An alleged attack on key delegation in the Trusted Platform Module

King Ables
Supervisor: Dr. Mark Ryan

School of Computer Science
University of Birmingham

MSc Advanced Computer Science
First semester mini-project

12 January 2009

Abstract

The Trusted Platform Module (TPM) is a chip included in most Intel-based computers manufactured today. Specified by the Trusted Computing Group (TCG) and manufactured by many different companies, the TPM provides hardware-based cryptographic functions in an effort to create a more trustworthy environment that is not vulnerable to malware or intentional misuse.

After an overview of some of the capabilities of the TPM and some of the tools that can be used to access its functions, this report examines a potential weakness in the specification of the TPM alleged in another paper. If valid, this weakness could allow an attacker to misuse the TPM key delegation function in such a way that access to a key might be granted even when access is not authorized.

The allegation was made by way of an automated analysis of the TCG's TPM specification. This report also describes an implementation of the proposed attack. The attack was executed to verify and assess the severity of the alleged vulnerability. Development of the attack code also required additional functionality be developed for the tool used to access the TPM.

When executed on a TPM-enabled system, the TPM withstood the attack and the illegitimate request for access to the alternate key was correctly denied. Some possible explanations for this are also examined.

Keywords

TPM; TCG; delegate; delegation; OSAP; DSAP; TPM_DSAP; TPM_Delegate_CreateKeyDelegation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The goal of this mini-project is to gain an understanding of the Trusted Platform Module (TPM) and to investigate a potential attack against it.

Specific objectives of the mini-project include:

- understand TPM use and session structure
- evaluate methods of accessing the TPM
- install and use one of these methods to perform TPM operations
- evaluate the proposed attack scenario
- attempt the attack on a TPM-enabled system

Chapter 2 of this report presents some background material on Trusted Computing and the Trusted Platform Module to familiarize the reader with the environment of the attack. While this is a very limited overview of the TPM and its functionality, it covers the areas required to understand the attack.

Chapter 3 surveys several alternative methods of accessing the TPM. The projects considered for use were:

- jTSS
- TPM/J
- TrouSerS

TPM/J, a Java-based API, is used in this mini-project.

Also in Chapter 3, the vector for the attack is examined. In his Master's thesis ([Lin05]), Amerson Lin describes a potential weakness in the specification of the TPM uncovered during his work in automated analysis of the TPM API. This vulnerability is examined as a potential attack on the TPM. Lin alleges that the flaw makes the TPM susceptible to a key handle switching attack where a user might gain illegitimate access to a key while using a different key legitimately. Because the problem was discovered during automated analysis of the API, no actual attack was attempted. So is the TPM truly vulnerable?

Chapter 4 tells how this question is answered. To execute Lin's attack, modifications to TPM/J were required. The capability to create delegated keys

and establish the special type of transport session necessary to use them is not included in the current release of TPM/J (nor in any of the other tools that were considered). Once TPM/J supported these TPM functions, source code for the attack itself was developed. When executed, the attack ultimately failed and the TPM denied access to the key as it should.

That the TPM withstood the attempted key switching attack indicates some misunderstanding in Lin's analysis of how the TPM authenticates its transport session for a delegated key. Chapter 5 suggests some possible explanations for the TPM's observed success in defending against the attack.

Chapter 2

Trusted Computing background

The notion of trust can be elusive. Trust between human beings can be sensed but not necessarily measured. Trust between electronic devices could be measured, but is difficult to define.

Many different levels of trust may also exist. A trustworthy operation may execute on untrustworthy data. Trustworthy data may be sent to an untrusted program.

2.1 Who can you trust?

It is not always possible to know what data, programs, or systems are trustworthy. The user often cannot be trusted. Even if he or she cares about maintaining a secure environment, many are fooled into giving away important data like passwords. The application may not be trustworthy, either. Malware or faulty code can cause any software operation to fail, or worse, perform intentionally incorrectly. Even the operating system, which is itself only software, can be corrupted. If malware penetrates the kernel or the boot sector, security of the operating system cannot be guaranteed.

If all software is potentially vulnerable, the hardware is left as the only place where we can be guaranteed any level of trust.

2.2 Trusted Computing Group

The Trusted Computing Group (TCG) was formed in 2003.[1]

With approximately 130 members, the "Promoter[2] members" of the TCG, as listed on their web site `http://www.trustedcomputinggroup.org`, include:

- AMD
- Fujitsu

[1]The TCG was created from what had been the Trusted Computing Platform Alliance (TCPA), which was established in 1999.

[2]Other TCG membership levels are "Contributor" and "Adopter."

- Hewlett-Packard
- IBM
- Infineon
- Intel
- Lenovo
- Microsoft
- Seagate
- Sun Microsystems
- Wave Systems

The TCG is a non-profit organization whose charter is to produce standards to promote Trusted Computing. The TCG provides architecture and functional specifications, interface definitions, and even marketing and public relations material.

Two of these standards are the specifications for the Trusted Platform Module and Trusted Computing Software Stack.

2.3 Trusted Platform Module

The Trusted Platform Module (TPM) is a chip embedded in many electronic devices today. According to [Kay05], IDC says that a TPM chip can be found on the motherboard of more than 90% of all Intel-based desktop and laptop computers. IDC goes on to say that 100 million deployed personal computers contained TPMs in 2005 and, by 2010, manufacturers are forecast to be shipping more than 250 million units with TPMs every year.

Few applications currently use the TPM. Two of note are Microsoft Bitlocker, a disk encryption application, and HP Protect Tools, a security manager application.

2.3.1 Manufacturers

TPM chips are manufactured by several companies:

- Atmel
- Broadcom
- Infineon
- Intel
- Sinosun
- STMicroelectronics
- NuvoTon (formerly Winbond)

Current TPM chips implement the latest version (1.2) of the TCG TPM specification.

2.3.2 Characteristics

The TPM is an opt-in device, that is, one must explicitly activate it before it will operate. This was done to address the fears that computer systems would automatically deny access to certain operations without the owner's consent.

TPMs currently ship deactivated. The system owner must perform a "take ownership" action to activate it. The owner can also reset or deactivate it.

TPMs in the United States and Europe implement RSA[3] encryption. TPM chips used in China implement Elliptical Curve Cryptography (ECC). The TPM used in this mini-project uses RSA key pairs and this report will always intend RSA key pairs when referring to TPM keys.

An active TPM provides key management, cryptographic, and protected storage functions to protect private key information. The cryptographic and key management functions include:

- generate 2048-bit RSA keys (public and private)
- encrypt (bind) and sign keys
- perform platform measurement, authentication, and attestation
- generate very nearly real random numbers
- create SHA-1 and HMAC hashes
- maintain monotonic counters

Most results of TPM functions are handed back to an application in an encrypted *blob*[4] rather than storing them on the chip. A key blob contains several pieces of data about a TPM key, most significant of which is its private-key and public-key values. The key blob also contains authorization data required to use the key. Referred to in the TPM specifications [Tru06a, Tru06b] as AuthData, this is the SHA-1[5] hash of a password or phrase. The key's AuthData is required to use the key.

The private part of a key is encrypted so that only the TPM can decrypt it. This way, though the data may be stored anywhere, it is protected from access or modification.

