Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

**Практическое задание №9**

**Тема «Изучение стандартных средств для реализации симметричного и ассиметричного шифрование с использованием SubtleCrypto в js» (Вариант12)**

Руководитель: Ржеутская Н. В.

Выполнил:

Студент 2 курса 1 группы ФИТ

Немкович Анастасия Вадимовна

Минск 2023

**Цель**

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

**Теоретическое введение**

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.

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

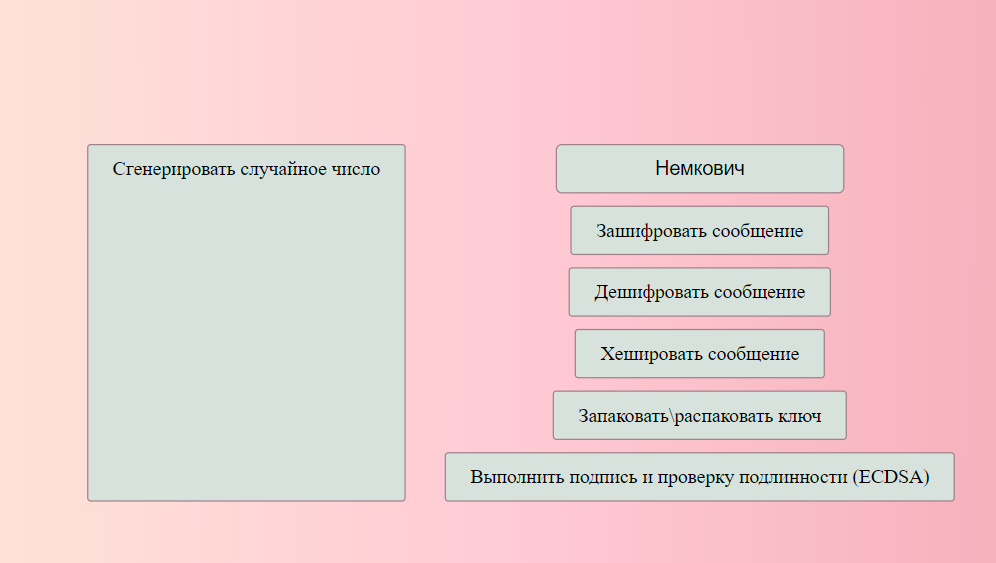
Значение extractable является логическим значением и указывает на возможность экспорта ключа.

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.

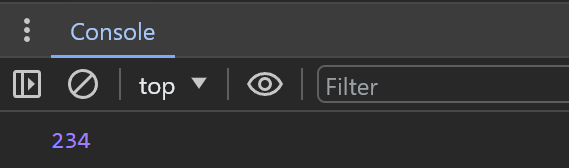
**Ход выполнения**

1. Ознакомиться с созданием криптографического приложения



1. Выполнить генерацию и вывод в консоль случайный чисел.

|  |
| --- |
| *//функция для генерации случайного числа*  button.addEventListener("click", () => {    const array = new Uint8Array(1);    console.log(crypto.getRandomValues(array)[0]);  }); |

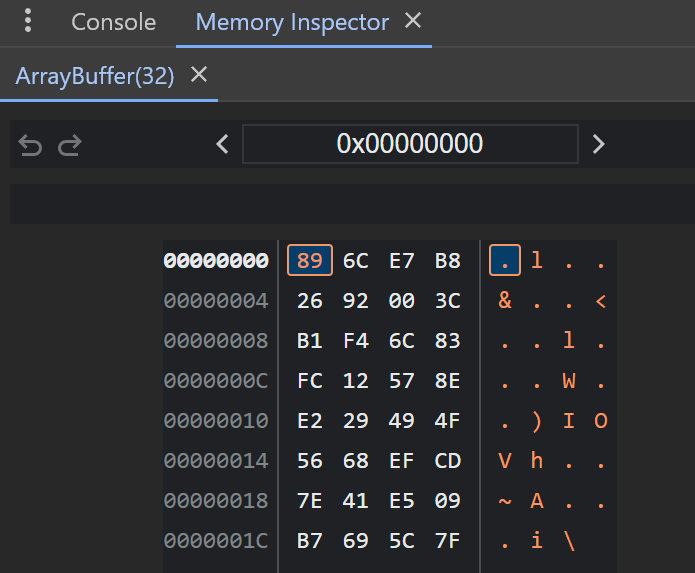


1. Выполнить шифрование, дешифрование и хеширование своей фамилии по указанным алгоритмам.

