rules on rank and corank and which are proved by inspection of the rules.

We shall again observe that although these results are specific to Kerberos 5, the methodology we used is general. Indeed, the overall structure of statements and proofs would not differ substantially for other protocols, nor would the way they are organized.

##### 4.4.1 Authentication of KAS to client

We begin with the formal correspondent of Property 1, which ascertains that whenever the client processes a well-formed reply from the KAS, then the KAS indeed generated it (assuming the long-term key they share has not been compromised). This is a typical example of a data origin authentication statement, all of which have the form “if a principal  $A$  receives a message  $\{m\}_k$  encrypted with a key  $k$  that  $A$  shares with  $B$  and  $k$  is unavailable to the attacker, then  $B$  must have generated  $m$  and originated  $\{m\}_k$ ”. Theorem 3 specializes this statement to the scenario under consideration and formalizes it relative to the MSR specification. The presence of the opaque ticket slightly complicates this result.

**Theorem 3.** For  $C : \text{client } R_C$ ,  $k_C : \text{dbK } R_C C$ ,  $T : \text{TGS } R_C$ ,  $AK : \text{shK } C T$ ,  $n_1 : \text{nonce}$ , and  $t_K : \text{time}$ , if the initial state of a finite trace contains neither  $I(k_C)$  nor any fact of positive  $k_C$ -rank relative to  $(AK, n_1, t_K, T)$ , and if the fact  $N(C, X, \{AK, n_1, t_K, T\}_{k_C})$  appears in some state of a trace for some  $X : \text{msg}$ , then at some point earlier in the trace some  $K : \text{KAS } R_C$  fired rule  $\alpha_{2.1}$ , consuming the fact  $N(C, T, n_1)$ , freshly generating the key  $AK : \text{shK } C T$ , and producing the fact  $N(C, \{AK, C, t_K, \text{Rnil}\}_{k_T}, \{AK, n_1, t_K, T\}_{k_C})$  for some  $k_T : \text{dbK } R_C T$ .

*Proof.* The fact  $N(C, \{AK, C, t_K, \text{Rnil}\}_{k_T}, \{AK, n_1, t_K, T\}_{k_C})$  has  $k_C$ -rank 1 relative to  $(AK, n_1, t_K, T)$ ; we consider which protocol participants could fire a rule that increases this rank. By hypothesis, no such fact appeared in the initial state of the trace. Additionally,  $k_C$  was not initially known to the intruder and so is never known to the intruder (because long-term keys are not leaked or transmitted in a message); thus the intruder could never fire a rule increasing  $k_C$ -rank relative to  $AK, n_1, t_K, T$ . Inspection of the principal rules shows that the only honest principal who will fire such a rule is some  $K : \text{KAS } R_C$ .  $\square$

#### 4.4.2 Confidentiality of intra-realm $AK$

**Theorem 4.** For  $R_C : \text{realm}$ ,  $C : \text{client } R_C$ ,  $T : \text{TGS } R_C$ ,  $C, T \neq I$ ,  $k_C : \text{dbK } R_C C$ ,  $k_T : \text{dbK } R_C T$ ,  $AK : \text{shK } C T$ , and  $n_1 : \text{nonce}$ , if the initial state of a finite trace does not contain  $I(k_C)$  or  $I(k_T)$  and some  $K : \text{KAS}$  fires rule  $\alpha_{2.1}$ , freshly generating  $AK$  and creating the fact  $N(C, \{AK, C, t_K, \text{Rnil}\}_{k_T}, \{AK, n_1, t_K, T\}_{k_C})$ , then no state of the trace contains the fact  $I(AK)$ .

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

If any  $\text{KAS}$  fires a rule which decreases  $\{k_C, k_T\}$ -corank relative to  $AK$ , then that rule freshly generates  $AK$  and, if the newly created fact in the resulting multiset is  $N(C, \{AK, C, t_K, \text{Rnil}\}_{k_T}, \{AK, n_1, t_K, T\}_{k_C})$ , the  $\{k_C, k_T\}$ -corank relative to  $AK$  of this multiset equals 1. Because  $AK$  is freshly generated here, no previous multiset in the trace contained a fact with finite  $\{k_C, k_T\}$ -corank relative to  $AK$ , nor can any  $\text{KAS}$  later fire a rule which decreases  $\{k_C, k_T\}$ -corank relative to  $AK$ .

