```
// Key generation function
function generateKeys() {
  let p = generateLargePrime();
  let q = generateLargePrime();
  let n = p * q;
  let W = generateSuperIncreasingSequence(n);
  let b = (n / gcd(W)) + 1;
  let a = generateCoprime(n);
  return {
    publicKey: [n, W],
    privateKey: [p, q, a]
  };
}

// Encryption function
function encrypt(message, publicKey) {
  let n = publicKey[0];
  let W = publicKey[1];
  let binaryMessage = messageToBinary(message);
  let C = 0;
  for (let i = 0; i < binaryMessage.length; i++) {
    C += binaryMessage[i] * W[i];
  }
  return C % n;
}

// Decryption function
function decrypt(encryptedMessage, privateKey) {
  let p = privateKey[0];
  let q = privateKey[1];
  let a = privateKey[2];
  let M = (encryptedMessage ** (a ** (p - 1))) % (p * q);
  return binaryToMessage(M);
}

// Helper functions
function generateLargePrime() {
  // Code to generate a large prime number
}

function generateSuperIncreasingSequence(n) {
  // Code to generate a super-increasing sequence of integers
}

function gcd(arr) {
  // Code to compute the greatest common divisor of an array of integers
}

function generateCoprime(n) {
  // Code to generate a number that is coprime with n
}

function messageToBinary(message) {
  // Code to convert a message to a binary representation
}

function binaryToMessage(binary) {
  // Code to convert a binary representation to a message
}
```