Алгоритм 3. NewKey — перешифрование данных на серверах с помощью набора $\widetilde{\Gamma} = \{\widetilde{\Gamma}^1, \ldots, \widetilde{\Gamma}^r\}$

1: Пользователь передаёт на сервер S_j, $j = 1, \ldots, r$, соответствующие этому серверу p векторов: $\widetilde{\Gamma}^{j \bmod (r+1)+\lfloor j/(r+1)\rfloor}, \ldots, \widetilde{\Gamma}^{(j+p-1) \bmod (r+1)+\lfloor (j+p-1)/(r+1)\rfloor}$.

2: Сервер S_j обновляет хранящиеся у него части: $\mathrm{DB}_\Gamma^l := \mathrm{DB}_\Gamma^l \oplus \widetilde{\Gamma}^l$, где $l \in \{j \bmod (r+1) + \lfloor j/(r+1)\rfloor, \ldots, (j+p-1) \bmod (r+1) + \lfloor (j+p-1)/(r+1)\rfloor\}$, $j = 1, \ldots, r$.

Алгоритм 4. SetBlock — перезапись i-го блока в базе DB новым значением $\widetilde{d}_i$

1: Пользователь получает i-й блок базы ($d_i = \mathbf{GetBlock}(i)$), генерирует новый набор $\widetilde{\Gamma} = \{\widetilde{\Gamma}^1, \ldots, \widetilde{\Gamma}^r\}$ ($\widetilde{\Gamma} \leftarrow$ `rnd_gen`) и для каждого $\widetilde{\Gamma}^j = (\widetilde{\gamma}_1^j, \ldots, \widetilde{\gamma}_b^j)$ из $\widetilde{\Gamma}$ переопределяет $\widetilde{\gamma}_i^j$: $\widetilde{\gamma}_i^j := \widetilde{\gamma}_i^j \oplus (\widetilde{d}_i \oplus d_i)$.

2: Пользователь выполняет протокол перешифрования $\mathbf{NewKey}(\widetilde{\Gamma})$.

Показано, что

1) протоколы **GetBlock** и **SetBlock** обеспечивают анонимность соответственно запрашиваемых и записываемых данных;
2) протоколы **DistribDB**, **NewKey** и **SetBlock** обеспечивают конфиденциальность хранимых на серверах данных;
3) любая коалиция мощности $t < \lceil r/p \rceil$ не может нарушить конфиденциальность базы данных, а любая коалиция мощности $t \geqslant r - p + 1$ однозначно дешифрует базу данных.

ЛИТЕРАТУРА

1. *Pfitzmann A. and Hansen M.* A terminology for talking about privacy by data minimization: Anonymity, Unlinkability, Undetectability, Unobservability, Pseudonymity, and Identity Management. `https://dud.inf.tu-dresden.de/literatur/Anon_Terminology_v0.18.pdf`. Дата обращения 30.03.2016.
2. *Chor B., Goldreich O., Kushilevitz E., and Sudan M.* Private information retrieval // J. ACM. 1998. V. 45(6). P. 965–981.
3. *Demmler D., Herzberg A., and Schneider T.* RAID-PIR: Practical Multi-Server PIR // Proc. 6th edition of the ACM Workshop on Cloud Computing Security. N. Y., USA: 2014. P. 45–56.

УДК 004.94 DOI 10.17223/2226308X/9/34

МЕТОД ЗАПУТЫВАНИЯ ПРОГРАММНОЙ РЕАЛИЗАЦИИ СХЕМЫ HMAC ДЛЯ НЕДОВЕРЕННОЙ СРЕДЫ

Д. Н. Колегов, О. В. Брославский, Н. Е. Олексов

Предлагается метод обфускации схемы аутентификации сообщений HMAC для реализации в недоверенных средах.

Ключевые слова: *white-box cryptography, коды аутентификации сообщений, HMAC, обфускация, защита приложений.*

При разработке защищённых веб-приложений часто необходимо реализовывать алгоритмы выработки кодов аутентификации сообщений (MAC) на языке JavaScript

в браузере пользователя. При этом нарушитель имеет не только доступ к исходному коду криптографических алгоритмов и ключевой информации, но и возможность отладки и изменения программы, что позволяет рассматривать браузер как недоверенную среду. В данном контексте возможность вычисления кодов аутентификации, в отличие от обладания ключевой информацией, может не представлять существенного интереса для нарушителя.

В качестве примера рассмотрим веб-приложение, в котором для уменьшения поверхности атаки используется аутентификация HTTP-запросов на стороне клиента. Для осуществления подобной аутентификации возникает необходимость использования MAC для контролируемых параметров HTTP-запроса. В данном случае возможность вычислять MAC представляет для нарушителя существенно меньший интерес, чем сама ключевая информация. Зная ключ и алгоритм выработки MAC, нарушитель способен использовать эффективные автоматизированные средства для анализа веб-приложения. Обладая же лишь возможностью вычисления MAC при помощи исходного алгоритма, нарушитель вынужден генерировать все запросы через веб-браузер, используя стандартную функциональность веб-приложения, что существенно затрудняет и замедляет анализ приложения.

