CSC 495/680 Assignment 3

Due Monday, November 15

Objective: To experiment with and learn about key migration, both of regular keys and the newer certifiable migratable keys (CMKs) supported by version 1.2 TPMs.

Exercises: For this assignment, you are to write four programs which will enable you to create a migratable bind key and migrate it from one system to another — to show that this worked, you will encrypt (bind) some test data using this key on one system, and then show that you can decrypt it on two different systems. The migratable bind key will be a child of the non-migratable system key ("SK") on each system that it resides on.

Program 1: A program to create a migratable bind key. In your last assignment, you wrote a program to create a non-migratable bind key underneath SK, and then save that key blob to a file. For program 1, you should modify this program so that it creates a *migratable key* — if you didn't get this completed in the last assignment, you can use my solution to the last assignment, which is available in `/scratch/495/newkey-file.c`.

Your migratable key should use a simple but non-trivial migration auth secret (in other words, don't use the "well known secret"). You can either prompt the user to enter the migration secret, or you can simply use a hard-coded value. In order to set this up, you will need to make three TSS calls between when the key object is created and when you call `Tspi_Key_CreateKey` to actually create the key. When you create the key object, it does not have a migration policy associated with it — so you first need to create a migration policy. To do this, use `Tspi_Context_CreateObject` to make a policy object with subtype `TSS_POLICY_MIGRATION`. Next, use `Tspi_Policy_AssignToObject` to assign your new policy object to the key object. And then finally, use `Tspi_Policy_SetSecret` to set the secret in the migration policy — use `TSS_SECRET_MODE_PLAIN` for the secret mode. For details on how to use these functions, consult the handy-dandy man pages.

Program 2: A program to extract a systems "SK" to a file. In order to migrate a key, you need to have a "destination key." This can be the key of a migration authority, or the new parent of the key on the destination system. However, the important part is that it is a key from a *remote* system — not the one you are currently working on! In this assignment, you are to migrate your key to underneath the SK of a different system, so we need to export a copy of the public key to use in the migration process.

A fairly simple modification to the `newkey-file.c` program will do what you need

here — that program already loads the SK, and has code for extracting a key blob and writing it to a file. Just modify the program so that it doesn't create a new key, but rather just extracts the key blob from the SK and then writes that to a file. You could then, for example, run this on `labhost7` and extract the SK to save it in a file named something like `labhost7-sk.dat`. That file can then be used as the "destination key" if you create your migration blob on `labhost6`.

Program 3: A program to create a migration blob to start the migration process from the original system. This program should take the migratable key produced by Program 1, and the destination key saved by Program 2, and create a migration blob (both "blob" and "random" parts) so the migratable key can be moved to the system that produced the destination key file. The main function you will use for this is `Tspi_Key_CreateMigrationBlob`, although you will first have to create an authorization ticket (see below). Note that for our simple scenario you *could* use the `TSS_MS_REWRAP` scheme; however, to illustrate the more powerful migration capabilities, your program should use `TSS_MS_MIGRATE`.

Creating an authorization ticket using `Tspi_TPM_AuthorizeMigrationTicket` is a little tricky, because it is an owner-authorized command — that means you have to first get the TPM object and then set the usage policy for the TPM to the owner secret. What makes this difficult is that the systems in the lab all use an owner secret written in Unicode (as the TSS standard specifies), and Unicode is somewhat difficult to work with in C — fortunately, the Trousers library provides a function to make this easier. Unfortunately, it's not documented. To avoid unnecessary difficulty in figuring this out, I'll just give you the code:

```
char ownerSecret[] = "SPANOwner";

res = Tspi_Context_GetTpmObject(hContext, &hTPM);
checkres(res, "GetTpmObject");

res = Tspi_GetPolicyObject(hTPM, TSS_POLICY_USAGE, &hOwnerSecret);
checkres(res, "GetPolicyObject-TPM");

UINT32 ownerSecretLen = strlen(ownerSecret);
BYTE *unicodeSecret = Trspi_Native_To_UNICODE((BYTE *)ownerSecret,
                                              &ownerSecretLen);

res = Tspi_Policy_SetSecret(hOwnerSecret, TSS_SECRET_MODE_PLAIN,
                            ownerSecretLen-2,
                            (BYTE *)unicodeSecret);
checkres(res, "SetSecret-TPM");
```

Program 4: A program to convert a migration blob to install the migratable key on a new system. The final program to write will take the key blob and random parts from Program 3,

and install the key on the destination system using `Tspi_Key_ConvertMigrationBlob`. Once that function is called and you have converted the key, you get get the key blob from your newly-converted key, and save it to a file (again, the `newkey-file.c` program has code that helps out!).