Additionally, no client, TGS, or server rule decreases  $\{k_C, k_T\}$ -corank relative to  $AK$ .

The facts  $I(k_C)$  and  $I(k_T)$  do not appear in the initial state of the trace; these keys can never be lost during the protocol, so these facts never appear in the trace under consideration. By hypothesis,  $K$  freshly generates  $AK$ , so  $I$  cannot freshly generate  $AK$ . Thus  $I$  does not fire any rule which decreases  $\{k_C, k_T\}$ -corank relative to  $AK$ .

As a result, no fact of  $\{k_C, k_T\}$ -corank 0 relative to  $AK$ , in particular  $I(AK)$ ,

occurs in any multiset of the trace.  $\square$

#### 4.4.3 Confidentiality of cross-realm $AK$

**Theorem 5.** For  $C : \text{client } R_C, T : \text{TGS } R_i, T' : \text{rTGS } R_i R_j, C, T, T' \neq I, k_{R_i, T_{i+1}} : \text{dbK } R_i T'$ , if no state of the trace contains either  $I(k_{R_i, T_{i+1}})$  or  $I(AK)$ , and if some  $T : \text{TGS } R_i$  fires rule  $\alpha_{A'.1}$ , freshly generating  $AK' : \text{shK } C T'$  and creating the fact  $N(C, \{AK', C, t_T^{CR}, (R_k \hat{=} Rs)\}_{k_{R_i, T_{i+1}}}, \{AK', n_3^{CR}, t_T^{CR}, T'\}_{AK})$  for some  $R_k : \text{realm}, Rs : \text{Rset}, t_T^{CR} : \text{time}, n_3^{CR} : \text{nonce}$ , then no state of the trace contains the fact  $I(AK')$ .

*Proof.* (Sketch) We claim that no fact with  $\{AK, k_{R_i, T_{i+1}}\}$ -corank equal to 0 relative to  $AK'$  in the trace.

If any TGS fires a rule which decreases  $\{AK, k_{R_i, T_{i+1}}\}$ -corank relative to  $AK'$ , then that rule is  $\alpha_{A'.1}$  and freshly generates  $AK'$ . Furthermore, if the newly created fact in the resulting multiset is  $N(C, \{AK', C, t_T^{CR}, (R_k \hat{=} Rs)\}_{k_{R_i, T_{i+1}}}, \{AK', n_3^{CR}, t_T^{CR}, T'\}_{AK})$  then the  $\{AK, k_{R_i, T_{i+1}}\}$ -corank relative to  $AK'$  of this multiset equals 1. Because  $AK'$  is freshly generated here, no previous multiset in the trace contained a fact with finite  $\{AK, k_{R_i, T_{i+1}}\}$ -corank relative to  $AK'$ , nor can any TGS later fire a rule which decreases  $\{AK, k_{R_i, T_{i+1}}\}$ -corank relative to  $AK'$ .

If some  $K : \text{KAS}$  fires a rule that decreases  $\{AK, k_{R_i, T_{i+1}}\}$ -corank relative to  $k : \text{shK } C T'$ , then that rule firing freshly generates  $k$ ; this means  $k \neq AK'$ , which is freshly generated by  $T$ . Additionally, no client or server rule decreases  $\{AK, k_{R_i, T_{i+1}}\}$ -corank relative to  $AK'$ .

The fact  $I(k_{R_i, T_{i+1}})$  does not appear in the initial state of the trace; this long-term key cannot be lost during the protocol. Combined with the assumption on  $AK$ , neither of the facts  $I(AK)$  and  $I(k_{R_i, T_{i+1}})$  ever appear in the trace under consideration. By hypothesis,  $T$  freshly generates  $AK$ , so  $I$  cannot do this; thus  $I$  does not fire any rule which decreases  $\{AK, k_{R_i, T_{i+1}}\}$ -corank relative to  $AK'$ .

