CITI Technical Report 00-2

Secure Coprocessor Integration with Kerberos V5

Naomaru Itoi
itoi@eecs.umich.edu

Abstract

The nightmare of *Trusted Third Party (T3P)* based protocol users is compromise of the T3P. Because the compromised T3P can read and modify any user information, the entire user group becomes vulnerable to secret revelation and user impersonation. *Kerberos*, the most widely used network authentication protocol, is no exception. When the Kerberos *Key Distribution Center (KDC)* is compromised, all the user keys are exposed, thus revealing all the encrypted data and allowing an adversary to impersonate any user. If an adversary has physical access to the KDC host, or can obtain administrator rights, KDC compromise is possible, and catastrophic. To solve this problem, and to demonstrate the capabilities of secure hardware, we have integrated the IBM 4758 secure coprocessor into Kerberos V5 KDC. As a result of the integration, our implemented KDC preserves security even if the KDC host has been compromised.

March 22, 2000

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

Secure Coprocessor Integration with Kerberos V5

Naomaru Itoi [*]

Center for Information Technology Integration
University of Michigan
`itoi@eecs.umich.edu`

April 19, 2000

1 Introduction

Over the past decades, numerous security protocols have been developed and have been quite successful at improving computer system security, providing safe handling of critical information. For example, we can feel fairly comfortable storing private letters, financial records, medical information, and so on on computers today.

Although we have a fair amount of confidence in the security of current computer systems, there still remains one large security dread; **you really do not know what your computer is doing.** Indeed, with current commodity computer technology, it is quite difficult to have confidence in system integrity[1] because (1) physical security tends to be overlooked in commodity hardware, (2) software bugs inevitably introduce security threats, and (3) new systems introduce new problems. Most of the security software available today ignores these difficulties, and simply asserts system integrity. The reasoning behind this assertion is that physical attacks are more difficult to execute than software attacks.

However, this assumption can no longer be considered reasonable as the value of information stored in computers increases. For example, the CIC Security Working Group reported the theft of a medical server that contained highly private information, such as social security numbers and medical histories of many donors at a university hospital. Considering the private nature of the stolen information, the damage the incident caused was extremely serious [19].

The time is right to begin addressing the flawed assumption that physical attack is unlikely and that system integrity is intact [30]. One approach is to employ secure hardware with the following considerations in mind. First, such hardware should be physically tamper resistant. Second, to minimize software flaws, it should be simple and throughly tested. Secure hardware is now being mass produced and is becoming more widely available (e.g., [21, 9, 40]), so it can now be more readily integrated with existing computer infrastructures. In this effort, we secure the most critical component in current computer systems: the trusted third party in the most widely used network authentication protocol, namely, the *Kerberos Key Distribution Center (KDC).*

We integrated the IBM 4758 secure coprocessor into the Kerberos KDC, which has resulted in the implemented KDC preserving critical secrets even when compromised. This paper presents the motivation, design, implementation, and performance evaluation of the project.

1.1 Fundamental Security Problems

Physical Security

Many researchers have identified the problem of physical security of distributed computer systems [20, 47, 34]. Unlike mainframe computers of the past,

*This project has been carried out in the IBM T. J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598, in the summer of 1999.

[1] System integrity is intact when "no unauthorized modification has been made." [11]

in isolated computer centers, today's computer environment consists of physically distributed personal computers and workstations, connected by networks. Such an environment is difficult to protect because computers are geographically distributed, making it more difficult to control physical access. Further, PCs and workstations have weaker physical protection than mainframes in that physical access to computational and storage devices is typically possible by simply opening the cover of the computer. For example, a hard disk drive can be easily removed from a personal computer. Once it is removed, an adversary can mount it on his own computer to access it, or can make a copy and analyze it off-line. Some PCs and workstations have locks, but these tend to be of low quality and easily defeated [14].

Bootstrap Process

Arbaugh et al. have pointed out that without a secure bootstrap process, the integrity of operating system kernels cannot be trusted because malicious code (e.g., Trojan horse) can be injected in the bootstrap process [1]. For example, typical PCs can be booted from floppy disks, thus allowing arbitrary operating system kernels to run, even malicious ones or ones with Trojan horses. Some of them allow the administrator to set a BIOS password, preventing booting unless the password is entered. However, an adversary can reset the password by resetting the BIOS [14].

Software Flaws

Bugs and design flaws in software, which are unavoidable, can be exploited. For example, buffer overflow in an administrator privileged (root) process can allow an adversary to run arbitrary code with administrator privileges. Vulnerable software ranges from operating systems to applications. Some examples are as follows:

- operating systems (e.g., erroneous permission of DLL cache on Windows NT 4.0 [12])
- basic system programs (e.g., buffer overflow in `df`, `eject`, `login`, etc., in IRIX [5])
- daemons (e.g., buffer overflow in `wu-ftpd` [8], buffer overflow in IIS web server [7, 35] and bugs in sample files in IIS [45])
- applications (e.g., buffer overflow in `sendmail` [6])
- network protocols (e.g., flaws in ICMP Router Discovery Protocol allowing man-in-the-middle attack [36])
- security software (e.g., poor encryption of shelllock [32] and Password Appraiser sending Windows passwords in the clear on the Internet [31])

Such vulnerabilities can be quite serious. For instance, they may yield administrative rights to an adversary, crash the computer system, or leak critical information.

The computer security community deals with such flaws by publishing countermeasures as soon as vulnerabilities are found. However, searching for vulnerabilities is an endless chore, as it is impossible to be confident that the software is bug-free. In addition, computer systems are developing quite rapidly, and new systems tend to bring new problems.

For example, the new functionality of Java [18] enabled client side programming on the Internet. At the same time however, a design flaw in Java caused a mismatch between the language and the bytecode, leaving the Java Virtual Machine open to attacks [28], and implementation bugs made Internet browsers vulnerable [10]. In many ways, the new technology itself enabled new kind of attacks [43, 27].

It is dangerous to assume the integrity of an operating system's kernel and software, as most software does. This is problematic especially for security critical software, such as trusted third parties.

