Security on the Move: Indirect Authentication Using Kerberos

Armando Fox and Steven D. Gribble
University of California at Berkeley
{fox,gribble}@cs.berkeley.edu

Abstract

Even as mobile computing and network computing are gaining momentum, Internet security is sharing the spotlight. Security and authentication on open networks is difficult even without the additional risks posed by wireless media. At the same time, the hardware and software constraints imposed by "small" mobile devices such as PDA's and smart phones are leading to the pervasive use of application-level proxies to mediate between clients and servers. To date, less attention has been focused specifically on securing the connection between the client and an application-level proxy, independently of any particular service. We describe how to provide secure access to application-level proxied services using an indirect protocol called Charon, which leverages the strong protocol and deployed infrastructure of Kerberos IV. Charon partitions the Kerberos client functionality into a very simple client module that does little more than DES encryption, and a Unix-hosted unprivileged proxy module that runs the remainder of the Kerberos client protocol and can interoperate with other proxy code that provides application-level proxy services. The client module is extremely lightweight and amenable to implementation even on the most modest PDA-class devices. The proxy is trusted as much as any other Kerberized service, but it never learns the user's Kerberos password or ticket-granting service session key. As a result of this partitioning, clients can continue to enjoy the benefits of proxy-mediated access in a secure way, in addition to gaining secure access to and interoperability with existing Kerberized services.

1 Introduction

1.1 Security, Open Networks, and Mobile Computing

The explosive growth of the Internet and the World Wide Web has fueled the image of personal digital assistants (PDA's), laptop computers, and mobile devices of all kinds as information retrieval devices, rather than stand-alone "islands" of computation—mobile users want *access* to the Internet and their desktop, not a new "killer app" for mobile devices. Wireless infrastructure is rapidly being deployed to enable "untethered" mobile computing [14].

Unfortunately, security and authentication in unprotected networks such as the Internet is a difficult problem [33, 7, 27], and the wireless medium is physically easier to compromise than wired media. Much attention has been focused recently on securing this link [4, 11, 18]. However, link-level encryption and authentication solve only part of the problem. Users still need to authenticate themselves in an end-to-end fashion [38] to *services* higher than this level, for example, filesystem access, remote login, database access, etc. Users of network computing devices [23], and the service providers to which those devices connect, will need authentication for connection to the service and privacy when retrieving content, especially sensitive content such as email.

1.2 Proxied Services

PDA's and personal communications devices are in every respect very modest compared to laptops: they have smaller screens, less memory, less powerful processors, stripped-down operating systems and programmatic API's, and weak (typically wireless) communication. To compensate for these differences, the strategy of mediating access via *proxies*—computation inserted between the client and server—is becoming pervasive. In some cases proxies are transparent to the user, such as when they provide network-level protocol encapsulation [25, 37, 39], link encryption [4, 11, 18], or performance optimization [5, 24]. In other cases, however, the client applications communicate explicitly via a proxy that operates on application data directly; recent examples include transcoding data from the Web [10, 16, 45], controlled access through firewalls [34], and distributed document caching [1, 8]. Such *application-level proxied services* are essential to mobile computer users, because the computation performed by the proxy on the client's behalf can tailor and adapt the content to the specific needs of each client. Our goal is to enable secure access to such services, while minimizing the client-side complexity that such secure access imposes.

When the end service and the proxy are in a trusted infrastructure (e.g. a corporate intranet or private networks), strong link-level encryption may suffice if it is available and securely implemented. When the end service is in an untrusted infrastructure but supports end-to-end encryption, the proxy can be used to tunnel encrypted data, but the user loses the benefits of the computation the proxy would normally perform, since the proxy cannot access and transform the decrypted data.

We therefore address the "intermediate" case, in which the end-to-end encryption is split at the proxy: The client-to-proxy and proxy-to-server connections are both encrypted, possibly with different keys, and the proxy is trusted enough to operate on the data on the user's behalf.

1.3 Charon: Secure Indirect Access to Proxied Services

As our contribution to enabling secure access to proxied services, we present Charon, a proxied implementation of Kerberos [40] for *indirect authentication* and secure communications with personal mobile devices. By indirect, we mean that most of the computational resources needed to conduct the Kerberos protocol and establish a secure channel are located at a *proxy*, a process running on a desktop workstation in the wired infrastructure. This approach simplifies the client software considerably: minimal client-side software is sufficient to preserve Kerberos semantics and security guarantees, while off-loading substantial protocol processing, credentials management, and other housekeeping tasks to a proxy in which minimal trust must be placed.

Charon provides the three secure-access models that we motivated in the previous section:

- A means of authenticating clients to a service attachment point (proxy) using a Kerberos-based protocol;

- Establishment of a secure communication channel to that attachment point, on which future logical channels can be multiplexed, thus providing encryption to the proxy when link-level encryption is weak or unavailable;
- Both end-to-end and proxied access to two-way-authenticated Kerberized services in an existing Kerberos infrastructure.

In the remainder of this section we motivate our approach by showing why PDA's, network computers, smart personal communicators [20], and similar devices represent an interesting design point, despite placing significant constraints on software development. We describe how Kerberos's properties make it a desirable mechanism for our approach, why it is difficult to port to such clients, and how our approach circumvents these constraints. Section 2 describes the Charon indirect protocol in detail, compares it with the unmodified Kerberos protocol, and describes the models of trust and secure access that Charon provides to the client. Section 3 describes our implementation of Charon (both client and proxy components). The client-side is implemented on a Sony Magic Link PDA, whose characteristics are representative of the clients we are targeting. Section 4 compares Charon with Kerberos in terms of resistance to a variety of attacks. We close with comments on Charon's role within larger mobile computing and proxy-based computing projects, related work, and work in progress.

1.4 Why Kerberos?

Kerberos [40] is a time-tested, widely-deployed system that provides authentication and the establishment of secure channels in open networks. Each client or service in an administrative domain is called a *principal* in Kerberos, and to each principal there corresponds a conventional-cryptography key known only to the principal and the Kerberos key distribution center (KDC). Kerberos clients authenticate themselves to servers by presenting *tickets* for each service. Tickets are capabilities distributed by a central trusted server within each administrative domain, and are constructed so that only clients possessing the appropriate key(s) are able to decrypt and use them. Kerberos includes specific features to prevent forgery of client or server identity, detect replay attacks [33], establish secure channels between endpoints through safe distribution of temporary session keys, allow mutual authentication, and minimize the likelihood that the user's Kerberos password will be compromised (it never leaves the user's workstation, and all traces of it are destroyed once the user has authenticated herself). The strengths and weaknesses of Kerberos are analyzed in detail in [7].

