МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ  
Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ   
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационных систем и технологий

Специальность 1—98 01 03 «Программное обеспечение информационной

безопасности мобильных систем»

**Отчет**

по дисциплине «Основы информационной безопасности»

**Исполнитель**

студент(ка) 2 курса 7 группы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Банкузов М.О.

подпись, дата

**Руководитель**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ржеутская Н. В.

должность, ученая степень, ученое звание подпись, дата

**Практическое задание № 9**

**Тема «Изучение стандартных средств для реализации симметричного и ассиметричного шифрование с использованием SubtleCrypto в js»**

Цель: Изучить интерфейс **SubtleCrypto**, Основные классы и структуры данных, разработать приложение для шифрования файлов использующих симметричные и ассиметричные алгоритмы шифрования

**Криптография в Js**

Web Crypto API - это интерфейс, позволяющий использовать криптографические примитивы для построения систем с использованием криптографии. Данный интерфейсвключают в себя возможность генерировать, использовать и применять пары криптографических ключей; шифровать и дешифровать сообщения; надежно генерировать случайные числа

Некоторые браузеры реализовали интерфейс под названием Crypto, без точной структуры. Чтобы избежать путаницы, методы и свойства этого интерфейса были удалены из браузеров, реализующих Web Crypto API, и все методы Web Crypto API доступны в новом интерфейсе: SubtleCrypto.

Интерфейс SubtleCrypto Web Crypto API предоставляет ряд низкоуровневых криптографических функций. Доступ к функциям SubtleCrypto осуществляется через объёкт Crypto.subtle .

Этот объект содержит набор методов для выполнения общих криптографических функций, таких как шифрование, хеширование, подписывание и генерация ключей. Поскольку все криптографические операции выполняются с необработанными двоичными данными, каждый метод SubtleCrypto имеет дело с типами ArrayBuffer и ArrayBufferView. Из-за того, что строки так часто становятся предметом криптографических операций, классы TextEncoder и TextDecoder будут часто использоваться вместе с SubtleCrypto для преобразования в строки и обратно.

Одной из проблем криптографии является генерация случайных чисел. Если будет использоватся Math.random() то будет сгенерировано псевдослучайное число которое использует генератор PRNG (pseudorandom number generator). Поскольку сгенерированное число внутреннее состояние PRNG то использование данного алгоритма для криптографии неприемлемо. Для решения данной задачи решить эту проблему, криптографически безопасный генератор псевдослучайных чисел (cryptographically secure pseudorandom number generator, CSPRNG) дополнительно включает в себя источник энтропии качестве входных данных, такие системных свойств, которые проявляют непредсказуемое поведение. Это алгоритм медленнее PRNG, но значения, генерируемые CSPRNG, являются остаточно непредсказуемыми для криптографических целей. Код для генерации несколких случайных чисел можно увидеть в листинге 1.

const array = new Uint8Array(1);

for (let i=0; i<5; ++i) {

console.log(crypto.getRandomValues(array));

}

Листинг 1 – Код генерации случайных чисел

**Хеширование**

Хеширование в SubtleCrypto представлено следующими алгоритмами.

* Sha1
* Sha256
* Sha385
* Sha512

Метод SubtleCrypto.digest() используется для создания хеша сообщения. Ниже представлен пример для хеширования сообщения представлен в листинге 2.

const text = 'Hash using sha256';

async function digestMessage(message) {

const encoder = new TextEncoder();

const data = encoder.encode(message);

const hash = await crypto.subtle.digest(**'SHA-256'**, data);

return hash;

}

digestMessage(text)

.then((digestBuffer) => console.log(digestBuffer.byteLength));

Листинг 2 – Пример хеширования данных

Как видно из примера выше то для указания алгоритма хеширования используется первый параметр. Также стоит упомянуть что результат хеширования обычно используется в 16-ой форме. Для преобразования стоит добавить код представленный в листинге 3.

const hashArray = Array.from(new Uint8Array(hash));

const hashHex = hashArray.map((b) => b.toString(16).padStart(2, '0')).join('');

return hashHex;

Листинг 3 – Пример преобразование хеша в 16-ричную форму.

**Генерация ключей**

Генерация случайного CryptoKey выполняется с помощью метода SubtleCrypto.generateKey(algorithm, extractable, keyUsages). В этот метод передается объект params, указывающий целевой алгоритм, логическое значение,указывающее, должен ли ключ извлекаться из объекта CryptoKey, и массив строк — keyUsages, указывающий, с какими методами

Поскольку разные алгоритмы используют разный набор данных для ключей то в первый параметр содержит сооствутсвующее название алгоритма.

* RSA (RSASSA-PKCS1-v1\_5, RSA-PSS, or RSA-OAEP )использует объект RsaHashedKeyGenParams.
* ECDSA и ECDHи спользует объект EcKeyGenParams.
* HMAC использует объект HmacKeyGenParams.
* AES (AES-CTR, AES-CBC, AES-GCM, AES-KW) использует объект AesKeyGenParams

Значение extractable является логическим значением и указывает на возможность экспорта ключа.

Третий параметр keyUsages описывает, с какими алгоритмами можно использовать ключ.