1.2 Trusted Third Party

Several trusted third party (T3P) based security protocols are in use today. T3P is a central authority in a network that defines and enforces security policies for the other members of the network. A certificate authority (CA) is a T3P in a public key based protocol that uses certificates for authorization. A key distribution center (KDC) is a T3P in a secret key based protocol that stores secret keys of the members. The natural match between a T3P based model and real-world security management lends T3P based protocols configurability and scalability, making them widely accepted.

The T3P is a critical point of attack on a network. The damage caused by a T3P compromise is extremely serious. In particular, it is catastrophic to have the KDC compromised, as the keys of all the members can be obtained by an adversary. With all the member keys in hand, the adversary can decrypt all the secrets encrypted with the keys, and can impersonate any member. To recover from KDC compromise, all keys must be revoked and regenerated, affecting every member. Compared with KDC, CA has better characteristics when compromised because CA stores public keys, but not private keys. However, CA compromise is still quite damaging, as an adversary can impersonate anyone by crafting bogus certificates.

Therefore, to keep systems secure, T3Ps must have the highest security. However, as described in the previous section, fundamental security problems pose a significant challenge to obtaining high levels of security with current computer systems.

1.3 IBM 4758 Secure Coprocessor

We address the problem stated above by bringing a secure coprocessor into the mix. A secure coprocessor is a "computational device that can be trusted to execute its software correctly, despite physical attack" [38].

We employ the IBM 4758 secure coprocessor because of its superior security and programmability. The 4758 is a PCI card with a tamper-resistant and tamper-responding secure coprocessor.

IBM 4758 Security

The 4758 is physically protected with layers of epoxy and metal so that it does not leak information out of the barrier, has electromagnetic shielding, and cannot be accessed without the card detecting it. The card detects opening attempts, penetration attempts, temperature attacks, and radiation attacks.

This has three types of storage: RAM, battery-backed up RAM (BBRAM), and flash memory. On detecting an attack, the card responds by resetting all the data in RAM and BBRAM, thus preventing an adversary from obtaining any information. RAM is 4 MB of volatile memory. BBRAM is 8.5 kilobytes of non-volatile secure memory. Flash is 1 MB of non-volatile memory.[2]

Validated with the FIPS 140-1 Level 4 standards, this coprocessor is one of the most trustworthy and secure coprocessors [39].

IBM 4758 Programmability

In addition to its security, the 4758 has very good programmability. Applications that run in the card are written in C, and can be debugged with a runtime debugger [13].

It has a very fast cryptographic accelerator (20 MB/s bulk DES and 20 signatures/s RSA[3]), which allows for efficient implementation of security protocols. [38, 21].

It is natural to use the most secure hardware for the most critical component. To demonstrate the potential of secure hardware integration in T3P protocols, and to counter one of the fundamental security limitations of Kerberos, we integrated the 4758 into Kerberos V5 KDC.

1.4 Paper Composition

This paper presents the secure coprocessor integration with Kerberos KDC project. The next section provides the motivation behind the project by referring to related work. Section 3 describes the design of

[2]Parameters such as storage size and cryptographic performance are reported for the 4758 Model 1. The 4758 Model 2 has improved size and performance [21].

[3]50 MB/sec DES and 200 signatures/s RSA on 4758 Model 2

the integrated protocols. Section 4 presents the prototype implementation. Performance evaluation of the prototype is presented in Section 5. Discussion and future work are in Section 6.

In this document, it is assumed that readers have some knowledge of the mechanisms of Kerberos. Readers who are not familiar with them are advised to consult available literature [4, 41, 26, 24, 25].

2 Related Work

This section reviews the work most closely related to our research. Section 2.1 introduces Kerberos. Sections 2.2 - 2.5 describe approaches taken by researchers to address goals similar to ours, and the relationship between their approaches and ours. Section 2.6 discusses secure hardware integration with the Kerberos client.

2.1 Kerberos

Kerberos [41, 26, 25] is a very widely used authentication protocol. It is a secret key, T3P protocol based on the Needham-Schroeder protocol [33]. Kerberos KDC offers two services, *Authentication Service (AS)*, and *Ticket Granting Service (TGS)*. AS authenticates members (*principals*), while TGS establishes a session key between two principals. For example, Alice needs to run AS with the Kerberos KDC to prove she is Alice, and needs to run TGS with the Kerberos KDC to obtain a ticket, which is later sent to Bob to establish a session key between them. Every principal in the protocol, i.e., users, services [4], and computers, is assigned a secret key, which is shared between the principal and the KDC.

Kerberos is used in universities to protect their computer network environments. CMU, Cornell, MIT, Stanford, the University of Michigan, and many more embrace it. It is also part of the products of many corporations such as Transarc, Cisco, Qualcomm, IBM, and Microsoft, whose Windows 2000 employs it as a fundamental network authentication method. Many network applications are modified to work with Kerberos, including login, ftp, telnet, PAM, ssh, AFS, and DFS. Its security has been thoroughly analyzed [2, 26], and it scales quite well. For example, three replicated Kerberos servers at the University of Michigan serve 180,000 users. It is also quite portable. Both the clients and the servers run on almost any UNIX or Windows systems. We believe that Kerberos will continue to be an important security system.

A huge security issue for Kerberos is the common problem of KDCs, that is, it yields all the keys when compromised [2, 30]. Kerberos stores a master key, which encrypts the other keys, in cleartext in a file (/usr/local/var/krb5kdc/.k5.DOMAINNAME). Keys of the principals, i.e., the user keys and the service keys, are encrypted with the master key and are stored on hard disks. Therefore, if an adversary has administrative rights for the KDC's computer or physical access to its disks, all the keys can be stolen.

[4]Examples of services are login service and ftp service.