1.5 Porting Kerberos is Out of the Question

Traditional Kerberos clients (e.g. *krlogin, ktelnet*) access Kerberos client-side services by linking against a Kerberos library, *libkrb*, which in turn makes use of an encryption library, *libdes*. Three factors complicate a direct port of Kerberos to PDA's:

- limited hardware and software resources
- heterogeneous development environments that do not provide the Unix idioms Kerberos requires
- application and library API's that are incompatible with the Kerberos Unix API, at either the language or the linkage level

We describe each difficulty in turn.

Today's PDA's lack the memory, computing power, and persistent storage necessary to support the Kerberos library. As a first order approximation, observe that the runtime resident set size of *kinit*, a trivial Kerberos client that performs initial user authentication, is 292KB under Linux 2.0.12. In contrast, the Sony Magic Link PIC-2000 PDA, whose capabilities are representative of today's devices, has a total of 2MB of SRAM which functions as both the persistent store and a shared heap for all applications, the operating system, and the GUI (there is no swap in the Unix sense, and the virtual memory facilities are limited).

Indeed, it is meaningless to speak of "porting" most Unix software to today's PDA's: even with more memory and a faster CPU, the development environments for most PDA's are so different from Unix (and from each other) that in many cases we may only speak of *rewriting* the software to achieve the same functionality. Unix idioms do not apply; Unix-like raw I/O facilities are usually not provided, and the standard set of Unix libraries implementing kernel abstractions such as memory allocation services and standard I/O are absent. For example, the Magic Cap subset of C (Magic Cap is the OS and development environment for the Magic Link PDA) forbids globally or statically declared variables, is missing most of the functions typically found in the UNIX standard C library, and forces programmers to interact directly with the modem in order to communicate with remote machines. Subset implementations of TCP/IP are only now becoming available for PDA's, but to date they have proven to be too heavyweight for all but the most trivial applications [36]. (Java [42] may address some of these issues, but there will still be significant obstacles to a full port of Kerberos to Java-enabled devices, as we describe later in Section 3.4.)

Kerberos is particularly problematic code to port: besides its size and complexity (9,000 lines of C source), it relies heavily on Unix idioms and libraries, use of Unix I/O (including Berkeley sockets) and memory services, and knowledge of hardware semantics such as the byte ordering and word-size of the host architecture. None of this functionality is easy to implement or mimic on current PDA class devices, or on most non-UNIX systems for that matter. Even if the libraries could be compiled, many current PDA development environments do not support Unix-like linkage conventions for calling into the library, or else they provide a language model so different from conventional C that passing around complex Kerberos data structures would be cumbersome at best.

Faced with rewriting Kerberos as opposed to just porting it, we found it reasonable to rewrite only the minimal amount necessary to preserve Kerberos security guarantees to the client and provide an encrypted channel for client-to-proxy communication. In this way, the Kerberos client-side functionality is partitioned between the client and the proxy, with the proxy absorbing most of the Unix-specific functionality that makes Kerberos difficult to port to begin with, and the client handling DES encryption and decryption and some other tasks that are easy to implement and necessary in order to preserve the correctness of the Kerberos protocol. Since we already rely on proxies to perform substantial computing on behalf of clients, we pushed Kerberos functionality to the proxy wherever possible without weakening the security of the system.

2 Charon Design

2.1 An Overview of the Charon Architecture

Charon is designed to be deployed in an environment that consists of heterogeneous and lightweight clients, services, proxies, and the (unmodified) Kerberos infrastructure. In our architecture, proxies play two important roles: they provide application-level adaptation for impoverished clients [9, 17] and can optionally function as a protocol gateway and logical channel multiplexor for non-TCP/IP clients. In the latter case, access to application-level proxied services is supported by a simple RPC-like messaging protocol, that simplifies message parsing on the client and requires

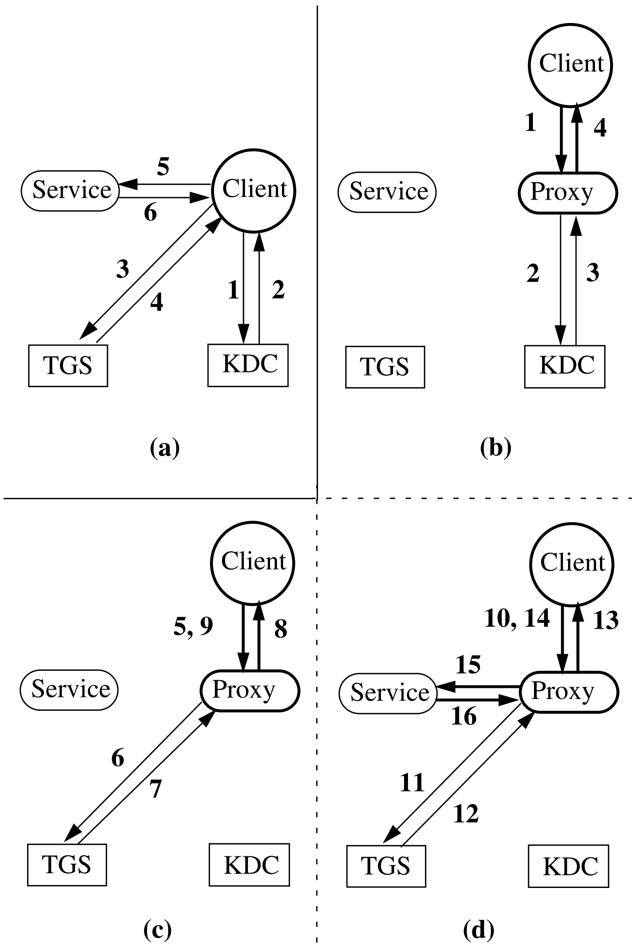

Figure 1: (a) The unmodified Kerberos protocol. Clients interact with the Kerberos key distribution center (KDC), ticket-granting service (TGS), and Kerberized services. (b) The first step of the handshake phase - obtaining a ticket-granting ticket (TGT) via Charon. (c) The second step of the handshake phase - obtaining a session key for use with the proxy. (d) Obtaining a service key via Charon.

only reliable stream or datagram delivery. When a client running Charon attempts to connect to the proxy, the proxy-side Charon protocol module receives the request and attempts to authenticate the client.

Charon interaction consists of two distinct phases: the *handshake* phase, in which the client authenticates itself to the proxy via Kerberos and establishes a secure channel with it, and the *service access* phase, in which the proxy accesses Kerberized services on the client's behalf. The Charon protocol module on the proxy and the Charon client-side software are responsible for the flow of control during both phases.

Handshaking is accomplished by treating the proxy as a Kerberized service. The handshake consists of two steps. During the first step (illustrated in figure 1b), the client uses the proxy as an intelligent router to obtain a TGT, which will then be managed by the proxy. From the point of view of the KDC and TGS, the proxy appears to be a normal Kerberos client during this phase.

During the second step (figure 1c), the client obtains a "proxy service" ticket from the TGS; during this step the proxy functions not only as a router but also as a Kerberized service, for which the TGS can create a session key to be securely disclosed via the Kerberos messages. After this two-step handshake phase has succeeded, the session key can be used to secure the channel between the client and the proxy, and therefore all logical connections multiplexed onto it.

