CITI Technical Report 98-7

Smartcard Integration with Kerberos V5

Naomaru Itoi
`itoi@eecs.umich.edu`

Peter Honeyman
`honey@citi.umich.edu`

Abstract

We describe our design and implementation of smartcard integration with Kerberos V5. Authentication is among the most important applications for smartcards and is one of the critical requirements for computer security. By augmenting Kerberos V5 with tamper-resistant hardware, we enhance the security of Kerberos V5 and offer a potential "killer application" leading to wider adoption of smartcard technology.

December 3, 1998

Center for Information Technology Integration
University of Michigan
519 West William Street
Ann Arbor, MI 48103-4943

Smartcard Integration with Kerberos V5

Naomaru Itoi and Peter Honeyman
`itoi@eecs.umich.edu, honey@citi.umich.edu`

1 Introduction

Smartcards are a rapidly emerging technology that have received much attention both from industry and academia. Smartcards can make significant impact on current computer systems because of their inherent security and mobility.

According to market researcher Dataquest, the smartcard market will grow from 544 million units in 1995 to 3.4 billion units by 2001. However, the vast majority of smartcards are used in Europe; 90 percent of worldwide smartcard shipments went to Europe in 1995. Only 2 percent were shipped to the Americas [4].

Smartcards are not popular in the United States because there is no "killer application" for smartcards here. Smartcards were introduced to Europe by government telecommunications monopolies in the form of phone cards, but the telecommunications industry in the US is private and decentralized.

These cultural and economic differences are common to other smartcard applications prevalent worldwide, such as health care and banking. In addition, credit cards are more successful in the US than in Europe, in part due to the prevalence of online verification, which is universally available in the US. This keeps fraud rates low – reportedly 0.07% [11] – which allows card issuers to indemnify customers for any loss over 50 USD. Consequently, issuers, customers, and merchants are equipped and satisfied with magstripe cards and readers, which feature low cost and broad familiarity [3].

The information technology business sector might provide the killer application for the smartcard industry in the United States because the demand for secure computer environments is huge and growing. There is growing fear of hackers attacking sensitive information on the Internet. Smartcards can provide a secure authentication system when combined with sound authentication protocols, and can significantly improve computer system security wherever authentication plays a critical role in the computer security scheme, *i.e.*, everywhere.

Here at the University of Michigan, smartcards are already deployed and used for storing a small amount of cash. Thus, we have a good setting for extending the deployment of smartcards in the computer environment:

- Students and faculty are familiar with using smartcards.
- An infrastructure is well established, *e.g.*, many vending machines and shops have smartcard readers.
- There is a serious security problem that can be solved by integrating smartcards into the computer environment.
- University technologists, especially at the Center for Information Technology Integration (CITI), have skill sets and resources to develop smartcard applications.

The goal of our project is to develop, build, and deploy a smartcard-integrated computer environment. We want to provide a smartcard in everyone's pocket that handles computer authentication, computer profiles, electronic cash, banking, identification, course registration, paying rents, submitting resume, copy machines, *etc.* [6].

The centrally administered computing environment at the University of Michigan is protected by Kerberos, the most widely used network authentication protocol [21, 13]. Kerberos is also key to the security infrastructure at MIT (where Kerberos was invented), Cornell, Carnegie-Mellon, and Stanford, as well as in commercial product offerings from Microsoft and Oracle.

Kerberos suffers from some inherent security pitfalls, principally its reliance on passwords selected by users. In recent years, CITI staff have used password guessing attacks [6, 7] on the University of Michigan Kerberos servers with (disappointing) success, quickly obtaining thousands of passwords on each occasion. To improve the security of Kerberos and the infrastructure it protects, we intend to replace passwords with randomly generated Kerberos keys stored on a smartcard.

The remainder of this paper is organized as follows. In Section 2, we describe why and how smartcards can enhance the security of Kerberos. In Section 3, we explain the protocol we use to integrate Kerberos with smartcards. Section 4 contains implementation details for those who want to port our program to other operating systems or to use other types of smartcards. (Readers who are not interested in implementation details may want to skip the section.) Performance is evaluated in Section 5. Section 6 discusses related work and smartcards we have examined. Future directions are described in Section 7, followed by concluding remarks in Section 8.