2.2 Public Key Based Authentication Systems

Several public key authentication systems have been designed and implemented [44, 37, 42] that are compatible with or related to Kerberos. Many of these systems are similar to ours in the sense that they try to protect the trusted third party. The logic to support them is that public key based authentication systems fail more gracefully than secret key based systems when T3P is compromised. Indeed, CA does not store private keys, thus maintaining forward secrecy and preventing an adversary from getting immediate impersonation ability. However, these systems amplify the value of our work because of the following:

- Even in public key systems, the trusted third party (CA) is the most critical point of attack. By obtaining the CA's private key, an adversary can modify certificates, issue bogus certificates, and modify certificate revocation lists, to impersonate members. Therefore, it is vital to protect the CA with secure hardware.
- Because of both the computational overhead of public key cryptography and the necessity for key revocation, we have concerns over how public

key based authentication systems scale. In contrast, we know the secret key based system scales quite well. We believe that secret key based Kerberos will be in service for a long time.

Therefore, we believe that public key augmentation to Kerberos complements our work.

2.3 Sandboxing

The sandbox approach limits the access rights of untrusted software. For example, the Java sandbox system prohibits Java applets from accessing filesystems, and from connecting to arbitrary computers [17]. Goldberg et al. developed a system to restrict the use of system calls by helper applications in a browser [16]. Lepreau et al. developed a recursive virtual machine (RVM) model, which consists of nested virtual machines. Operating system resources are "donated" by the parent VM to child VMs; the children cannot access the parent's resources unless they are donated [29].

The sandbox approach is similar to ours in the sense that both limit the power of untrusted software and minimize the damage caused by it. However, our approach is more thorough because we do not trust even the operating system kernels. In addition to malicious software, we can counter physical attacks. The main drawbacks of our system are that it requires additional hardware, and needs major modification in application software.

2.4 Type-Safe Languages

Some languages, such as Modula-3, Standard ML, and Java, are type-safe and minimize pointer related bugs, such as the buffer overflow. Operating systems have been built over such languages (e.g., SPIN is written with Modula-3 [3]). If all software is written in type-safe languages, the computer is more secure because it does not have pointer related problems. Yet, even though it has been proven possible, it is not practical to rebuild an entire computer environment from scratch with type-safe languages because the cost is too great.

2.5 Secret Sharing

Secret sharing mechanisms have been developed by researchers who have identified that computer systems can be compromised or intruded [20, 46]. They split a secret (e.g., key) into n pieces, and distribute them to n computers. To recover the secret, at least k ($k < n$) computers must reveal their pieces. Therefore, an adversary compromising fewer than k computers cannot obtain the secret.

Using a secret sharing approach raises the bar of the system's security by forcing intruders to have to break into k computers to compromise the entire system, instead of one. In contrast, the secure hardware approach makes the compromise of even a single computer much more difficult.

2.6 Smartcard Integration with Kerberos Client

Smartcard integration with the Kerberos client enhances security of Kerberos by taking advantage of secure hardware in the form of smartcard [22]. This work (smartcard/Kerberos client) and our work (secure coprocessor/Kerberos server) complement each other.

3 Design

As described in Section 1.1, we prefer not to trust the host computer on which the Kerberos KDC runs. Thus, we designed our protocol to survive a host "hijack". Here are the design policies:

- keys never leave the 4758 in clear
- all cryptographic operations are executed in the 4758

More concretely, the master key is stored in the battery-backed up RAM in the 4758 and never leaves. Because of storage limitations, user keys are stored in the host and encrypted with the master key. The 4758 has BBRAM of 8.5 kilobytes and 1 megabytes of flash memory, allowing it to securely store many DES keys. However, storage in the host is more abundant than storage in the card, and a Kerberos realm, for example, at a university, may require a huge number of keys, so we decided to store them in the host.

When user keys are used, for example, to encrypt a ticket, they are downloaded from the host to the 4758, decrypted there with the master key, used, and then deleted from its memory. Session keys are also generated in the 4758, augmented into tickets, and encrypted in the 4758 before being shipped to clients.

process_as_req

Figure 1 shows how the authentication request (AS_REQ) is handled in Kerberos V5. The keys (the user keys of Alice and Bob, the `krbtgt` key, and the master key) are used in the host. The master key is not shown in the figure, but is used to decrypt the user keys. If the host is compromised, all the keys are revealed.

To solve this problem, we designed the protocol with the 4758 in Figure 2. Note that all the encryption and decryption is done in the 4758, and no key is in the host in the clear. Even if the host is compromised, the keys are still safe and an adversary can do no harm, (although a denial of service attack is possible.)

process_tgs_req

Figures 3 and 4 show ticket granting service request (TGS_REQ) handling both with and without the 4758.

In the protocol using the 4758, all the encryption is done in the 4758, and no key is in the host in the clear.

4 Implementation

We implemented the AS and TGS protocols described in Section 3 by modifying Kerberos V5-1.0.6 distributed by MIT. The host platform is Linux 2.0.36 (RedHat 5.2) on an IBM PC. The card platform is the IBM 4758 Model 1 secure coprocessor, with the Secure Cryptographic Coprocessor toolkit version 1.33.

4.1 Outline

The implementation was carried out in the following three steps.

- analysis of `process_as_req()` and `process_tgs_req()`, which implement AS and TGS to identify which portions of the functions should be moved to the 4758
- implementation of the card side functions that have functionality equivalent to the portions identified in the first step
- modification of the host side program to make calls to the implemented functions in the card

4.2 Step1: Functionality Analysis

There are six parts to be moved in AS: three calls to key decryption and one each to preauthentication, ticket encryption, and reply encryption. Likewise, there are six parts in TGS: two calls to key decryption and one each to ticket decryption, authenticator decryption, ticket encryption, and reply encryption.

As the performance evaluation in Section 5 shows, the overhead of calling a function in the 4758 is high. Therefore, to obtain high performance, the six calls are combined into one call. However, as cryptographic code and non-cryptographic code are tightly coupled together in Kerberos, doing so changes the order of execution and breaks modularity, thus significantly complicating the host side code. For this prototype, we decided to make six calls in each AS and TGS, valuing simplicity and manageability over performance. A detailed look at the overhead in Section 5.2 explains our decision.