The client, now in the service access phase, can then request indirect access to other Kerberized services (figure 1d).

Kerberos can similarly be thought of as a two-phase protocol, in which the first phase (e.g. *kinit*) consists of getting credentials, and the second phase (e.g. *krlogin*) consists of presenting them to obtain service tickets. An important property that distinguishes our architecture from unmodified Kerberos is that the proxy-side Charon module never gains possession of enough information necessary to independently negotiate for services on behalf of the client. Specifically, the Charon client-side never reveals the user's Kerberos key to anyone, and it also never gives the TGS session key (which Kerberos normally stores on the client workstation) to the proxy-side module. This TGS session key is required in order to construct and encrypt timestamped authenticators (see [40] or Appendix A) that must accompany each request for service; therefore the proxy-side Charon module cannot obtain service on behalf of the client without the client's explicit cooperation.

2.2 The Handshake Phase

In describing the messages exchanged in the Charon protocol, we use the following terminology. Readers desiring a review of the Kerberos protocol are referred to Appendix A.

Kerberized service: A service (e.g. *rlogin*) that requires its clients to authenticate themselves using Kerberos, and is capable of authenticating itself to its clients using Kerberos. These services are hosted on *Kerberos servers*, on which credentials for the services have been securely installed by the Kerberos administrator.

C: the client's principal (i.e. the user). The user knows a password from which the key K_c can be derived.

Ch: the Charon agent (part of the proxy process) running at the proxy workstation.

P: the proxy service as a Kerberos principal.

S: the principal corresponding to some Kerberized service, for example, Kerberized *rlogin*.

KDC: key distribution center, a secure database that knows the secret keys of every Kerberos principal in its *realm* (administrative domain).

TGS: The Kerberos ticket-granting server.

TGT: The initial ticket returned by the TGS which, when decrypted by the client using K_c, can be used to prove the client's identity in future transactions with the KDC (in conjunction with a valid *authenticator*) without requiring the principal to supply K_c again.

A_p: a Kerberos authenticator constructed by principal P. A_p is encrypted using a key shared only by P and the TGS and is timestamped to prevent replay attacks.

K_x: the key corresponding to Kerberos principal *x*, where *x* may be a user or a Kerberized service. Each key is known only to its principal and to the KDC.

$K_{x,y}$: a one-time[1] session key generated by the KDC and securely disclosed to principals *x* and *y* so that they may encrypt their communication.

1: Bellovin and Merrick have observed [7] that a Kerberos service session key is really a *multi-session key,* since it is used for all transactions with that service for the lifetime of the ticket. They describe a simple change to the protocol to allow the use of true one-time session keys.

$T_{c,s}$: a ticket authorizing client C to access service S.

$\{x\}K$: message x encrypted under key K.

The goal of the handshake phase of the Charon protocol is for the client and proxy to establish a secure communications channel, mutually authenticating themselves and obtaining a session key to use when encrypting traffic. A summary of the messages exchanged during the first step of the Charon unshackle phase is as follows. (These 4 messages replace messages 1-2 in the unmodified Kerberos protocol presented in Appendix A.)

1. The client indicates its desire to establish secure authenticated channel with Charon. The format of this message is somewhat simpler than that of message 1 in the unmodified Kerberos protocol.
2. Charon requests a TGT on the client's behalf. This message is identical to message 1 of the unmodified Kerberos protocol.
3. The TGT packet is returned to Charon; this message is identical to message 2 of the unmodified protocol.

   Formally: $KDC \rightarrow Ch$: $\{K_{c,tgs}, \{T_{c,tgs}\}K_{tgs}\}K_c$
4. The TGT packet is forwarded verbatim to the client. The client prompts the user for his/her Kerberos password, converts it into a DES key, and uses DES to decrypt the TGT packet.

   Formally: $Ch \rightarrow C$: $\{K_{c,tgs}, \{T_{c,tgs}\}K_{tgs}\}K_c$

At this point in our protocol, in addition to the client's key K_c, the client possesses the contents of the TGT packet: the TGT itself $\{T_{c,tgs}\}K_{tgs}$, and a session key $K_{c,tgs}$ to use when talking with the TGS. The proxy possesses only the contents of message 3 above (the encrypted TGT packet); in particular it cannot decrypt message 3 (since it does not know K_c), and therefore it cannot possess $K_{c,tgs}$. The client will then request a ticket for the proxy service in the second step of the handshake phase:

5. Request a ticket for the "proxy service". The client must construct an authenticator A_c and encrypt it using $K_{c,tgs}$, obtained in step 4. (Compare this with message 3 in the unmodified Kerberos protocol: in that case, the Kerberos client program can construct the authenticator directly, since $K_{c,tgs}$ is stored in the ticket file on the user's workstation.) Since a timestamp is included in the authenticator, the client's clock must be synchronized with that of the TGS (to within 5 minutes for Kerberos IV). We do not address how synchronization should be achieved, but note that lack of synchronization can only result in denied service; it does not weaken the security of the system.

   Formally: $C \rightarrow Ch$: P, $\{T_{c,tgs}\}K_{tgs}$, $\{A_c\}K_{c,tgs}$
6. Charon relays the information to the TGS as a well-formed Kerberos request. Note that the forwarded message contains the (encrypted) TGT originally returned in message 3 and the authenticator constructed by the client; both are necessary for the TGS to honor the request. This is an important property: since only the client can construct the authenticator, Charon cannot construct a valid request for a service without the client's cooperation.

   Formally: $Ch \rightarrow TGS$: P, $\{T_{c,tgs}\}K_{tgs}$, $\{A_c\}K_{c,tgs}$
7. The TGS returns the proxy ticket and a session key for use between the client and the proxy, all encrypted with $K_{c,tgs}$, so that only the client can decrypt this message. Note that there are *two* copies of $K_{c,p}$ in this message: one copy is contained in the encrypted ticket $T_{c,p}$, and only the entity that can decrypt the ticket can get that copy. In this case the entity is Charon, which alone possesses the key K_p.

   Formally: $TGS \rightarrow Ch$: $\{K_{c,p}, \{T_{c,p}\}K_p\}K_{c,tgs}$
8. The ticket and key are forwarded verbatim to the client, which decrypts them using $K_{c,tgs}$ and extracts $K_{c,p}$.

   Formally: $Ch \rightarrow C$: $\{K_{c,p}, \{T_{c,p}\}K_p\}K_{c,tgs}$
9. The client forwards the proxy ticket and a constructed authenticator to Charon. The authenticator proves that the client knows $K_{c,p}$. Since Charon has access to the Kerberos credentials K_p of the proxy, it can decrypt $\{T_{c,p}\}K_p$ and extract $K_{c,p}$ from $T_{c,p}$.

