“use strict”;

/\*\*\*\*\*\*\*\*\* External Imports \*\*\*\*\*\*\*\*/

Const { stringToBuffer, bufferToString, encodeBuffer, decodeBuffer, getRandomBytes } = require(“./lib”);

Const { subtle } = require(‘crypto’).webcrypto;

/\*\*\*\*\*\*\*\*\* Constants \*\*\*\*\*\*\*\*/

Const PBKDF2\_ITERATIONS = 100000; // number of iterations for PBKDF2 algorithm

Const MAX\_PASSWORD\_LENGTH = 64; // we can assume no password is longer than this many characters

/\*\*\*\*\*\*\*\*\* Implementation \*\*\*\*\*\*\*\*/

Class Keychain {

Constructor() {

This.data = {}; // Store member variables that you intend to be public here

This.secrets = {}; // Store member variables that you intend to be private here

}

Static async init(password) {

Const keychain = new Keychain();

Keychain.secrets.masterKey = await keychain.deriveKey(password);

Return keychain;

}

Async deriveKey(password) {

Const salt = getRandomBytes(16);

Const keyMaterial = await subtle.importKey(

‘raw’,

stringToBuffer(password),

{ name: ‘PBKDF2’ },

False,

[‘deriveKey’]

);

Const key = await subtle.deriveKey(

{

Name: ‘PBKDF2’,

Salt: salt,

Iterations: PBKDF2\_ITERATIONS,

Hash: ‘SHA-256’

},

keyMaterial,

{ name: ‘AES-GCM’, length: 256 },

False,

[‘encrypt’, ‘decrypt’]

);

This.secrets.salt = encodeBuffer(salt);

Return key;

}

Async set(name, value) {

Const encodedName = encodeBuffer(stringToBuffer(name));

Const encodedValue = encodeBuffer(stringToBuffer(value));

This.data[encodedName] = encodedValue;

}

Async get(name) {

Const encodedName = encodeBuffer(stringToBuffer(name));

Const encodedValue = this.data[encodedName];

If (encodedValue) {

Return bufferToString(decodeBuffer(encodedValue));

}

Return null;

}

Async remove(name) {

Const encodedName = encodeBuffer(stringToBuffer(name));

If (encodedName in this.data) {

Delete this.data[encodedName];

Return true;

}

Return false;

}

Async dump() {

Const kvs = { …this.data }; // Include the KVS object in the dump

Const dataString = JSON.stringify({ kvs });

Const dataBuffer = stringToBuffer(dataString);

Const hashBuffer = await subtle.digest(‘SHA-256’, dataBuffer);

Const hashString = encodeBuffer(hashBuffer);

Return [dataString, hashString];

}

Static async load(password, repr, trustedDataCheck) {

Const parsedData = JSON.parse(repr);

If (trustedDataCheck) {

Const dataBuffer = stringToBuffer(repr);

Const hashBuffer = await subtle.digest(‘SHA-256’, dataBuffer);

Const hashString = encodeBuffer(hashBuffer);

If (hashString !== trustedDataCheck) {

Throw new Error(“Integrity check failed”);

}

}

Const keychain = new Keychain();

Try {

// Attempt to derive the master key using the provided password

Keychain.secrets.masterKey = await keychain.deriveKey(password);

} catch (error) {

Return false; // If an error is caught, return false indicating invalid password

}

Keychain.data = parsedData.kvs; // Restore the KVS object from the parsed data

Return keychain;

}

}

Module.exports = { Keychain };