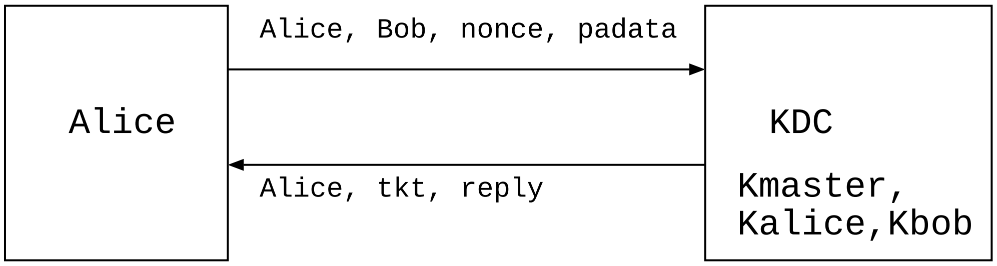

Figure 1: original AS_REQ processing in Kerberos V5

Alice	The principal who wants to be authenticated.
Bob	The principal with whom Alice wants to communicate. (Bob is the "krbtgt" when AS_REQ is used to obtain TGT.)
PAdata	{`current time`}`Kalice` : Preauthentication data to prove that Alice knows the right Kalice.
Kses	Session key
Tkt	{`Alice, Bob, Kses`}`Kbob` : Ticket forwarded by Alice to Bob to prove that Alice carried out authentication with the KDC. If Bob is "krbtgt", the tkt is the TGT ({`Alice, krbtgt, Kses`}`Kkrbtgt`)
Reply	{`Bob, nonce, Kses`}`Kalice` : Alice decrypts it to get the session key.

The parties send additional information, such as message types, protocol version number, flags, and start/expiration/renew-until time. We omit them in this figure because they are treated in the same manner with or without the 4758.

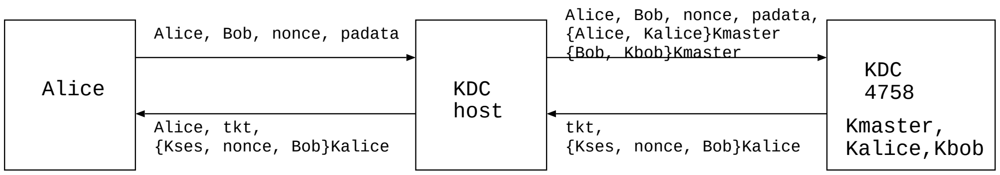

Figure 2: AS_REQ processing in Kerberos V5 with the 4758

Security critical tasks, e.g., en(de)cryption and random key generation, are moved from the host to the 4758. The host sends the 4758 the information needed for such tasks, e.g., the nonce sent by Alice, and the encrypted keys of Alice and Bob. The 4758 generates a reply to Alice, and sends it back to the host.

The entries encrypted with the master key e.g., {`Alice, Kalice`}`Kmaster`, are generated and decrypted in the 4758. The KDC host stores them (encrypted) in the Kerberos database and sends them to the 4758 when needed. The KDC host does not know the master key.

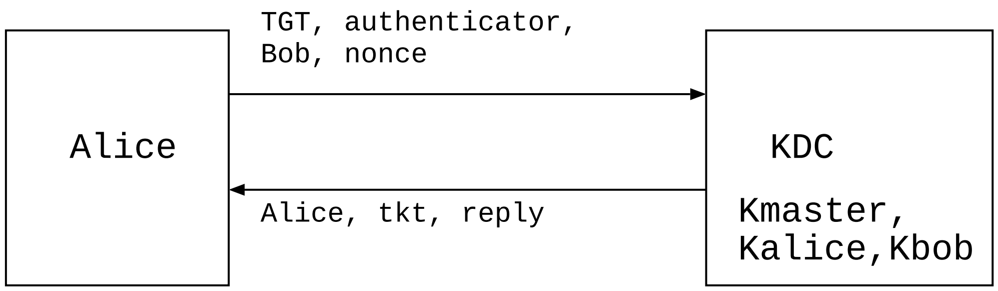

Figure 3: original TGS_REQ processing in Kerberos V5

Alice	The principal who wants to be authenticated.
Bob	The principal with whom Alice wants to communicate. Bob is "krbtgt" when TGS_REQ is used to obtain TGT.
PAdata	Preauthentication data (TGT and Authenticator). and knows the right Kses.
TGT	{Alice, krbtgt, Kses}Kkrbtgt : Ticket Granting Ticket, which proves that Alice carried out authentication with KDC.
Authenticator	{Alice, time, (subkey)}Kses
K	Key in TGT or subkey in authenticator
Kses	Session key
Tkt	`{Alice, Bob, Kses'}K` : New ticket for Alice and Bob.
Reply	`{Bob, nonce, Kses'}K` : Alice decrypts it to get the session key.

The parties send additional information, such as message types, protocol version number, flags, and start/expiration/renew-until time. We omit them in this figure because they are treated in the same manner with or without the 4758.

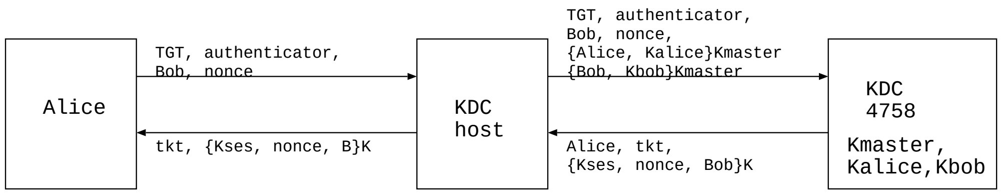

Figure 4: TGS_REQ processing in Kerberos V5 with the 4758

Security critical tasks, e.g., en(de)cryption and random key generation, are moved from the host to the 4758. The host sends the 4758 the information needed for such tasks, e.g., the nonce sent by Alice, the encrypted keys of Alice and Bob. The 4758 generates a reply to Alice, and sends it back to the host.

The entries encrypted with the master key, e.g., `{Alice, Kalice}Kmaster`, are generated and decrypted in the 4758. KDC host stores them (encrypted) in the Kerberos database and sends them to the 4758 when needed. The KDC host does not know the master key.

