const *crypto* = require("crypto");

//Algorithm implementation

function theHash(*B*, *H*, *K*, *Text*) {

  const Blength = *B*;

  const KBuffer = Buffer.from(*K*, "hex");

  // Step 1

  let Ko;

  if (KBuffer.length === Blength) {

    Ko = KBuffer;

  } else if (KBuffer.length > Blength) {

    // Step 2

    const hash = *crypto*.createHash(*H*);

    const hashedK = hash.update(KBuffer).digest();

    const zeros = Buffer.alloc(Blength - KBuffer.length, 0x00);

    Ko = Buffer.concat([hashedK, zeros], Blength);

  } else {

    // Step 3

    const zeros = Buffer.alloc(Blength - KBuffer.length, 0x00);

    Ko = Buffer.concat([KBuffer, zeros], Blength);

  }

  const ipad = Buffer.alloc(Blength, 0x36);

  const opad = Buffer.alloc(Blength, 0x5c);

  // Step 4

  const step4Result = Buffer.from(Ko.map((*byte*, *index*) => *byte* ^ ipad[*index*]));

  // Step 5

  const step5Result = Buffer.concat([step4Result, Buffer.from(*Text*, "hex")]);

  // Step 6

  const hashStep6 = *crypto*.createHash(*H*);

  const step6Result = hashStep6.update(step5Result).digest();

  // Step 7

  const step7Result = Buffer.from(Ko.map((*byte*, *index*) => *byte* ^ opad[*index*]));

  // Step 8

  const step8Result = Buffer.concat([step7Result, step6Result]);

  // Step 9

  const hashStep9 = *crypto*.createHash(*H*);

  const step9Result = hashStep9.update(step8Result).digest();

  return step9Result;

}

// Test the function

const B = 64; // Block size in bytes

const H = "sha256"; // Approved hash function

const K = "0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b"; // Hexadecimal key

const Text = "4869205468657265"; // Data to be hashed

const result = theHash(B, H, K, Text);

console.log(result.toString("hex"));