   Formally: $C \rightarrow Ch$: $\{T_{c,p}\}K_p$, $\{A_c\}K_{c,p}$

The two-step handshake phase requires five messages to be exchanged between the PDA client and Charon. If the handshake completes successfully, the following are true:

- Both the client and the proxy are in possession of a session key $K_{c,p}$, which is then used to encrypt further communication between the client and the proxy. Since in our architecture the proxy also functions as a logical-channel multiplexor, we can encrypt any of the logical channels carried on the client-to-proxy connection.
- None of the encryption keys (i.e. the client/TGS session key $K_{c,tgs}$, the client's Kerberos key K_c, the proxy's Kerberos key K_p, and the client/proxy session key $K_{c,p}$) and neither of the tickets $T_{c,tgs}$ and $T_{c,p}$ have ever been transmitted in the clear.
- Charon believes in the identity of the user.
- Charon does not possess $K_{c,tgs}$.

2.3 The Service Access Phase

Under normal Kerberos, access to a service is requested by presenting the TGS with a ticket and an authenticator, and the name of the desired service. A file on the user's workstation contains the user's credentials, namely the TGT, service tickets, and session keys, including the TGS session key $K_{c,tgs}$ necessary to construct the authenticator for requesting new service tickets and to decrypt returned service tickets.

Since Charon possesses neither $K_{c,tgs}$ nor the Kerberos password that was used to obtain it, messages 10 through 14 in figure 1d (which are identical to messages 5 through 9 but with the proxy service P replaced by the appropriate service name) are used to obtain the service ticket, from which the client can extract a session key for the service. The session key and service ticket are then securely relayed to Charon using the key $K_{c,p}$ obtained during the handshake phase. The proxy can then interact with the service on the client's behalf (messages 15 and 16). For example, the proxy can securely retrieve mail via a Kerberized POP service, distill [17] MIME images embedded in the messages to suit the client's display, truncate messages to a client-specified maximum size, and finally relay them in a format most convenient for that client.

An alternative approach to service access that places more trust in Charon is for the client to reveal $K_{c,tgs}$ to Charon over the established secure channel, thus allowing Charon to negotiate for Kerberized services directly. (This approach is similar to that presented in [15].) Charon can now construct the authenticators that must accompany the TGT on each request, and therefore does not need to interact with the client to access new services. In this approach, Charon still doesn't have the user's Kerberos password, but because it has $K_{c,tgs}$, it can do more damage should it be compromised. Specifically, an attacker who controls $K_{c,tgs}$ can impersonate the client for the lifetime of the TGT, requesting additional services that the client has not authorized. The ticket lifetime, which is specified at the time the TGT is requested, may be as lengthy as several hours, which presents a potentially large window during which attackers could cause damage. This second approach potentially increases convenience to the user at the cost of decreased security. We examine security in detail in Section 4.

3 Charon Implementation and Performance

3.1 The Charon Proxy Module

For rapid prototyping and portability, we have been using Perl 5 [43] as our proxy prototyping language. Accessing the Kerberos IV library *libkrb.a* from Perl required some glue code, which was relatively painless to create due to the extensive facilities Perl 5 provides for exactly this linking process. Kerberos clients compose and decompose encrypted text via calls to the Kerberos library, which in turn calls an encryption library. Kerberos was specifically designed this way so that the encryption module would be easily replaceable, although in practice all currently deployed implementations of Kerberos IV use DES [32] via the *libdes.a* library.

Our proxy consists of a set of modules that provide various proxied services to a connected client. The Charon module takes control when a client first connects to the proxy. After authenticating the client during the handshake phase, Charon remains available to the proxy to provide access to Kerberized services and to provide encrypted streams as an option for any service. The Charon module consists of about 500 lines of Perl; a "minimal" proxy, including Charon and all proxy support code but no access to other proxied services, consists of about 1500 lines of Perl.

3.2 Sony Magic Link Client-side Implementation

We chose to implement the client-side component of Charon on a Sony Magic Link PDA [41], which features a relatively rich OS (MagicCap) and a C compiler. Charon places minimal requirements on the client-side component. Specifically, the client must be able to do the following:

- Construct authenticators, which have a relatively simple format and include a timestamp.
- Extract session keys from tickets, which happen to be the first 8 bytes of a ticket, eliminating the need for the client to parse the ticket structure.[2]
- Communicate with the proxy over some sort of reliable stream or datagram protocol. In our implementation, we used a dial-in connection and the LAPM-capable [26] modem built into the Magic Link, obviating the need for a TCP/IP stack on the client.
- Perform DES encryption and decryption.

The DES library is the most complex component of the client-side Charon implementation, since idiosyncrasies of the MagicCap OS development environment forced us to rewrite much of it. This result compares favorably with the approach of porting all of Kerberos to clients such as the Magic Link PDA. (It even compares favorably to porting Kerberos to other Unix platforms [2], given Kerberos' reliance on low-level machine characteristics.) Our design and implementation pushes this complexity off of the client while maintaining the security and integrity of the abstract Kerberos protocol. Specifically, the Charon partitioning relieves the client of the following tasks and resource requirements:

- intimate knowledge of most of the details of the Kerberos IV implementation protocols and data formats;
- TCP/IP connectivity (to communicate with both the Kerberos infrastructure and application services);
- ticket management and file system interaction;
- intimate knowledge of application service protocols and data formats.

Our reimplementation of DES contains about 1200 lines of source code, resulting in a compiled image size of about 30KB. The rest of the Charon client, including a modest user interface, required another 750 lines of source (15KB compiled), for a total footprint of approximately 45KB on the PDA.

This figure merits some explanation. In many PDA OS's, including Magic Cap and Newton OS, applications once opened are never closed until they are uninstalled (although they may run in the background). The Charon footprint of 45KB represents all of the memory it will ever use on the Magic Link (including data and allocated memory[3]). In contrast, although the *kinit* binary is only 32KB (stripped) under Linux 2.0.12, its runtime image at peak is 292KB, due to the inclusion of shared libraries, data and stack segments, and application-level buffers. This difference in memory allocation models underscores the need for economy of size when dealing with PDA class devices.

3.3 Adding Access to Kerberized Services

In traditional Kerberos, a Kerberos client program authenticates itself to a server via two Kerberos library functions: `get_ad_tkt` to obtain a service ticket for the service (using an existing TGT), and `krb_sendauth` to present the ticket to the remote server and effect (possibly mutual) authentication. The Charon module provides direct access from Perl to `krb_sendauth`, but it cannot call `get_ad_tkt` directly because that routine tries to read the session key $K_{c,tgs}$ (which Kerberos normally stores on the user's workstation) for two operations:

- to construct the authenticator that must accompany any request to the TGS;
- to decrypt the response returned by the TGS.

