УДК 004

М. Д. Былинский

ЗАЩИТА ПРИЛОЖЕНИЙ JAVASCRIPT С ПОМОЩЬЮ WEB CRYPTOGRAPHY API

Балтийский федеральный университет им. И. Канта, Калининград, Россия
Поступила в редакцию 02.07.2022 г.
Принята к публикации 26.07.2022 г.

Для цитирования: *Былинский М.Д.* Защита приложений Javascript с помощью Web Cryptography API // Вестник Балтийского федерального университета им. И. Канта. Сер. Физико-математические и технические науки. 2022. №1. С. 53–60.

По мере роста количества веб-приложений возрастает потребность обычных пользователей в более безопасных веб-приложениях, и веб-разработчики пытаются соответствовать этим ожиданиям. Данная статья посвящена рассмотрению основных концепций Web Cryptography API, что определяет криптографические примитивы, которые должны быть развернуты в браузерах и веб-приложениях JavaScript. Целью статьи является теоретическое обоснование применения Web Crypto API в собственной среде JavaScript для защиты веб-приложений. Раскрыты ключевые понятия в рамках Web Crypto API; описаны дизайн и общие интерфейсы для использования криптографических алгоритмов; обозначены возможные ограничения Web Crypto API; рассмотрены совместимость и принцип работы с криптомодулем Node.js, в частности разработан прототип WebCrypto для Node.js. Результаты исследования предоставляют конкретные доказательства того, что веб-приложения, использующие Web Crypto API, становятся более безопасными, поскольку они обеспечивают соблюдение шаблонов использования ключей, соответствующих известным передовым методам криптографии. Полученные данные свидетельствуют о том, что следует уделять больше внимания установлению взаимодействия между Web Crypto API и существующим криптомодулем Node.js.

Ключевые слова: Web Cryptography API, криптография, криптографические примитивы, Node.js, JavaScript, прототип WebCrypto, аутентифицированное шифрование

Значимость криптографии

По мере того как программное обеспечение становится все более модульным, растет потребность в защите веб-приложений. Криптография становится все более важна в современной разработке приложений с многочисленными вариантами использования, будь то сквозное шифрование в приложении для обмена сообщениями, схема аутентификации для онлайн-банкинга или доказательство целостности важных данных.

До недавнего времени в браузерах не было встроенных криптографических API, что привело к появлению большого количества библиотек JavaScript, реализующих криптографию для веб-приложений. Как

бы хорошо ни были спроектированы и написаны эти библиотеки, программисты были вынуждены использовать JavaScript, который оказался особенно неподходящим языком программирования для криптографических алгоритмов из-за своей высокоуровневой природы, отсутствия 64-битной целочисленной арифметики, доступа к аппаратным функциям и многопоточности [14, p. 10].

Сообщество Node.js рано осознало важность криптографии и уже более 10 лет предоставляет различные криптографические функции с помощью криптомодуля Node.js [5].

В 2017 г. Консорциум World Wide Web (W3C) опубликовал Web Cryptography API, что позволяет приложениям JavaScript в браузерах использовать общие криптографические функции без необходимости обращения к каким-либо сторонним библиотекам. Эти функции, независимо от того, предоставляются они через криптографический сервис API или через криптомодуль Node.js, часто называют «криптографическими примитивами». Несмотря на то что эти механизмы хорошо зарекомендовали себя и считаются безопасными сами по себе, их легко использовать неправильно и, возможно, небезопасным образом [10].

Дизайн Web Crypto API

На момент написания статьи все популярные браузеры обеспечивают реализацию Web Crypto API для приложений JavaScript через полуглобальный криптообъект (англ. semi-global crypto object) [3]. Теперь рассмотрим API и способы его использования.

