Программирование: как использовать Web Crypto API в браузере в 2026 году

15.11.2026 · PDF-версия статьи с сайта cms.dolinais.ru

Зачем нужна криптография в браузере

Современные веб-приложения всё чаще обрабатывают чувствительные данные на клиенте: генерируют пароли, считают хеши файлов, подписывают запросы. Для этого не обязательно отправлять данные на сервер — в браузере доступен **Web Crypto API** (интерфейс для криптографических операций, встроенный в браузер). Он позволяет выполнять хеширование, генерацию ключей, шифрование и проверку подписей без сторонних библиотек. В 2026 году API поддерживается всеми актуальными браузерами (Chrome, Firefox, Safari, Edge), но имеет ряд особенностей, которые важно учитывать.

В этой статье разберём, как использовать Web Crypto API для типовых задач, какие ограничения существуют и как избежать распространённых ошибок. Примеры кода приведены на JavaScript (ES2022) и протестированы в Chrome 120+ и Firefox 115+.

Основные возможности Web Crypto API

API предоставляет низкоуровневые криптографические примитивы через объект crypto.subtle. Доступны следующие операции:

- **Хеширование** — алгоритмы SHA-1, SHA-256, SHA-384, SHA-512.
- **Генерация ключей** — для симметричных (AES) и асимметричных (RSA, ECDSA) алгоритмов.
- **Шифрование и расшифровка** — AES-CBC, AES-GCM, RSA-OAEP.
- **Подпись и проверка** — HMAC, RSASSA-PKCS1-v1_5, ECDSA.
- **Согласование ключей** — ECDH.
- **Генерация случайных чисел** — crypto.getRandomValues().

Все методы crypto.subtle асинхронны и возвращают Promise. Это связано с тем, что криптографические операции могут быть ресурсоёмкими и выполняются вне основного потока.

Примеры использования

Хеширование строки

Простейшая операция — получить SHA-256 хеш от строки. Используется для проверки целостности данных или создания уникальных идентификаторов.

```
async function sha256(message) {
  const encoder = new TextEncoder();
  const data = encoder.encode(message);
  const hashBuffer = await crypto.subtle.digest('SHA-256', data);
  return Array.from(new Uint8Array(hashBuffer))
    .map(b => b.toString(16).padStart(2, '0'))
    .join('');
```

```
}

sha256('hello').then(console.log); // '2cf24dba...'
```

Обратите внимание: crypto.subtle.digest принимает ArrayBuffer или TypedArray, поэтому строку нужно преобразовать через TextEncoder.

Генерация ключа AES-GCM и шифрование

Для симметричного шифрования часто используют AES-GCM — он обеспечивает конфиденциальность и аутентификацию. Пример создания ключа и шифрования данных:

```
async function encryptData(plainText, key) {
  const encoder = new TextEncoder();
  const iv = crypto.getRandomValues(new Uint8Array(12)); // 96 ??? ??? GCM
  const ciphertext = await crypto.subtle.encrypt(
    { name: 'AES-GCM', iv },
    key,
    encoder.encode(plainText)
  );
  return { iv, ciphertext };
}

async function generateAesKey() {
  return crypto.subtle.generateKey(
    { name: 'AES-GCM', length: 256 },
    true, // ??????????? ????
    ['encrypt', 'decrypt']
  );
}
```

Ключ можно экспортировать в формате JWK или raw для сохранения, но помните: хранение ключей на клиенте снижает безопасность.

Подпись данных с помощью ECDSA

Асимметричные подписи позволяют подтвердить авторство данных. В примере создаётся пара ключей ECDSA (кривая P-256), затем данные подписываются и проверяются.

```
async function signAndVerify() {
  const keyPair = await crypto.subtle.generateKey(
    { name: 'ECDSA', namedCurve: 'P-256' },
    true,
    ['sign', 'verify']
  );
  const data = new TextEncoder().encode('message');
  const signature = await crypto.subtle.sign(
```

```
    { name: 'ECDSA', hash: 'SHA-256' },
    keyPair.privateKey,
    data
  );
  const isValid = await crypto.subtle.verify(
    { name: 'ECDSA', hash: 'SHA-256' },
    keyPair.publicKey,
    signature,
    data
  );
  console.log(isValid); // true
}
```

Ограничения и подводные камни

Несмотря на удобство, Web Crypto API имеет несколько ограничений, которые могут стать неожиданностью.

- **Только безопасный контекст** — API доступен только на страницах, загруженных по HTTPS (или localhost). Это требование безопасности.
- **Асинхронность** — все методы crypto.subtle возвращают Promise, поэтому код нужно строить соответствующим образом.
- **Ограниченный набор алгоритмов** — например, нет поддержки SHA-3, некоторых режимов блочного шифрования. Для экзотических алгоритмов придётся использовать сторонние библиотеки.
- **Извлекаемость ключей** — при генерации можно указать extractable: true, но это снижает безопасность: ключ можно экспортировать и украсть. Для долговременных ключей лучше использовать неизвлекаемые (extractable: false).
- **Производительность** — криптографические операции в браузере могут быть медленными на слабых устройствах. Тяжёлые вычисления (например, генерация RSA-ключей большой длины) лучше выполнять на сервере.

Также стоит помнить, что Web Crypto API не предназначен для защиты от XSS-атак: если злоумышленник может выполнить свой JavaScript на странице, он получит доступ к ключам и данным.

Практические рекомендации

Чтобы использовать Web Crypto API эффективно и безопасно, следуйте этим советам:

1. **Используйте современные алгоритмы** — AES-GCM для симметричного шифрования, ECDSA или Ed25519 для подписей (если поддерживается), SHA-256 для хеширования.
2. **Не храните долговременные ключи на клиенте** — если нужно сохранить ключ, используйте IndexedDB с шифрованием или делегируйте управление ключами серверу.
3. **Проверяйте поддержку браузерами** — хотя API поддерживается широко, некоторые алгоритмы могут быть недоступны в старых версиях. Используйте feature detection.
4. **Обрабатывайте ошибки** — криптографические операции могут завершиться ошибкой (например, при неверном ключе). Всегда добавляйте try/catch.
5. **Тестируйте на разных платформах** — поведение может отличаться в разных браузерах, особенно в мобильных.

> Помните: криптография в браузере не заменяет серверную защиту. Она лишь дополняет её, снижая риски при передаче и обработке данных на клиенте.

Заключение

Web Crypto API — мощный инструмент для выполнения криптографических операций в браузере. Он позволяет создавать безопасные клиентские приложения без сторонних библиотек. Однако важно учитывать ограничения и следовать рекомендациям, чтобы не создать ложное чувство безопасности. В 2026 году API продолжает развиваться: появляются новые алгоритмы и улучшения, поэтому следите за обновлениями спецификации W3C.

Источник: https://cms.dolinais.ru/programmirovanie-kak-ispolzovat-web-crypto-api-v-brauzere-v-2026-godu