Recall that as described in section 2.2, $K_{c,tgs}$ is known only to the client, and is *not* stored on Charon's workstation. Therefore Charon must ask the client to construct the authenticator to be passed to the TGS, and when the TGS sends a reply, Charon must pass the reply packet to the client for decryption.

Charon therefore provides `charon_get_ad_tkt`, which is a function that obtains a ticket using the Charon protocol messages 5-9 rather than the unmodified Kerberos messages 3-4. `charon_get_ad_tkt` is a functional replacement for the standard Kerberos `get_ad_tkt`, and can be called by any proxy module that wants to provide client access to a Kerberized service.

2: Clients may wish to verify the service name, instance, and realm in the initially returned proxy ticket, in order to prevent themselves from being tricked into connecting to another Kerberized service instead of the proxy via a man-in-the-middle attack. If clients do not perform this verification, the worst that can happen is an erroneous connection to a Kerberized service that could then impersonate the proxy. Only a Kerberized service could mount this "attack" successfully, so this caveat is consistent with the principle that Kerberized services are trustworthy in the Kerberos infrastructure.

3: The Magic Link does use a stack, but the stack size is typically on the order of 5 to 10 KB.

3.4 "Porting" to Other Devices

A wide variety of devices could potentially benefit from the Charon protocol, including a large number of heterogeneous clients and laptop computers of differing operating systems, computational prowess, and communication abilities. Attempting to design software that can be easily ported across this large class of target devices is difficult at best, and currently seems intractable for software of any complexity.

Java [42] may someday alleviate this problem, but Java-enabled PDA's with the rich runtime required by desktop Java implementations have yet to appear. When they do appear, a full port of Kerberos to Java will be complicated by Kerberos' extensive byte manipulation and pointer casting. However, the benefit of an indirect implementation goes beyond these obstacles: With our architecture, all PDA applications see an extremely simple and uniform interface to secure communication with proxied services (Kerberized or not), and the proxy-to-client connection for any application can be encrypted transparently to the application. (In contrast, the Kerberos implementations of *rlogin* and *telnet* contain additional explicit code to encrypt the channel and replace the traditional Unix login with a Kerberos handshake.)

In view of these considerations, we designed both the Charon bytestream protocol and the required client-side components to be as simple as possible. Rewriting these components on new devices is intended to be relatively simple. For example, in our implementation, the client needs to understand only one kind of transaction (Kerberos messages can take a number of different formats), and the file that describes the format and contents of Charon protocol messages is used to automatically generate code for different target platforms to parse and verify the integrity of the messages.

3.5 Performance of DES on PDA

Our recent experience porting a GIF renderer [17] and audio stream decoder [22] to the Magic Link device strongly suggested that the limiting factor in Charon performance would be the speed of DES encryption and decryption. We have found that the Charon handshake phase takes about 2 seconds (using the Magic Link's 2400 baud modem) when talking to a "dummy" test server that generates a canned set of responses for the client, and about 5 seconds when talking to the real Charon module and interacting with the Kerberos infrastructure. These results confirm that the "untrusted Charon" model, in which the client never relinquishes $K_{c,tgs}$ to the proxy but instead constructs an authenticator for every new service request, yields acceptable performance even on PDA-class devices.

In order to decide whether DES encryption on the PDA could keep up with the expected network throughput of the device, we ran a series of DES throughput benchmarks on our unoptimized DES implementation. The benchmarks consisted of repeatedly encrypting buffers of various sizes, and computing the encryption throughput as a function of the input buffer size. The results (summarized in Table 1) confirm that DES throughput can adequately keep up with network throughput speeds of up to 32 kilobits per second, assuming that the CPU is dedicated to the task of encryption. Note that our Magic Link DES implementation has not been optimized for performance, so it is conceivable that even greater throughput could be achieved with more effort.

DES throughput is seen to increase for increasing buffer sizes up to a maximum of approximately 5.2 KB/s at an input buffer size of 2048 bytes, after which it decreases to a stable value of 4.10 KB/s for larger buffer sizes. The initial increase in throughput is explained by the relative decrease in function call overhead for larger buffer sizes. The drop in throughput for input buffers greater than 2048 bytes is explained by the fact that the on-chip static RAM cache of the Magic Link's CPU (a Motorola 68349 "Dragon" processor) is 4 KB in size—at most 2048 bytes for the plaintext input buffer and 2048 bytes for the encrypted text output buffer can fit in such a cache. For larger input buffer sizes, the input and output buffer together exceed the capacity of the cache.

Buffer size (bytes)	DES Throughput (kilobytes/sec)
8	4.92 ± .01
32	5.00 ± .01
128	5.21 ± .03
512	5.22 ± .03
2048	5.02 ± .02
8192	4.10 ± .03
12288	4.10 ± .03

Table 1: DES throughput on a Sony Magic Link PDA

4 Security of Proxied Applications

This section examines potential attacks on a Charon implementation and on the protocol itself. There are three agents in our system: the client, the proxy (including Charon), and the Kerberos infrastructure agents (in particular, the TGS and the KDC). Each of these could potentially be compromised by an attacker. We omit discussion of attacks on the Kerberos protocol and agents, as these are treated in detail elsewhere [7, 12]. Notice that while Charon corresponds to a much simpler *implementation* of certain Kerberos protocol messages, the *abstract* properties of the messages (from which Kerberos' strength derives) are identical to those of Kerberos.

4.1 Attacks by Malicious Clients

Charon inherits Kerberos' resistance to attacks by malicious clients. Although a TGT that has been decrypted by a client can be captured by sniffing network traffic, it is useless without the session key $K_{c,tgs}$, which is never transmitted in the clear but is required to construct an authenticator. Replay attacks that include captured authenticators are also futile because of the timestamp embedded in the authenticator[4] and because the attacker does not possess $K_{c,tgs}$.

4.2 Attacks on the Charon Host Workstation

Subversion of the workstation hosting the proxy side of Charon has the same security implications as subversion of, e.g., a Kerberized host running *krlogind*—a client may believe it has connected to a legitimate Kerberized service rather than a subverted one. If the proxy workstation is compromised, the attacker can obtain the Kerberos credentials stored in the *krb.srvtab* file, which in turn can be used to obtain all session keys, secret keys, and tickets that Charon ever possesses. Fortunately, this includes neither the client's key K_c, nor the session key $K_{c,tgs}$ for constructing authenticators (since it has been stripped from all tickets stored in the ticket file). Therefore, services already requested by a client

4: There is a well-known Kerberos attack that involves compromising a network time service (see, for instance, [7]). We do not address this attack, but merely assert that our protocol is no worse that Kerberos in this regard.

can be commandeered, but no new services can be initiated without the user's explicit cooperation.

Networked filesystem attacks resulting in unauthorized write access to a filesystem [13] or on-the-wire binary patching [21] can also result in compromised binaries being executed on the workstation, thus compromising the workstation. We guard against this by running Charon and the proxy from a local file system, trusting the compiler that generated the executables. Note that these techniques cannot be used to trick the client into revealing her secret key by executing a compromised binary of *kinit* or *kpasswd*, since in Charon the critical *kinit* functionality resides on the PDA client, which by nature cannot be subverted by such an attack.

