LAB Assignment #7 for ECE 525

Assigned: Tue., Mar. 23, 2017
Due: Thur., March. 30, 2017

Description: Add a Mechanism to Generate the HELP Parameters can carry out Token Authentication.

1) This lab adds to the code you created in all previous labs.

2) The following parameters exist in your existing code:

- LFSR_seed_low
- LFSR_seed_high
- Reference mean
- Reference range
- Modulus
- Margin

As a security measure, we will generate values for these parameters randomly, and both the verifier (laptop) and token (FPGA) will participate in selecting these parameters. This will be accomplished by generating nonces (random numbers) on both the token and verifier. Both the verifier and verifier will carry out the following operations:

```
int RANDOM;

if ( (RANDOM = open("/dev/urandom", O_RDONLY)) == -1 )
   {
   printf("ERROR: Could not open /dev/urandom\n");
   fflush(stdout);
   exit(EXIT_FAILURE);
   }
printf("Successfully open '/dev/urandom'\n");
```

Obtain nonces on both the token and verifier (call verifier nonce 'verifier_n2') using:

```
if ( read(RANDOM, token_n1, 8) == -1 )
   {
   printf("ERROR: GenNonceExchange(): Read /dev/urandom failed!\n");
   fflush(stdout);
   exit(EXIT_FAILURE);
   }
```

The 'token_n1' and 'verifier_n2' nonces should be defined as, e.g., 'unsigned char token_n1[8]'.

3) The verifier should send its nonce to the token. The token receives 'verifier_n2' nonce and computes an 'XOR_nonce' that is a bitwise XOR of the token and verifier 8 nonce bytes.

4) The XOR_nonce is then sent back to the verifier (all 8 bytes).

5) Both the token and verifier will use the following code to compute the parameters:

```
LFSR_seed_low = 0x000007FF & ((XOR_nonce[1] << 8) + XOR_nonce[0]);
LFSR_seed_high = 0x000007FF & ((XOR_nonce[3] << 8) + XOR_nonce[2]);
ref_mean = ((unsigned)(0x3F & XOR_nonce[4])) + mean_lower;
new_range = (0x1F & XOR_nonce[5]) + range_lower;
if ( (0x01 & XOR_nonce[6]) == 0 )
   Margin = 2;
else
   Margin = 3;

if ( (0x01 & XOR_nonce[6]) == 0 )
   *Modulus = 12 + ((unsigned)(0x07 & XOR_nonce[7]) << 1);
else
   *Modulus = 18 + ((unsigned)(0x03 & XOR_nonce[7]) << 1);
```

Use the following for 'mean_lower' and 'range_lower'

```
float mean_lower = -10.0;
float range_lower = 80.0;
```

6) You should ensure that these randomly generated parameters are used by your existing code. As mentioned, use the LFSR_seed_low as the seed in Offset operation.

7) Note that the above operations are carried out after the vectors are received but before ComputePNDiffs, TVCOMP, ComputeModulus are called.

8) The last lab required that you generate Helper Data and a Strong Bitstring on the token and verifier. As discussed in class, you can increase reliability significantly if you carry out the following:

- Generate Helper Data on both the token and verifier and have the token send its Helper Data and Strong Bitstring to the verifier.
- The verifier computes Helper Data using the stored enrollment data and bitwise 'AND's it with the token's Helper Data.
- It then generates its own Strong Bitstring using the AND'ed Helper Data.
- It then eliminates bits in the token's Strong Bitstring that are re-classified as weak after the two Helper Data bitstrings are AND'ed together.

Once the two Strong Bitstrings are created, they can be compared to see if they match.

Prepare to demo your code in class.