CMPUT 274 - Tangible Computing **Fall 2019**

Assignment #2: Encrypted Communication - Part 2

Category	Mark
1. `Random Number Generation` Every time you need a random number in the range $[2^k, 2^{k+1})$ for some k, you read the least significant bit of analog read k times with a slight delay to get k random bits. Then you add 2^k. **Deduction Examples** • Using the builtin random function (severe deduction). • Using more than 1 bit from each analog reading. • Not correctly constructing a number in the range..	/20
2. Generating Primes Generate random primes by getting random integers and checking they are prime. Getting a new random number or incrementing the current number (with wrap-around) if necessary. **Deduction Examples** • The numbers generated are not always primes. • Using a slow primality testing algorithm: $O(\sqrt{n})$ is fast enough.	/20
3. `Generating the Key Parts.` Correctly computes $n = p \cdot q$ for distinct primes p, q and randomly generates e and the associated d according to the specification. **Deduction Examples** • Incorrect calculations: the keys do not satisfy $e \cdot d \equiv 1 \bmod \phi(n)$ (or other issues). • Not generating e randomly according to the specification.	/30
4. `Handshaking` Shares the keys properly between the client and server. **Deduction Examples** • Fails to share keys, especially after reasonable "resets". • Does not handshake properly with the instructor's solution (or does not chat properly either).	/30

Note:
There are many aspects to this assignment. If you cannot complete one part, it is far better to just do something that works so you can have the remaining parts evaluated. For example, if you are not able to generate the keys randomly then you can just hard code some keys like in part 1 and mention this in your README. You would receive significant deductions, but we can still give some credit for other parts. If this is the case, please mention carefully what you did get working in your README.

Note: It should take only a fraction of a second to encrypt and decrypt a byte. Processing around 8 characters per second is acceptable. If you type really fast, you might notice a minor delay and that is ok. If you type really fast for a long time and overflow the Arduino's communication buffer, no problem (we won't try to enter more than 64 characters at a time into the buffer as long as you process characters fast enough).

Another Note: We will continue to avoid the use of the `break` command to break out of loops. Using it will result in a deduction.

Global Variables: The only global variables you should use are constants related to the wiring. You can use pass-by-reference to "return" multiple values from a function, this might be helpful in the handshaking (for example).

Setup Routine: You are not required to determine the client/server role in the `setup()` function. This can be done outside of `setup()`.

Further Notes: Additional deductions (to the overall grade) may be applied at the grader's discretion if the submission

- Does not compile when using our Arduino `Makefile`. Significant deductions apply if this happens, possibly resulting in a grade of 0 overall. You must submit code that can be compiled, even if it does not address all tasks. *Make sure your code compiles on the VM using our `Makefile` before you submit it!*
- Has extremely poor code structure, style, or modularity. Uses global variables apart from constants that do not change throughout the programs execution like pin numbers or the hard-coded key values.
- Does not have correct function names, function parameters, or file names for any one of these that is specified in the description. Remember, you are required to include a `README` file (no .txt extension to the filename) and to compress all files in your solution into either a single `.zip` file or a single `.tar.gz` file.

Style
Your submission must use good style. Generally speaking, good style rules from Python that are applicable to C++ still apply: comments must be placed appropriately, variable names must make sense, you should exercise good use of functions, etc. With C++, you also **must have proper indenting**. While a C++ program with bad indenting can still be compiled, you must indent properly.

Finally, your program must have a `main()` function that calls a `setup()` function to initialize things like the Arduino, the pin modes, serial communication, and anything else that is appropriate here.