The University of North Carolina at Greensboro — Handout 7
CSC 495/680: Trusted Computing — November 22, 2010
Prof. Stephen R. Tate

CSC 495/680 Assignment 4

Due Monday, December 6

General Information: The purpose of this assignment is to give students several ways to improve their class average. This assignment is optional, but if students choose to do one of these exercises then the grade will replace their lowest assignment grade. Students can elect to write a quality tutorial on some aspect of trusted computing, or can do a coding assignment.

Tutorial Writing Options: For this option, you can write a tutorial on any significant aspect of trusted computing — examples of topics are the following:

1. Key migration (TPM v1.1 style) — concepts and mechanics
2. CMK migration — concepts and mechanics
3. Identity management — creating, certifying, and using identity keys
4. Integrity measurements — PCRs, static vs. dynamic root of trust, etc.

I want to stress that these should be high-quality, professional style tutorials — they must be designed to teach, not just to show that you know the material. Tutorials should include high-quality pictures/diagrams, clear scenario examples, and tested code samples. These will be published on the web, and a common "style" file will be used for all tutorials (which will be provided to those choosing this option).

Only one student may write about any specific topic. Additional topics are possible, but must be approved by the instructor.

Coding Option: If you prefer to write code rather than a tutorial, you can extend the last assignment (on key migration) into a full-fledged key management and backup system, based on *keys stored in the TSS persistent storage*. Keys should be managed as follows: each user will create a migratable user root key below SK, and additional migratable keys will be created below this. Note that there is a small challenge to this: The labhost machines, by default, share a common filesystem, and hence the user persistent storage is the same on all the labhost machines — however, keys under SK can only work on the one machine they were created on. The easy solution to this is to have a different persistent storage database on each machine (the more difficult solution is to manage UUIDs very, very carefully), and code to use a distinct persistent storage database per machine will be provided to you on the labhost machines.

The precise details of how your system works are up to you (although you can, and probably should, run designs by me to make sure they satisfy the requirements), but you should provide at least the following capabilities:

- A program that sets up a machine for use — this should create a user root key and store it in persistent storage.
- A function which can be called by any program to create a new key and register it in user persistent storage. This should support any key type supported by TSS (sign, bind, etc.) — the key type will be provided by the application calling your function, and the job of your function is simply to create the key and register it. The process for assigning and managing UUIDs is up to you, but must be sensible.
- A program to create a backup archive for your keys. This will create a migration blob for the user root key for a given destination key (provided in a file, for example), and then bundle this in a file with *all* of the user's registered keys. If the user has created 100 keys, all 100 keys should be represented in this file. The random part of the migration blob should be saved in a separate file.
- A program which takes a backup archive created as just described, and a file with the user root's "random part" of the migration, and installs the entire key hierarchy on the current machine. You'll need to consider what to do if the current machine already has a user root key on that machine, or if UUIDs are duplicated.