Final activity: Test it! Assuming you've got all of the previous programs running properly, you can follow this sequence of operations to test that everything works together — I'll use `labhost6` as the source host and `labhost7` as the destination host, but you can use any two hosts that you'd like. I would like you to turn in a record showing your actions in this part — an easy way to do this is with the `script` program on the labhost machines.

Here is a sequence of operations for testing: First, run Program 2 on `labhost7` to save that systems SK to a file `labhost7-sk.dat`. Next, log in to `labhost6`, and use Program 1 to create a migratable bind key, and save it in a file named something like `labhost6-bindkey.dat`. Now use Program 3 on `labhost6` in order to create a pair of files (blob and random) with the previously-saved `labhost7-sk.dat` as the destination key. While you are still logged in to `labhost6`, create a simple (and short – under 20 bytes) test input file, and use the "bind" program from your previous assignment (or my solution) to encrypt this file. Call the encrypted file something like textttencrypted.dat. Now use "unbind" from the previous assignment to decrypt this with `labhost6-bindkey.dat` and make sure you get back the original data. Finally, log back in to labhost7, and use Program 4 to complete the migration process, and create a migrated copy of the bind key in a file named something like `labhost7-bindkey.dat`. Check that this file is a correctly migrated key by using "unbind" from the previous assignment and the `labhost7-bindkey.dat` to decrypt the same encrypted file that you created on `labhost6`.

If you could decrypt this encrypted file on both `labhost6` and `labhost7`, then congratulations! You have just migrated a TPM key!

For the exercises part of this assignment, email me the sourcecode for all 4 of your programs, and print out all the source code as well as the recorded interaction in the final activity to turn in during class.

Questions: To consider key management in some realistic settings, answer the following questions.

1. Consider the following scenario: You are working for a company, and the company's system administrator (or system administration group) controls the TPM Owner secret for all systems owned by the company — individual users like you do not get this secret (passphrase). The company's policy on key management is as follows: keys can be backed up to a company server, but can only be installed and used on a new system if the system administrator approves the new system. Keys cannot be migrated or copied directly from one system to another without the system administrator's approval.

   Describe clearly how this can be done with regular migratable keys. Give a full description of the process, starting with when the system is first bought by the company and

ownership is taken of the TPM. Describe how a key management system set up by the system administrator could work — who interacts at what times, who authorizes commands, etc. In addition to describing the steps involved, justify as clearly as you can how the security goals are enforced (e.g., why can't a user copy keys to a new system on their own?).

2. Consider the following scenario: You are a private individual with your own computer — you are both the owner of the system and the main user, so you know both the owner secret and all key authorizations (usage and migration auth secrets). You have bought a piece of commercial software from a company, and they would like to use a key on your system in parts of their software. However, to maintain the integrity of their software, they would like the key to be restricted so it is only present or usable on systems that you register with them — ideally this is just for a single system, since that is all you have licensed the software for, but to enable system upgrades and system replacements the company will authorize some additional systems in certain circumstances. So the goal is a key that is migratable, but only under conditions that this outside party (who doesn't know the system secrets) authorizes.

   Standard migratable keys cannot be used in this scenario: give a clear argument why not.

   This scenario *can* be implemented using CMKs — describe clearly how this can be done with CMKs. Give a full description of the process, starting with when the system is first bought and ownership is taken of the TPM. Describe how this CMK-based system could work — who interacts at what times, who authorizes commands, etc. In addition to describing the steps involved, justify as clearly as you can how the security goals are enforced (e.g., why can't a user copy keys to a new system on their own?).