As a result, no fact of  $\{AK, k_{R_i, T_{i+1}}\}$ -corank 0 relative to  $AK'$ , in particular  $I(AK')$ , occurs in any multiset of the trace.  $\square$

#### 4.4.4 Authentication of ticket-granting ticket and authenticator

The following theorem authenticates the origin of the TGT and authenticator when these are processed as part of a request for a cross-realm ticket. A very similar theorem, which we omit, authenticates the origins of these objects when they are presented as part of a request for an intra-realm service ticket.

**Theorem 6.** For  $R_C, R_i, R_j, R_k, R_n$  : realm,  $C$  : client  $R_C$ ,  $T$  : TGS  $R_i$ ,  $AK$  : shK  $C T$ ,  $t, t'$  : time,  $Rs$  : Rset,  $k_{R_k, T_i}$  : dbK  $R_k T$ ,  $t_C$  : time,  $T_n$  : TGS  $R_n$ , and  $n_3$  : nonce, if the initial state of the trace did not contain  $I(k_{R_k, T_i})$  or any fact with positive  $k_{R_k, T_i}$ -rank relative to  $AK, C, t, Rs$ , if no state of the trace contains  $I(k)$  for  $T_k$  : TGS  $R_k$  and  $k$  : shK  $C T_k$ , and if  $T$  fires rule  $\alpha_{4'.1}$ , consuming the fact  $N(\{AK, C, t, Rs\}_{k_{R_k, T_i}}, \{C, t'\}_{AK}, C, T_n, n_3)$ , freshly generating  $AK' : \text{shK } C T'$ , and creating the fact  $N(C, \{AK', C, t'', (R_k \hat{=} Rs)\}_{k_{R_i, T_{i+1}}}, \{AK', n_3, t'', T'\}_{AK})$  for some  $t''$  : time,  $T'$  : TGS  $R_j$ ,  $k_{R_i, T_{i+1}}$  : dbK  $R_i T'$ , then earlier in the trace some  $T''$  : TGS  $R_k$  fired rule  $\alpha_{4'.1}$ , freshly generating  $AK$  and creating the fact  $N(C, \{AK, C, t, Rs\}_{k_{R_k, T_i}}, \{AK, n'_3, t''', T\}_{AK''})$  for some  $n'_3$  : nonce,  $t'''$  : time, and  $AK'' : \text{shK } C T''$ . Furthermore, after this earlier firing of rule  $\alpha_{4'.1}$ ,  $C$  fired rule  $\alpha_{3.1}$ , freshly generating  $n_3^{CR'}$  : nonce and creating the fact  $N(X, \{C, t'\}_{AK}, C, T_m, n_3^{CR'})$  for some  $X$  : msg,  $R_m$  : realm, and  $T_m$  : TGS  $R_m$ .

*Proof.* (Sketch)  $T$ 's firing of rule  $\alpha_{4'.1}$  consumes a fact of  $k_{R_k, T_i}$ -rank 1 relative to  $AK, C, t, Rs$ . As this rank was 0 for every fact in the initial state of the trace, some rule firing during the trace must have increased this rank.

No client or server can fire a rule that increases  $k_{R_k, T_i}$ -rank relative to  $AK, C, t, Rs$ ; if  $R_k \neq R_i$ , then no KAS can either. If  $R_k = R_i$ , then no TGS can fire a rule that increases this rank because the **CloserRealm** constraint returns a different realm than the one in which the TGS consulting it is located. For the remainder of this proof sketch, we assume we fall into the  $R_k \neq R_i$  case; the  $R_k = R_i$  case is similar.

Because  $I(k_{R_k, T_i})$  does not appear in the initial state of the trace, and thus in any state of the trace, the intruder cannot increase the rank in question. Thus some TGS must have fired a rule to increase this rank. Rule  $\alpha_{4.1}$  cannot increase  $k_{R_k, T_i}$ -rank relative to  $AK, C, t, Rs$ , so this must have been a firing of rule  $\alpha_{4'.1}$ . By inspection, we see that this firing of rule *ruleTCcross* must have: been done by some  $T''$  : TGS  $R_k$ , freshly generated  $AK$ , and created the fact  $N(C, \{AK, C, t, Rs\}_{k_{R_k, T_i}}, \{AK, n'_3, t''', T\}_{AK''})$  for some  $n'_3$  : nonce,  $t'''$  : time, and  $AK'' : \text{shK } C T''$ .

