Seriously Confidential. Dispatch received 2019-09-18 03:05 BST. From: [redacted].

To: Team 2

Cc: [redacted] , Cabinet Office, [redacted]

Due to the recent [redacted] we (SIS) require a clean-room implementation of an ESME capable of SMPP v3.4 communication.

We have 24 hours to deployment. It is of utmost importance that you implement ONLY what is specified in order to meet this timeline.

We must avoid detection, so all work MUST be completed within normal working hours. Any violation of this condition will result in [redacted] of you and any family members.

Order [redacted] mandates the use of “mob programming” techniques for tasks that fall under Section 13a of [redacted] 2013. The majority of your team’s time must be spent working together on a SINGLE computer. Team members must take up “typist”, “driver” and “advisor” roles, and these must rotate at least every hour. Ask your supervisor if you require more definition of these roles.

You MUST NOT discuss any aspect of your tasks with Team 1, at any time. Remember the [redacted] and [redacted]. Understand the consequences of violating these.

You may choose any technology (platform, programming language, available libraries) for this task, but the code must execute on your supervisor’s laptop at 2019-09-19 11:00 BST. You must write a command-line program that executes in a standard Linux Bash shell.

You MUST avoid using any library code that implements any telecoms-related code, most importantly anything directly related to SMPP, due to [redacted] , obviously.

You MUST implement unit tests* of all aspects of your code, and you must provide when asked an explanation of how your system can support comprehensive acceptance tests.

> ** Unit tests must be capable of executing on our off-grid system, which has no writeable storage or network connectivity.*

For definitive information about the SMPP v3.4 protocol, refer to https://www.openmarket.com/docs/Content/downloads/SMPP-v3-4-Issue1-2.pdf, but for informal understanding purposes we recommend referring to the OpenMarket links given below.

Your program should prompt for messages to send one at a time. Afer sending each message, it should read any responses, and then prompt again to send. There is no requirement for asynchronous operation.

Complete these tasks in the specified order. Each of these tasks has value, and the release of your ~~loved ones~~ compensation is linked to each task you complete. You are not expected to complete all tasks.

1. Start the program and connect to a TCP host and port (host and port supplied as command-line parameters).
2. Prompt the operator on the command line for a message to send (This will contain the [redacted]). Send a valid SUBMIT_SM PDU (similar to that described here: https://www.openmarket.com/docs/Content/apis/v4smpp/mt.htm). You must send a valid PDU, but the only important part for this task is the Short Message. You may assume the connection is bound: you are NOT required to send a BIND_TRANSMITTER message first – focus on SUBMIT_SM only. You only need to support Short Messages containing ASCII characters. Encode the short message using GSM 03.38 encoding, with one byte per character.
3. After submitting, wait for a SUBMIT_SM_RESP PDU (similar to that described at https://www.openmarket.com/docs/Content/apis/v4smpp/mt.htm#submitrespheader). Use this PDU to understand whether the message was received, and print this information to the command line. Keep hold of the Message ID supplied in the PDU.
4. Check for any DELIVER_SM PDU (similar to https://www.openmarket.com/docs/Content/apis/v4smpp/deliveryreceipt.htm). If the Short Message section of the PDU begins “id:XXX”, and the XXX part matches the Message ID from any previously-sent message, print a message to the command line, specifying the message, and whether or not it was delivered. Read the TLV with tag 0x2153 (type: short) to decide the message status. Do not use the Short Message. The TLV value will be 4 for success, or any other value (or missing TLV) for failure. If no PDU arrives within 3 seconds, stop waiting and prompt for the next message to send (it may come later though).
5. Implement optional bind logic (similar to https://www.openmarket.com/docs/Content/apis/v4smpp/bind.htm): if a System ID and Password are provided on the command line, begin by emitting a valid BIND_TRANSMITTER PDU, which contains the System ID and Password. Wait for a BIND_TRANSMITTER_RESP PDU indicating success before prompting the operator for a message.
6. [redacted]