Получение криптографически защищенных случайных данных

Первую примечательную особенность Web Crypto API составляет `crypto.getRandomValues`, который в настоящее время является единственным способом для веб-приложений получать криптографически безопасные случайные данные:

```
  const twentyBytes = crypto.getRandomValues(new
Uint8Array(20));
  console.log(twentyBytes);
```

Все остальные функции Web Crypto API доступны через объект `crypto.subtle`. В отличие от `crypto.getRandomValues`, все функции `crypto.subtle` возвращают объекты Promise, как указано в спецификации языка ECMAScript 2015, что позволяет браузерам выполнять необходимые вычисления асинхронно в фоновом режиме вместо блокировки цикла обработки событий JavaScript.

Общие интерфейсы для использования криптографических алгоритмов

Web Crypto API предоставляет набор универсальных интерфейсов для выполнения операций с использованием различных криптографических алгоритмов, которые идентифицируются стандартизированными и в основном говорящими сами за себя именами, такими как AES-CTR, RSA-OAEP, SHA-256 и PBKDF2 [12].

Все операции принимают объект, идентифицирующий алгоритм и опции, если это необходимо. Например, этот фрагмент кода генерирует ключ AES, а затем шифрует сообщение с использованием режима сцепления блоков шифротекста (англ. Cipher Block Chaining mode (CBC)):

```
   const key = await crypto.subtle.generateKey(
     // The algorithm is AES in CBC mode, with a key
length
     // of 256 bits.
     {
       name: 'AES-CBC',
       length: 256
     },
     // Allow extracting the key material (see below).
     true,
     // Restrict usage of this key to encryption.
     ['encrypt']
   );

   // AES-CBC requires a 128-bit initialization vector
(iv).
   const iv = crypto.getRandomValues(new Uint8Array(16));

   // This is the plaintext:
   const encoder = new TextEncoder();
   const message = encoder.encode('Hello world!');

   // Finally, encrypt the plaintext, and obtain the
ciphertext.
   const ciphertext = await crypto.subtle.encrypt(
     // The algorithm is still AES-CBC. In addition,
the
     // 128-bit initialization vector must be specified.
     {
       name: 'AES-CBC',
       iv
     },
     // The encryption key. This must be an AES-CBC key,
     // otherwise, this function will reject.
     key,
     // The plaintext to encrypt.
     message
   );
```

Web Cryptography API использует экземпляры класса `ArrayBuffer` для представления последовательности байтов, но большинство функций также принимают любой тип `TypedArray` в качестве входных данных [13]. Результатом `crypto.subtle.encrypt` будет `ArrayBuffer`, и, вероятно, его необходимо преобразовать в другой тип данных или формат для хранения или передачи зашифрованных данных.

Шаблоны использования соответствуют лучшим практикам в криптографии

В отличие от многих других криптографических библиотек, Web Crypto API применяет некоторые шаблоны использования ключей, соответствующие известным передовым методам криптографии [15]. Ключи можно использовать и получать к ним доступ только через класс `CryptoKey`, который накладывает определенные ограничения: материал ключа может быть извлечен из `CryptoKey` только в том случае, если его свойство `extractable` было явно установлено в `true` во время его создания.

Точно так же каждый `CryptoKey` имеет алгоритм (`algorithm`) и атрибут использования (`usages`). Если ключ используется для операции (например, для создания цифровой подписи) в контексте алгоритма (например, RSASSA-PSS), операция завершается ошибкой, если алгоритм (`algorithm`) ключа не равен запрошенному алгоритму или если использование (`usages`) свойства не содержит запрошенное имя операции [8].

Эти ключевые свойства сохраняются при импорте и экспорте ключей в формате JSON Web Key (`JWK`), но не в любом другом поддерживаемом формате. JWK представляет собой JSON-представление ключевого материала и поэтому является хорошим выбором для хранения или передачи данных в среде, которая традиционно имеет очень ограниченную поддержку двоичных данных [11, p. 9].

Неизменяемые объекты WebCrypto без состояния

Хороший аспект дизайна API заключается в том, что там, где другие библиотеки часто требуют, чтобы объекты проходили сложные жизненные циклы, все объекты WebCrypto не имеют состояния (англ. stateless) и неизменяемы (англ. immutable), что делает невозможным случайное использование ключей или алгоритмов в недопустимом или неожиданном состоянии.