$T$ 's firing of rule  $\alpha_{4'.1}$  also consumes a fact of  $AK$ -rank 1 relative to  $C, t'$ . As  $AK$  was freshly generated by  $T''$ 's firing of rule  $\alpha_{4'.1}$ , every fact in the initial state of the trace had  $AK$  rank 0 relative to  $C, t'$  and so some rule firing during the trace (and after  $T''$ 's firing of rule  $\alpha_{4'.1}$  to generate  $AK$ ) must have increased this rank. By inspection, no KAS, TGS, or server rule can increase  $AK$ -rank relative to  $C, t'$ . By hypothesis, the conditions of Theorem 5 are satisfied, so  $I(AK)$  does not appear in the trace; thus no intruder rule can increase this rank. Thus the firing of some client rule must have increased  $AK$ -rank relative to  $C, t'$ . By inspection, this must have been rule  $\alpha_{3.1}$ , fired by  $C$  to freshly generate  $n_3^{CR'}$  : nonce and create the fact  $N(X, \{C, t'\}_{AK}, C, T_m, n_3^{CR'})$  for some  $X$  : msg,  $R_m$  : realm, and  $T_m$  : TGS  $R_m$ .  $\square$

## 5 Conclusions and Future Work

In this paper, we have reported on a multi-year effort aimed at applying the techniques of formal analysis to a real-world protocol as opposed to the simplified abstractions commonly examined in the formal methods literature. The popular Kerberos 5 authentication protocol [10] provided a practically relevant case study. The overall result has been the most detailed formal verification of any deployed protocol to date (to our knowledge). Because real-world protocols are significantly more complex than their commonly-studied abstractions, this effort pioneered substantial advances to the protocol verification methodology that relies on theorem proving: the use of dependently-typed languages as flexible formalization vehicles, the interleaving of authentication and confidentiality proofs based on distinct but cooperating mathematical tools, and the coordinated refinement of specifications, properties and proofs. These methods allowed us to prove numerous properties of Kerberos 5, in particular authentication, confidentiality and structural soundness, hence confirming that it is a robust and well-designed protocol. We also discovered a number of innocuous but unexpected behaviors that we called “anomalies”. Our analysis of the cross-realm authentication support of Kerberos 5 clearly exposed the fragility of this protocol with respect to the notion of trust.

Discussions about this work started our interactions with the IETF Kerberos working group; these have continued in connection with other parts of our project, and the renewed interest in anonymous tickets has connected to our work here.

The Kerberos protocol suite is complex and continually subject to extension as new real-world authentication needs are addressed. This provides additional interesting problems for study; we are now in the process of analyzing some of these protocol extensions.

The recent development of a prototype for our specification language, MSR, opens the doors to automating, at least partially, the proof methodology underlying this and future verification efforts.

### Acknowledgments

We are grateful to Alan Jeffrey, John Mitchell, Clifford Neuman, Ken Raeburn, and Sam Hartman for a number of helpful comments on various shorter versions of this work. Our interactions with participants in the IETF Kerberos working group regarding this and other aspects of our ongoing Kerberos analysis have been most helpful. We are indebted to the anonymous reviewers for the numerous suggestions for improving the original version of this paper.