The TPM also provides platform integrity checking and reporting of the BIOS, Master Boot Record (MBR), and the boot sector. It records hash values and reports them if asked. It does not prevent the system from booting.

The TPM has sensors that can detect a physical attack to allow it to protect its on-board data. While it is possible to physically strip layers off of a TPM chip, doing so will guarantee the destruction of the data contained therein.

[3] RSA is a widely-used public-key encryption algorithm named for its inventors, Ron Rivest, Adi Shamir, and Leonard Adleman.

[4] While "blob" is an apt description of this combination of data values, the TPM specification actually makes an acronym out of it by referring to it as a "binary large object."

[5] Secure Hash Algorithm-1 is no longer considered as secure in all uses, but it is sufficiently so for the way it is used by the TPM.

2.3.3 On-board memory

The TPM contains a small amount of memory. Non-volatile memory contains platform keys and ownership authorization data which are valid for as long as the TPM remains activated and over multiple power cycles. Data that are stored in non-volatile memory include:

- the Endorsement Key (EK) pair and its certificate (burned into the chip by the manufacturer)
- the Storage Root Key (SRK) pair and owner authorization data (created when ownership is established)

The private portions of the EK and SRK never leave the TPM.

Volatile memory contains data and state information related to the currently running operating system or keys loaded into the TPM. Data stored in volatile memory (thus reinitialized at each system restart) include:

- 24 Platform Configuration Registers (PCRs)[6]
- loaded (currently in use) keys
- key handles
- transport session handles

PCR values are set each time a system boots. They can later be checked against previously stored values to verify platform integrity. This is how the TPM implements platform attestation.

2.3.4 Keys

The TPM has two special keys that always exist when the TPM is in an active state.

The Storage Root Key (SRK) is the root of the storage tree of keys. Each TPM has exactly one SRK and it is stored in the TPM's non-volatile memory. The SRK is created by the "take ownership" process that must be performed to activate the TPM.

The Endorsement Key (EK) is burned into the chip at the time of manufacture. This key guarantees the chip is an authentic TPM and includes a digital certificate (i.e., the EK is signed by the TPM's manufacturer).

All other keys are stored (permanently) outside the TPM and only loaded into the TPM when in use.

A key residing in the TPM has the following attributes:

- a key handle
- a parent key handle
- a public key
- a private key
- owner AuthData

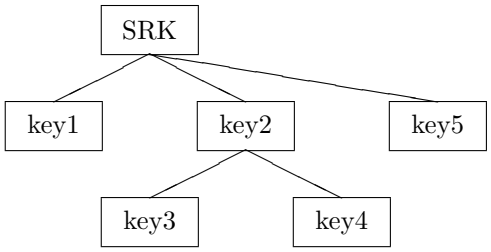

Figure 2.1: TPM key tree

Keys are logically arranged in a hierarchy, as shown in Figure 2.1, with the Storage Root Key (SRK) being the default parent key.

The SRK is called the "root of trust" for the platform since everything created or loaded under it will have had to specify the SRK's AuthData so the chain of trust extends to each key. Parent key AuthData must be specified to successfully create or load a key into the TPM. A key's AuthData must be specified to use a loaded key.

Private portions of keys that are returned to an application are encrypted with the parent's public key, so they are only retrievable by the TPM (via decryption using the parent key's private portion) and only when both keys are loaded.

2.3.5 Delegation of Authority

One may wish to allow another user to use a key without allowing that other user all the rights and privileges that come with ownership of the key. Anyone in possession of the key and the key's AuthData effectively has ownership privileges.

To address this, version 1.2 of the TPM specification documents ([Tru07b, Tru06a, Tru06b]) include the concept of key *delegation*. The recipient of a delegated key can use the key only for specific purposes which are specified at the time of delegation.

Delegate AuthData (distinct from owner AuthData) is added to the delegate blob to allow non-owner use of the key.

2.3.6 Transport sessions

The TPM can create an encrypted "tunnel" over which a process may communicate with the TPM without fear of key or authorization data being copied.

Three types of transport sessions are described by the TPM specification:

- OIAP - Object Independent Authorization Protocol
- OSAP - Object Specific Authorization Protocol
- DSAP - Delegate Specific Authorization Protocol

[6]This was 16 in the TPM 1.1 specification and may increase in future versions.

When performing multiple operations on a single object, OSAP and DSAP allow an application to specify AuthData once for an entire session rather than each time the object is used.

A DSAP session is required to use a delegated key.

2.4 Trusted Computing Software Stack

The Trusted Computing Software Stack (TSS) is the software required by any system to access its TPM, including the TPM device driver. The data structures and interfaces to the TSS are defined by TCG specifications.

PC manufacturer's include Windows drivers and BIOS hooks for the specific TPM chip they supply in their systems. A Linux TPM device driver and software stack is also available.

Chapter 3

Previous work

This mini-project builds on two areas of previous work: Java classes used to access the TPM and a security analysis of the TPM API identifying the attack vector.

3.1 Accessing the TPM

Because the TPM is a chip without a great deal of space for sophisticated API capabilities, accessing it through the device driver layer is inconvenient, to say the least. A complete TSS layer would provide the needed APIs, but many platforms do not yet supply any more than the most basic interface. In most cases, input data must be marshalled into long sequences of bytes and results unmarshalled back into useful data objects.

Three projects attempt to address this missing functionality with varying degrees of usefulness for the purposes of this mini-project. No project implements a TSS layer compliant with the TCG TSS specification for version 1.2. Therefore, no project supports key delegation or DSAP transport sessions since these functions are new in the TPM and TSS version 1.2 specifications.

Even so, all of these tools provide some level of useful abstraction that is superior to direct communication with the TPM device driver. For code development in this mini-project, TPM/J was chosen because of its relative ease of extensibility and support for multiple hardware platforms.

3.1.1 jTSS

jTSS is Java implementation of the TCG Software Stack (TSS). It is developed by the Institute for Applied Information Processing and Communications (IAIK) in the the Computer Science department at Graz University of Technology[1] in Austria. The code is distributed through SourceForge at `http://trustedjava.sourceforge.net`.

This code is in the early stage of development. The current release is the first release and is considered to be experimental. A companion package, jTpmTools, provides command-line interfaces to some TPM functions. jTSS and jTpmTools implement version 1.1 of the TSS specification.

[1] `http://www.iaik.tugraz.at`

3.1.2 TPM/J

TPM/J is an set of Java classes that closely mimics the TPM functions defined in the TSS. TPM/J is developed by the Computer Science and Artificial Intelligence Laboratory (CSAIL) at the Massachusetts Institute of Technology (MIT). The code is available from the CSAIL web site at `http://projects.csail.mit.edu/tc/tpmj`.

TPM/J includes some command-line versions of common TPM functions implemented with the Java classes. The code is no longer actively developed and does not fully implement the TSS specification, but it does support use of version 1.2 TPM devices.

3.1.3 TrouSerS

TrouSerS is an open-source TCG Software Stack (TSS) implemented in Java. The code is distributed through SourceForge at `http://trousers.sourceforge.net` and implements the TCG TSS 1.1 (TSS 1.2 support is currently under development).

TrouSerS currently runs only on Linux platforms.

