Overview

The goal of this challenge is to write a small program that handles the parsing and verification of UDP packets, to demonstrate a small example of overall design architecture and code aesthetic.

Provided are a few files required for bootstrap as well as example input for testing purposes. Note that a variety of code inputs will be used to evaluate the efficacy of the program: the example provided should be used as a general test-case (but should work for any general variation of input).

Packet Structure

Below is an outline of the structure of an incoming packet. These incoming packets have been created with proper network byte ordering.

```
=================================================================

 ( 4 bytes ) Unique Packet ID for the checksummed binary
=================================================================

( 4 bytes ) Packet Sequence # (Total Checksums Processed)
=================================================================

 ( 2 bytes ) Multibyte Repeating XOR Key | ( 2 bytes ) # of Checksums
=================================================================

( Variable ) Repeating key XOR'd Cyclic Checksum CRC32 DWORDs
....
....
....

=================================================================

( 64 bytes ) RSA 512 SHA-256 Digital Signature (for above fields)
=================================================================
```

Objectives

Create a UDP server that handles the following criteria:

1. Verify the structural integrity of the packet
2. Verify the packet has a valid digital signature

   • Failing this, the server should write to a log file in the root directory of the project in a log file named

   verification_failures.log

3. Verify checksums are being sent correctly
   • Failing this, the server should write to a log file in the root directory of the project in a log file named

   checksum_failures.log

4. Introduce an artificial “delay” for writing to the log file, the duration will be passed as a command line argument (in seconds)

Log File Structure

For verification failures, the log format should follow the structure:

0x42 (Packet ID - in hex)
3703 (Packet sequence number) fd2bc562a95c4924d27f9f81de052fbab650f0c2989ee9f4e826244e7c1f0e66 (received hash) 26a4fcaa2167342136272e1d2814b7c73ac995e1229fea8bffa536600cc57921 (expected hash) \n (trailing newline)

For checksum failures, the log format should follow the structure:

0x42 (Packet ID - in hex)
1109 (Packet sequence number) 1119 (Cyclic checksum iteration) 2165e3dd (received crc32) 2165e24d (expected crc32)
\n (trailing newline)

Command Line Arguments

Several command line arguments will be passed at runtime, which your submission must handle:

• –keys: a dictionary of {packet_id: key_file_path} mappings
– ex: --keys '{"0x42": "key.bin", "0x1337": "super_secret_key.bin"}'

• –binaries: a dictionary of {packet_id: binary_path} mappings
– ex: --binaries '{"0x42": "cat.jpg", "0x1337": "kitten.jpg"}'

• -d: delay, (in seconds) for writing to log files – ex: -d '180'

• -p: port, to receive packets on – ex: -p '1337'

Included Files

- **send.py** - Will send prerecorded UDP packets to 127.0.0.1 on port 1337. First packet is guaranteed to be correct. Packet IDs are 0x42.
- **server.py** - The main entry point of your server.
- **payload_dump.bin** - Pickled raw pre-recorded packet data that is sent out (used for simulation testing).
- **key.bin** - Raw binary bytes of RSA 512 bit public key and exponent. Used to verify signature of incoming

packets.

- **cat.jpg** - The source binary used for reference against the cyclic checksum values. Corresponds to packet ID

  0x42.

- **verify.py** - Will validate that your server correctly logs the first checksum & verification failures.