LAB Assignment #2 for ECE 525

Description: Run HELP Enrollment Process

This lab is the first one in a sequence, where each will build on previous labs and culminate in the project for this course. The goal of these labs and projects is to have you implement an actual PUF-based authentication scheme that will authenticate your token (which is plugged into the UNM server or is in your possession if you chose to buy one) with a trusted authority (a server). I have provided a set of 19 hardware implementations of the HELP PUF, as FPGA bitstreams. Each implementation is identical in design but is placed in a different location on the PL side of the FPGA (see PUFs II(A) through II(J) screencasts). Therefore, each of these bitstream implementations represents a different token despite the fact that they are located on the same FPGA device. I will refer to the 19 implementations as *instances* in the following.

The first step of the process is to run enrollment, which is what you will do in this laboratory. For HELP, enrollment collects digitized timing data for a set of paths and stores the timing data in a separate file for each instance. The timing data can then be combined into one large file, called the *MasterDB.txt* file, and used for subsequent (in-field) authentications (PUF-based authentication will be covered in detail in the screencasts including the HELP protocol that you will use here).

I have also provided the C code that you will use to collect enrollment data for the 19 instances. The enrollment process involves running one C program on the server, called *verifier_enrollment*, and a second C program on the Zybo board, called *token_enrollment.elf*. The two programs use a network connection to communicate with each other. The verifier sends challenges to the token and stores timing data sent by the token to a data file. The *token_enrollment.elf* program coordinates with a PL-side implementation of the HELP algorithm. The VHDL code implementation of HELP programmed into the PL-side carries out clock strobing to measure path delays of paths defined within a combinational logic function, which is also embedded on the PL-side. The combinational logic function is a component of the AES encryption algorithm, and as we discussed in the HELP PUF screencast, serves as the source of Entropy for HELP. The digitized path delays are transferred through a GPIO interface between the PL and PS-sides on the FPGA. The token program running on the PS-side collects this data and transmits it across the network to the verifier.

The enrollment process is completely automated, i.e., once it is started, the timing data for all 19 instances is collected one-at-a-time and stored in a set of 19 data files on the verifier. The PL-side programming operation is carried out within the C program as each instance is enrolled. Therefore, you will not do any programming in this lab but rather will be exposed to the Xilinx SDK tool, networking and other components of the experimental testbed. The sequence of steps you need to follow are given below. A separate screencast, titled *SDKInstruction_HELPEnrollment*, gives details on using the Xilinx SDK tool. The remaining laboratories and project will build on what you do in this lab so I encourage you to use this lab to become familiar with SDK, linux and the networking environment.

0) Create a directory called lab2 and cd into it.

```
mkdir lab2
cd lab2
```

1) Download the following files from lab2 on UNM learn.

From the PROTOCOL directory on UNM learn to your laptop or UNM server in a directory called PROTOCOL:
common.h
common.c
compile.csh
construct_MasterDB.csh
KG_FU_TVChar_NumSeeds_15_optimalKEK_TVN_0.60_WID_1.10.txt
KG_FU_TVChar_NumSeeds_15_optimalKEK_TVN_0.60_WID_1.10_masks.txt
Makefile_VE
verifier_common.c
verifier_common.h
verifier_enrollment.c
README.txt

From the SDK directory on UNM learn to your laptop or UNM server in a directory called SDK:
common.h
common.c
token_common.c
token_common.h
token_enrollment.c

From the BITSTREAM directory on UNM learn to your laptop or UNM server in a directory called BITSTREAM:
*.bin (all 19 xxx.bin files)
design_1_wrapper.hdf

2) Compile the verifier code by running ‘compile.csh’ on your laptop or UNM server. This creates ‘verifier_enrollment’ which you will run below.

```
cd PROTOCOL
./compile.csh
```

3) You do NOT need to create a Vivado project because we are giving you the bitstreams for the board. You will need a hardware description of the project we built (when we created the bitstreams) to pass to sdk so it knows what the hardware platform looks like. The hardware platform is described in design_1_wrapper.hdf. You need to run the following command from the BITSTREAM directory:

```
xsdk -vmargs -Dorg.eclipse.swt.internal.gtk.cairoGraphics="false" &
```

Use the current BITSTREAM directory when prompted with ‘Select Workspace Directory’. Once SDK opens, under ‘File’, ‘New’, ‘other’, choose ‘Hardware Platform Specification’ under ‘Xilinx’, then browse to the ‘design_1_wrapper.hdf’ file under ‘Target Hardware Specification’. Select it and click ‘Finish’. This should add ‘design_1_wrapper_hw_platform_0’ to your sdk Project Explorer.