## References

- [1] J. Clark, J. Jacob, A survey of authentication protocol literature, Tech. rep., Department of Computer Science, University of York, web Draft Version 1.0 available from <http://www.cs.york.ac.uk/~jac> (1997).
- [2] R. Needham, M. Schroeder, Using Encryption for Authentication in Large Networks of Computers, *Comm. ACM* 21 (12) (1978) 993–999.
- [3] D. Dolev, A. Yao, On the security of public-key protocols, *IEEE Trans. Info. Theory* 2 (29) (1983) 198–208.
- [4] M. Backes, B. Pfitzmann, M. Waidner, A composable cryptographic library with nested operations, in: *Proceedings of 10th ACM Conference on Computer and Communications Security (CCS)*, ACM Press, Washington, DC, 2003, pp. 220–230.
- [5] M. Bellare, P. Rogaway, Entity authentication and key distribution, in: *In Advances in Cryptology: CRYPTO’93*, Springer Verlag LNCS 773, 1993, pp. 232–249.
- [6] R. Canetti, Universally composable security: A new paradigm for cryptographic protocols, in: *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001*, IEEE Computer Society, Las Vegas, NV, 2001, pp. 136–145.
- [7] I. Cervesato, Fine-Grained MSR Specifications for Quantitative Security Analysis, in: *Fourth Workshop on Issues in the Theory of Security — WITS’04*, Barcelona, Spain, 2004, pp. 111–127.
- [8] C. Meadows, Analysis of the Internet Key Exchange Protocol using the NRL Protocol Analyzer, in: *Proc. IEEE Symp. Security and Privacy*, 1999, pp. 216–231.
- [9] C. Neuman, T. Ts’o, Kerberos: An Authentication Service for Computer Networks, *IEEE Communications* 32 (9) (1994) 33–38.
- [10] C. Neuman, T. Yu, S. Hartman, K. Raeburn, The Kerberos Network Authentication Service (V5), <http://www.ietf.org/rfc/rfc4120> (July 2005).
- [11] J. Garman, *Kerberos: The Definitive Guide*, O’Reilly, 2003.
- [12] K. Schutz, welcome Speech, IETF Kerberos Working Group September 2005 Interim Meeting.
- [13] G. Bella, Inductive Verification of Cryptographic Protocols, Ph.D. thesis, University of Cambridge, <http://www.cl.cam.ac.uk/~gb221/papers/bella14.ps.gz> (March 2000).
- [14] G. Bella, L. C. Paulson, Kerberos Version IV: Inductive Analysis of the Secrecy Goals, in: *Proc. ESORICS’98*, Springer LNCS 1485, 1998, pp. 361–375.
- [15] J. C. Mitchell, M. Mitchell, U. Stern, Automated Analysis of Cryptographic Protocols Using  $\text{Mur}\varphi$ , in: *Proc. IEEE Symp. Security and Privacy*, IEEE Computer Society Press, 1997, pp. 141–153.