4.3 Attacks on the Charon Proxy

We maintain that the required trust in the Charon proxy is not unreasonable: it is the same trust placed in existing Kerberized services such as *krlogind*. Suppose an attacker has no access to the host workstation (and therefore the Charon service keys), but attempts to impersonate Charon via a man-in-the-middle attack on the network connecting the client to Charon. Since the attacker cannot access the Charon service keys, it will be unable to complete the handshake, and the client will know something is amiss.

On the other hand, suppose the handshake completes successfully. This means that either the user has connected to a legitimate Charon, or else the user has connected to a compromised Charon (which implies that the workstation and possibly parts of the filesystem have been compromised as well, as would be required for successfully impersonating a Kerberized service). In this case, all hope is lost, as would be true if the user unwittingly connected to a compromised *krlogind*. The compromised Charon can capture user traffic, although the user's Kerberos password is still safe. If, on the other hand, Charon is legitimate, replacing an (already running) Charon binary with a compromised one on the fly is nontrivial at best.

4.4 Strength of Charon's Protocol

In [12], a logic for examining the integrity of an authentication protocol is described. This logic was used to formally verify the correctness of the Kerberos protocol, and to help extract the list of assumptions that must be made for the protocol to remain valid.

Because the Charon implementation preserves all the semantics of the Kerberos handshake messages, the same proof of the Kerberos protocol from [12] can be used unmodified to prove the correctness of the Charon handshake protocol. This result implies that the same set of assumptions must be made when using Charon that must be made for Kerberos, and that our modifications to the Kerberos protocol do not change its ideal, abstract logical properties. This is intuitive in retrospect; Charon acts as a data tunnel during the first half of the authentication handshake, and as both a data tunnel and a Kerberized service during the second half of the handshake. When acting as a data tunnel, it can be considered as part of the network infrastructure, which Kerberos already assumes to be untrustworthy. If the client decides to relinquish service session keys to the proxy, obviously the security properties and assumptions of the Charon protocol diverge from those of Kerberos.

The client must assume that the proxy is trustworthy and uncompromised. Although strictly speaking, this same assumption must be made when clients converse with Kerberized services such as *krlogin*, relinquishing session keys to a proxy means that the proxy could potentially perform actions as the client, with neither the knowledge nor consent of the client.

The specific assumptions that must be made about the Kerberos (and therefore Charon) protocol are:

- all principals must trust that the KDC and TGS are not compromised, which implies that the secret keys shared with them are safe;
- principals must trust that the KDC and TGS generate good session keys, and that those session keys are never known to anyone besides the KDC, the TGS, and the principals participating in the session;
- principals must believe that timestamps embedded in tickets and authenticators are "fresh", i.e. that duplicated or old tickets and authenticators used in a replay attack can be detected and rejected.

The fundamental implication of these assumptions is that parties that want to communicate with each other securely must trust each other, and that all parties involved (principals, the KDC, and the TGS) are uncompromised, including the clocks that those parties access.

4.5 Charon Security

The security properties achieved by an uncompromised Charon implementation are as follows:

1. Charon may be trusted with temporary session keys for particular services it contacts on the client's behalf, but not with the user's Kerberos password or with sufficient information to impersonate the user in case Charon is compromised. Each time an additional service is requested, the client's cooperation is required to gain access.
2. If Charon is undermined, the worst the attackers can do is obtain valid tickets for a currently-accessed Kerberized service. These tickets can be given short lifetimes to minimize damage.
3. Eavesdropping on the PDA-to-proxy connection is no more useful than eavesdropping on any (untrusted) network connection in a Kerberos infrastructure.
4. The abstract Kerberos protocol is used unmodified. In particular, the same proof used to demonstrate its correctness and security [12] applies to Charon. From this point of view, Charon represents a *repartitioning* of the implementation of the unmodified abstract Kerberos protocol, rather than a modification to the abstract Kerberos protocol itself.

4.6 Who Does Charon Run As?

Since Kerberos performs all authentication using tickets and authenticators rather than user ID's, the proxy side of Charon can run as an unprivileged program. All that is required is that Charon must be able to exclusively read its service key file, which can be stored on the local filesystem and made accessible only to a fictitious (unprivileged) user *charon*. If, however, the user wants to access any non-Kerberized services that authenticate by UID (e.g. traditional NFS), this model will fail. One obvious solution is to allow Charon to switch to the user's UID after authentication, but this requires some part of Charon to run as *setuid root*—just as current programs like *rlogind* do. Since programs of nontrivial complexity represent inherent security weaknesses which have been successfully exploited [6], we prefer to avoid the *setuid* method and are plan to investigate alternate ways to provide legitimate users access to non-Kerberized UID-authenticated services.

Note that this entire issue is moot for services such as Web access, where any authentication occurs at the application level using separate mechanisms (though Kerberos authentication for the Web is under investigation [44]).

4.7 End-to-End vs. Proxied Kerberized Services

Once the client has authenticated itself to the proxy, there are two models for client access to Kerberized services, particularly those such as *krlogin* that provide an encrypted channel:

- Proxied service: The proxy knows the session key for the service. It decrypts data from the server, optionally performs some transformations on it, and optionally re-encrypts it before passing it to the client.
- End-to-end service: The proxy does *not* know the session key for the service; it acts purely as a router (and possibly network gateway) for the client.

We motivate the proxied service model by pointing to the success of our proxy architecture at addressing client variation with respect to hardware, software and connectivity [17]. For example, we have built (separately) an MH-compatible MIME email client for the Sony Magic Link. This client retrieves MIME email via a proxy which performs *distillation* [16] on included MIME images, optimizing the images for display on the PDA's screen. Clearly, such transformations are impossible unless the proxy has access to decrypted data.

The proxy can, of course, re-encrypt the data (using the client-to-proxy session key $K_{c,p}$ obtained as described in section 2.2) before forwarding it to the client. Re-encryption may be unnecessary if the network can perform separate link-level encryption, as is the case with GSM [11]. But in any case, granting the proxy session keys to Kerberized services implies a certain amount of trust in the proxy.

5 Other Applications and Work in Progress

5.1 Charon, Daedalus, and GloMop

The Daedalus project [29] is investigating wireless overlay networks and cross-roaming agreements between administrative domains. Proxied services are provided to wireless clients in the Daedalus system via a proxy architecture called GloMop, which is described extensively elsewhere [9, 17]. The Charon module of the GloMop proxy will be used within the Daedalus architecture in two ways:

- to authenticate users of the GloMop proxy in their home domain (Kerberos realm);
- to authenticate users visiting another domain, by verifying that they can authenticate themselves to some home domain with which the visited domain has a pre-established trust agreement.