4.3 Step2: Card Side Functions

4.3.1 Authentication Service

Key Decryption

User keys are stored in the host and encrypted with the master key. The card decrypts the keys before using them. The host-side `decrypt_key()` function sends keys to the card, decrypts them and then stores them in RAM for future use. The function is called three times in AS: first for Alice's key for preauthentication, second for Bob's key for ticket encryption, and third for Alice's key for reply encryption. [5]

Preauthentication

Preauthentication is the step in which Alice proves her identity to the KDC by proving knowledge of her key. By default, preauthentication takes place in the following three steps:

- Alice sends to the KDC a timestamp encrypted with her key : {time}Kalice.
- The KDC decrypts {time}Kalice.
- The KDC verifies that the request is really generated by Alice by determining whether the time value falls within clock skew allowed. KDC goes on to the next step in AS if the answer is yes. Otherwise, KDC rejects Alice.

Because this step requires the use of Alice's user key, this function is moved to the card.

Ticket Encryption

A ticket is a data structure sent from the KDC to Alice to establish a session key. Roughly speaking, it is {`Alice, Bob, Kses`}`Kbob`. Part of the ticket is not security critical (e.g., Alice's and Bob's names), and is generated in the host. Afterward, the ticket is sent to the card, filled with the session key generated in the card, encrypted with Kbob, and sent to Alice. The card stores the session key for future use because the reply will include it as described in the next paragraph.

[5] We can save one call by caching the key in preauthentication and using it in reply encryption. We did not try this optimization for the prototype; performance is not yet our goal.

Reply Encryption

Similar to the ticket, the reply {`Bob, nonce, Kses`}`Kalice` includes a public part, which is encoded in the host and is sent to the card. The session key, generated in the ticket encryption function, is inserted into the reply. The card then encrypts the reply with Alice's key.

4.3.2 Ticket Granting Service

As in AS, six calls are made to the card in TGS: two calls to key decryption, and one each to ticket decryption, authenticator decryption, ticket encryption, and reply encryption. Some of the functions are common in AS; we explain only the functions that do not appear in AS.

Ticket Decryption

TGS decrypts the ticket granting ticket, or TGT ({Alice, krbtgt, Kses}Kkrbtgt), to obtain Alice's name and the session key. Because it involves the TGS key (Kkrbtgt), and the session key is in the TGT, this step must be carried out in the card. The card decrypts the TGT and returns it in the clear to the host excluding the session key. The session key must not leave the card, so it is stored in RAM in the card. Later it is used in authenticator decryption and reply encryption.

Authenticator Decryption

The authenticator {Alice, time, (subkey)}Kses is decrypted in the card and is returned to the host in the clear.

4.4 Step 3: Host Side Modification with Secure Hardware RPC

As with other secure hardware [23], the communication methods between the host and the 4758 are primitive. For example, the only type provided is an array of bytes. It is the developers' responsibility to convert types such as `int, short, char,` and more complicated data structures, into and out of the string of bytes. It is a burden for developers to deal with low-level programming, e.g., marshaling and unmarshaling data structures, dealing with endian problems, message buffer handling, and error handling.

To provide a better abstraction, we developed the *Secure Hardware Remote Procedure Call (SHRPC)*, which parses the interface definition file and generates C programs to handle the low-level communication details. With SHRPC, procedure call abstraction is provided to the host. As a consequence, modification in the host side is merely to call SHRPC stub functions, e.g.., `decrypt_key()`, instead of equivalent but more elaborate functions in the host.

Although *interface definition language (IDL)* should follow some standards, such as `rpcgen`, we picked our own simple IDL for rapid implementation. The interface definition file for the Kerberos / 4758 integration looks like this:

```
# Interface Declaration File
#  for the Kerberos V5 / 4758 Project
# 8/6/1999, Naomaru Itoi
PROG:   krb5_4758
FUNC:   decrypt_key
IN:
int type
# type :
#   0: client key
#   1: server key
string enc_key
OUT:
int tick
...
```

The interface definition above defines a function `decrypt_key()` in a program `krb5_4758`, an input data structure with an integer and a string of bytes (encrypted key), and an output data structure with an integer (for performance evaluation).

SHRPC generates the following:

- a header file to define the `string` type, the input data structure, and the output data structure
- utility functions, such as marshaling and unmarshaling
- host side stub functions, which are called from the application program in the host
- a card side template program, which receives the call from the host, and dispatches it to an appropriate service function in the card

SHRPC reduces the developers' work to having only to write (1) the application that calls the host side stub, and (2) the service function in the card.

-

5 Performance Evaluation

We evaluated the performance of the prototype in the following environment: IBM Netfinity PC with Intel 30 MHz Pentium; the IBM 4758 secure coprocessor model 1; the KDC and the Kerberos clients running on the same computer to avoid network delay.

Each measurement was carried out ten times and an average is presented in tables. Variance was very small.

5.1 Overall Result

This section describes the performance of AS. `kinit` is the client program used to initiate the AS request. The total time kinit spends with or without 4758 is shown. To exclude the time spent for password typing, the password is hard coded in the `kinit` program. `kinit` with the 4758 takes 34% more time than `kinit` without it.

	time (sec)
`kinit` without 4758	0.0611
`kinit` with 4758	0.0820

`sclient` is the client we used to initiate the TGS request. `sclient` with the 4758 takes 33% more time than `sclient` without it.

	time (sec)
`sclient` without 4758	0.0719
`sclient` with 4758	0.0953

4758 integration introduces approximately 33% of overhead in both cases. We look into the details in the following sections.

5.2 Communication Overhead

In this section, we examine communication overhead. We measure the total time spent for the six cryptographic operations described in Section 4.2, the time spent in the card, and derive the communication overhead. As shown in Figure 5, the total time is the sum of the card time and the overhead.