- [16] I. Cervesato, A. D. Jaggard, A. Scedrov, C. Walstad, Specifying Kerberos 5 Cross-Realm Authentication, in: Proc. WITS'05, ACM Digital Library, 2005, pp. 12–26.
- [17] I. Cervesato, Typed MSR: Syntax and Examples, in: V. Gorodetski, V. Skormin, L. Popyack (Eds.), First International Workshop on Mathematical Methods, Models and Architectures for Computer Networks Security — MMM'01, Springer-Verlag LNCS 2052, St. Petersburg, Russia, 2001, pp. 159–177.
- [18] N. A. Durgin, P. D. Lincoln, J. C. Mitchell, A. Scedrov, Multiset Rewriting and the Complexity of Bounded Security Protocols, *J. Comp. Security* 12 (2) (2004) 247–311.
- [19] I. Cervesato, C. Meadows, D. Pavlovic, An Encapsulated Authentication Logic for Reasoning About Key Distribution Protocol, in: Eighteenth Computer Security Foundations Workshop — CSFW-18, IEEE Computer Society Press, Aix-en-Provence, France, 2005, pp. 48–61.
- [20] S. Bistarelli, I. Cervesato, G. Lenzini, R. Marangoni, F. Martinelli, On Representing Biological Systems through Multiset Rewriting, in: Proc. EUROCAST'03, Springer-Verlag LNCS 2809, 2003, pp. 415–426.
- [21] I. Cervesato, M.-O. Stehr, Representing the MSR Cryptoprotocol Specification Language in an Extension of Rewriting Logic with Dependent Types, in: N. Martí-Oliet (Ed.), Fifth International Workshop on Rewriting Logic and its Applications — WRLA'04, Elsevier ENTCS 117, Barcelona, Spain, 2004, pp. 183–207.
- [22] E. M. Clarke, S. Jha, W. R. Marrero, Using state space exploration and a natural deduction style message derivation engine to verify security protocols, in: Proc. IFIP Working Conference on Programming Concepts and Methods (PROCOMET), 1998, pp. 87–106.
- [23] T. Yu, S. Hartman, K. Raeburn, The perils of unauthenticated encryption: Kerberos version 4, in: Proc. NDSS'04, 2004, <http://www.isoc.org/isoc/conferences/ndss/04/proceedings/Papers/Yu.pdf>.
- [24] F. Butler, I. Cervesato, A. D. Jaggard, A. Scedrov, A Formal Analysis of Some Properties of Kerberos 5 Using MSR, in: Fifteenth Computer Security Foundations Workshop — CSFW-15, IEEE Computer Society Press, Cape Breton, NS, Canada, 2002, pp. 175–190.
- [25] F. Butler, I. Cervesato, A. D. Jaggard, A. Scedrov, Verifying Confidentiality and Authentication in Kerberos 5, in: K. Futatsugi, F. Mizoguchi, N. Yonezaki (Eds.), Software Security - Theories and Systems — ISSS 2003, Springer-Verlag LNCS 3233, Tokyo, Japan, 2003, pp. 1–24.
- [26] F. Butler, I. Cervesato, A. D. Jaggard, A. Scedrov, A Formal Analysis of Some Properties of Kerberos 5 Using MSR, Tech. Rep. MS-CIS-04-04, University of Pennsylvania Department of Computer and Information Science, <ftp://ftp.cis.upenn.edu/pub/papers/scedrov/ms-cis-04-04.pdf> (April 2004).
- [27] A. Birrell, B. W. Lampson, R. M. Needham, M. D. Schroeder, A Global Authentication Service without Global Trust, in: IEEE Symposium on Security and Privacy, 1986, pp. 223–230.

- [28] A Secure European System for Applications in a Multi-vendor Environment, <https://www.cosic.esat.kuleuven.ac.be/sesame/>.
- [29] V. D. Gligor, S.-W. Luan, J. N. Pato, On Inter-Realm Authentication in Large Distributed Systems, *J. Computer Security* 2 (2–3) (1993) 137–158.
- [30] C. Neuman, J. Kohl, T. Ts'o, K. Raeburn, T. Yu, The Kerberos Network Authentication Service (V5), Internet draft, expires 20 May 2002. <http://www.ietf.org/internet-drafts/draft-ietf-cat-kerberos-revisions-10.txt> (November 20 2001).
- [31] L. Zhu, P. Leach, K. Jaganathan, Anonymity Support for Kerberos, Internet draft, expires April 18, 2006. <http://www.ietf.org/internet-drafts/draft-zhu-kerb-anon-00.txt> (October 15 2005).
- [32] K. Raeburn, Encryption and Checksum Specifications for Kerberos 5, <http://www.ietf.org/rfc/rfc3961.txt> (Feb. 2005).
- [33] D. Gollmann, Authentication—Myths and Misconceptions, *Progress in Computer Science and Applied Logic* 20 (2001) 203–225.
- [34] G. Bella, L. C. Paulson, Using Isabelle to Prove Properties of the Kerberos Authentication System, in: H. Orman, C. Meadows (Eds.), *Proc. DIMACS'97, Workshop on Design and Formal Verification of Security Protocols (CD-ROM)*, 1997, <http://www.cl.cam.ac.uk/~gb221/papers/bella4.ps.gz>.
- [35] J. Meseguer, Conditional rewriting logic as a unified model of concurrency, *Theoretical Computer Science* 96 (1992) 73–155.
- [36] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, J. Quesada, *Maude: Specification and Programming in Rewriting Logic*, SRI International, <http://maude.csl.sri.com> (January 1999).
- [37] S. Schneider, Verifying Authentication Protocols in CSP, *IEEE Trans. Software Eng.* 24 (9) (1998) 741–758.
- [38] F. J. Thayer Fábrega, J. Herzog, J. D. Guttman, Honest ideals on strand spaces, in: *Proc. CSFW'98*, 1998, pp. 66–78.
- [39] J. Heather, S. Schneider, A Decision Procedure for the Existence of a Rank Function, *J. Computer Security* 13 (2) (2005) 317–344.
- [40] C. Meadows, D. Pavlovic, Deriving, attacking and defending the gdoi protocol, in: *Proc. ESORICS 2004*, Springer-Verlag LNCS 3193, 2004, pp. 33–53.
- [41] C. Neuman, personal communication (June 2002).
- [42] Kerberos working group meeting at IETF-64, November 2005. Summary archived at <ftp://ftp.ietf.org/ietf-mail-archive/krb-wg/>.
- [43] I. Cervesato, N. Durgin, P. D. Lincoln, J. C. Mitchell, A. Scedrov, A Comparison between Strand Spaces and Multiset Rewriting for Security Protocol Analysis, *Journal of Computer Security* 13 (2) (2005) 265–316.