3.2 Attack vector

Several software attacks on the TPM have been contemplated in recent years, most via protocol or API analysis with verification tools:

- [BCLM05] describes an OIAP transport session replay attack
- [GRS+07] proposes a method of undermining the integrity of remote attestation
- [CR08b] examines methods of mitigating a plausible offline dictionary attack
- [CR08a] theorizes a sort of man-in-the-middle attack on the TPM based on the weakness inherent in shared authorization data

The attack attempted in this mini-project is another idea uncovered by a verification analysis of the TPM. In his MIT Master's thesis ([Lin05]), Amerson Lin analyzes the security of some of the functions of the TPM. In his analysis, he identifies three potential security vulnerabilities that could occur if the chip designers made poor choices due to ambiguities in the TPM specification.

Lin's thesis mainly concerns itself with the automated analysis, but in the course of his narrative, he alludes to a potential flaw in the specification of the TPM regarding key delegation.

On page 81 of his thesis (quoted in Figure 3.1), Lin somewhat casually mentions that his model showed a DSAP session might be vulnerable to a key handle switching attack, such as in the following steps:

1. user1 creates delegate blob for a key
2. user1 gives user2 the delegate blob and delegate AuthData
3. user2 establishes a valid DSAP session with the delegated key

> "Our model found that the DSAP transport session was susceptible to a key-handle switching attack ... The DSAP session command does not check that the target resource in the input parameters matches the resource specified in delegation because the usage password of the target resource was not used to generate the session shared-secret. Rather, it is the delegation password that was used. Therefore, it is possible for an adversary to specify a completely arbitrary resource handle while still providing the delegated password. The command would still validate and the user would be given usage access to the resource."

Figure 3.1: Quote from Amerson Lin's thesis

4. user2 requests an operation in the session on another (unrelated) key, for which he does not have delegate privileges nor valid AuthData

Lin claims this will allow the user access to the unrelated key because, since the delegate AuthData was used to authenticate the transport session rather than the owner AuthData, there is no way for the TPM to check the owner AuthData of the second key.

The questions is: is Lin's assertion correct? Is this a vulnerability in the TPM? This mini-project will answer this question by attempting the attack he envisions.

Chapter 4

Executing the attack

To attempt Lin's attack on the TPM, a system with an activated TPM (i.e., including the proper libraries and drivers), the Java developer and run-time environments, and TPM/J is required. After setting up such a system, application code can be written to perform TPM operations via existing TPM/J functions.

TPM/J does not, however, provide all the necessary functionality to be able to implement Lin's attack. Therefore, new code must be written for TPM/J to implement missing functions, after which, application code to implement the attack can be developed.

4.1 System set up

Most Intel-based computers contain a TPM, but the hardware documentation for the system should be consulted to verify it.

4.1.1 TPM driver

Windows systems should have the proper device driver already installed by the vendor. Linux systems may require optional software to install the TPM device driver, depending on the distribution.

The target platform in this mini-project is a Lenovo system running Windows XP. The TPM driver was downloaded from `http://www.lenovo.com` and included a library `TPMDDL.DLL`.

When running on a Windows platform, TPM/J looks for a library called `IFXTPM.DLL`. When renamed and installed in `C:\WINDOWS\System32`, the library from Lenovo satisfies the requirement.

4.1.2 TPM/J

TPM/J can be downloaded from `http://projects.csail.mit.edu/tc/tpmj` and installed in a system directory or in a user directory. The TPM/J User's Guide ([Sar07b]) lists the Java Runtime Environment (JRE) version 5.0 as a minimum requirement.

Dependencies

TPM/J uses the Bouncy Castle Provider library[1] for some encryption functionality. This is provided with TPM/J in the file `bcprov-jdk15-131.jar` and is required for some TPM/J classes to function.

On Windows platforms, TPM/J also requires the included library `IFXTPMJNIProxy.dll`.

Definitions

Two steps are required to define the locations of these TPM/J dependencies for the JRE. The TPM/J User's Guide lists them and TPM/J includes example scripts.

Assuming TPM/J has been installed on a Windows platform in `C:\TPMJ`, the definitions required are as follows:

1. Add the location of the required library to the java command:

```
java -Djava.library.path=C:\TPMJ\lib
```

2. Define CLASSPATH to point to the location of TPM/J and the Bouncy Castle Provider library upon which it depends , such as:

```
CLASSPATH=
 "C:\TPMJ\lib\tpmj.jar;C:\TPMJ\lib\bcprov-jdk15-131.jar"
```

   Note that the delimiter between components in a CLASSPATH definition is a semicolon (;) on a Windows system but is a colon (:) on a Linux system. This value may also be specified on the command line using the `-cp` argument to the `java` command.

Activation

The TPM must be active for TPM/J to be able to do much of anything useful. TPM/J provides the class `edu.mit.csail.tpmj.tools.`**TPMTakeOwnership** that can be run from the command line in order to "take ownership" and activate the TPM. This process is described in the User's Guide ([Sar07b]).

4.2 TPM/J architecture

TPM/J provides both high-level "convenience" functions that hide some of the complexity of communicating with the TPM and lower-level classes that allow more flexibility in operation to access TPM functionality from a Java program. The TPM/J Developer's Guide ([Sar07a]) gives a brief but helpful overview.

The basic process to access the TPM is to initialize the TPM driver and create a driver object. This object is then passed to most other command or

[1] See `http://www.bouncycastle.org`

session classes to create their corresponding objects. Each command object has a corresponding output object where the results of the TPM command are stored and accessible to the calling program.

The session classes implement the TPM OSAP and OIAP sessions. The current release of TPM/J does not support DSAP sessions. Command classes correspond to TPM commands.

4.2.1 TPM driver object

The first thing any program must do to access the TPM is use the TPM/J convenience functions *initTPMDriver()* and *getTPMDriver()* to initialize the driver and create a driver object, such as:

```
TPMUtilityFuncs.initTPMDriver();
TPMDriver tpmDriver = TPMUtilityFuncs.getTPMDriver();
```

When it has finished accessing the TPM, the program should also use the *cleanupTPMDriver()* convenience function:

```
TPMUtilityFuncs.cleanupTPMDriver();
```

4.2.2 Command objects

Once the TPM driver has been created, commands can be sent to the TPM. Some commands do not require authorization because they return no protected data. For example, anyone can read the value of one of the Platform Configuration Registers (PCRs), so no authorization is required for this function.

Each TPM/J command class includes an *execute()* method. To use a TPM command, create the command object with any required arguments, and then run this method. When not using an authorization session, the execute method takes the TPM driver object as its argument. For example, the following code could be used to read the value of an arbitrary PCR:

```
TPM_PCRRead cmd = new TPM_PCRRead(5);
TPM_PCRReadOutput output = cmd.execute(tpmDriver);
TPM_PCRVALUE pcrVal = output.getOutDigest();
```

This is a simplified version of the operation. TPM/J command class *execute()* methods throw an exception (TPMException) that must be caught by the calling program.

TPM commands requiring authorization are more complex. Authorization is provided via a *transport session* and the command(s) to operate on the object are sent to the TPM using this session. TPM/J provides higher-level convenience functions for the most common operations that require authorization. These classes manage the transport session behind the scenes. If multiple operations on an object (or multiple objects) are to be performed, however, the application must manage its own transport session(s).