|  |  |  |
| --- | --- | --- |
| 12 | AES-CTR | SHA-512 |

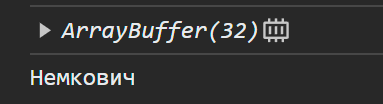
Шифрование:

|  |
| --- |
| *// Асинхронная функция для шифрования сообщения.*  async function encodeMessage() {  *// Генерация ключа для шифрования.*    let key = await window.crypto.subtle.generateKey(decryptKeyParams, true, [      "encrypt",      "decrypt",    ]);    decryptKey = key; *// Сохранение ключа для последующей расшифровки.*  *// Кодирование исходного текста сообщения в байты.*    const originalPlaintext = new TextEncoder().encode(codeMessage.value);  *// Шифрование сообщения и сохранение результата в byteMessage.*    const ciphertext = await crypto.subtle.encrypt(      encryptDecryptParams,      key,      originalPlaintext    );    byteMessage = ciphertext;  *// Вывод зашифрованного сообщения в консоль.*    console.log(ciphertext);  } |



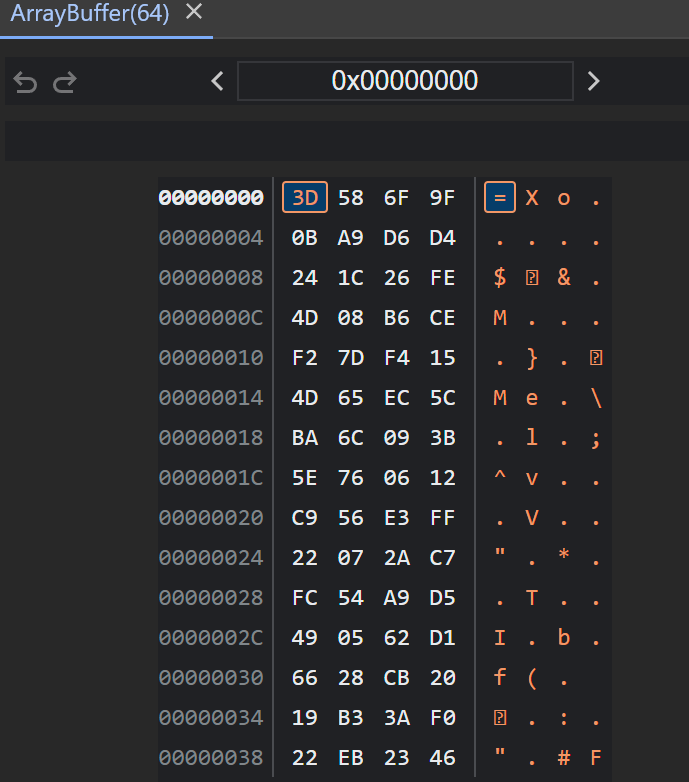
Дешифрование:

|  |
| --- |
| async function decodeMessage() {  *// Расшифровка сообщения и преобразование результата в строку.*    const decryptedPlaintext = await crypto.subtle.decrypt(      encryptDecryptParams,      decryptKey,      byteMessage    );  *// Вывод расшифрованного текста в консоль.*    console.log(new TextDecoder().decode(decryptedPlaintext));  }  decode.addEventListener("click", decodeMessage); |



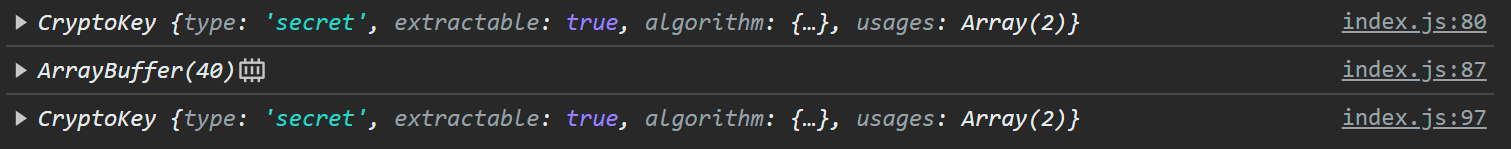
Хеширование:

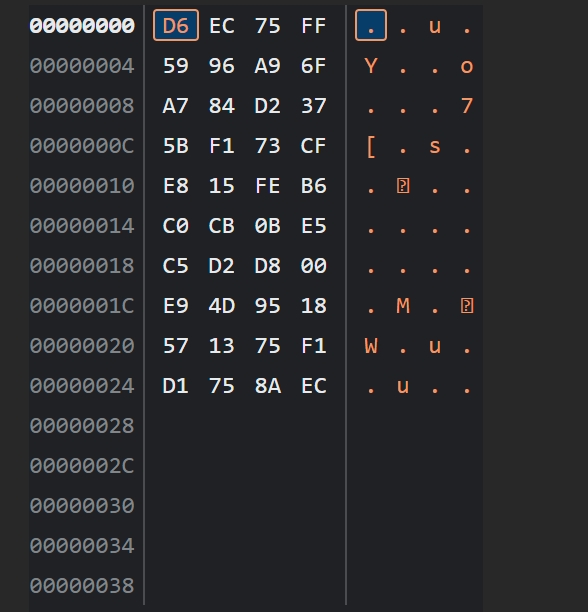
|  |
| --- |
| *// Асинхронная функция для вычисления хеша сообщения.*  async function digestMessage(message) {  *// Создание нового кодировщика текста.*    const encoder = new TextEncoder();  *// Кодирование сообщения в байты.*    const data = encoder.encode(message);  *// Вычисление хеша с помощью алгоритма SHA-512 и возвращение результата.*    const hash = await crypto.subtle.digest("SHA-512", data);    return hash;  }  hash.addEventListener("click", () => {    digestMessage(codeMessage.value).then((digestBuffer) =>  *// Вывод в консоль буфера ArrayBuffer результата хеширования.*      console.log(digestBuffer)    );  *// Очистка поля ввода.*    codeMessage.value = "";  }); |



1. Продемонстрировать упаковку и распаковку ключа, полученного в предыдущем задании используя алгоритм AES-KW.

|  |
| --- |
| async function WrapKey() {  *// Параметры для генерации ключа обертывания.*    const keyFormat = "raw";    const extractable = true;    const wrappingKeyAlgoIdentifier = "AES-KW";    const wrappingKeyUsages = ["wrapKey", "unwrapKey"];    const wrappingKeyParams = {      name: wrappingKeyAlgoIdentifier,      length: 256,    };  *// Генерация ключа обертывания.*    const wrappingKey = await crypto.subtle.generateKey(      wrappingKeyParams,      extractable,      wrappingKeyUsages    );  *// Вывод ключа обертывания в консоль.*    console.log(wrappingKey);  *// Обертывание ключа расшифровки и вывод результата в консоль.*    const wrappedKey = await crypto.subtle.wrapKey(      keyFormat,      decryptKey,      wrappingKey,      wrappingKeyAlgoIdentifier    );    console.log(wrappedKey);  *// Распаковка ключа и вывод в консоль.*    const unwrappedKey = await crypto.subtle.unwrapKey(      keyFormat,      wrappedKey,      wrappingKey,      wrappingKeyParams,      decryptKeyParams,      extractable,      ["encrypt", "decrypt"]    );    console.log(unwrappedKey);  } |





1. Выполнить процедуру подписи сообщения и проверку подлинности с использованием ECDSA.

|  |
| --- |
| pack.addEventListener("click", WrapKey);  *// Функция для создания и проверки цифровой подписи.*  let sign = async function () {  *// Параметры для генерации ключевой пары.*    const keyParams = {      name: "ECDSA",      namedCurve: "P-256",    };    const keyUsages = ["sign", "verify"];  *// Генерация ключевой пары.*    const { publicKey, privateKey } = await crypto.subtle.generateKey(      keyParams,      true,      keyUsages    );  *// Кодирование сообщения для подписи.*    const message = new TextEncoder().encode("Mes to sign");  *// Параметры для создания и проверки подписи.*    const signParams = {      name: "ECDSA",      hash: "SHA-512",    };  *// Создание подписи.*    const signature = await crypto.subtle.sign(signParams, privateKey, message);  *// Проверка подписи.*    const verified = await crypto.subtle.verify(      signParams,      publicKey,      signature,      message    );  *// Вывод параметров подписи, публичного ключа и сообщения в консоль.*    console.log(signParams, publicKey, message);  *// Вывод результата проверки подписи в консоль.*    console.log(verified);  };  *// Добавление обработчика события на кнопку 'signa' для события 'click'.*  signa.addEventListener("click", () => {    sign();  }); |

****