Ограничения Web Crypto API

Одним из текущих ограничений Web Crypto API является отсутствие поддержки любой потоковой передачи. Одновременная обработка больших объемов данных обычно относительно неэффективна, поэтому Node.js предоставляет потоковые интерфейсы для многих криптографических функций, включая симметричное шифрование и дешифрование, хеширование, создание цифровой подписи и ее проверку.

Поддержка браузерами криптографических API различается, и хотя все распространенные браузеры реализуют API, нет гарантии, что конкретный алгоритм поддерживается во всех браузерах [4]. Если веб-приложение требует определенного алгоритма, и предполагается, что

оно будет работать в браузерах, которые могут его не поддерживать, может потребоваться динамическое переключение на реализацию JavaScript («polyfill»).

Некоторые аспекты API неясны в рекомендации W3C, и поведение браузеров отличается [9, p. 129]. Например, Firefox 73, в отличие от Google Chrome 80, позволяет разработчикам создавать ключи, которые нельзя ни извлечь, ни использовать, что означает, что их атрибуту `extractable` присвоено значение `false`, а атрибуту `usages` — пустой массив. Точно также Web Crypto API не требует, чтобы браузеры выдавали разумные сообщения об ошибках, и многие браузеры этого не делают, что приводит к разочарованию при отладке программного кода, использующего API.

В рекомендации Web Crypto API также предлагаются нестандартные функции, такие как параметр длины для AES в режиме счетчика (англ. counter mode), который позволяет пользователям ограничивать количество битов вектора инициализации, используемых в качестве счетчика. Эти функции доступны не во всех реализациях, а некоторые реализации могут игнорировать такие опции, что затрудняет обнаружение функций.

Совместимость с криптомодулем Node.js

Официальные дистрибутивы Node.js поддерживают все функции Web Crypto API, но модуль шифрования Node.js предоставляет другой API, а также другие функции и классы. Функции Web Crypto API, как правило, более общие. Например, все симметричное и асимметричное шифрование обрабатывается через `crypto.subtle.encrypt`, тогда как Node.js предоставляет отдельные функции `crypto.createCipheriv`, `crypto.publicEncrypt` и `crypto.privateEncrypt`, в зависимости от того, какой тип шифрования необходимо выполнить.

В большинстве случаев обмен данными между WebCrypto и Node.js crypto API относительно прост, но есть некоторые исключения. В некоторых случаях Web Crypto API представляет данные иначе, чем Node.js. Например, аутентифицированное симметричное шифрование, такое как AES-GCM, создает зашифрованный текст **`C`** и тег аутентификации **`T`** как отдельные последовательности в Node.js, но как единую конкатенированную последовательность **`C | T`** в WebCrypto. И поскольку **`T`** имеет фиксированный размер, преобразование между отдельными значениями и конкатенированной последовательностью легко реализовать в JavaScript.

Обмен ключами между Web Crypto API и Node.js в настоящее время ограничен отсутствием поддержки формата веб-ключа JSON (`jwk`) в Node.js [7]. Однако другие форматы кодирования ключей, используемые Web Crypto API (например, `spki`, `pkcs8` и `raw`), обычно хорошо поддерживаются и предлагают подходящую замену.

Команда Node.js недавно добавила в криптомодуль Node.js различные функции для обеспечения лучшей совместимости, например, под-

держку подписей RSASSA-PSS, настраиваемые хэш-функции для RSA-OAEP, функции для создания пары асимметричных ключей и преобразования между различными форматами ключей и подписей. Благодаря этим дополнениям к Node.js, веб-приложения могут рассчитывать на превосходную совместимость между Node.js и современными веб-браузерами.

