ANSWERS

A7.10 Project: a secure sensor packet

Data representation · A level · OCR H446 1.3.1, AQA 7517 4.5.4.4, Eduqas A500QS 2.3 · about 40 min

BugBotLab

What this lesson is about

Pack a distance reading into fixed point bytes with flags and a checksum, encrypt it with a one-time pad and send it by radio.

- Designing the packet
- Encrypting it

Questions

6 marks in all

1. A byte stores a distance in unsigned fixed point with 7 bits before the binary point and 1 bit after. Give the 8 bits for 36.5 cm. [1 mark]

> **Answer:** 01001001. Doubling moves the point one place: 36.5 x 2 = 73, which is 01001001, or 0100100.1 with the point shown.

2. For that format (unsigned, 7 bits before the point and 1 bit after), what are the range and precision? [1 mark]

- ☐ **A** 0 to 127.5, in steps of 0.5
- ☐ **B** 0 to 255, in steps of 1
- ☐ **C** -64 to 63.5, in steps of 0.5
- ☐ **D** 0 to 127, in steps of 0.1

> **Answer:** A. The largest pattern, 11111111, is 127.5 and the smallest place value is one half.

3. Which operation tests whether bit 0 of the flags byte is set? [1 mark]

- ☐ **A** AND with 00000001, then check the result is not zero
- ☐ **B** OR with 00000001
- ☐ **C** XOR with 00000001
- ☐ **D** Shift left by 1

> **Answer:** A. AND with a mask keeps only the bit being tested. OR would set it and XOR would flip it.

4. What does this program print? [1 mark]

```
packet = [0x44, 0x10, 0x81]
check = sum(packet) % 256
print(format(check, "02X"))
```

> **Answer:**
>
> ```
> D5
> ```
>
> 68 + 16 + 129 = 213, which is D5 in hex and fits in one byte.

5. What is the denary value of 01000100 XOR 00111010? [1 mark]

> **Answer:** 126. XOR gives 1 where the bits differ: 01111110, which is 126.

6. The receiver decrypts the packet. Why should it check the checksum before using the reading? [1 mark]

- ☐ **A** Interference may have changed bits, and a wrong reading could make the robot act wrongly
- ☐ **B** Decryption always introduces errors
- ☐ **C** The checksum contains the key
- ☐ **D** The reading cannot be read until the checksum is removed

> **Answer:** A. Encryption keeps the data private but does not detect corruption. The checksum detects most corrupted packets so they can be thrown away.

The task: a secure sensor packet

The robot faces a wall. `PAD` is a list of four whole numbers from 0 to 255. 1. Read the sensor once with `distance()`, a reading in cm (a float, from 0 to 127.5 on this mat). 2. Build the four packet bytes as whole numbers from 0 to 255: the type byte `ord("D")`; the reading byte, the reading doubled with its fraction dropped (`int`); the flags byte with bit 7 always set and bit 0 set only when the reading is under 30 cm, built with shifts and OR; and the checksum, the sum of the first three bytes modulo 256. 3. Print `packet:` followed by the four bytes as two uppercase hex digits each, separated by single spaces. 4. Encrypt the packet by XORing each byte with the `PAD` byte in the same position, and print `encrypted:` followed by the four encrypted bytes in the same format. 5. Send `PKT` followed by the encrypted bytes as eight hex digits with no spaces, built by your program. 6. Now be the receiver, using only the text you sent: turn each pair of hex digits back into a byte, XOR with `PAD`, and check that the checksum matches and bit 7 of the flags is set. If both are true, print `received: <cm> cm, checksum ok`, where `<cm>` is the reading byte divided by 2 (for example `51.0`), and set the LED to green. Otherwise print `received: checksum failed` and set the LED to red. Build every byte from the reading; do not type any of the packet in.

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()

PAD = [0x3A, 0x91, 0x5C, 0xE7]

def hex_bytes(data, sep):
    return sep.join(format(b, "02X") for b in data)

d = distance()
```

The hint students can ask for: Build the four bytes one at a time: the type letter's code, the reading doubled to keep one fraction bit, the flags from shifted 1s ORed together, and the checksum of the first three. XOR with the pad, send, then do the whole thing in reverse as the receiver would and check the checksum before you trust the reading.

A solution

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()

PAD = [0x3A, 0x91, 0x5C, 0xE7]

def hex_bytes(data, sep):
    return sep.join(format(b, "02X") for b in data)

def vernam(data, key):
    return [data[i] ^ key[i] for i in range(len(data))]

d = distance()
kind = ord("D")
reading = int(d * 2)
flags = 1 << 7
if d < 30:
    flags = flags | (1 << 0)
check = (kind + reading + flags) % 256
packet = [kind, reading, flags, check]
print("packet:", hex_bytes(packet, " "))
```

```
secret = vernam(packet, PAD)
print("encrypted:", hex_bytes(secret, " "))
text = "PKT " + hex_bytes(secret, "")
send(text)

digits = text.split(" ")[1]
arrived = [int(digits[i:i + 2], 16) for i in range(0, len(digits), 2)]
back = vernam(arrived, PAD)
if sum(back[:3]) % 256 == back[3] and back[2] & (1 << 7):
    print("received:", back[1] / 2, "cm, checksum ok")
    led("green")
else:
    print("received: checksum failed")
    led("red")
```

Any program that meets the task's checks is marked correct in the simulator; this is one way, not the only way.