4) Under ‘File’, ‘New’, ‘Application’, use ‘token_enrollment’ as the Project name, select linux as OS Platform, and then empty application. This creates token_enrollment in the Project Explorer. Add the source files from your SDK directory to the token_enrollment application and build the application (see screencast).

5) The token_enrollment.elf file will be in the token_enrollment/Debug directory created under the BITSTREAM directory. Use scp to copy token_enrollment.elf file to the Zybo board using:

```
    cd BITSTREAMS/token_enrollment/Debug
    scp token_enrollment.elf root@192.168.1.10:/
```

Replace the .1. with the IP you have been assigned.

6) Copy all *.bin files in the BITSTREAM directory to the /tmp directory on your Zybo board.

```
    cd BITSTREAMS
    scp *.bin root@192.168.1.10:/tmp
```

7) Execute ‘verifier_enrollment’

```
    cd PROTOCOL
    ./verifier_enrollment 192.168.1.20
```

You MUST change the .1. to the IP you have been assigned.

NOTE: If after running ./verifier_enrollment command on your laptop or UNM server, you get the following message

```
    ERROR: OpenSocketServer(): failed to bind!
```

Wait 30 seconds and try again. Keep trying, it will eventually connect to the socket and print

```
    Waiting for incoming connections from clients for Enrollment ....
```

8) In a separate Xterm, ssh to the Zybo board and run token_enrollment.elf.

```
    ssh root@192.168.1.10 (type root for the password)
    cd /
    ./token_enrollment C1 192.168.1.20
```

Change the chip name from C1 to the number you have been assigned.

NOTE: If after running ./token_enrollment command on the Zybo board, you get the following message

```
    -sh: token_enrollment.elf: not found
```

Then you must change the compiler settings in SDK as follows:

```
    Right click on token_enrollment in the Project Explorer pane
    Select Properties (very bottom of drop-down menu)
    Expand C/C++ Build tab
    Select Settings
    Click on ARM v7 Linux gcc assembler
    Replace string in Command field with arm-xilinx-linux-gnueabi-gcc
    Click on ARM v7 Linux gcc compiler
    Replace string in Command field with arm-xilinx-linux-gnueabi-gcc
    Click on ARM v7 Linux gcc linker
    Replace string in Command field with arm-xilinx-linux-gnueabi-gcc
    Press OK
```

This will recompile your code with the correct compiler. Transfer the token_enrollment.elf file to the Zybo board, overwriting the version that produces the error:

```
    scp token_enrollment.elf root@192.168.1.10:/
```

Replace the .1. with the IP you have been assigned.

9) If all goes well, the token_enrollment.elf command will finish and a set of 19 data files file by the name ‘C1_V*_KG_FU_25C_1.00V_E_PUFNums.txt’ will be created on your laptop or UNM server in the PROTOCOL directory. This is the enrollment data for the 19 instances of the HELP PUF obtained from your Zybo board.

10) Edit construct_MasterDB.csh and replace the C1 with the chip number you have been assigned and run it.

```
    ./construct_MasterDB.csh
```

This script simply creates a MasterDB.txt file that is the concatenation of all the individual data files. You can delete the 19 individual ‘C1_V*_KG_FU_25C_1.00V_E_PUFNums.txt’ files.

NOTE: The instance of HELP that is programmed onto the Zybo board is the last one programmed when you ran enrollment. For labs beyond this one, you can reprogram the board with any of the 19 HELP instances by running the *program_FPGA.elf* program that I have provided in the BITSTREAMS directory for this lab. Download *program_FPGA.elf* from UNM learn, transfer it to your Zybo board and run it as follows:

```
    program_FPGA.elf 10
```

This programs the FPGA with the 10th bitstream from the /tmp directory.

NOTE: Do not attempt to program the FPGA using xsdk or vivado on the UNM server. I have not provided the .bit files that you would need for this, nor do you have permission to the usb programming ports. Please use the *program_FPGA.elf* process instead.