Прототип WebCrypto для Node.js

Существует небольшое количество сторонних реализаций Web Crypto API для Node.js, и команда Node.js находится в процессе оценки потенциала Web Crypto API для приложений Node.js, включая реализацию прототипа, написанного на JavaScript:

Асинхронность. Спецификация WebCrypto требует, чтобы почти все операции выполнялись асинхронно, однако Node.js реализует очень мало операций асинхронно [6]. Обычно это не проблема, поскольку: 1) большинство криптографических функций невероятно быстры по сравнению с оверхедом (англ. overhead), который неразрывно связан с асинхронностью, 2) так как Node.js реализует большинство криптографических функций через эффективные потоковые интерфейсы. WebCrypto не имеет потоковых интерфейсов, а только одноразовые API. Таким образом, шифрование, хэширование, подпись или проверка больших объемов данных в WebCrypto затруднены без базовых асинхронных API.

Структура. Основной экспорт реализован в `lib/index.js` и представляет интерфейс `crypto`, как определено в разделе 10 спецификации WebCrypto. Он содержит:

1) `subtle` атрибут реализован в `lib/subtle.js`, включая все методы, описанные в разделе 14.3 спецификации WebCrypto. Эти методы обычно делегируют работу одной или нескольким криптографическим операциям, перечисленным в разделе 18.2.2 и реализованным в `lib/algorithms/`;

2) функция `getRandomValues` реализована в `lib/random.js`.

Тесты. Каталог `test` содержит небольшое количество модульных тестов. Все эти тесты необходимо проходить после каждого коммита. Можно запускать модульные тесты с помощью `npm test`.

Отчет о покрытии можно создать с помощью команды `npm runcover`.

Подмножество тестов веб-платформы также можно использовать для тестирования. Целесообразно инициализировать подмодуль `test/wpt/wpt`, чтобы использовать их. Можно запустить WPT, используя `npm run wpt`. Предлагаемые изменения не обязательно должны проходить все WPT, но они не должны нарушать тесты, которые прошли без изменений.

Линтинг. Этот репозиторий использует ESLint. Необходимо использовать `npm run lint` для проверки кода.

Использование Web Crypto API в Node.js

Одним из преимуществ использования Web Crypto API является возможность повторного использования одного и того же кода для браузерных приложений и приложений Node.js [2]. Однако некоторые характеристики веб-стандарта потенциально лучше подходят для браузеров, чем для Node.js, включая отсутствие поддержки потоковой передачи. Использование объектов Promise имеет смысл в современном JavaScript, но может привести к значительным накладным расходам на сервере при выполнении множества кратковременных криптографических операций, которые могут выполняться так же быстро, как один вызов функции в криптомодуле Node.js [1]. Также кажется неудобным то, что Web Crypto API работает с экземплярами `ArrayBuffer`, тогда как приложения Node.js в основном работают с экземплярами класса `Buffer`.

Ближайшая цель команды Node.js — обеспечить совместимость между Web Crypto API и существующим криптомодулем Node.js. Разумеется, по мере приближения к цели возникают определенные трудности, тем не менее пока можно отметить действительно основательный подход. Криптография, JavaScript и Node.js — быстро меняющиеся технологии, и такие стандарты, как Web Crypto API, должны будут соответствующим образом адаптироваться, как и их реализации.

Список литературы

1. *Лоре А.* Проектирование веб-API / пер. с англ. Д.А. Беликова. М., 2020.

2. *Зима В.М., Молдовян А.А., Молдовян Н.А.* Безопасность глобальных сетевых технологий. СПБ., 2003.

3. *Мао Венбо.* Современная криптография. Теория и практика. М., 2005.

4. *Сабанов А.Г., Скиба В.Ю.* Некоторые аспекты защиты электронного документооборота // Проблемы информационной безопасности. Компьютерные системы. 2012. №2. С. 42—45.

5. *Deveria A.* Web Cryptography. 2020. URL: https://caniuse. com/cryptography (дата обращения: 13.05.2022).