2 How can smartcards help in Kerberos?

Bellovin and Merritt enumerate problems of Kerberos that "are not solvable without employing special-purpose hardware, no matter what the design of the protocol." [2] The problems are:

- Need for secure encryption device
- Need for secure key storage
- Dictionary attack on passwords

We explain these problems, and describe countermeasures that take advantage of strong security feature of smartcards.

2.1 Need for external encryption device

In the Kerberos protocol, a user key, K_u, is shared between a user and a *Key Distribution Center* (KDC), a trusted third party. K_u is derived from a password: a workstation reads the password from a user, converts it to K_u, and uses it to decrypt a *ticket granting ticket* (TGT), an initial credential in Kerberos. The protocol is shown in Figure 1.

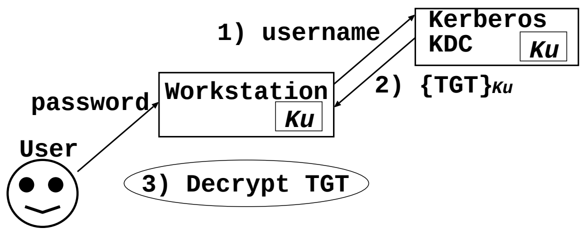

Figure 1: Kerberos authentication protocol without a smartcard

1) When a user attempts to login to a workstation, the workstation sends a request to the KDC. 2) KDC generates a TGT, encrypts it with K_u, and sends it back to the workstation. 3) The workstation asks the user for a password, hashes it into key K_u, and uses the key to decrypt the TGT. If the TGT decrypts properly, the user is authenticated and is allowed to login.

In this protocol, K_u is exposed to two parties, a user and a workstation. A key memorized by a user can be vulnerable because she can tell it to another person, or an adversary might "shoulder surf" it when she types it. A key in a workstation can be vulnerable if the workstation is not securely protected or cannot be trusted for other reasons. For example, if an adversary can scan the entire physical memory of the workstation, he can obtain the key. Along the same lines, if someone has administrative access rights to the workstation, it is straightforward to install a rogue login program in the workstation that stores a user's password in the adversary's directory. (This is called a Trojan horse attack.)

To solve these problems, it is desirable to decrypt the TGT outside a workstation. Therefore, an external encryption device is required.

2.2 Need for secure key storage

Kerberos stores some keys in computers, *e.g.*, session keys in a workstation and user keys in

KDC. However, typical computers cannot store information securely. Information in a computer system is stored either in memory or in a hard disk, but neither is sufficiently secure. A secret in a hard disk is hard to protect because:

- A powerful adversary can access (read and write) it.
- It is usually backed up in mass storage devices, which may lack sufficient physical or cryptographic protection.

A secret in memory is also hard to protect because :

- Memory can be physically scanned by a powerful adversary.
- It may be paged out to hard disks, which can be scanned.

Therefore, secure storage outside a workstation and KDC is an important goal.

2.3 Dictionary Attack

When a user chooses a poor password, the derived user key K_u, is subject to a dictionary attack. Dictionary attack is performed as follows:

1. Create a list of common words, names, etc.
2. Derive keys from the words in the list.
3. Obtain a <plaintext, ciphertext> pair.
4. Decrypt the ciphertext with the derived keys.
5. If the plaintext is recovered correctly, the key used for decryption is revealed.

For example, if the password is a short English word, an adversary can try all English words in the dictionary and quickly discover the password.

Kerberos is vulnerable to dictionary attack because:

1. It is a password-based authentication protocol.
2. It easily gives up a <plaintext, ciphertext> pair to the adversary.

Test runs of dictionary attack in the University of Michigan Kerberos realm have yielded passwords for more than 5% of the user accounts, *i.e.*, over 4,000 accounts [6].[1]

To solve problem (2), pre-authentication is introduced in Kerberos V5. The Kerberos authentication protocol with pre-authentication is depicted in Figure 2.

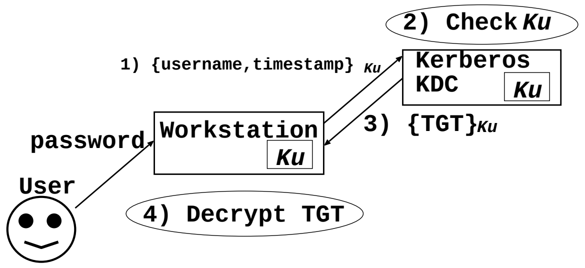

Figure 2: Kerberos authentication protocol with pre-authentication

In this scenario, KDC ensures that the client knows K_u before issuing a TGT. 1) When the client requests the TGT, it sends a username and a timestamp encrypted with K_u. 2) If KDC can successfully decrypt with K_u and recover the username and a valid timestamp, it is sure that the client knows K_u. If not, KDC assumes someone is forging the client to obtain a <plaintext, ciphertext> pair and rejects the request. 3) After pre-authentication, KDC sends TGT encrypted by K_u to the workstation and the protocol continues as depicted in Figure 1.