4.2.3 Session objects

TPM/J supports two types of transport sessions. The Object Specific Authorization Protocol (OSAP) is used to run one or more TPM commands on a single object (e.g., a key). The Object Independent Authorization Protocol (OSAP) is used to run commands on more than one object. TPM/J does not support the Delegate Specific Authorization Protocol (DSAP).

Session objects are similar to TPM command objects (in fact, the session classes are extended from the **TPMCommand** base class). To open an OSAP session on a previously loaded key, a program would create a *TPMOSAPSession* object:

```
TPMOSAPSession session = new TPMOSAPSession(tpmDriver);
session.startSession(entityType, keyHandle, authData);
```

Again, this is simplified because the **TPMOSAPSession** class methods throw exceptions on errors. In this case, `entityType` is a constant that indicates the object being used is a key, `keyHandle` is the handle returned when the key was loaded into the TPM, and `authData` is the authorization data (the SHA-1 hash of the password).

Now that the program has a session handle, it can be passed to other TPM command objects to perform commands on the key. For example, to get the public key value of the key, create a **TPM_GetPubKey** object and run its *execute()* method passing it the OSAP session object:

```
TPM_GetPubKey pubKey = new TPM_GetPubKey(keyHandle);
TPM_GetPubKeyOutput output = pubKey.execute(session, true);
System.out.println("Public Key: " + output.getPubKey() );
```

The boolean value `true` passed to the *execute()* method tells the TPM to keep this session open. Using a value of `false` here will close the session after the operation has completed.

4.3 Identify missing functionality

Lin's attack on the TPM requires the ability to create a delegate key blob and use it in a DSAP transport session.

4.3.1 DSAP session

The classes that would be required to support a DSAP transport session are similar to those that already exist to support the OSAP transport session:

- **TPM_DSAP**
- **TPM_DSAPOutput**
- **TPMDSAPSession**

The major differences between the DSAP session and the OSAP session are:

- the entity passed is a delegate blob
- the DSAP session creates the session shared secret from the delegate AuthData in the delegate blob rather than the owner AuthData of the delegated object

4.3.2 Creation of delegate key

In order to use (or in fact, to even need) a DSAP transport session, one must have a delegate key blob which is created by the **TPM_Delegate_CreateKeyDelegation** TPM command. TPM/J does not implement a class to run this TPM command, so the following new classes will be required:

- **TPM_Delegate_CreateKeyDelegation**
- **TPM_Delegate_CreateKeyDelegationOutput**

The **TPM_Delegate_CreateKeyDelegation** TPM command creates a blob of data containing a key and new delegate AuthData that can be used by another user or process. The blob also contains a bitmask value specifying which specific TPM commands are allowed in the delegation.

The TPM has another command called **TPM_Delegate_CreateOwnerDelegation** that delegates ownership privileges of a key to another user. That function is not required to attempt this attack, so it was not implemented.

4.3.3 Access to the Family Table

The TPM maintains internal tables to support key delegation. One of these tables is called the Family Table.

Key delegations are created in groups called *families*. Often a single delegation makes up a family, but multiple key delegations can be grouped together as a family.

The purpose is to allow the delegating entity (user or process) to change the delegation of a group of keys. Once a delegation has been created and the delegate blob distributed, there must be a way of invalidating the key at a later point in time (for example, if an employee to whom a key has been delegated is terminated).

Each delegation must belong to a valid family. Each family in the Family Table maintains a *verification count* for the family. A delegate blob is also given a verification count when it is created. So long as the delegate blob's verification count matches the verification count in the Family Table in the TPM, the key in the delegate blob may be used. To invalidate a delegation the verification count in the Family Table entry is incremented. Because the verification count of any delegate blob no longer matches the verification count of the family, the delegation is rendered invalid. New delegate blobs can then be issued to the proper entities.

Part of the process of delegating a key is adding it to the TPM's Family Table. Therefore, support for accessing and updating the Family Table is required.

Each family table entry contains information about the specific family of delegations. Each family is uniquely identified by its family ID (integer) and it's family label (byte).

4.4 Implement new TPM/J classes

The TPM/J Developer's Guide ([Sar07a]) provides direction in developing other TPM/J classes. Most of the difficulty of developing a new TPM/J class centers around parameter passing to and from the TPM and the methods required to implement this properly. One incorrect size count or ordering of data causes a TPM command to fail.

4.4.1 Family table classes

The lowest level of functionality identified in the previous section is the creation and modification of Family Table entries.

The TPM command used to create entries in the Family Table is **TPM_Delegate_Manage**. Therefore, the new classes required for TPM/J will be:

- **TPM_Delegate_Manage**
- **TPM_Delegate_ManageOutput**

TPM_Delegate_Manage can operate on an existing Family Table entry or to create a new entry in the Family Table. When used to create a new entry, *TPM_Delegate_ManageOutput()* is used to retrieve the `familyID`. Figure 4.1 lists the methods defined by the output class.

TPM_Delegate_Manage is also used to "invalidate" a family, which removes the entry from the Family Table.

Method	Purpose
getFamilyID()	returns the `familyID` of the Family Table entry represented by the object

Table 4.1: **TPM_Delegate_ManageOutput** public method

Simplified code to create and enable an entry in the Family Table using an already established OIAP session would work like this:

```
TPM_Delegate_Manage cmd =
  new TPM_Delegate_Manage(TPMConsts.TPM_FAMILY_CREATE, label);

TPM_Delegate_ManageOutput output =
  session.executeAuth1Cmd(cmd, sessionSecret, true);
```

```
int familyID = output.getFamilyID();

cmd = new TPM_Delegate_Manage(TPMConsts.TPM_FAMILY_ENABLE,
                              familyID, label);
output = session.executeAuth1Cmd(cmd, sessionSecret, false);
```

The *(()*executeAuth1Cmd) is similar to the *execute()* method, but also deals with the required authorization information sent to the TPM. This method is defined for classes whose corresponding TPM commands require authorization.

Another TPM command used to read the Family Table is **TPM_Delegate_ReadTable**. This command is necessary to get the verification count, as well as other data about a Family Table entry. Therefore, two more TPM/J classes will be required:

- **TPM_Delegate_ReadTable**
- **TPM_Delegate_ReadTableOutput**

TPM_Delegate_ReadTable retrieves the data for a single entry in the Family Table. Then the methods defined by **TPM_Delegate_ReadTableOutput** shown in Table 4.2 are used to get the values.

Method	**Purpose**
getFamilyTableLength()	returns number of entries in the Family Table
getFamilyTableEntryLabel()	returns the label of a specified Family Table entry
getFamilyTableEntryFamilyID()	returns the `familyID` of a specified Family Table entry
getFamilyTableEntryVerificationCount()	returns the value of the verification count for the specified entry in the Family Table
getFamilyTableEntryFlags()	returns the flags associated with the specified Family Table entry

Table 4.2: **TPM_Delegate_ReadTableOutput** public methods

No authorization is required to read the Family Table, so simplified code to create a *TPM_Delegate_ReadTable* object will work like this:

```
TPM_Delegate_ReadTable cmd = new TPM_Delegate_ReadTable();
table = cmd.execute(tpmDriver);
```

Then any of the methods in Table 4.2 can be used to get information about any specific Family Table entry. For example, to get the verification count for a known `familyID`:

```
int n = table.getNumberOfEntries();
int verCount = 0;

for (int i=1; i <= n; i++)
{
```

```
      if (familyID == table.getFamilyID(i)) {
         verCount = table.getVerificationCount(i);
      }
   }
```

4.4.2 Delegate key classes

The TPM command used to create delegate key blobs is **TPM_Delegate_CreateKeyDelegation**. Therefore, the new classes required for TPM/J will be:

- **TPM_Delegate_CreateKeyDelegation**
- **TPM_Delegate_CreateKeyDelegationOutput**

Figure 4.3 lists the methods defined by the output class.