6. *Laurie B., Langley A., Kasper E.* RFC 6962 Certificate Transparency. Experimental, IETF, 2013. URL: https://tools. ietf. org/html/rfc6962 (дата обращения: 10.05.2022).

7. *Torlak E., Taghdiri M., Dennis G., Near J.P.* Applications and extensions of alloy: past, present and future // Mathematical Structures in Computer Science. 2013. Vol. 23. P. 315—333.

8. *Near J.P., Jackson D.* Derailer: Interactive security analysis for web applications. 2014. URL: https://dspace.mit.edu/handle/1721.1/100435 (дата обращения: 10.05.2022).

9. *Cairns K., Halpin H., Steel G.* Security Analysis of the W3C Web Cryptography API // Proceedings of Security Standardisation Research (SSR). Gaithersberg, 2017. P. 112—140.

10. *Watson M.* Web Cryptography API // Tech. Rep. Cambridge: W3C, Jan. 2017. URL: https://www.w3.org/TR/WebCryptoAPI/ (дата обращения: 08.05.2022).

11. *Jones M.* JSON Web Key (JWK). IETF. May 2015. Proposed Standard. URL: https://www.rfc-editor.org/rfc/rfc7517 (дата обращения: 10.05.2022).

12. *Turner S.* Asymmetric Key Packages. IETF. August 2010. Proposed Standard. URL: https://www.rfc-editor.org/rfc/rfc5958 (дата обращения: 12.05.2022).

13. *Stanford* Computer Security Lab. Stanford Javascript Crypto Library (SJCL). 2019. URL: http://bitwiseshiftleft.github.io/sjcl/ (дата обращения: 10.05.2022).

14. *Ptacek Th.* Javascript Cryptography Considered Harmful. 2011. URL: https://www.nccgroup.trust/us/about-us/newsroom-and-events/blog/2011/august/javascript-cryptography-considered-harmful/ (дата обращения: 11.05.2022).

15. *Perrin T.* Web Cryptography API. Editor's draft, W3C, 2014. URL: http://github.com/trevp/curve25519webcrypto (дата обращения: 08.05.2022).

Об авторе

Мстислав Дмитриевич Былинский — магистрант, Балтийский федеральный университет им. И. Канта, Калининград, Россия.

E-mail: mcstyle@yandex.ru

M.D. Bylinsky

SECURING JAVASCRIPT APPLICATIONS WITH THE WEB CRYPTOGRAPHY API

Immanuel Kant Baltic Federal University, Kaliningrad, Russia

Received 02 July 2022
Accepted 26 July 2022

To cite this article: Bylinsky M.D. 2022, Securing JavaScript applications with Web Cryptography API, *Vestnik of Immanuel Kant Baltic Federal University. Series: Physical-mathematical and technical sciences,* №1. P. 53—60.

With an increasing number of web applications, the need of ordinary users to have more secure web applications has increased and web developers are attempting to match those expectations. This article is devoted to consideration of the basic concepts of the Web Cryptography API since it defines cryptographic primitives to be deployed across browsers and JavaScript web applications. The purpose of the article is the theoretical justification for the application of Web Crypto API within native JavaScript environments in order to secure web applications. The article deals with the key definitions within the Web Crypto API; describes design and generic interfaces for using cryptographic algorithms; indicates possible limitations of the Web Crypto API; reviews its compatibility and implementations for Node.js cryptomodule, for instance, a WebCrypto prototype for Node.js has been developed. The results of study provide concrete evidence that web applications that use Web Crypto API become more secure as it enforces usage patterns of keys that correlate to known best practices in cryptography. The findings suggest that more emphasis should be placed on providing interoperability between the Web Cryptography API and the existing Node.js crypto module.

Keywords: Web Cryptography API, cryptography, cryptographic primitives, Node.js, JavaScript, WebCrypto prototype, authenticated encryption

The author

Mstislav D. Bylinsky, Master's Student, Immanuel Kant Baltic Federal University, Kaliningrad, Russia.

E-mail: mcstyle@yandex.ru