The second function can be realized by exploiting support in Kerberos for inter-realm authentication. It requires only that a shared secret be established between any pair of KDC's whose Kerberos realms have established cross-roaming agreements. Charon is an attractive option for Daedalus because it is a strong protocol, because it uses a widely deployed authentication infrastructure, and because it requires only minimal changes to each domain's Kerberos database to establish a cross-authentication channel.

5.2 Kerberos V

Kerberos V [30] has explicit support for both proxied authentication and hierarchical inter-realm authentication forwarding. These are attractive features for proxy-based mobile computing. However, the Kerberos V implementation is still in flux, and there is considerable debate over the effectiveness of the implementations of these and other new features [7]. It will evidently take some time for these problems to be resolved and for Kerberos V to be widely implemented and deployed. Until then, we rely on the proven, highly scrutinized, and widely-disseminated Kerberos IV.

5.3 *PDAkinit*—Authentication From Your PDA

One "hole" in the existing Kerberos implementation is that users must type their Kerberos passwords to the *kinit* program to obtain an initial TGT. These keystrokes are fairly easy to capture, by snooping unprotected traffic to the X server, tampering with keyboard device drivers, or simply peering over somebody's shoulder.

On the other hand, if PDA's are used for authentication, an attacker would have to compromise software on the PDA's in order to capture their Kerberos passwords. Since PDA's are very personal devices that are rarely left unattended or connected to a network in such a manner than an over-the-network attack is possible, they are harder to compromise so as to capture keystrokes and transmit them to an attacker. Thus, performing the initial Kerberos handshake from a PDA rather than the keyboard may appeal to very security-conscious users.

5.4 Related Work

In [15], an agent called a "translator" is described that securely accesses Kerberized services on behalf of clients of the Institutional File System (IFS). In this system, each client acts like a Kerberos service - the translator initially acquires a ticket for the client service, and uses that ticket to establish a secure channel to the client. The client then relinquishes its TGS session key $T_{c,tgs}$ to the translator over this secure channel, at which point the translator can then initiate service sessions without further interaction with the client. This approach has the advantage that the clients' interaction with the translator is reduced to at best a single interaction, but suffers from the weakness that a compromised translator can initiate new sessions without the knowledge or permission of clients. This system also did not partition complexity across the client and translator - each client ran a full implementation of Kerberos IV.

Application-level transcoding proxies are becoming more and more common. Some proxies provide applications with enhanced functionality or access to data [45, 10]. Others attempt to adapt content to better suit clients with hardware, software, or network limitations [28, 35, 16, 17].

Proxies may also provide an integrated combination of both network and application level services [24, 46, 31]. Charon belongs to this category: although its primary goal is to reduce the application-level complexity of interaction between the client device and the Kerberos infrastructure, the simple requirements for its client connection allow it to secondarily act as a protocol gateway between the TCP/IP Kerberos infrastructure and a client using a reliable-stream protocol as simple as LAPM [26].

The application architecture of the Daedalus wireless networking project [29], called GloMop, is built around the idea of proxied services; Charon was originally implemented in this context, although its applicability is potentially farther-reaching and its implementation essentially independent of the GloMop architecture.

5.5 Conclusions

The Charon system provides indirect authentication and secure communication between a lightweight client and a proxy, and enables the provision of secure proxied services to the client. Charon represents a client/proxy partitioning of Kerberos IV functionality that minimizes the resources needed on the client, which is absolutely essential when dealing with PDA-class devices, yet retains the desirable security properties of unmodified Kerberos IV. In particular, Charon has the following properties:

- Only DES encryption and decryption need be ported to clients, which typically suffer from limited resources and dissimilar development environments.
- Rather than understanding multiple message formats and communicating with multiple Kerberos entities, the client needs to understand only a single message format and communicates exclusively with a single entity (the Charon proxy), which can also serve as a network gateway if necessary.
- Neither the user's Kerberos password nor the key required to construct Kerberos authenticators (used when requesting access to new services) ever leave the client.
- Charon has the same immunity to protocol-based attacks as Kerberos does, and is more immune to certain end-to-end attacks because of the nature of the devices on which it is designed to run, providing an alternate means of Kerberos authentication for security-conscious users.

5.6 Acknowledgments

Charon was done as a semester project for a course in wireless mobile communications, with the initial goal of providing an attractive authentication and cross-roaming mechanism for Daedalus. Eric Brewer pointed out the potential applicability of indirect authentication to network computers, intelligent personal communicators, and ISP's.

Eric Anderson, Trevor Pering, and Mark Stemm provided valuable comments on both content and presentation. Ian Goldberg and David Wagner, cypherpunks extraordinaire, provided useful insights on both the practical aspects and the security aspects of the Charon design and implementation. Venkata Padmanabhan suggested the snappy catch phrase "Security on the Move" for the paper title. The reviewers' comments helped us restructure the paper to emphasize Charon's role in a proxy architecture.

6 Bibliography

[1] M. Abrams *et al*. Caching proxies: limitations and potentials. In *Proceedings of the Fourth International World Wide Web Conference*, Boston, MA, USA, December 1995.

[2] E. Anderson. Personal communication, May 1996.

[3] Apple Computer, Inc. Newton internet enabler. Home page "http://devworld.apple.com/dev/newton/tools/nie.html".

[4] A. Aziz and W. Diffie. Privacy and authentication for wireless local area networks. *IEEE Personal Communications*, pages 25–31, First Quarter 1994.

[5] H. Balakrishnan, S. Seshan, E. Amir, and R. H. Katz. Improving reliable transport and handoff performance in cellular wireless networks. *ACM Wireless Networks*, vol. 1, no. 4, December 1995.

[6] S. M. Bellovin. Personal communication, May 1996.

[7] S. M. Bellovin and M. Merritt. Limitations of the kerberos authentication system. In *Proceedings USENIX Winter Conference 1991*, Dallas, Texas, USA, 1991.

[8] C. Mic Bowman *et al*. The harvest information discovery and access system. In *Proceedings of the Second International World Wide Web Conference*, pages 763-771, Chicago, Illinois, October 1994.

[9] Eric A. Brewer et al. The GloMop project (home page). "http://www.cs.berkeley.edu/~fox/glomop".

[10] C. Brooks, M. S. Mazer, S. Meeks, and J. Miller. Application-specific proxy servers as http stream transducers. In *Proceedings of the Fourth International World Wide Web Conference*, December 1995.

[11] D. Brown. Techniques for privacy and authentication in personal communication systems. *IEEE Personal Communications*, August 1995.

[12] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, February 1990.

[13] W. R. Cheswick and S. M. Bellovin. *Firewalls and Internet Security: Repelling the Wily Hacker*. Addison-Wesley Professional Computing Series. Addison-Wesley, 1994.

[14] D. Cox. Wireless personal communications: What is it? *IEEE Personal Communications*, pages 20–35, April 1995.