Method	**Purpose**
getBlobSize()	returns the length of the delegate blob
getBlob()	returns the delegate blob

Table 4.3: **TPM_Delegate_CreateKeyDelegationOutput** public method

The simplified code to create a delegate blob, using an already established OSAP session, will work like this:

```
TPM_Delegate_CreateKeyDelegation cmd =
   new TPM_Delegate_CreateKeyDelegation(keyHandle,
                  delegateAuthData, familyID, label,
                  verificationCount, delegateMask);
TPM_Delegate_CreateKeyDelegationOutput output =
   cmd.execute(session, false);

byte[] delegateBlob = output.getBlob();
```

4.4.3 DSAP session classes

Three classes implement OSAP sessions in TPM/J so support for DSAP will require the corresponding three classes:

- **TPMDSAPSession**
- **TPM_DSAP**
- **TPM_DSAPOutput**

These classes operate the same way as the OSAP classes except they operate on a delegate blob and delegate AuthData instead of a regular object, like a key blob, and its owner AuthData.

The most significant modification from its counterpart is that of the **TPM_DSAP** class, so as an example, it is listed in Appendix C on page 34.

The rest of the source code is included in the CD-ROM but not included in this report for the sake of page length.

When the support code described in this section was written and added to the TPM/J infrastructure, it was then possible to create a delegate key blob for a key, specifying new delegate AuthData, and to use that blob in a DSAP session. Any attempt to use the delegate blob with the wrong AuthData (even with the original owner AuthData for the key) resulted in a `TPM_AUTH_FAIL` as expected.

4.5 Implement attack code

All that remains now is to develop the application code using the new-and-improved TPM/J to attempt Lin's attack on the TPM.

Because some of the operations required are non-trivial and required in several different places in the program, they have been implemented as new application classes along with the new **Attack** class. The **Attack** class will make use of additional classes that manage key blobs and access to the TPM Family Table.

4.5.1 The Key class

The **Key** class is used to create a new key or perform TPM operations on an existing key. All these operations are performed over an OSAP session that is established by the method and terminated when the operation is complete.

Method	Purpose
Key()	the constructor returns a new key object containing a key blob created by the TPM
load()	loads the key into the TPM
getPublic()	returns the public key portion of the key
evict()	removes the key from the TPM

Table 4.4: **Key** public methods

A new key can be created, loaded into the TPM, and later removed from the TPM with the following code fragment:

```
String password="my pass phrase";
Key key = new Key(password);
int keyHandle = key.load();
System.out.println( "keyHandle = 0x" +
                    Integer.toHexString(keyHandle) );
...

key.evict();
```

Table 4.4 shows the public methods of this class.

4.5.2 Family Table classes

Two classes are required to support access to the TPM's Family Table.

The **Family** class is used to create or remove (invalidate) a Family Table entry in the TPM. **Family** issues **TPM_Delegate_Manage** commands to the TPM via an OIAP session that it creates and terminates when its operation is complete. The methods are listed in Table 4.5.

Method	**Purpose**
create()	create an enabled entry in the TPM's Family Table with the provided 1-byte label
invalidate()	invalidate (remove) an entry in the TPM's Family Table corresponding to the given family ID and label
getFamilyID()	Unused: returns the family ID stored in the object, NOT from the TPM
getLabel()	Unused: returns the label stored in the object, NOT from the TPM

Table 4.5: **Family** public methods

The **FamilyTable** class is used to access the data in the TPM's Family Table. **FamilyTable** issues **TPM_Delegate_ReadTable** commands to the TPM. No transport session is necessary because **TPM_Delegate_ReadTable** does not require authorization. The methods are listed in Table 4.6.

Method	**Purpose**
FamilyTable()	the constructor creates a *FamilyTable* object containing a snapshot of the current TPM Family Table
reload()	reload the Family Table information from the TPM
getNumberOfEntries()	return the number of entries in the Family Table
getLabel()	return the 1-byte label for the a Family Table entry
getFamilyID()	return the family ID for a given Family Table entry
getVerificationCount()	return the verification count for a given Family Table entry
getFamilyVerificationCount()	return the verification count for a given family ID
getFlags()	return the flags field for a given Family Table entry

Table 4.6: **FamilyTable** public methods

A third class called **ClearFamilyTable** uses both of these classes to access Family Table data and invalidate all rows in the Family Table. This was a requirement during development as the TPM can only maintain eight rows in the Family Table and new entries in the Family Table entries were being created on a regular basis. **ClearFamilyTable** has no public methods, it has only a *main()* method so that it can be run from the command line.

4.5.3 The Attack class

The final class developed is **Attack**, the class that implements Amerson Lin's attack on the TPM.

The full source code listing for the **Attack** class (`Attack.java`) can be found in Appendix C beginning on page 35 and on the CD-ROM.

Attack has no public methods, it has only a *main()* method so that it can be run from the command line.

The basic algorithm of the **Attack** class is as follows:

1. Create a key, key1, which we will delegate, and load it into the TPM.
2. Get the public key portion of key1 to show we can use it.
3. Create a key, key2, which we will not delegate, and load it into the TPM.
4. Get the public key portion of key2 to show we can use it.
5. Create a Family Table entry for the delegation.
6. Get the verification count from the Family Table entry for the delegation.
7. Create a delegate blob from key1 with new AuthData and specifying **TPM_GetPubKey** as a delegated command.
8. Open a DSAP session with the delegate blob and the delegate AuthData.
9. Get the public key portion of (delegated) key1.
10. Attempt to get the public key portion of key2.

4.6 Attack result

The **Attack** class successfully creates a delegate blob and opens a DSAP session with it. The TPM command **TPM_GetPubKey** successfully executes and the public part of key1 is returned via the DSAP session.

When the delegate blob is created to delegate a different TPM command and used in the DSAP session, the **TPM_GetPubKey** (correctly) fails with the TPM error `TPM_BAD_DELEGATE`. A properly delegated command can be run on the delegated key, multiple times if attempted, successfully. TPM commands not included in the delegation fail as expected.

When attempting to use key2 with the delegate blob, the TPM refuses access. Using the key handle for key2 when opening the DSAP session results in the error `TPM_INVALID_KEYHANDLE` and the DSAP session is not initiated.

When opening the DSAP session with the handle for key1 (i.e., properly), and then attempting to use key2 in the session itself (and for the proper delegated command), the TPM returns `TPM_AUTH_FAIL`.

Chapter 5

Conclusion

The TPM withstood the attack envisioned by Lin. But why?

Lin's contention (in Figure 3.1 on page 11) was that because the session shared secret is generated from the delegate AuthData rather than the owner AuthData (which is true) that the TPM would not notice a change in key being used in the transport sessions. However, this presumes that the session shared secret is the only verification being performed on the session.