Pre-authentication prevents an adversary from getting a <plaintext, ciphertext> pair just by requesting it, and thus raises the bar of security to the adversary. However, the adversary can still eavesdrop a network to obtain a <plaintext, ciphertext> pair. Also note that it is very easy for the adversary to recognize a plaintext because it includes well known entries such as a user name and a realm name.

As long as Kerberos uses passwords for secure information, dictionary attack cannot be solved completely. Therefore, it is desirable to replace passwords with randomly generated

[1] The most common password was "love."

bits stored in tamper-resistant hardware [17].

A smartcard is an ideal device to solve the problems outlined here. The countermeasures are described in the next section.

3 Design

In this section, we describe a method intended to enhance the security of Kerberos. It takes advantage of a smartcard to solve the problems stated in Section 2.

From the discussion in Section 2, our design goals are:

- Use randomly generated bits for K_u.

  We can prevent dictionary attack by using a random key instead of a user chosen password. However, we then require a way for users to possess their keys, as it is impossible (and insecure!) to expect anyone to remember a random string of any substantial size.

- Store a user key in a smartcard.

  A smartcard can serve as an external key storage because it is designed to be tamper-proof with restricted communication mechanisms.

- Decrypt TGT in a smartcard.

  A smartcard can perform decryption as an external encryption device because it has DES en(de)cryption mechanisms.[2]

- Do not modify KDC.

  If KDC must be modified to implement the smartcard augmentations, then our efforts will offer enhanced security in our local Kerberos realm, but nowhere else. We also want our improvements to enhance the security of Kerberos realms beyond our administrative control.

3.1 Protocol

Figure 3 shows our Kerberos authentication protocol with a smartcard. Steps 1) and 2) are identical to the original protocol (Figure 1). 3) When the workstation receives the TGT, it does not decrypt it by itself. Instead, it sends the TGT to a smartcard. 4) The smartcard then decrypts the TGT, and returns the TGT in plaintext to the workstation. 5) If the workstation confirms that the decrypted TGT is correct, the protocol is finished and the user is authenticated. The protocol satisfies the goals we stated above; TGT is decrypted in the smartcard, K_u never leaves the smartcard, K_u can be random bits, and KDC is not modified [3].

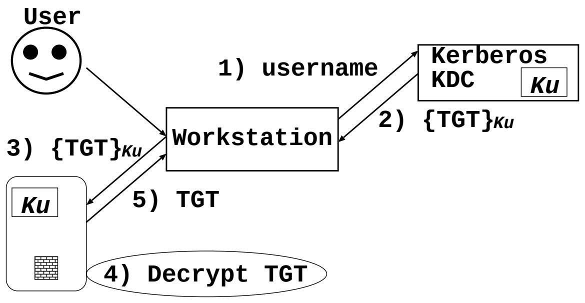

Figure 3: Kerberos authentication protocol with a smartcard

4 Implementation

We implemented the smartcard integrated Kerberos protocol described in Section 3. We now detail the modifications we made to the Kerberos library, the DES library, and the Kerberos client.

TGT decryption is implemented with a STARCOS version 2.1 smartcard from Giesecke & Devrient. This card offers superior performance by providing native *cipher block chaining* (CBC) for long messages. (A Kerberos V5 TGT is over 200 bytes long.) The development platform is OpenBSD-2.2 on Pentium 133MHz PC. The code base is Kerberos version 1.0.5 released by MIT.

[2] Or claims to. Many smartcards claim to offer DES but they in fact do not. We discuss this further in Section 6.2

[3] In fact, KDC in Kerberos V5-1.0.5 must be modified by one line to run the protocol due to a bug in Kerberos. However, this modification will not be necessary in later version of Kerberos. We discuss it in Section 4.1.