С целью затруднения извлечения нарушителем ключевой информации из приложения принято использовать положения модели «White-box Cryptography» [1]. В настоящее время для недоверенных сред известны лишь примеры реализации симметричных алгоритмов шифрования [2, 3] и неизвестно ни одного метода для схемы HMAC.

Одним из способов выработки MAC являются ключевые хэш-функции. Известны два основных способа построения ключевых хэш-функций. Первый — использование блочных шифров в режиме генерации имитовставки. Однако полученные таким образом MAC, как правило, имеют недостаточную длину, а алгоритм их вычисления может быть неэффективен. Кроме того, проблема получения реализации блочных шифров для недоверенных сред частично изучена, и на данный момент существует ряд таких реализаций шифров DES [2] и AES [3].

Второй способ — получение ключевых хэш-функций на основе бесключевых. Данный подход позволяет строить эффективно вычислимые ключевые хэш-функции и более гибко выбирать длину получаемого MAC. Для описания этого алгоритма рассмотрим классическую схему вычисления хэш-функций на основе одношагового сжимающего отображения.

В качестве сжимающего отображения выбирается функция двух переменных $f(x, y)$, где x и y — двоичные слова длины m и n соответственно. Важно отметить, что отображение f в данном построении полностью определяет свойства получаемой хэш-функции, а потому f должна быть односторонней и устойчивой к коллизиям. Далее для вычисления $h(M)$ сообщение M разбивается на блоки $M_1, \ldots, M_N$ длины m, которые последовательно передаются на вход сжимающему отображению следующим образом:

$$b_0 = \nu, \quad b_i = f(M_i, b_{i-1}), \quad i = 1, \ldots, N, \quad h(M) = b_N, \tag{1}$$

где ν — некоторое фиксированное начальное значение.

В случае, если длина сообщения M не кратна m, последний блок сообщения дополняется некоторым специальным образом до полного. Обозначим операцию дополнения $pad(x, p)$, результат её вычисления — сообщение x, дополненное до длины m при помощи p. Помимо непосредственно дописывания дополнения, для операции $pad(x, p)$ допустима также модификация сообщения x, например сложение x и p по модулю 2^m, как это происходит в алгоритме [4].

Очевидно, для построения ключевой хэш-функции на основе бесключевой ключ MAC должен быть некоторым образом добавлен к исходному сообщению. Ввиду того, что непосредственно приписывание ключа к началу или концу сообщения может существенно ослабить получаемую ключевую хэш-функцию, в [5] предлагается следующая схема построения:

$$H(k, x) = h(k, p_1, h(k, p_2, x)),$$

где p_1 и p_2 — дополнения ключа k до длины, кратной m, используемые при выполнении операций $pad(k, p_1)$ и $pad(k, p_2)$, а h — бесключевая хэш-функция, вычисляемая по схеме (1) .

Обратим внимание, что значение k, в соответствии с требованиями схемы, целиком попадает в первые блоки вычисления обоих хеш-значений. Данное наблюдение позволяет предвычислить значение h на данных блоках и тем самым избежать использования k в реализации алгоритма в открытом (или легко вычислимом) виде.

Разобьём ключ k на блоки $K_1, K_2, \ldots, K_l$ длины m. Положим

$$epad(k, p) = K_1||K_2|| \ldots ||pad(K_l, p),$$

где «||» — конкатенация строк.

Обозначим h_1 и h_2 хэш-функции, в которых b_0 равно $h(epad(k, p_1))$ и $h(epad(k, p_2))$ соответственно, а дальнейшее вычисление производится по схеме (1). Тогда нетрудно заметить, что

$$H(k, x) = h(k, p_1, h(k, p_2, x)) = h_1(h_2(x)).$$

То есть ключ k не используется в алгоритме в открытом виде, а только в составе свёрток $h(epad(k, p_1))$ и $h(epad(k, p_2))$. Извлечение ключа из данных свёрток является вычислительно трудной задачей ввиду требований к сжимающим отображениям, указанным выше.

Полученный алгоритм вычисления H удовлетворяет всем условиям модели «White-box Cryptography» [1]. Предложенный метод реализован для общепринятого алгоритма HMAC [4] на языке JavaScript в рамках экспериментального проекта jCrypto [6].

ЛИТЕРАТУРА

1. White-Box Cryptography: Protecting Cryptographic Keys in Software Applications. `http://www.whiteboxcrypto.com/`
2. *Chow W. S., Eisen P., Johnson H., and Van Oorschot P. C.* A White-box DES Implementation for DRM Applications // LNCS. 2003. V. 2696. P. 1–15.
3. *Chow W. S., Eisen P., Johnson H., and Van Oorschot P. C.* White-Box Cryptography and an AES Implementation // LNCS. 2003. V. 2595. P. 250–270.
4. RFC 2104. HMAC: Keyed-Hashing for Message Authentication. `https://tools.ietf.org/html/rfc2104`
5. *Menezes A. J., Van Oorschot P. C., and Vanstone S. A.* Handbook of Applied Cryptography. N. Y.: CRC Press, 1997.
6. jCrypto: White-Box Cryptography Tools for ECMAScript Language. `https://github.com/tsu-iscd/jcrypto`