The definition of **TPM_DSAP** in the TPM specification does not specifically discuss authorization as much as the definition of **TPM_OSAP**. However, the two commands are very similar in many respects and the definition of **TPM_OSAP** does state ([Tru06b, p. 177] line 3064) that the OSAP session must track "any other internal TPM state that the TPM needs to manage the session." While not specifying what that state might be, it is not difficult to imagine that it might include the key handle for which the session was created (at least in the case of OSAP, and possibly DSAP).

The shared secret is used as the encryption key for the transport session, but the TPM specification does not indicate that it is used for any type of session validation.

An additional attack was attempted with two keys that both used the same AuthData values. The delegate blob was also created with this same AuthData. So if any AuthData was all that was being used to validate the use of an object in a DSAP session, this would work. But this test also failed, indicating that something other than AuthData is being used to validate the use of the key.

Another possibility is noted in the definition of **TPM_DSAP** in [Tru06b]. On p. 182, line 3169, the specification states:

> "Each ordinal that uses the DSAP session MUST validate that `TPM_PERMANENT_DATA -> restrictDelegate` does not restrict delegation, based on `keyHandle -> keyUsage` and `keyHandle -> keyFlags`, return `TPM_INVALID_KEYUSAGE` on error."

This indicates that each TPM command (ordinal) is responsible for allowing or denying use of a key and that it is based on the key handle. If this interpretation is correct, then switching a key handle on any command in a DSAP session should fail, as observed.

Unfortunately, many of the inner workings of the TPM are not explicitly defined by the specification and therefore left open to interpretation by the

vendor. So while it is difficult to be sure what the TPM is using to verify the key, experimentation indicates that it is doing so properly.

5.1 Future Work

To date, this attack has only been run against an Atmel TPM chip. The intent is to also run it on systems containing TPM chips manufactured by other vendors to see if any difference is observed.

Presuming other chips perform the same way, the specific question regarding the alleged vulnerability in the TPM will have been answered.

It is the case, however, that the TPM-related code that was developed for this mini-project was incomplete. Other functionality that is part of the TPM was not implemented in TPM/J because it was not required for this mini-project and time did not permit a complete implementation. Before this code could be used in another TPM/J-based application, the implementation would need to be completed.

- Creating a delegate owner blob, and therefore owner delegation in general, is not supported.
- **TPM_Delegate_ReadTable** only supports the Family Table. Fully implemented, it would also support the Delegate Table which is also required to implement owner delegation.
- DSAP sessions, as implemented, do not support the "one-shot" uses comparable to other TPM/J session classes.
- The `TPM_DELEGATE_PUBLIC` structure defined in **TPM_Delegate_CreateKeyDelegation** is a quick-and-dirty version and should instead be implemented with additional new TPM/J classes.
- Many selection options in **TPM_Delegate_CreateKeyDelegation** are not supported (e.g., PCR selection options).

Bibliography

[BCLM05] Danilo Bruschi, Lorenzo Cavallaro, Andrea Lanzi, and Mattia Monga, *Replay attack in TCG specification and solution*, ACSAC '05: Proceedings of the 21st Annual Computer Security Applications Conference (Washington, DC, USA), IEEE Computer Society, 2005, pp. 127–137.

[CR08a] Liqun Chen and Mark Ryan, *Attack, solution and verification for share authorisation data in TCG TPM*, draft under review, 2008.

[CR08b] ———, *Offline dictionary attack on TCG TPM weak authorisation data, and solution*, pp. 43–56, Vieweg & Teubner, May 2008.

[CYC+08] David Challener, Kent Yoder, Ryan Catherman, David Safford, and Leendert Van Doorn, *A Practical Guide to Trusted Computing*, IBM Press, Upper Saddle River, NJ, 2008.

[GRS+07] S. Gürgens, C. Rudolph, D. Scheuermann, M. Atts, and R. Plaga, *Security evaluation of scenarios based on the TCG's TPM specification*, Computer Security - ESORICS 2007 (Joachim Biskup and Javier Lopez, eds.), Lecture Notes in Computer Science, vol. 4734, Springer Verlag, 2007.

[Gun08] Vandana Gunupudi, *Exploring trusted platform module capabilities: A theoretical and experimental study*, Ph.D. thesis, University of North Texas, Denton, Texas, USA, May 2008.

[Kay05] Roger Kay, *The future of trusted computing*, slides from GovSec 2005 keynote, May 2005.

[Lin05] Amerson H. Lin, *Automated analysis of security APIs*, Master's thesis, Massachussetts Institute of Technology, May 2005.

[Rya08] Mark D. Ryan, *Introduction to the TPM 1.2*, not yet published, July 2008.

[Sar07a] Luis Sarmenta, *TPM/J Developer's Guide*, Massachussetts Institute of Technology, April 2007.

[Sar07b] ———, *TPM/J User's Guide*, Massachussetts Institute of Technology, April 2007.

[Tru06a] Trusted Computing Group, Inc., *TPM Main Part 2 TPM Structures*, Trusted Computing Group, October 2006.

[Tru06b] ———, *TPM Main Part 3 Commands*, Trusted Computing Group, October 2006.

[Tru07a] ———, *TCG Specification Architecture Overview*, Trusted Computing Group, August 2007, draft.

[Tru07b] ———, *TPM Main Part 1 Design Principles, Revision 103*, Trusted Computing Group, July 2007.

The bibliography has been prepared using the BibTEX `amsalpha` style provided by the American Mathematical Society, Providence, RI (1995).

Appendix B

Statement of information search strategy

B.1 Parameters of literature search

B.1.1 Forms of literature

The literature search for this mini-project was performed on all typical categories:

- conference papers
- technical reports
- journal articles
- theses
- books

Even though books are often not as timely, and therefore, not as useful, the book search did turn up several recent books on the TPM. The one included in the bibliography was written by some of the authors of the TPM specification.

B.1.2 Geographical/language coverage

The search was not restricted to either a specific geography or language. The searches did turn up several Chinese references that were unusable (but many references from Chinese conferences or companies were in English). While the non-English references were not helpful, it did show how much TPM work is being done in academia in China, which is interesting in itself.

B.1.3 Retrospective coverage and currency

While the TPM is a fairly recent technological development, it is useful to restrict the searches to post-2000 to eliminate other subjects using the "TPM" acronym. By restricting the date range, more references found really were about the Trusted Platform Module.

B.2 Search tools

The following search tools were used in the literature search for this mini-project:

- University of Birmingham eLibrary
- Engineering Village/Inspec
- Science Citation Index/ISI Web of Knowledge
- ProQuest/Dissertation Abstracts International
- Index to Theses
- Amazon and Google

B.3 Search statements

The advantage of the TPM technology being a recent development in the industry is searches do not return an overabundance of results for a generic search on "Trusted Platform Module," and the results that are returned are likely applicable.

This mini-project is related to a specific subset of the TPM, key delegation. While finding generic TPM references was easy, finding any that dealt specifically with key delegation proved difficult. This indicates that few people have tried to use it yet. Because it is part of only the latest version of the TPM specification, this is not unexpected.

The search terms used in searches for this mini-project were:

- Trusted Platform Module
- Trusted Computing Group
- TPM and TCG
- TPM and trusted
- TPM and delegate
- TPM and delegation
- TPM and DSAP

B.4 Evaluation of the searches

A search of *The University of Birmingham eLibrary* (`http://elibrary.bham.ac.uk`) for TPM-related journal papers was fruitless.

