CMPUT 274 - Tangible Computing **Fall 2019**

Assignment #2: Encrypted Communication - Part 1

Category	Mark
1. `Identifying Client/Server Role` When the Arduino starts, it reads pin 13 and correctly determines its role as a client or server. The correct keys are then identified. **Deduction Examples** • Determining the incorrect role. • Using the wrong pin number. • Using keys that are different than in the assignment specification.	/10
2. `mulmod` Correctly implementing mulmod using only 32-bit types (see the worksheet). **Deduction Examples** • Implicitly using 64-bit types (eg. just doing `(a*b)%m` anyway and trusting the compiler to use 64-bits for the multiplication). • Not using `% m` after every arithmetic step. **IMPORTANT**: You will get a 0 on this part if you just use 64-bit types and do not implement the algorithm that the worksheet on eClass helps you develop!	/30
3. `Reading and writing bytes.` Bytes are correctly read and written between `Serial` and `Serial3`, as necessary to perform the communication. **Deduction Examples** • Not printing the byte back to the serial monitor in addition to sending the encrypted data. • Not sending `\n` after sending a carriage return. • Sending the 4 bytes of a `uint32_t` across `Serial3` in the incorrect order (see the assignment description for code that sends/receives in the correct order).	/30
4. `Encryption / Decryption.` Encryption and decryption are performed correctly and efficiently using the fast exponentiation algorithm from the lectures. This fast exponentiation algorithm calls the `mulmod` function to perform the multiplication steps. **Deduction Examples** • Using slow versions of the algorithms. • Using the wrong keys. • Deviating from the specification in a way that causes your program to not be able to successfully chat with an Arduino running a correct solution.	/30

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 or to the specific key values from the assignment description (which are also constant values). To avoid a small deduction to the overall grade due to poor style, once you determine which keys to use, the variables n, e, d, m you store them in must be local to some function and passed to appropriate functions when encryption/decryption must be performed.

More precisely: if you put n, e, d, m in the global space, a 5% style deduction will be applied.

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.