4.1 Adding an encryption system in Kerberos library

Kerberos V5 uses a look-up table to provide for easy replacement and development of encryption systems [12]. The look-up table associates an encryption type to cryptographic functions, such as encryption, decryption, and checksum functions, and data structures, such as a key structure. It is simple to add a new encryption system entry by adding an entry to the look-up table.

There are several encryption system types defined in the RFC[12] and implemented in Kerberos V5-1.0.5 including:

- NO encryption
- DES in CBC mode with a CRC-32 checksum (`des-cbc-crc`)
- DES in CBC with MD5 (`des-cbc-md5`)

We created a new encryption system, DES in CBC with MD5 with a smartcard (`des-cbc-md5-sc`). We added a new entry `des-cbc-md5-sc` in the look-up table. The entry is defined in `des_md5.c` (Figure 4).

```
krb5_cryptosystem_entry
  mit_des_md5_sc_cryptosystem_entry {
    EncryptionType  ENCTYPE_DES_CBC_MD5_SC;
    DecryptionFunc  mit_des_md5_sc_decrypt_func();
    // Other members are identical to des-cbc-md5
  };
```

Figure 4: Smartcard cryptosystem entry

`mit_des_md5_sc_decrypt_func()` is a new function that uses a smartcard for decryption. The other members of the entry are not modified.

Although the default hash method in Kerberos V5-1.0.5 is CRC, implementation of des-cbc-crc in Kerberos V5-1.0.5 has a bug. In the Kerberos 5 specification, the *initialization vector* (IV) of DES-CBC mode is defined to be 0 [12]. However, `des-cbc-crc` uses the user key as the IV. This error can not be fixed easily because Kerberos 5 is already deployed widely. The G&D smartcard cannot use the key as an IV without passing it as an argument to the card, which defeats our goal of eliminating the key on the workstation.

To our relief, `des-cbc-md5` uses 0 as the IV, complying with the RFC. Rumor has it that the next version of Kerberos will use `des-cbc-md5` by default.

4.2 Modifying DES library

`mit_des_md5_sc_decrypt_func()` calls the DES_CBC encryption function in `f_cbc.c`. We created a new DES_CBC function `mit_des_cbc_sc_encrypt()` that calls a DES function in a smartcard instead of a Kerberos DES library. STARCOS version 2.1 can handle up to 112 bytes in one command. The TGT, whose length is approximately 200 bytes, is divided into two pieces, decrypted in a smartcard piece by piece, and combined into one TGT in the workstation.

The specific commands, or APDUs in ISO 7816-4, sent to the smartcard are as follows. (Readers not familiar with smartcard APDUs are advised to consult the ISO 7816-4 specification [8] or Guthery and Jurgensen's book [5].)

- Send "decrypt" APDU with 112 (0x70) bytes of encrypted data.

```
0x80 0xf8 0x81 0x81 0x70 data ...
```

- Send "get response" APDU to upload 112 bytes of plaintext data.

```
0x00 0xc0 0x00 0x00 0x70
```

4.3 Modifying kinit

In the authentication function `get_in_tkt()`, an encryption system can be chosen as an argument. We modified `kinit.c` so that it does not request a password from the user, and specified encryption type `des-cbc-md5-sc` instead of `des-cbc-md5`.

5 Performance Evaluation

Here we evaluate the performance of our Kerberos modifications.

5.1 Performance Evaluation

We ran the authentication protocol described in Section 3.1 by executing the modified `kinit` program five times and logged salient performance data. The authentication time fluctuates within a relatively small range (1.53 - 1.57 sec.), averaging 1.55 sec. We analyze performance in detail in the following sections.

5.2 Time line

Figure 5 shows a time line of one of the five trials.

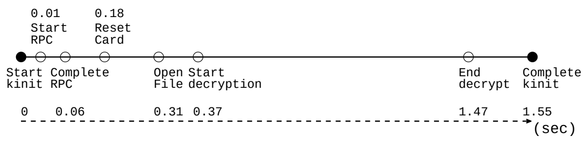

Figure 5: Time line

Decryption is the obvious bottleneck of the protocol, taking 1.1 sec., or 70% of the elapsed time. Decryption is slow – with an 8-bit data path and a 3.5 MHz clock, a smartcard is much slower than a workstation. In addition, about half of the elapsed time is due to the slow (9.6 Kbps) communication with the smartcard.[4]

5.3 Breakdown

Table 1 shows how much time is spent in each part of the protocol. Time is in ms.

part name	time (ms)	std. deviation
decryption	823	0.754
(non comm part)	498	0.0219
(comm part)	325	0.766
get response	304	0.0459
card reset + open file	245	0.00340
RPC	59.8	9.65
pre-processing	6.19	0.0583
post-processing	83.3	4.80
total	1521	8.76

Table 1: Authentication time breakdown

The decryption step is separable into two parts: the time to send the encrypted ticket to the smartcard, and the actual decryption time. Get response reads plaintext from a smartcard. Open file is required to select a key file. RPC time is for communication with KDC.