A search of *Engineering Village* (also called *Inspec*, `http://www.engineeringvillage2.org`) resulted in no directly applicable references. The initial search using "Trusted Platform Module" resulted in over 2000 results that comprised anything that had anything to do with the TPM. This was clearly not sufficient search refinement. However, adding any mention of delegation to the search reduces the results to zero. Several scans

of combinations that brought only tens or hundreds of results revealed sources that were not very interesting, nor were they applicable to this mini-project.

The search of the *Science Citation Index* (part of the *ISI Web of Knowledge*, `http://wok.mimas.ac.uk`) resulted in 28 references citing some volume of the TPM Specification documents. Of these, only five conference papers were closely related enough to consider. Three of these were from European conferences, one was published in China, and one in Australia. None of the references found related directly to key delegation.

A subsequent search of the main part of the *ISI Web of Knowledge* resulted in 60-200 (depending on the exact keywords used) references, none directly related to this mini-project. One reference, however, was later provided by the project supervisor as an example of other software-related security attacks on the TPM so is included in the bibliography.

A citation search of *ProQuest* (formerly *Dissertation Abstracts International*, `http://proquest.umi.com/login`) resulted in two doctoral dissertation references, both from the United States. One of these was applicable enough to be used for background material but was not directly applicable to the topic of the mini-project.

A subsequent regular search of *ProQuest* resulted in 89 TPM-related results but none applicable to the specific topic of key delegation (with the single exception of a 1-page magazine article reprint that only mentions it).

A search of *Index to Theses* (`http://www.theses.com`) resulted in no thesis references. All references to "TPM" appeared to be related to medicine or manufacturing.

Though not as well-respected as these search tools, common Internet search sites like `http://www.amazon.co.uk` and `http://www.google.com` proved quite useful. A search on Amazon turned up several recent books on the TPM, one of which was used for background material in this project and did provide limited information on key delegation. Google, of course, returned thousands of results, but by searching using strings of text from abstracts of papers found through other sources, one could locate a copy of a paper that had only been found as an abstract previously (sometimes on the author's personal web site). These references may not be as permanent and universal, but they often provided quick access to the information that was needed. This is, no doubt, due to the more applied nature of this mini-project.

Appendix C

Selected source code

For the sake of page length, only selected source code is included in this text. All source code developed for this mini-project is contained on the accompanying CD-ROM. The execution environment can be created with the contents of this CD-ROM, a Java Runtime Environment, and the TPM/J distribution from MIT (see page 13).

C.1 TPM_DSAP.java

```
 1 package edu.mit.csail.tpmj.commands;

 2 /*
 3  * based on TPM_OSAP
 4  */

 5 import edu.mit.csail.tpmj.TPMConsts;
 6 import edu.mit.csail.tpmj.TPMException;
 7 import edu.mit.csail.tpmj.drivers.TPMDriver;
 8 import edu.mit.csail.tpmj.structs.*;
 9 import edu.mit.csail.tpmj.util.ByteArrayReadWriter;

10 /*
11  * @param entityType - TPM_ET_DEL_KEY_BLOB or TPM_ET_DEL_OWNER_BLOB
12  * @param keyHandle - handle of (loaded) delegated key
13  * @param nonceOddDSAP - odd nonce (managed by session obj)
14  * @param entityValueSize - size of blob
15  * @param entityValue - blob
16  */
17 public class TPM_DSAP extends TPMCommand
18 {
19     private short entityType;
20     private int entityValueSize;
21     private byte[] entityValue;
22     private int keyHandle;
23     private TPM_NONCE nonceOddDSAP;

24     public TPM_DSAP( short entityType, int keyHandle, TPM_NONCE nonceOddDSAP,
25             int entityValueSize, byte[] entityValue )
26     {
27         super( TPMConsts.TPM_TAG_RQU_COMMAND, TPMConsts.TPM_ORD_DSAP );
28         this.keyHandle = keyHandle;
29         this.entityType = entityType;
30         this.entityValue = entityValue;
31         this.entityValueSize = entityValueSize;
32         this.setNonceOddDSAP( nonceOddDSAP );

33         this.setParamSize( 40 + this.entityValueSize );
34     }
```

```
35     public TPM_NONCE getNonceOddDSAP()
36     {
37         return nonceOddDSAP;
38     }

39     public void setNonceOddDSAP( TPM_NONCE nonceOddDSAP )
40     {
41         this.nonceOddDSAP = nonceOddDSAP;
42     }

43     public Class getReturnType()
44     {
45         return TPM_DSAPOutput.class;
46     }

47     @Override
48     public TPM_DSAPOutput execute( TPMDriver tpmDriver ) throws TPMException
49     {
50         return (TPM_DSAPOutput) super.execute( tpmDriver );
51     }

52     @Override
53     public byte[] toBytes()
54     {
55         return this.createHeaderAndBody( this.entityType, this.keyHandle,
56                 this.nonceOddDSAP, this.entityValueSize, this.entityValue );
57     }

58     @Override
59     public void fromBytes( byte[] source, int offset )
60     {
61         this.readHeader( source, offset );
62         ByteArrayReadWriter brw = this.createBodyReadWriter( source, offset );

63         this.entityType = brw.readShort();
64         this.keyHandle = brw.readInt32();
65         this.nonceOddDSAP = new TPM_NONCE();
66         brw.readStruct( this.nonceOddDSAP );
67         this.entityValueSize = brw.readInt32();
68         this.entityValue = brw.readBytes(this.entityValueSize);
69     }

70 }
```