[15] B. Doster and J. Rees. Third party authentication in the institutional file system. *CITI Technical Report 92-1*, Center for Information Technology Integration, University of Michigan, February 1992.

[16] A. Fox and E. A. Brewer. Reducing WWW latency and bandwidth requirements via real-time distillation. In *Proceedings of the Fifth International World Wide Web Conference*, Paris, France, May 1996. World Wide Web Consortium.

[17] A. Fox, S. Gribble, E. Brewer, and E. Amir. Adapting to network and client variability via on-demand dynamic distillation. In *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VII)*, Cambridge, MA, October 1996.

[18] Y. Frankel *et al*. Security issues in a cdpd wireless network. *IEEE Personal Communications*, 2(4):16–27, August 1995.

[19] Geos operating system for PDA's (home page). "http://www.geoworks.com/htmpages/sso.htm".

[20] Geoworks and Nokia. Nokia 9000 communicator (home page). "http://www.geoworks.com/htmpages/9000.htm".

[21] I. Goldberg, D. Wagner, E. A. Brewer, and P. Gauthier. Basic flaws in internet security and commerce. "http://www.cs.berkeley.edu/~gauthier/endpoint-security.html".

[22] S. Gribble. Real-time streams to PDA's. "http://www.cs.berkeley.edu/~gribble/cs294-3_multimedia/project/project.html".

[23] T. R. Halfhill. Inside the web pc. *Byte Magazine*, pages 44–56, March 1996.

[24] IBM Corporation. IBM ARTour publications page. "http://www.raleigh.ibm.com/art/artpub.htm".

[25] J. Ioannidis, D. Duchamp, and G. Maguire, Jr. IP-based protocols for mobile internetworking. In *Proceedings of SIGCOMM '91*, pages 235-245, Zurich, Switzerland, September 1991.

[26] ITU-T Recommendation v.42. International Telecommunication Union, March 93.

[27] H. Jerome and J. Saltzer. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, September 1975.

[28] A. D. Joseph *et al*. Rover: A toolkit for mobile information access. In *Proceedings of the Fifteenth Symposium on Operating Systems Principles*, Copper Mountain Resort, Colorado, December 1995.

[29] R. H. Katz *et al*. The daedalus project (home page). "http://

daedalus.cs.berkeley.edu".

[30] J. Kohl and C. Neuman. The kerberos network authentication service (V5). *Request for Comments (RFC) 1510*, September 1993.

[31] M. Liljeberg *et al*. Enhanced services for world wide web in mobile WAN environment. *University of Helsinki CS Technical Report No. C-1996-28*, April 1996.

[32] National Bureau of Standards. Data encryption standard. Technical Report Publication 46, Federal Information Processing Standards, 1977.

[33] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *CACM*, 21(12):993–999, December 1978.

[34] Netscape Communications Corporation. The netscape proxy server (home page). "http://www.pinncomp.com/netscape/proxy_server.html".

[35] B. D. Noble, M. Price, and M. Satyanarayanan. A programming interface for application-aware adaptation in mobile computing. *Proceedings of the Second USENIX Symposium on Mobile and Location-Independent Computing*, Ann Arbor, MI, USA, April 1995.

[36] Ray Rischpater. Using magic cap's TCP/IP stack. *PDA Developers* 4.2, March/April 1996.

[37] J. Romkey. A nonstandard for the transmission of ip datagrams over serial lines: SLIP. *Request for Comments (RFC) 1055*, June 1988.

[38] J. H. Saltzer, D. P. Reed, and D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems*, vol. 2, no. 4, November 1984.

[39] W. Simpson, Ed. The point-to-point protocol (PPP). *Request for Comments (RFC) 1661*, July 1994.

[40] J. G. Steiner, C. Neuman, and J. I. Schiller. Kerberos: an authentication service for open network systems. In *Proceedings USENIX Winter Conference 1988*, pages 191–202, Dallas, Texas, USA, February 1988.

[41] Sony Corporation. Sony Magic Link PDA (home page). "http://www.sel.sony.com/SEL/Magic/".

[42] Sun Labs. The java language: A white paper. Available at "http://java.sun.com".

[43] L. Wall and R. L. Schwartz. *Programming perl*. O'Reilly & Associates, Inc., 1991.

[44] World-Wide Web Consortium Security Group. A presentation on incorpating kerberos authentication into HTTP. Available at "http://www.ncsa.uiuc.edu/InformationServers/adam/w3c/KRB1.HTM".

[45] K.-P. Yee. Shoduoka mediator service. "http://www.lfw.org/shodouka".

[46] B. Zenel. A proxy based filtering mechanism for the mobile environment. Thesis Proposal, March 1996.

Appendix A: Review of the Kerberos Authentication Protocol

In this section we review the unmodified Kerberos protocol, so that we can then describe Charon by indicating how it differs from Kerberos. The Kerberos authentication protocol is an example of the Needham and Schroeder [33] key distribution protocol. The protocol is depicted in figure 1a—we briefly review it using the following terminology. Recall that in Kerberos, each entity capable of authenticating itself is referred to as a *principal*, and that to each principal corresponds a secret *key* known only to itself and the Kerberos Key Distribution Center (KDC).

In the first stage of the protocol, the client attempts to obtain a ticket-granting ticket (TGT); this step requires the client's principal to supply his Kerberos password. Once obtained, the TGT allows the client to request tickets for additional services from the TGS, without requiring the client's principal to retype the Kerberos password each time. These additional tickets are presented to the appropriate service as a means of proving the authenticity of the client's request for service. The exact messages exchanged are summarized as follows:

Messages 1-2: the client obtains a TGT. These are the messages exchanged by the *kinit* program.

1. C→KDC:
   The client requests a TGT from the TGS.
2. KDC→C: $\{K_{c,tgs}, \{T_{c,tgs}\}K_{tgs}\}K_c$
   The KDC returns the TGT (encrypted with the TGS's key) and a session key for use with the TGS, all encrypted with the client's key.

Messages 3-4: the client obtains a ticket for a service

3. C→TGS: S, $\{T_{c,tgs}\}K_{tgs}$, $\{A_c\}K_{c,tgs}$
   After decrypting message 2 using K_c, the client requests a ticket for service S, presenting the TGT (to authenticate itself) and a constructed authenticator (to guard against replay attacks) as credentials.
4. TGS→C: $\{K_{c,s}, \{T_{c,s}\}K_s\}K_{c,tgs}$
   The TGS returns a ticket for the service encrypted with the service's key, and a session key for use with that service, all encrypted with $K_{c,tgs}$.

Messages 5-6: the client requests service

5. C→S: $\{T_{c,s}\}K_s$, $\{A_c\}K_{c,s}$
   After decrypting message 4 using $K_{c,tgs}$, the client presents the ticket $T_{c,s}$ and authenticator A_c to the service.
6. S→C: service begins.