HW07 – ECE 404

Input text:

The phony war is over and it will soon be time to discover who's hot and who's not on the 2023 Formula 1 grid. Red Bull ended last season in dominant shape, winning all bar one of the grand prix in the second half of the 22-round championship. Because of that - and their 2021 budget cap breach - they have less time to spend on developing their RB19. Will that allow Ferrari and Mercedes to reduce their advantage?

Output text:

5b11ec306b005aa885c0fb9c7c286caf9e261538495944b9550d8698aeea61f552ad85c564210088bd3f2566
9c89da2fdd79ee8024f1eb8d1c0bffe948637191

Brief Explanation of Code:

This homework requires us to use the BitVector module to create an implementation of the SHA512 algorithm. To begin with, we use 8 64-bit words to initialize the 1024-bit hash buffer before we start scanning the input message block for its hashing. Then we use round constants (also known as K constants) to perform round based processing of each 1024-bit input message block. There is a 64-bit constant for each of the 80 rounds. These constants can be stored as an array of BitVector objects.

Primarily, the SHA512 algorithm can be broken down into four major steps. The first step involves padding the message so that its length is an integer multiple of the block size which is 1024 bits. This padding accounts for the fact that the last 128 bits of the padded input must store the length of the input message. Then we initialize an array of 'words' to store the message schedule for a block of the input message.

The next step would be to create a message schedule for the 1024-bit input block. This message schedule contains 80 words, each 64-bits long. The first 16 words of the message schedule are obtained directly from the 1024-bit input block. Now we need to expand the first 16 64-bit words of the message schedule into a full schedule that contains 128 64-bit words. Before we can start Step 3, we need to store the hash buffer contents obtained from the previous input message block in the variables a, b, c, d, e, f, g, h.

In Step 3, we carry out a round-based processing of a 1024-bit input message block. There are a total of 80 rounds and the calculations carried out in each round are referred to as calculating a "round function". The round function for the i-th round consists of permuting the previously calculated contents of the hash buffer registers as stored in the temporary variables a, b, c, d, e, f, g and replacing the values of two of these variables with values that depend of the i-th word in the message schedule, words[i], and i-th round constant, K[i].

Lastly, for Step 4, the values in the temporary variables a, b, c, d, e, f, g, h after 80 rounds of processing are mixed with the contents of the hash buffer as calculated for the previous block of the input message. Now, we simple concatenate the contents of the hash buffer to obtain a 1024-element BitVector object, and obtain the hex representation of the binary hash value, which is then written to an output file.