	Total	Card Time	Overhead
AS w/o 4758	0.00054	-	-
AS w/ 4758	0.02535	0.00866	0.01669
TGS w/o 4758	0.00032	-	-
TGS w/ 4758	0.02748	0.00866	0.01882

Communication overhead is approximately twice as much as the card time in both AS and TGS. This is an obvious bottleneck and there is an obvious optimization. Theoretically, the number of calls can be reduced from six in each AS and TGS to one in AS and two in TGS. All operations can be done at once in AS. In TGS, the TGT (ticket granting ticket) must be decrypted to obtain the name of the client before the KDC looks up its database. In contrast, ticket encryption and reply encryption must happen after the database lookup. Therefore, TGS requires two calls. This optimization would reduce the card time to 0.00278 seconds in AS and 0.00314 seconds in TGS, thus reducing the overhead of using 4758 to 11% in AS and 15% in TGS.

5.3 Card Time Details

Although communication overhead was the bottleneck, it is also useful to study the details of the time spent in the card. Breakdown of AS and TGS is shown in the following table. For each function, total time and time spent in main components are presented.

AS

function	contents	time (sec)
decrypt_key	24B DES decryption	0.000957
	TOTAL	0.001109
kdc_preauth	40B DES decryption	0.000997
	CPGetTime	0.000023
	TOTAL	0.001445
encrypt_tk	168B DES encryption	0.001191
	random number gen	0.000352
	random number gen	0.000352
	168B CRC	0.000041
	TOTAL	0.002078
encode_kdc	216B DES encryption	0.001270
	random number gen	0.000352
	216B CRC	0.000053
	TOTAL	0.001809

TGS

function	contents	time (sec)
decrypt_key	24B DES decryption	0.000957
	TOTAL	0.001115
decrypt_tk	168B DES decryption	0.001172
	168B CRC	0.000041
	TOTAL	0.001324
rd_rec_dec	120B DES decryption	0.001113
	120B CRC	0.000031
	TOTAL	0.001230
encrypt_tk	168B DES encryption	0.001191
	random number gen	0.000352
	random number gen	0.000352
	168B CRC	0.000041
	TOTAL	0.002105
encode_kdc	184B DES encryption	0.001211
	random number gen	0.000352
	184B CRC	0.000045
	TOTAL	0.001773

DES operation takes the longest time. Considering that the hardware implemented DES takes much longer time than the software implemented CRC even

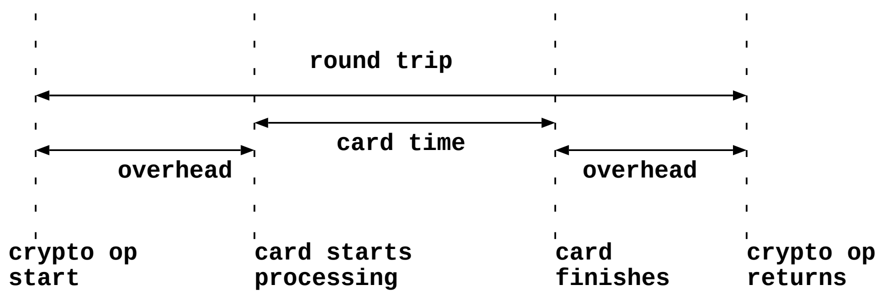

Figure 5: total, card time, and overhead

though the hardware itself is quite fast (20 MB/s [6]), we believe the most of the DES operation time is spent in making a system call to DES hardware and setting up a key schedule. For an application that operates on such small data (100 - 200 bytes), which we believe many authentication and authorization systems do, it is good to provide (1) software implementation of crypto operations to save system call overhead and (2) a decoupled API for DES key scheduling separate from DES operation. (2) is helpful because some keys are used repeatedly, e.g., a master key.

6 Discussion

6.1 On This Project

We showed the capability of secure hardware to enhance the security of a trusted third party protocol. The Kerberos KDC with the 4758 can preserve security even if the host has been compromised. Because all secrets are stored and used in the 4758, to compromise the implemented KDC, an adversary must compromise the 4758, which is much more difficult to compromise than the host because of the 4758's protection. The 4758 does not leak information out of the card, thus preventing an adversary from eavesdropping secrets. It responds to a tamper attempt by deleting its state, thus protecting secrets from physical attacks. The integration of the 4758 into the Kerberos KDC results in a KDC as difficult to compromise as possible with current technology.

[6] 50 MB/s on Model 2.

6.2 Lessons Learned

Integration of secure hardware into a security protocol can be significantly simplified if the original implementers of the protocol anticipate the use of secure hardware. Complication of our work comes from cryptographic operations and non-cryptographic operations being tightly coupled in a program, e.g., they coexist in one function. If they are decoupled cleanly in an initial implementation, the work of integration is merely to move the crypto code to the secure hardware. Moreover, we believe the separation is good for portability of the protocol, e.g., to switch from one encryption system to another.

6.3 Future Work

Several steps must be taken before this project is deployed.

Administration

We have not addressed problems associated with administration: changing passwords, adding / removing principals, changing the the KDC's policy, etc.

The Kerberos Database (`KDB`) is the database in which Kerberos stores its critical information, e.g., the keys and the principal attributes; it is accessed by administrators through `kadmind`. Because the data in the `KDB` are sensitive, the entries are encrypted with the master key. As a consequence, in the 4758 integrated KDC, administration requests must go through the 4758.

An adversary can attack a Kerberos/4758 system by attacking the channel between the administrator and the 4758. For example, one possible attack, which could reduce the advantage of integrating the 4758 into Kerberos, is a Trojan horse in the administrator's terminal. If it can interrupt the operations by the administrator, it can steal sensitive information, e.g., user passwords. In fact, this secure I/O problem is a general concern for any security system, which requires the administrator be trustworthy, and the administrator's terminal be secure.

We plan to address these concerns by establishing an encrypted connection between a system administrator and the 4758 with Kerberos authentication, and using the connection to securely transfer requests by the administrator to the 4758.

