тересе к данному протоколу, но даже с учетом того, что не все отличия носили принципиальный характер, такое частое обновление затрудняло процесс внедрения, особенно на уровне аппаратной поддержки в сетевых коммутаторах. Необходимо отметить, что механизм выбора конкретной версии протокола для взаимодействия контроллера и коммутатора появился лишь в 2012 году в версии 1.3.1, что наряду с внедрением расширяемых структур данных в формате TLV (Type-Length-Value, Тип-Длина-Значение) позволило улучшить ситуацию с совместимостью оборудования.

Регулярное появление новых спецификаций было обусловлено тем, что первые редакции протокола были ориентированы на академические исследования и не отражали требований, предъявляемых к коммутаторам компьютерных сетей, которые должны обеспечить корректную обработку произвольного трафика. Показательным является тот факт, что протокол OpenFlow не предусматривает механизмов конфигурирования сетевого оборудования. Для исправления ситуации ONF пришлось разрабатывать параллельное семейство протоколов OF-CONFIG [5]. Протокол OpenFlow предполагает интенсивный опрос коммутатора со стороны контроллера, механизмы асинхронной нотификации об изменении состояния недостаточны. Очевидно, что в условиях реальной сети, когда один контроллер управляет как минимум десятками коммутаторов, такое решение не может обеспечить достаточной эффективности. Кроме этого, архитектура современных коммутаторов такова, что основные ресурсы сосредоточены в плоскости передачи данных. Коммуникационные интерфейсы не обеспечивают достаточной производительности. Переход к асинхронным нотификациям привел к необходимости разработки дополнительного механизма взаимодействия контроллера и коммутатора [6], внедрение которого только начинается. Расширения существующей спецификации необходимы и для организации эффективного взаимодействия с оптоволоконными сетями, широко использующими коммутацию каналов [7]. Без такого взаимодействия внедрение ПКС будет сильно ограничено. Нельзя не отметить, что аппаратная реализация OpenFlow сложна, и современное поколение коммутаторов не было предназначено для таких протоколов и в данный момент пока находится на стадии лабораторных испытаний. Так же, стоить отметить большие перспективы развития данной технологии.

**Литература**

1. Peev M. The SECOQC quantum key distribution network in Vienna [Текст] /M. Peev, C. Pacher, R. All´eaume, C. Barreiro et al // New J. Phys. – 2009. – V.11. – P.075001.

2. Open Networking Foundation [Электронный ресурс] - Режим доступа: https://www.opennetworking.org (дата обращения: 15.12.2015)

3. Open Networking Foundation White Paper Software-Defined Networking: The New Norm for Networks [Электронный ресурс] - Режим доступа: https://www.opennetworking.org/images/stories/downloads/sdn-resources/ white-papers/wp-sdn-newnorm.pdf (дата обращения: 22.11.2015))





4. ONF TS-025 OpenFlow Switch Specification Version 1.5.1 / Open Networking Foundation - 2015

5. ONF TS-016 OF-CONFIG 1.2 OpenFlow Management and Configuration Protocol / Open Networking Foundation - 2014

6. ONF TS-014 OpenFlow Notifications Framework OpenFlow Management Version 1.0 / Open Networking Foundation - 2013

7. ONF TS-022 Optical Transport Protocol Extensions Version 1.0 March 15,2015 / Open Networking Foundation - 2015

А.С. Кирьянцев, И.А. Стефанова

## АЛГОРИТМ ДИНАМИЧЕСКОЙ ГЕНЕРАЦИИ КЛЮЧЕЙ И ПОДПИСЕЙ В ПРИЛОЖЕНИИ CRYP2CHAT

(Поволжский государственный университет телекоммуникаций и информатики)

Современное общество характеризуется интенсивным обменом, накоплением информации в электронном виде и её обработкой в компьютерных сетях, что требует оперативного решения постоянно возникающих проблем защиты информационного содержания и информационной безопасности.

Данная тема является весьма актуальной и подогрета заявлениями и публикациями технического специалиста, бывшего сотрудника ЦРУ Э. Сноудена, о том, что агентство национальной безопасности (АНБ) США ведёт нечестную игру в направлении прослушивания граждан по всему миру при помощи существующих информационных сетей и сетей связи.