- [44] I. Cervesato, Expressing type-flaw attacks in a strongly-typed language, Invited talk at the Second Workshop on Foundations for Secure/Survivable Systems and Networks, Tokyo, Japan, slides available as <http://theory.stanford.edu/~iliano/talks.html> (2001).
- [45] I. Cervesato, A Specification Language for Crypto-Protocols based on Multiset Rewriting, Dependent Types and Subsorting, in: G. Delzanno, S. Etalle, M. Gabbrielli (Eds.), Workshop on Specification, Analysis and Validation for Emerging Technologies — SAVE'01, Paphos, Cyprus, 2001, pp. 1–22.

## A Additional Roles

Figure A.1 shows the client role for requesting ticket-granting tickets.

$$\begin{array}{l}
\forall C : \text{client } R_C \\
\left[ \begin{array}{l}
\forall X : \text{msg } \forall T : \text{TGS } R_T \quad \forall T_n : \text{ts } R_n \quad \forall S : \text{server } R_S \quad \forall AK : \text{shK } C T \\
\forall t_C : \text{time} \\
\\
\begin{array}{c}
\exists n_3 : \text{nonce} \\
\text{Auth}_C(X, T, AK) \xrightarrow{\alpha_{3,1}} \left[ \begin{array}{l}
\text{N}(X, \{C, t_C\}_{AK}, C, T_n, n_3) \\
\text{MemTGE}(C, T_n, T, AK, n_3) \\
\text{Auth}_C(X, T, AK)
\end{array} \right] \\
\\
\text{if } \text{DesiredHop}(C, S, R_T, R_n), \text{clock}_C(t_C) \\
\\
\forall T_j : \text{ts } R_j \quad \forall Z : \text{msg } \quad \forall AK' : \text{shK } C T_j \quad \forall n_3 : \text{nonce} \quad \forall t_T : \text{time} \\
\left[ \begin{array}{l}
\text{N}(C, Z, \{AK', n_3, t_T, T_j\}_{AK}) \\
\text{MemTGE}(C, T_n, T, AK, n_3)
\end{array} \right] \xrightarrow{\alpha_{3,2}} \text{Auth}_C(Z, T_j, AK')
\end{array}
\end{array} \right]
\end{array}$$

Fig. A.1. The client's role in TGS exchange.

Figure A.2 shows the TGS role for responding to a request for an intra-realm service ticket, *i.e.*, for a server in the TGS's own realm.

Figure A.3 shows the TGS role for responding to a request for a cross-realm TGT, *i.e.*, for a TGS in a different realm than the TGS's.

Figure A.4 shows the client role for requesting a service ticket.

Figure A.5 shows the server role.

