Data Integrity and Protection

Annalise Tarhan

March 18, 2021

1 First run checksum.py with no arguments. Compute the additive, XOR-based, and Fletcher checksums.

Seed 0
Decimal: 216 194 107 66
Binary: 0b11011000 0b11000010 0b01101011 0b01000010

Additive
(216+194+107+66) % 256
583 % 256
71 = 0b1000111

Xor
0b11011000 $\oplus$ 0b11000010 $\oplus$ 0b01101011 $\oplus$ 0b01000010
0b00011010 $\oplus$ 0b00101001
0b00110011 = 51

Fletcher
s1 = (0+216) % 255 = 216
s2 = (0+216) % 255 = 216
s1 = (216+194) % 255 = 155
s2 = (216+155) % 255 = 116
s1 = (155+107) % 255 = 7
s2 = (116+7) % 255 = 123
final s1 = (7+66) % 255 = 73
final s2 = (123+73) % 255 = 196
(0b01001001, 0b11000100)

2 Now do the same, but vary the seed.

Seed 1
Decimal: 34 216 195 65
Binary: 0b00100010 0b11011000 0b11000011 0b01000001

Additive
(34+216+195+65) % 256
510 % 256
254 = 0b11111110

Xor
0b00100010 $\oplus$ 0b11011000 $\oplus$ 0b11000011 $\oplus$ 0b01000001
0b11111010 $\oplus$ 0b10000010
0b01111000 = 120

Fletcher
s1 = (0+34) % 255 = 34
s2 = (0+34) % 255 = 34
s1 = (34+216) % 255 = 250
s2 = (34+250) % 255 = 29
s1 = (250+195) % 255 = 190
s2 = (29+190) % 255 = 219
final s1 = (190+65) % 255 = 0
final s2 = (219+0) % 255 = 219
(0b00000000, 0b11011011)

3 Can you pass in a 4-byte data value that does not contain only zeroes and leads the additive and XOR-based checksums having the same value? In general, when does this occur?

Decimal: 138 133 160 80
Binary: 0b10001010 0b10000101 0b10100000 0b01010000
This always occurs when none of the inputs have ones in the same place. There are some inputs, including this one, where three of the numbers have a one in the same place, but because it is the highest digit, the carryover in the addition is modded off.

4 Now pass in a 4-byte value that you know will produce different checksum values for additive and XOR. In general, when does this occur?

Decimal: 15 15 204 51
Binary: 0b00001111 0b00001111 0b11001100 0b00110011
The values will usually be different as long as some of the inputs have a one in the same place. There are exceptions, as in the previous problem.

5 Use the simulator to compute checksums twice, once each for a different set of numbers. The two number strings should be different but should produce the same additive checksum. In general, when will the additive checksum be the same, even though the data values are different?

100,234,182,5
75,23,11,156
They will be the same when the sums of the inputs are the same modulo 256.

6 Now do the same for the XOR checksum.

82,252,214,25
174,0,109,162
The xor result is the same for 0,0 and 1,1, as well as for 0,1 and 1,0, so either pair can be substituted for the other without affecting the checksum.

7 Now look at the following sets of data values. The first is -D 1,2,3,4. What will the different checksums be for this data? Now compare it to computing these three checksums over -D 4,3,2,1. What do you notice about these three checksums? How does Fletcher compare to the other two? How is Fletcher generally "better" than something like the simple additive checksum?

The additive and xor checksums are the same for both, 10 and 4 respectively. The Fletcher checksum differentiates, giving 10,20 for the first set and 10,30 for the second. Based on these results, Fletcher is the only checksum of the three that changes based on the order of the data it is given.

8 Given a particular input of your choosing, can you find other data values that lead to the same Fletcher checksum? When, in general, does this occur? Start with a single data string and see if you can replace one of those numbers but end up with the same Fletcher checksum.

Adding 255 to any of the input values yields the same Fletcher checksum, since all of the calculations are modulo 255.