This will partially achieve our goals because the administrator is authenticated via Kerberos, and communication is encrypted. However, it is not possible to provide a completely trusted terminal with current commercial hardware, even with secure hardware such as the 4758, because even if the processors and storage are trusted, the I/O devices may not be. For example, a keyboard or a display instrumented with a hardware eavesdropper can steal administrators' keystrokes. However, it is much easier to keep a terminal secure during administration than to keep a Kerberos server secure in 24 hours a day, seven days a week fashion. Therefore, we defer solving this problem of secure I/O.

Performance Optimization

As described in Section 5.2, the six calls to the card in AS and TGS should be combined into one and two calls respectively to optimize the performance. The drawback of this optimization is that it changes the Kerberos code significantly. In the Kerberos/Cartel meeting in July of 1999, we sensed that such a radical change would pose a major challenge to Kerberos developers with regard to maintaining the source code. Therefore, we decided to first implement a prototype to determine what the computer systems community thinks about it before proceeding to the deployment step.

Brute Force Attack to Master Key

If an adversary has access to messages passed between the host and the 4758, he or she can obtain a plaintext-ciphertext pair. Some messages are encrypted with single DES and the master key. This is problematic because given a plaintext-ciphertext pair, DES key can be cracked by a brute force attack in a week [15]. Kerberos distribution from MIT supports triple DES, eliminating this threat.

Replay Attack

An adversary can use a replay attack to impersonate Alice if he or she hijacks the host and has Alice's obsolete password. Here we describe a possible attack and the countermeasure.

Our Kerberos/4758 protocol stores the master key inside the 4758, which encrypts the other keys with this master key and stores the ciphertext on the host. An adversary (Mallory) cannot access these plaintext keys even if she compromises the host because she does not know the master key, which never leaves the 4758.

However, without additional measures, such a protocol suffers from replay attacks if Mallory can learn one of Alice's old passwords. The replay attack is carried out as follows:

- Mallory obtains an old password of Alice, Pa.
- Because Mallory has complete access to the host, she can obtain {Alice, Kalice}Kmaster.
- Alice, knowing that her password is stolen, changes her password to Pa'. At this point, the old password Pa and the corresponding key Kalice are obsolete.
- Mallory types the obsolete password, Pa. Pa is hashed to the key Kalice. The KDC hijacked by Mallory sends {Alice, Kalice}Kmaster to the 4758. If the 4758 does not know Kalice is obsolete, it thinks Kalice is fresh, and sends a reply encrypted by Kalice to the KDC/Mallory. Mallory successfully decrypts the reply, thus impersonating Alice.

To avoid this attack, we use key version numbering and obsolete key caching. First, all the keys in the Kerberos database have a key version number, N. This key version number is different from the key version number used in the original Kerberos V5 protocol. An encrypted key entry contains this version number, i.e., {Alice, Kalice, N}. When Alice changes her password, Alice's current key version number is updated to N+1. The 4758 generates a new key entry {Alice, Kalice', N+1}, sends the entry back to the host, and caches a pair {Alice, N+1} in its internal memory.

The 4758 checks the cache whenever it receives a key from the host. If the version numbers do not match, then the key received is obsolete. To avoid cache overflow, once in a while (e.g., daily) the 4758 regenerates the new N and computes the new entries for all the keys, and sends them back to the host.

The cache should not overflow too quickly. If the cache size is 1MB and each entry is 32 bytes, then the maximum number of entries in the cache is 32K entries — which we imagine exceeds the maximum number of password changes in a day. (Furthermore, some preliminary tests indicate that the time needed for cryptographic operations to regenerate the cache is not excessive.)

7 Conclusion

This paper demonstrates the ability of secure hardware to improve the security of current computer environments. We can shrink the security boundary of the trusted third party from a workstation to a secure coprocessor, which is a smaller and more secure component. The implemented Kerberos KDC survives host compromise.

8 Acknowledgement

I thank Sean Smith, Ronald Perez, and Dawn Song at IBM research for their advice and discussion, especially for bringing up the replay attack described in Section 6.3, and suggesting the countermeasures. I thank Peter Honeyman at the University of Michigan, Mark Lindemann and Joan Dyer in IBM research, and the Kerberos developers for their help and discussion. I thank Chris Feak and Evan Cordes at the University of Michigan for assistance on English writing.

References

[1] William A. Arbaugh, David J. Farber, and Jonathan M. Smith. A secure and reliable bootstrap architecture. In *1997 IEEE Symposium on Security and Privacy*, Oakland, CA, May 1997.

[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] Brian N. Bershad, Stefan Savage, Przemyslaw Pardyak, Emin Gn Sirer, Marc Fiuczynski, David Becker, Susan Eggers, and Craig Chambers. Extensibility, safety and performance in the spin operating system. In *Proceedings of the Fifteenth ACM Symposium on Operating System Principles*, pages 267 – 284, December 1995.

[4] Bill Bryant. Designing an authentication system: a dialogue in four scenes, 1997. Afterword by Theodore Ts'o. http: //web.mit.edu/ kerberos/ www/ dialogue.html.

[5] CERT/CC. Cert advisory ca-97.21 (sgi buffer overflow vulnerabilities), July 1997. http:// www.cert.org/ advisories/ CA-97.21.sgi_buffer_overflow.html.

[6] CERT/CC. Cert advisory ca-97.05 (mime conversion buffer overflow in sendmail versions 8.8.3 and 8.8.4), January 1999. http:// www.cert.org/ advisories/ CA-97.05.sendmail.html.

[7] CERT/CC. Cert advisory ca-99-07 (iis buffer overflow), June 1999. http:// www.cert.org/ advisories/ CA-99-07-IIS-Buffer-Overflow.html.

[8] CERT/CC. Cert advisory ca-99-13 (multiple vulnerabilities in wu-ftpd), Oct 1999. http:// www.cert.org/ advisories/ CA-99-13-wuftpd.html.

[9] Chrysalis-its. http:// www.chrysalis-its.com/.

[10] Drew Dean, Edward W. Felten, and Dan S. Wallach. Java security: From hotjava to netscape and beyond. In *Proceedings of the IEEE Symposium on Security and Privacy*, 1996. http:// www.cs.princeton.edu/ sip/ pub/ secure96.html.