Для обмена сообщений в реальном времени существуют программы – мессенджеры, которые могут применяться для передачи текстовых сообщений, звуковых сигналов, картинок, видео, игр, а так же для организации телеконференций путем шифрования сообщений пользователей сети. Обычно мессенджеры работают совместно с сервером и являются клиентскими программами со своими правилами работы и особенностями КУ ним можно отнести, например, программы ICQ, Skype. Основной недостаток этих программ заключается в том, что они оставляют метаданные на центральном сервере в незашифрованном виде, что позволяет, при необходимости, узнать информацию о самих абонентах, времени их общения, количестве сообщений в обычной сессии. Для устранения этого недостатка разработана программная реализация шифрования данных, на клиентской стороне с помощью приложения Cryp2Chat.

В интернет мессенджерах отсутствуют таки функции как:

– проверка наличия MITM (Men in the middle) – атаки,
– наличие «чистого» (без данных) сервера,
– самоуничтожение сообщения после закрытия сессии.

MITM-атака является самым распространенным способом атаки для кражи данных пользователя, когда атакующий способен читать и видоизменять по

своей воле сообщения, которыми обмениваются отправитель и получатель, причем ни один из них не может догадаться о присутствии атакующего в канале. И для него не важно – используете ли вы криптографический протокол SSL или нет. Атакующий подключается к каналу между абонентами и осуществляет активное вмешательство в протокол передачи. При этом он может удалять, искажать информацию или навязывая ложную.

Под «чистым» сервером подразумевается тот факт, что общение двух абонентов не оставляет информации на сервере. В данном случае сервер работает как «репитер» и просто транслирует зашифрованное сообщение между клиентами. После окончания сессии доступ к переписке теряется безвозвратно.

Все эти проблемы решены в новом приложении – Cryp2Chat [1].

Приложение Cryp2Chat является авторским сервисом для обмена быстрыми сообщениями с поддержкой End2End шифрования.

Сервер прототипа данного приложения написан на Node.js (продвинутый JavaScript) с использованием библиотеки для веб сокетов – Socket.IO. В свою очередь Node.js – программная платформа, основанная на движке V8 (транслирующая JavaScript в машинный код), превращающая JavaScript из узкоспециализированного языка в язык общего назначения. Клиентская часть реализована на Html и JavaScript.

Для проксификации и увеличения надежности канала передачи предлагается использование сети TOR (The Onion Router). На компьютере клиента запускается прокси-сервер, который подключается к сети TOR [2], используя многоуровневое шифрование (рис.1). Прежде чем передать пакет данных серверу, он проходит через три случайных компьютера. Перед отправлением пакет шифруется тремя ключами: для каждого из трех компьютеров соответственно. Кроме того, сеть TOR может обеспечивать анонимность для серверов.

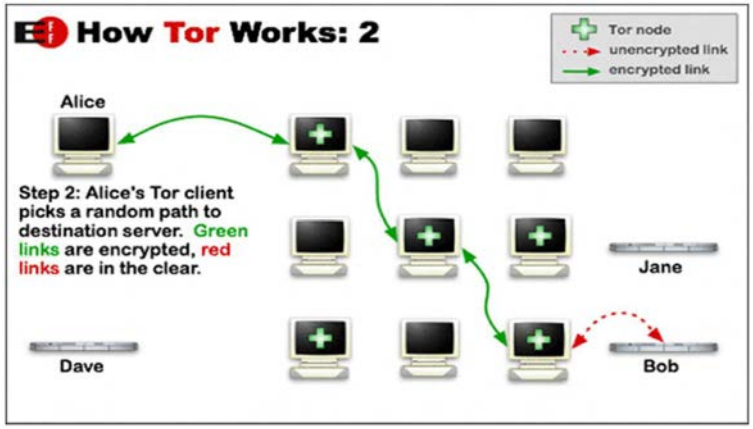


Рис. 1. Схема работы сети TOR





Передача потенциально опасной информации (террористические акты, продажа наркотиков) предотвращена. т.к. контроль обмена данными ведется с помощью электронной подписи. При регистрации пользователь генерирует подпись – это RSA ключ, который передается на сервер, хранится там и никогда не изменяется. При передаче сообщения сервер проверяет подпись, и если данная подпись отсутствует на сервере, то это сообщение не передается.