Total time to authenticate is 1.52 sec. Smartcard-related tasks – initialization, communication, and decryption – dominate, taking 1.37 sec. The rest of time, including RPC communication with KDC, is 0.15 sec. Of the 1.37 sec. of smartcard time, communication takes 0.629 sec., decryption takes 0.498 sec., and initialization takes 0.245 sec. Communication is the bottleneck; we anticipate improving performance by communicating at higher date rates.

[4]Many smartcards can be configured to communicate at higher speeds, up to 115 Kbps.

6 Discussion

6.1 Related Work

Here we relate this effort to secure computer systems, secure bootstrapping, smartcard authentication, and smartcard integration with Kerberos.

Secure computer system

We refer to two efforts with our goal in common, a secure computer environment.

In their paper "Dyad: A System for Using Physically Secure Coprocessors", Tygar and Yee describe their secure hardware and operating system in small computer systems (*i.e.*, workstations and PCs) [22]. They build a coprocessor that is physically tamper-proof and has the ability to process and store secrets. The coprocessor provides 1) secure bootstrapping, 2) secure logging, and 3) copy protection. They build an operating system called Dyad to operate on a secure coprocessor.

Their approach is top-down: unlike most security protocols, they do not assume security of low level components of computers such as operating systems, because an adversary can reload the kernel. To solve this problem, they build special purpose hardware, and build operating systems around it. This approach differs from ours, as we will explain.

Another related effort is described in "Authentication in Distributed Systems: Theory and Practice", by Butler Lampson *et. al.* [14]. They develop a theory of authentication for distributed systems based on an access control model. They build tools necessary for secure systems, such as encrypted channels, boot strapping, naming, and program loading. Accompanying the design of these tools are formal proofs of their security. Finally, they build an operating system to take advantage of the tools.

Their approach is also a top-down approach. They first design the authentication theory, and build an operating system based on the theory, then prove it secure through the design process.

Both Dyad and TAOS take top-down approaches: they start with a well-developed theoretical framework, then design secure hardware to support the theory, then build operating systems based on them.

Although these approaches are substantive and technically sound, they are not practical for most existing computer environment because they build new operating systems from scratch. We take a more pragmatic and experimental approach and build from the bottom-up for rapid implementation and deployment. We employ currently available, secure, inexpensive hardware in the form of commercial smartcards, integrate them with prevalent standards, and fit them – effortlessly – into our existing computer environment.

A disadvantage of our approach is that we still rely on the security of hardware and operating systems, of which we cannot be sure. (Often, we have great doubts!) For example, if an operating system is completely replaced, it is quite possible for an adversary to use stolen credentials to access resources.

Our solution to this problem is to store all critical secrets in a smartcard. A smartcard is tamper-resistant hardware, so no matter what happens to the hardware and the operating system, we can be confident that the secrets in the smartcard remain safe. In the previous example, even if the operating system is compromised, critical information in a smartcard, such as authentication keys, can not be accessed by the adversary. Therefore, our approach significantly "raises the bar" of security in a computer system with relatively small cost.

Secure Bootstrap

In their paper "A Secure and Reliable Bootstrap Architecture", Arbaugh *et. al.* introduce AEGIS, a secure bootstrap process [1]. They add a small PROM to commodity hardware. The PROM is assumed to be secure, *i.e.*, it is not replaced by the adversary. The PROM contains execution code to start bootstrapping and digital signatures. During the bootstrap process, all execution code is verified by the digital signature. At the end of the bootstrap process, a commodity operating system, FreeBSD in their example, starts up. As the execution code in PROM is trusted and bootstrap process is trusted, the operating system is trusted when it starts.

AEGIS is similar to our approach in the sense that both try to minimize components that must be trusted, the added PROM in AEGIS and the smartcard in our case. Also, both use commodity hardware and software. AEGIS and our approach complement one another because AEGIS aims at starting an operating system securely, and we aim at establishing a secure computer environment built on top of secure operating systems.