C.2 Attack.java

```
 1 import edu.mit.csail.tpmj.*;
 2 import edu.mit.csail.tpmj.commands.*;
 3 import edu.mit.csail.tpmj.drivers.TPMDriver;
 4 import edu.mit.csail.tpmj.funcs.*;
 5 import edu.mit.csail.tpmj.structs.*;
 6 import edu.mit.csail.tpmj.util.*;

 7 public class Attack
 8 {
 9     /*
10      * Attempt Amerson Lin's attack on the TPM
11      *
12      * create a delegation blob for a key
13      * open a DSAP session with the delegated key
14      * attempt to use a second, undelegated key in the session
15      */
16     public static void main( String[] args )
17     {
18         TPMUtilityFuncs.initTPMDriver();
19         TPMDriver tpmDriver = TPMUtilityFuncs.getTPMDriver();

20         Key key1 = new Key("password1");
21         TPM_SECRET keyAuth1 = key1.getAuthData();
22         int keyHandle1 = key1.load();
23
24         System.out.println("Loaded key1: owner authData = " + keyAuth1 + "\n"
```

```
25                + "keyHandle = 0x" + Integer.toHexString(keyHandle1) + "\n"
26                + key1.getPublic() );

27        // create a second key and load it into the TPM

28        Key key2 = new Key("password2");
29        TPM_SECRET keyAuth2 = key2.getAuthData();
30        int keyHandle2 = key2.load();
31
32        System.out.println("Loaded key2: owner authData = " + keyAuth2 + "\n"
33                + "keyHandle = 0x" + Integer.toHexString(keyHandle2) + "\n"
34                + key2.getPublic() );

35        // create the family table entry for the delegation

36        Family familyEntry = new Family(tpmDriver);
37        byte familyLabel = new String("a").getBytes()[0];

38        int familyID = familyEntry.create(familyLabel);

39        if (familyID < 0) {
40            System.out.println("Error: cannot create a new family table entry.");
41        } else {
42            System.out.println("created family id "
43                    + familyEntry.getFamilyID() );
44        }

45        // TPM_Delegate_Manage does not return an entire family table entry,
46        // only the familyID, so there is no way for the Family object to know
47        // its verification count. So now you have to use TPM_Delegate_ReadTable
48        // to find the verification count in order to be able to specify it when
49        // you create the delegation.

50        FamilyTable table = new FamilyTable(tpmDriver);
51        int verificationCount = table.getFamilyVerificationCount(familyID);

52        // create delegate password for key 1

53        TPM_SECRET delAuth =
54            TPMToolsUtil.createTPM_SECRETFromPrefixedString("delegatepassword");
55        System.out.println("key1: delegate authData = " + delAuth);
56
57        // establish OSAP session using key1

58        TPMOSAPSession myOSAPSession=null;
59        myOSAPSession = new TPMOSAPSession( tpmDriver );

60        try
61        {
62            myOSAPSession.startSession( TPMConsts.TPM_ET_KEYHANDLE,
63                    keyHandle1, keyAuth1 );
64        }
65        catch ( TPMException e )
66        {
67            System.out.println("Error: cannot start OSAP session on key1.");
68            TPMToolsUtil.handleTPMException( e );
69        }

70        // create the delegate blob for key1
71        // delegate GetPubKey command
72
73        int delMask = TPMConsts.TPM_KEY_DELEGATE_GetPubKey;
74
75        TPM_Delegate_CreateKeyDelegationOutput delegationOutput = null;
76        TPM_Delegate_CreateKeyDelegation cmd =
77            new TPM_Delegate_CreateKeyDelegation( keyHandle1, delAuth, familyID,
78                    familyLabel, verificationCount, delMask );
79        try
80        {
81            delegationOutput = cmd.execute( myOSAPSession, false);
82        }
83        catch ( TPMException e )
84        {
85            System.out.println("Error: cannot create key delegation.");
```

```
86              familyEntry.invalidate(familyID, familyLabel);
87              TPMToolsUtil.handleTPMException( e );
88          }
89          System.out.println( "Delegate_CreateKeyDelegation: "
90                  + delegationOutput );

91          int delegateBlobSize = delegationOutput.getBlobSize();
92          byte[] delegateBlob = delegationOutput.getBlob();

93          // establish DSAP session using delegate blob and key handle of
94          // delegated key (key 1)

95          TPMDSAPSession myDSAPSession=null;
96          myDSAPSession = new TPMDSAPSession( tpmDriver );

97          try
98          {
99              myDSAPSession.startSession( TPMConsts.TPM_ET_DEL_KEY_BLOB,
100                     keyHandle1,
101                     delegateBlobSize, delegateBlob,
102                     delAuth);
103         }
104         catch ( TPMException e )
105         {
106             // this never appears to happen, you can start the session
107             // with any authorization. Subsequent commands will fail
108             // with TPM_AUTHFAIL, but the session exists. Clearly the
109             // shared secret used in the session validates NOTHING
110             // about the object you're using.
111             System.out.println("Error: cannot start DSAP Session");
112             TPMToolsUtil.handleTPMException( e );
113         }

114         // get the public key of the keyHandle, same as before but over DSAP session
115
116         TPM_GetPubKeyOutput pkOutput=null;
117         TPM_GetPubKey pk1 = new TPM_GetPubKey( keyHandle1 );
118         try
119         {
120             pkOutput = pk1.execute( myDSAPSession, true);
121         }
122         catch ( TPMException e )
123         {
124             System.out.println("Error: TPM_GetPubKey failed for delegated key.");
125             TPMToolsUtil.handleTPMException( e );
126         }
127         System.out.println( "PubKey1(DSAP): " + pkOutput.getPubKey() );

128         // try to get the public key of the 2nd key handle
129         // (that is not associated with the session)

130         TPM_GetPubKeyOutput pk2Output=null;
131         TPM_GetPubKey pk2 = new TPM_GetPubKey( keyHandle2 );
132         try
133         {
134             pk2Output = pk2.execute( myDSAPSession, false); // last one
135             System.out.println( "PubKey2(DSAP): " + pk2Output.getPubKey() );
136         }
137         catch ( TPMException e )
138         {
139             System.out.println("Error: cannot get TPM_GetPubKey for unrelated key");
140             TPMToolsUtil.handleTPMException( e );
141         }

142         // clean up
143
144         key1.evict();
145         key2.evict();
146         familyEntry.invalidate(familyID, familyLabel);

147         TPMUtilityFuncs.cleanupTPMDriver();
148     }
149 }
```

Appendix D

Files on the CD-ROM

Because of the length of the source code, complete source code is included on the CD-ROM only. Files on the CD-ROM include:

- Existing TPM/J source files that were modified:
  - TPMConsts.java
  - TPMConsts.java.diff.txt
- New source files added to TPM/J:
  - TPMDSAPSession.java
  - TPM_DSAP.java
  - TPM_DSAPOutput.java
  - TPM_Delegate_CreateKeyDelegation.java
  - TPM_Delegate_CreateKeyDelegationOutput.java
  - TPM_Delegate_Manage.java
  - TPM_Delegate_ManageOutput.java
  - TPM_Delegate_ReadTable.java
  - TPM_Delegate_ReadTableOutput.java
- Classes to support the Attack program
  - ClearFamilyTable.java
  - Family.java
  - FamilyTable.java
  - Key.java
- The Attack program
  - Attack.java
- The LaTeXand BibTeX files making up this report
  - report.bbl

  - report.bib
  - report.dvi
  - report.pdf
  - report.tex

- Class files and library to be able to run the attack
  - `attack.jar` includes TPM/J classes.
  - `bcprov-jdk15-131.jar` is required by TPM/J.
  - `IFXTPMJNIProxy.dll` is required for TPM access on Windows.
  - `run` is an example Korn shell script to run the program.

Appendix E

Running the code

Four classes in attack.jar can be run from the command line:

- **FamilyTable** lists the family table
- **Family** creates a family table entry
- **ClearFamilyTable** deletes all family table entries
- **Attack** runs attack on the TPM

`IFXTPMJNIProxy.dll` is required to access TPM functionality on Windows platforms. If your TPM requires any other library or DLL, it should be copied to the current directory or you can specify its location on the java command line:

```
java -Djava.library.path=<directory> <method>
```

where `<directory>` specifies the directory where the library is located and `<method>` is one of the four methods listed above.

The CLASSPATH variable must include both the Attack jar file and the Bouncy Castle jar file:

```
export CLASSPATH="./attack.jar:./bcprov-jdk15-131.jar"
```

or it can be specified on the java command line:

```
java -cp "./attack.jar:./bcprov-jdk15-131.jar" <method>
```

Be sure to take care to use the proper CLASSPATH delimiter (in either the variable definition or on the command line) for your platform. Windows platforms require semicolons (;) to distinguish from the colon (:) used in the drive letter. Unix and Linux platforms use a colon (:).

Index