$$\begin{array}{l}
\forall T_i \text{TGS } R_i \\
\left[ \begin{array}{l}
\forall R_i, R_k : \text{realm} \quad \forall C : \text{client } R_C \quad \forall S : \text{server } R_i \quad \forall AK : \text{shK } C T_i \\
\forall k_{R_k, T_i} : \text{dbK } R_k T_i \quad \forall k_{R_i, S} : \text{dbK } R_i S \quad \forall n_2 : \text{nonce} \quad \forall Rs : \text{Rset} \\
\forall t_{T_i}, t_C, t : \text{time.} \\
\left[ \begin{array}{l}
\left[ \text{N}(\{AK, C, t, Rs\}_{k_{R_k, T_i}}, \right. \\
\left. \{C, t_C\}_{AK}, C, S, n_2) \right] \\
\left[ \text{N}(C, \{SK, C, t_{T_i}, (R_i \hat{=} Rs)\}_{k_{R_i, S}}, \right. \\
\left. \{SK, n_3, t_{T_i}, S\}_{AK}) \right] \\
\text{if } \text{Valid}_{T_i}(C, S, t_C, (R_k \hat{=} Rs)), \text{ clock}_{T_i}(t_{T_i})
\end{array} \right] \\
\end{array} \right]
\end{array}$$

Fig. A.2. TGS response to an intra-realm service ticket request.

$$\begin{array}{l}
\forall T_i \text{TGS } R_i \\
\left[ \begin{array}{l}
\forall R_{i-1}, R_{i+1}, R_n : \text{realm} \quad \forall C : \text{client } R_C \quad \forall AK : \text{shK } C T_i \quad \forall T_n : \text{TGS } R_n \\
\forall T_{i+1} : \text{rTGS } R_i R_{i+1} \quad \forall k_{R_k, T_i} : \text{dbK } R_{i-1} T_i \quad \forall k_{R_i, T_{i+1}} : \text{dbK } R_i T_{i+1} \\
\forall n_2 : \text{nonce} \quad \forall Rs : \text{Rset} \quad \forall t_{T_i}, t_C, t : \text{time.} \\
\left[ \begin{array}{l}
\left[ \text{N}(\{AK, C, t, Rs\}_{k_{R_k, T_i}}, \right. \\
\left. \{C, t_C\}_{AK}, C, T_n, n_3) \right] \\
\left[ \text{N}(C, \{AK', C, t_{T_i}, (R_{i-1} \hat{=} Rs)\}_{k_{R_i, T_{i+1}}}, \right. \\
\left. \{AK', n_3, t_{T_i}, T_{i+1}\}_{AK}) \right] \\
\text{if } \text{Valid}_{T_i}(C, T_n, t_C, (R_{i-1} \hat{=} Rs)), \text{ CloserRealm}(T_n, T_{i+1}), \text{ clock}_{T_i}(t_{T_i})
\end{array} \right] \\
\end{array} \right]
\end{array}$$

Fig. A.3. TGS response to a cross-realm TGT request.

$$\begin{array}{l}
\forall C : \text{client } R_C \\
\left[ \begin{array}{l}
\forall S : \text{server } R_S \quad \forall SK : \text{shK } C S \quad \forall t'_C : \text{time} \quad \forall Y : \text{msg.} \\
\text{Auth}_C(Y, S, SK) \xrightarrow{\alpha_{5.1}} \left[ \begin{array}{l}
\text{N}(Y, \{C, t'_C\}_{SK}) \\
\text{Auth}_C(Y, S, SK)
\end{array} \right] \\
\text{if } \text{clock}_C(t_{C, S_{req}})
\end{array} \right]
\end{array}$$

Fig. A.4. The client's role in the application server exchange.

$$\forall S : \text{server } R_S \left[ \begin{array}{l}
\forall C : \text{client } R_C \quad \forall T_S : \text{TGS } R_S \quad \forall SK : \text{shK } C \ S \quad \forall t'_C, t_{T_S} : \text{time} \\
\forall k_{R_S, S} : \text{dbK } R_S \ S \quad \forall R_S : \text{Rset.} \\
\left[ \begin{array}{l}
\text{N}(\{SK, C, t_{T_S}, R_S\}_{k_{R_S, S}}) \\
\{C, t'_C\}_{SK}
\end{array} \right] \xrightarrow{\alpha_{6.1}} \left[ \begin{array}{l}
\text{N}(\{t'_C\}_{SK}) \\
\text{Mem}_S(C, SK, t'_C, R_S)
\end{array} \right] \\
\text{if } \text{Valid}_S(C, t'_C, (R_C \hat{R}_S \hat{R}_S))
\end{array} \right]$$

Fig. A.5. The server's role in the service ticket exchange.