* encrypt: Ключ используется для шифрования сообщений.
* decrypt: Ключ используется для расшифровки сообщений.
* sign: Ключ используется для подписи сообщений.
* verify: Ключ используется для проверки подписанного сообщения.
* deriveKey: Ключ используется для получения ключа.
* deriveBits: Ключ используется для получения битов.
* wrapKey: Ключ используется для упаковки ключа
* unwrapKey: Ключ используется для распоковки ключа.

Пример генерация ключа представлен в листинге 4.

(async function() {

const params = {

name: 'AES-CTR',

length: 128

};

const keyUsages = ['encrypt', 'decrypt'];

const key = await crypto.subtle.generateKey(params, false, keyUsages);

console.log(key);

})();

Листинг 4 – Пример генерации ключей.

**Шифрование и дешифрование**

Объект SubtleCrypto позволяет использовать как открытый ключ, так и симметричные алгоритмы для шифрования и дешифрования сообщений. Это может быть выполнено с использованием методов SubtleCrypto.encrypt() и SubtleCrypto.decrypt() соответственно. Ниже представлена часть кода для шифрования и дешифрования данных где algoIdentifier это название алгоритма и представлена в листинге 5.

const originalPlaintext = (new TextEncoder()).encode('Crypto');

const encryptDecryptParams = {

name: algoIdentifier,

iv: crypto.getRandomValues(new Uint8Array(16))

};

const ciphertext = await crypto.subtle.encrypt(encryptDecryptParams, key,

originalPlaintext);

console.log(ciphertext);

// ArrayBuffer(32) {}

const decryptedPlaintext = await crypto.subtle.decrypt(encryptDecryptParams,

key, ciphertext);

console.log((new TextDecoder()).decode(decryptedPlaintext));

Листинг 5 – Часть кода для шифрования и дешифрования.

**Создание цифровой подписи и проверка сообщений**

Объект SubtleCrypto позволяет использовать алгоритмы с открытым ключом для генерации подписей с использованием закрытого ключа или для проверки подписей с использованием открытого ключа. Они выполняются с использованием методов SubtleCrypto.sign() и SubtleCrypto.verify() соответственно. Для подписания сообщения требуется объект params, чтобы указать алгоритм и любые необходимые значения, частный CryptoKey и ArrayBuffer или ArrayBufferView для подписи. В примере, представленном в листинге 6 можно увидеть процесс создания цифровой подписи и проверки сообщения с цифровой подписью.

async function() {

const keyParams = {

name: 'ECDSA',

namedCurve: 'P-256'

};

const keyUsages = ['sign', 'verify'];

const {publicKey, privateKey} = await crypto.subtle.generateKey(keyParams,

true, keyUsages);

const message = (new TextEncoder()).encode('Mes to sign');

const signParams = {

name: 'ECDSA',

hash: 'SHA-256'

};

const signature = await crypto.subtle.sign(signParams, privateKey, message);

const verified = await crypto.subtle.verify(signParams, publicKey, signature,

message);

console.log(verified); // true

})();

Листинг 6 – Пример создания и проверки сообщений с цифровой подписью.

**Упаковка и распаковка ключа**

Объект SubtleCrypto позволяет упаковывать и распаковывать ключи, чтобы обеспечить передачу по ненадежному каналу. Это выполняется с использованием методов SubtleCrypto.wrapKey() и SubtleCrypto.unwrapKey() соответственно. Для переноса ключа требуется строка форматирования, экземпляр CryptoKey для переноса, CryptoKey для выполнения переноса и объект params для указания алгоритма и любых необходимых значений. В примере представленном в листинге 7 симметричный ключ AES-GCM, упаковывается с помощью AES-KW и, распаковывается обратно.

(async function() {

const keyFormat = 'raw';

const extractable = true;

const wrappingKeyAlgoIdentifier = 'AES-KW';

const wrappingKeyUsages = ['wrapKey', 'unwrapKey'];

const wrappingKeyParams = {

name: wrappingKeyAlgoIdentifier,

length: 256};

const keyAlgoIdentifier = 'AES-GCM';

const keyUsages = ['encrypt'];

const keyParams = {

name: keyAlgoIdentifier,

length: 256};

const wrappingKey = await crypto.subtle.generateKey(wrappingKeyParams,

extractable, wrappingKeyUsages);

console.log(wrappingKey);

const key = await crypto.subtle.generateKey(keyParams, extractable, keyUsages);

console.log(key);

const wrappedKey = await crypto.subtle.wrapKey(keyFormat, key, wrappingKey,

wrappingKeyAlgoIdentifier);

console.log(wrappedKey);

const unwrappedKey = await crypto.subtle.unwrapKey(keyFormat, wrappedKey,

wrappingKey, wrappingKeyParams, keyParams, extractable, keyUsages);

console.log(unwrappedKey);

})()

Листинг 7 – Пример упаковки и распаковки ключа.