Authentication with Smartcards

Several authentication protocols that use smartcards have been proposed. For example, Rubin proposes one-time password [18], Shoup and Rubin propose session key distribution in the third-party setting [20], Leach proposes the use of zero knowledge authentication [15], and Wang and Chang propose use of public key authentication in smartcards [23]. Each of these concentrates on one-to-one authentication, such as when a user logs in to a computer. This differs from our approach in that we integrate a smartcard into a standard authentication protocol already in heavy use. Among them, only Shoup and Rubin's protocol has actually been implemented with a smartcard [10].

Smartcard Integration with Kerberos

In their paper "Enhancing SESAME V4 with Smart Cards", Looi *et. al.* describe smartcard integration with SESAME V5, a European implementation of Kerberos V5 [16]. Their approach is very similar to ours. They describe two ways of accomplishing smartcard integration:

1. Store a user key in a smartcard, load the key into a workstation, and use it for decrypting TGT instead of a derived key from a password.

2. Decrypt TGT in a smartcard.

Method 1 is not as secure as method 2 because the user key is loaded in a workstation. If the workstation is not trusted, the key is vulnerable. For example, a Trojan horse attack can easily obtain the key. Method 2, identical to our method, had not been implemented at the time of their writing.

6.2 DES in Smartcards

Many vendors claim that their smartcards support DES, but we had a very hard time getting a smartcard that meets our requirements, even though all we need is pure, unadulterated DES. Here we list some of the DES-capable smartcards that let us down when examined closely:

- Schlumberger CryptoFlex

  It seems to have DES, but they do not open the API, so we are unable to issue the proper APDUs.

- Schlumberger MultiFlex

  Internal authentication command returns the first six bytes of the eight bytes of encrypted data.

- IBM MFC

  The smartcard encrypts a random number challenge presented by SCT_CMD_AUTHENTICATE command.

- MAOSCO MULTOS

  The card supplied with the developer's kit encrypts with a fixed key, `0x41, 0xad, 0x82, 0x23, 0xa9, 0x0b, 0xe2, 0xa1`. According to the manual, "for security reasons," DES is used with a "known cryptographic key."

- General Information Systems OSCAR

  The DES key is XOR'ed with a random number before it is used. According to their e-mail: "The keys are XOR'ed with a random number for security reasons." While this may help secure the serial link between the terminal and the reader, it makes the card useless for enterprise security deployment.

- Gemplus GPK

  The key size is limited to 40 bit, a flaw not shared by Kerberos.

Eventually found a smartcard that satisfied our needs: Giesecke & Devrient STARCOS.

7 Future Direction

Comparison among several smartcards

We plan to implement the Kerberos authentication protocol in several smartcards, *e.g.* Schlumberger CryptoFlex, IBM MFC, MULTOS, and so on.[5] We expect to find some differences in their performance because:

- Some of the smartcards have DES CBC mode.

- Some of the smartcards have key scheduling APIs.

- Communication speed differs among smartcards.

We also expect to find differences in user friendliness and stability among smartcards and developer's kits.

Kerberos tickets in a smartcard

As we argued in Section 2, it is desirable to store keys in a smartcard rather than in a workstation. Therefore, storing session keys in addition to the user key in a smartcard adds security to the protocol. If tickets are stored on a smartcard, it is secure to leave a workstation to have a cup of coffee as long as the user

[5] If we receive smartcards with DES. See our discussion in Section 6.2.

brings the smartcard with her. Although an adversary can access the console, he cannot access resources protected by Kerberos because he does not have session keys.

Smartcard integration with PAM and NT-PAM

We will address secure single sign-on. Combined with PAM [19] or Windows NT-PAM [9], smartcards can provide secure single sign-on [7] because they can store keys and passwords securely, and can be integrated into existing authentication protocols, as we have shown in this paper.

8 Conclusion

In this paper, we identified certain limitation of Kerberos and ways that a smartcard can counter them. We suggested a protocol that takes advantage of the secure features of a smartcard to enhance security of Kerberos. The protocol is implemented with a Giesecke & Devrient STARCOS smartcard and Kerberos V5-1.0.5. Performance evaluation shows the protocol runs reasonably fast.

Acknowledgment

We thank Andrew Webb and Giesecke & Devrient America, Inc. for providing us with STARCOS smartcards and the smarts to use them effectively.