[11] Dorothy Denning. *Cryptography and Data Security.* Addison-Wesley, 1983.

[12] dildog@l0pht.com. any local user can gain administator privileges and/or take full control over the system. L0pht Security Advisory, February 1999. http:// www.l0pht.com/ advisories.html.

[13] J. Dyer, R. Perez, S.W. Smith, and M. Lindemann. Application support architecture for a high-performance, programmable secure coprocessor. In *22nd National Information Systems Security Conference*, October 1999.

[14] Kevin Fenzi and Dave Wreski. Linux security howto, April 1999.

[15] Electronic Frontier Foundation. *Cracking DES - Secrets of Encryption Research, Wiretap Politics & Chip Design.* O'Reilly & Associates, Inc., 1 edition, 1998.

[16] Ian Goldberg, David Wagner, Randi Thomas, and Eric Brewer. A secure environment for untrusted helper applications. In *Proceedings of 6th USENIX Unix Security Symposium*, July 1996.

[17] Li Gong. Java security: Present and near future. IEEE Micro, May/June 1997.

[18] James Gosling and Henry McGilton. The java language environment. White Paper, May 1996. http:// java.sun.com/ docs/ white/ langenv/.

[19] CIC Security Working Group. *Final Report: Incident Cost Analysis and Modeling Project.* Committee on Institutional Cooperation, 1997.

[20] Maurice Herlihy and Doug Tygar. How to make replicated data secure. In *Proceedings of CRYPTO-87*, pages 379 – 391, 1988.

[21] IBM. Cryptographic cards home page. http:// www.ibm.com/ security/ cryptocards.

[22] Naomaru Itoi and Peter Honeyman. Smartcard integration with Kerberos V5. In *Proceedings of USENIX Workshop on Smartcard Technology*, Chicago, May 1999.

[23] Naomaru Itoi, Peter Honeyman, and Jim Rees. Scfs: A unix filesystem for smartcards. In *Proceedings of USENIX Workshop on Smartcard Technology*, Chicago, May 1999. To appear.

[24] Charlie Kaufman, Radia Perlman, and Mike Speciner. *Network Security.* Prentice Hall, 1995.

[25] John T. Kohl and B. Clifford Neuman. The Kerberos network authentication service (V5), September 1993. Request For Comments 1510.

[26] John T. Kohl, B. Clifford Neuman, and Theodore Y. T'so. *Distributed Open Systems*, chapter The Evolution of the Kerberos Authentication System, pages 78–94. IEEE Computer Society Press, 1994.

[27] Mark D. Ladue. Hostile applets home page. http:// metro.to/ mladue/ hostile-applets/.

[28] Mark D. Ladue. When java was one: Threats from hostile byte code. In *Proceedings of the 20th National Information Systems Security Conference*, 1997.

[29] Jay Lepreau, Brian Ford, and Mike Hibler. The persistent relevance of the local operating system to global applications. In *Proceedings of the 7th ACM SIGOPS European Workshop*, September 1996.

[30] Peter A. Loscocco, Stephen D. Smalley, Patrick A. Muckelbauer, Ruth C. Taylor, S. Jeff Turner, and John F. Farrell. The inevitability of failure: The flawed assumption of security in modern computing environments. In *21st National Information Systems Security Conference*, Crystal City, Virginia, October 1998. National Security Agency, NISSC. http://www.jya.com / paperF1.htm.

[31] mudge@l0pht.com. Users of the tool password apraiser are unwittingly publishing nt user passwords to the internet. L0pht Security Advisory, January 1999. http:// www.l0pht.com/ advisories.html.

[32] mudge@l0pht.com and lumpy. Users can deobfuscate and retrieve the hidden shell code. L0pht Security Advisory, October 1999. http:// www.l0pht.com/ advisories.html.

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

[34] Innovative Security Products. Computer security. White Paper, 1998.

[35] Ryan R Permeh (rrpermeh@RCONNECT.COM). Re: Retina vs. iis4, round 2, ko, June 1999. http:// www.rootshell.com/.

[36] sili@l0pht.com. Attackers can remotely add default route entries. L0pht Security Advisory, August 1999. http:// www.l0pht.com/ advisories.html.

[37] M. Sirbu and J. Chuang. Distributed authentication in kerberos using public key cryptography. In *Internet Society 1997 Symposium on Network and Distributed System Security*, 1997.

[38] S. W. Smith and S. H. Weingart. Building a high-performance, programmable secure coprocessor. *Computer Networks, (Special Issue on Computer Network Security)*, 31:831 – 860, April 1999.

[39] S.W. Smith, R. Perez, S.H. Weingart, and V. Austel. Validating a high-performance, programmable secure coprocessor. In *22nd National Information Systems Security Conference*, October 1999.

[40] Spyrus. http:// www.spyrus.com/.

[41] 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.

[42] Brian Tung, Matthew Hur, Ari Medvinsky, Sasha Medvinsky, John Wray, and Jonathan Trostle. Public key cryptography for cross-realm authentication in kerberos. IETF Internet Draft, expires December 1, 1999. draft-ietf-cat-kerberos-pk-cross-09.txt.

[43] J. D. Tygar and Alma Whitten. Www electronic commerce and java trojan horses. In *Proceedings of the Second USENIX Workshop on Electronic Commerce*, November 1996.

[44] M. Vandenwauver, R. Govaerts, and J. Vandewalle. Overview of authentication protocols: Kerberos and sesame. In *Proceedings 31st Annual IEEE Carnahan Conference on Security Technology*, pages 108 – 113, 1997.

[45] weld@l0pht.com. Web users can view sensitive information in microsoft iis 4.0 web server. L0pht Security Advisory, May 1999. http:// www.l0pht.com/ advisories.html.

[46] Thomas Wu, Michael Malkin, and Dan Boneh. Building intrusion-tolerant applications. In *Proceedings of USENIX Security Symposium*, August 1999.

[47] Bennet Yee. *Using Secure Coprocessors.* PhD thesis, Carnegie Mellon University, May 1994.