1. Ознакомиться с созданием криптографического приложения
2. Выполнить генерацию и вывод в консоль случайный чисел.
3. Выполнить шифрование, дешифрование и хеширование своей фамилии по указанным алгоритмам.

|  |  |  |
| --- | --- | --- |
| 2 | AES-CTR | SHA-256 |

1. Продемонстрировать упаковку и распаковку ключа, полученного в предыдущем задании используя алгоритм AES-KW.
2. Выполнить процедуру подписи сообщения и проверку подлинности с использованием RSA-PSS или ECDSA на выбор.

const crypto = require("crypto");  
const array = new Uint8Array(1);  
  
for (let i = 0; i < 5; ++i) {  
 crypto.*getRandomValues*(array);  
 console.log(array[0]);  
}  
console.log("---------------------------------------------------------------");  
  
function deriveKey(*password*, *salt*, *keyLength*) {  
 return crypto.*pbkdf2Sync*(*password*, *salt*, 100000, *keyLength*, 'sha256');  
}  
  
const encryptionKey = deriveKey('myEncryptionKey', 'mySalt', 32);  
  
*// Функция для упаковки ключа с использованием AES-KW*function wrapKey(*key*, *wrappingKey*) {  
 const cipher = crypto.*createCipheriv*('aes-256-ecb', *wrappingKey*, '');  
 const wrappedKey = Buffer.concat([cipher.update(*key*), cipher.final()]);  
 return wrappedKey;  
}  
  
*// Функция для распаковки ключа с использованием AES-KW*function unwrapKey(*wrappedKey*, *wrappingKey*) {  
 const decipher = crypto.*createDecipheriv*('aes-256-ecb', *wrappingKey*, '');  
 const unwrappedKey = Buffer.concat([decipher.update(*wrappedKey*), decipher.final()]);  
 return unwrappedKey;  
}  
  
*// Функция для шифрования строки с использованием AES-CTR*function encryptAESCTR(*text*, *key*) {  
 const iv = crypto.*randomBytes*(16); *// Генерация случайного инициализационного вектора* const cipher = crypto.*createCipheriv*('aes-256-ctr', *key*, iv);  
 const encrypted = Buffer.concat([cipher.update(*text*), cipher.final()]);  
 return {  
 iv: iv.toString('hex'),  
 encryptedData: encrypted.toString('hex')  
 };  
}  
  
*// Функция для дешифрования строки с использованием AES-CTR*function decryptAESCTR(*encryptedData*, *key*, *iv*) {  
 const decipher = crypto.*createDecipheriv*('aes-256-ctr', *key*, Buffer.from(*iv*, 'hex'));  
 const decrypted = Buffer.concat([decipher.update(Buffer.from(*encryptedData*, 'hex')), decipher.final()]);  
 return decrypted.toString();  
}  
  
*// Функция для хеширования строки с использованием SHA-256*function hashSHA256(*text*) {  
 const hash = crypto.*createHash*('sha256');  
 hash.update(*text*);  
 return hash.digest('hex');  
}  
  
*// Функция для подписи сообщения с использованием RSA-PSS*function signMessageRSA(*message*, *privateKey*) {  
 const sign = crypto.*createSign*('RSA-SHA256');  
 sign.update(*message*);  
 return sign.sign(*privateKey*, 'hex');  
}  
  
*// Функция для проверки подлинности подписи сообщения с использованием RSA-PSS*function verifyMessageRSA(*message*, *signature*, *publicKey*) {  
 const verify = crypto.*createVerify*('RSA-SHA256');  
 verify.update(*message*);  
 return verify.verify(*publicKey*, *signature*, 'hex');  
}  
  
*// Пример использования функций*const surname = 'Банкузов'; *// Замените на свою фамилию  
  
// Упаковка ключа*const wrappedKey = wrapKey(encryptionKey, encryptionKey);  
console.log('Упакованный ключ:', wrappedKey);  
  
*// Распаковка ключа*const unwrappedKey = unwrapKey(wrappedKey, encryptionKey);  
console.log('Распакованный ключ:', unwrappedKey);  
  
*// Шифрование*const encryptedData = encryptAESCTR(surname, unwrappedKey);  
console.log('Зашифрованные данные:', encryptedData);  
  
*// Дешифрование*const decryptedData = decryptAESCTR(encryptedData.encryptedData, unwrappedKey, encryptedData.iv);  
console.log('Расшифрованные данные:', decryptedData);  
  
*// Хеширование*const hashedData = hashSHA256(surname);  
console.log('Хеш данных:', hashedData);  
  
*// Подписание сообщения*const privateKey = crypto.*generateKeyPairSync*('rsa', {  
 modulusLength: 2048,  
 privateKeyEncoding: {  
 type: 'pkcs8',  
 format: 'pem'  
 },  
 publicKeyEncoding: {  
 type: 'spki',  
 format: 'pem'  
 }  
}).privateKey;  
const signature = signMessageRSA(surname, privateKey);  
console.log('Подпись сообщения:', signature);  
  
*// Проверка подлинности подписи*const publicKey = crypto.*createPublicKey*(privateKey);  
const isSignatureValid = verifyMessageRSA(surname, signature, publicKey);  
console.log('Проверка подлинности подписи:', isSignatureValid);  
console.log("---------------------------------------------------------------");