References

[1] William A. Arbaugh, David J. Farber, and Jonathan M. Smith. A secure and reliable bootstrap architecture. Technical Report MS-CIS-96-35, University of Pennsylvania, 1996. MS-CIS-96-35.

[2] S. M. Bellovin and M. Merritt. Limitations of the kerberos authentication system. In *Proceedings of the Winter 1991 Usenix Conference*, January 1991. ftp://research.att.com/dist/internet_security/ kerblimit.usenix.ps.

[3] Jorge Ferrari et al. *Smart Cards: A Case Study.* IBM Redbook, 1998. http://www.redbooks.ibm.com/ SG245239/sg245239.htm, Section 1.11.

[4] Smart Card Forum. factoids. http://www.smartcrd.com/info/more/Factoids.htm.

[5] Scott B. Guthery and Timothy M. Jurgensen. *Smart Card Developer's Kit.* MacMillan Technical Publishing, Indianapolis, Indiana, December 1997.

[6] Peter Honeyman. Ubiquitous smartcards at the university of michigan. http://www.citi.umich.edu/projects/sinciti/ smartcard/smartcard-vision.html, 1997.

[7] Peter Honeyman, William A. Adamson, and Jim Rees. Joining security realms: A single login for netware and kerberos. In *Proceedings of Fifth USENIX UNIX Security Symposium*. USENIX, June 1995. Salt Lake City.

[8] The International Organization for Standardization and The International Electrotechnical Commission. *ISO/IEC 7816-4 : Information technology - Identification cards - Integrated circuit(s) cards with contacts*, 9 1995.

[9] Naomaru Itoi and Peter Honeyman. Pluggable authentication module for windows nt. In *Proceedings of 2nd USENIX Windows NT Symposium*, Seattle, August 1998. USENIX.

[10] Rob Jerdonek, Peter Honeyman, Kevin Cofman, and Jim Reesand Kip Wheeler. Implementation of a provably secure, smartcard-based key distribution protocol. In *CARDIS'98*, Louvain-la-Neuve, Belgium, Sept. 1998. Third Smart Card Research and Advanced Application Conference.

[11] VISA Ken Ayer. Standardization in chip card security evaluations. Presentation in SCIA workshop, November 1998.

[12] John T. Kohl and B. Clifford Neuman. The kerberos network authentication service (v5), September 1993. Request For Comments 1510.

[13] John T. Kohl, B. Clifford Neuman, and Theodore Y. T'so. The evolution of the kerberos authentication system. *Distributed Open Systems*, pages 78–94, 1994. IEEE Computer Society Press.

[14] Butler Lampson, Martin Abadi, Machael Burrows, and Edward Wobber. Authentication in distributed systems: Theory and practice. In *Operating Systems Review*, volume 27-5, pages 165–182. ACM, December 1993.

[15] John Leach. Dynamic authentication for smartcards. *Computers & Security*, 14(5):385–389, 1995.

[16] Mark Looi, Paul Ashley, Loo Tang Seet, Richard Au, Gary Gaskell, and Mark Vandenwauver. Enhancing sesame v4 with smart cards. In *CARDIS'98*, Louvain-la-Neuve, Belgium, Sept. 1998. Third Smart Card Research and Advanced Application Conference.

[17] Joseph N. Pato. Using pre-authentication to avoid password guessing attacks, 1993. OSF DCE Request For Comments 26.0.

[18] Aviel D. Rubin. Independent one-time passwords. *USENIX Journal of Computer Systems*, February 1996.

[19] V. Samar and R. Schemers. Unified login with pluggable authentication modules (pam), October 1995. Request For Comments 86.0.

[20] Victor Shoup and Avi Rubin. Session key distribution using smart cards,. In *Proceedings of Eurocrypt '96*, pages 321–331, Saragossa, Spain, May 1996.

[21] Jennifer G. Steiner, Clifford Neuman, and Jeffrey I. Schiller. Kerberos: An authentication service for open network systems. In *Proceedings of the Winter 1988 USENIX Conference*. USENIX, February 1988.

[22] J. D. Tygar and Bennet Yee. Dyad: A system for using physically secure coprocessors. Technical report, Carnegie Mellon University, May 1991. CMU-CS-91-140R.

[23] Shiuh-Jeng Wang and Jin-Fu Chang. Smart card based secure password authentication scheme. *Computers & Security*, 15(3):231–237, 1996.