Таким образом, можно еще и контролировать передачу данных.

В приложении используется алгоритм динамической генерации ключей и подписей для данных с зависимостью от времени.

Принцип работы заключается в следующем: два абонента. генерируют симметричный ключ с помощью алгоритма Диффи-Хелмана. В результате получается «root» (корневой) ключ который в дальнейшем будет использоваться в генерации временного ключа и подписи сообщения. Генерация временного ключа происходит по алгоритму схожим с HMAC (и его стандартом RFC2104), то есть от ключа генерируется хеш, пример: md5( rootKey + Time). При генерации время, а точнее его секундное значение округляется, если значение времени более 30, то в большую, если менее 30, то в меньшую сторону. Именно этим ключом будет шифроваться сообщение, а также с помощью этого алгоритма можно генерировать подпись сообщения, для проверки самого факта изменения сообщения. В результате мы получаем стойкую систему динамических ключей для шифрования сообщения и подпись, при этом для генерации ключа не нужно обмениваться какими-то данными, это снижает риск компрометации сообщения и root-ключа в целом.

Для оценки времени подбора пароля в наихудшем случае воспользуемся формулой [3]:

$$t = (N^0 + N^1 + N^2 + N^L)/V, \quad (1)$$

где t – время, необходимое для гарантированного нахождения заданного пароля, V – количество перебираемых комбинаций в секунду, N – количество символов, из которых может состоять пароль, L длина пароля.

Количество символов в случае использования алгоритма md5 составляет 36. В это число входят 26 символов-букв латинского алфавита (a…z) и 10 символов – арабских цифр (0..9). Количество символов в результативном ключе для шифрования или ключе для подписи – 32 символа. Для вычисления скорости мы возьмём процессор intel i7 и видеокарту Radeon HD5850 1024 MB мощность которых будет ровна 65 000 паролей в секунду определённая опытным путём.

В результате подстановки значений в (1) получим оценочное время:

$$t = \frac{36^0 + 36^1 + 36^2 + 36^{32}}{6500} = 9.745x10^{44}c.$$

При переводе полученного количества времени секунд в более крупные величины мы получим результат в 3.09 $\times 10^{37}$ лет.

Вывод: данный алгоритм можно считать защищённым от подбора, так как время подбора ключа перевешает актуальное время работы с данными.

**Литература**

1. А. Кирьянцев, И. Стефанова. Создание приватного сервиса с использованием приложения CRYP2CHAT. Труды Института системного программирования РАН, Том 27. Выпуск 3. 2015 г. Стр. 279-290.
2. Tor: Overview URL: www.torproject.org/about/overview.html.en
3. Как устроен генератор паролей: www.scritub.com/limba/rusa/194620205.php

А.В. Козачок

## О НЕКОТОРЫХ ОПРЕДЕЛЕНИЯХ СТОЙКОСТИ НЕРАЗЛИЧИМОЙ ОБФУСКАЦИИ ПРОГРАММНОГО КОДА

(Академия ФСО России)

Обфускацией программы называется всякое ее преобразование, которое сохраняет вычисляемую программой функцию (эквивалентное преобразование), но при этом придает программе такую форму, что извлечение из текста программы (программного кода) ключевой информации об алгоритмах и структурах данных, реализованных в этой программе, становится трудоемкой задачей [1]. Вопрос о существовании обфускаторов, удовлетворяющих различным формальным определениям понятия стойкости, является основным предметом рассмотрения данной статьи.

В 2001 году впервые строгое математическое определение стойкости обфускации программного кода была предложено Бараком [2]: обфускация считается стойкой, если всякий противник может извлечь из текста обфусцированной программы за разумное (полиномиальное относительно размера входных данных) время не больше информации, чем можно было бы получить, проводя тестовые испытания этой программы как "черного ящика". Рассмотрим основные положения теории Барака.

**Определение 1.** (Обфускация в модели виртуального "черного ящика"). Пусть $\{C_\ell\}_{\ell\in N}$ – класс булевых схем, тогда обфускатором в модели виртуального "черного ящика" для класса схем $\{C_\ell\}_{\ell\in N}$ называется такая (полиномиальная вероятностная машина Тьюринга) PPT $O$, которая удовлетворяет следующим требованиям [1]:

– функциональная эквивалентность. Для любой схемы $C\in\{C_\ell\}_{\ell\in N}$ и любого входа $x$ существует пренебрежимо малая функция $\mu$, такая что:

$$Pr[(O(C))(x)\neq C(x)]\le\mu(|C|);$$

– полиномиальные издержки. Существуют полином $poly(\cdot)$ такой, что для любой схемы $C\in\{C_\ell\}_{\ell\in N}$:

$$|O(C)|\le poly(|C|);$$





– свойство виртуального "черного ящика": для любой PPT $A$ (противника) существуют такая PPT $S$ (симулятор) и пренебрежимо малая функция $\mu$ такая, что для любой схемы $C\in\{C_\ell\}_{\ell\in N}$:

$$|Pr[A(O(C))=1]-Pr[S^C(1^{|C|})=1]|\le\mu(|C|).$$

В работе Барака также было доказано утверждение о невозможности построения идеального обфускатора для всех классов булевых функций, но данное утверждение не исключает возможности построения обфускатора, стойкого в модели виртуального "черного ящика". Исследования стойкости обфускации также были проведены в работах [3–5]. В 2013 году была предложена новая конструкция [6], так называемая "обфускация неразличимости" (indistinguishability obfuscation - $iO$).

**Определение 2.** Обфускатором неразличимости $iO$ называется однородная полиномиальная вероятностная машина Тьюринга для булевых схем класса $\{C_\lambda\}$, где $\lambda$ – параметр безопасности, если обеспечивается выполнение следующих требований [6]:

– функциональная эквивалентность. Существует пренебрежимо малая функция $\mu(\lambda)$ такая, что для всех $\lambda\in\mathbb{N}$ и всякой схемы $C\in C_\lambda$:

$$\forall x: Pr[iO(C(x))=C(x)]\ge 1-\mu(\lambda);$$

– свойство виртуального "черного ящика" (стойкость). Для любой PPT $D$ (распознавателя), существует пренебрежимо малая функция $\alpha$ такая, что для всех $\lambda\in\mathbb{N}$ и любой пары эквивалентных схем $C_1,C_2\in C_\lambda$, имеющих одинаковый размер, распределения вероятностей случайных величин $iO(C_1)$ и $iO(C_2)$ вычислительно неразличимы, то есть выполняется соотношение:

$$\forall x:\left|Pr[D(iO(\lambda,C_1(x)))=1]-Pr[D(iO(\lambda,C_2(x)))=1]\right|\le\alpha(\lambda).$$

Это означает, что не существует алгоритма распознавания обфускации более эффективного, чем обычное предположение, сделанное на основе анализа входов и выходов обфусцированной программы (функции или обфусцированного набора инструкций).

Неразличимость обфусцированных программ означает, что если две эквивалентные программы (одинакового размера) $P_1,P_2$ такие, что $\forall x: P_1(x)=P_2(x)$, а $iO$ – это обфускатор неразличимости, который принимает на вход программу $P(x)$ и на выходе генерирует новую программу $iO(P(x))$, то $iO(P_1(x))$ и $iO(P_2(x))$ будут вычислительно неразличимы:

$$\exists P_1(x),P_2(x).\forall x: P_1(x)=P_2(x)\to iO(P_1(x))\approx iO(P_2(x)).$$

На основе данной конструкции можно построить доказательство о том, что возможно построение неразличимого обфускатора для всех классов булевых функций, стойкого в модели виртуального "черного ящика". Рассмотрим основу построения обфускатора неразличимости [7]. Обфускатор $iO$ принимает в качестве входных данных программу $f$, представленную в виде булевой функции, и на выходе генерирует обфусцированную программу $iO(f)$. Фор-