Содержание

От издательства 13

Вступительное слово 14

Об авторе 15

О рецензенте 16

Предисловие 17

Часть I. Введение 21

Глава 1. OpenSSL и другие библиотеки SSL/TLS 22

Что такое OpenSSL? 22
История OpenSSL 24
Что нового в OpenSSL 3.0? 24
Сравнение OpenSSL с GnuTLS 25
Сравнение OpenSSL с NSS 26
Сравнение OpenSSL с Botan 27
Сравнение OpenSSL с облегченными библиотеками TLS 27
Сравнение OpenSSL с LibreSSL 29
Сравнение OpenSSL с BoringSSL 30
Резюме 31

Часть II. Симметричная криптография 32

Глава 2. Симметричное шифрование и расшифрование 33

Технические требования 33
Что такое симметричное шифрование 34
Обзор симметричных шифров, поддерживаемых OpenSSL 35

Сравнение блочных и потоковых шифров 35
Стойкость симметричного шифра 37
Сколько битов стойкости достаточно? 38
Обзор шифра AES 38
Обзор шифров DES и 3DES 39
Обзор шифра RC4 40
Обзор шифра ChaCha20 40
Обзор других симметричных шифров, поддерживаемых OpenSSL 41
Национальные шифры 41
Семейство шифров RC 43
Другие шифры 43
Режимы работы блочных шифров 44
Обзор режима простой замены 44
Обзор режима простой замены с зацеплением 45
Обзор режима CTR 47
Обзор режима GCM 48
Обзор режима AES-GCM-SIV 51
Другие режимы работы блочных шифров 51
Выбор режима работы блочного шифра 52
Дополнение для блочных шифров 52
Как сгенерировать ключ симметричного шифрования 53
Скачивание и установка OpenSSL 54
Шифрование и расшифрование с применением AES в командной строке 56
Инициализация и очистка библиотеки OpenSSL 60
Компиляция и компоновка с OpenSSL 61
Шифрование с применением AES из программы 63
Реализация программы шифрования 64
Выполнение программы encrypt 67
Расшифрование с применением AES из программы 68
Реализация программы decrypt 68
Выполнение программы decrypt 70
Резюме 71

Глава 3. Хеш-значения сообщений 72

Технические требования 72
Что такое хеш-значение сообщения и криптографическая функция хеширования? 73
Зачем нужны хеши сообщений? 74
Проверка целостности данных 74
Хеш-значения как основа HMAC 74
Цифровые подписи 75
Сетевые протоколы 75
Проверка пароля 75
Идентификатор содержимого 75
Блокчейн и криптовалюты 76

Доказательство выполнения работы 76
Оценка стойкости криптографических функций хеширования 77
Обзор криптографических функций хеширования, поддерживаемых OpenSSL 78
Семейство функций хеширования SHA-2 78
Семейство криптографических функций SHA-3 79
Функции хеширования SHA-1 и SHA-0 81
Семейство функций хеширования MD 82
Семейство функций хеширования BLAKE2 83
Менее популярные функции хеширования, поддерживаемые OpenSSL 84
Национальные криптографические функции хеширования 84
Другие криптографические хеш-функции 84
Какую криптографическую функцию хеширования выбрать? 85
Вычисление хеш-значения в командной строке 85
Вычисление хеш-значения из программы 86
Реализация программы digest 87
Выполнение программы digest 88
Резюме 88

Глава 4. MAC и HMAC 89

Технические требования 89
Что такое имитовставка? 90
Стойкость функции вычисления MAC 90
HMAC – MAC на основе функции хеширования 91
MAC, шифрование и принцип криптографической обреченности 93
Вычисление HMAC в командной строке 95
Вычисление HMAC из программы 96
Реализация программы hmac 97
Выполнение программы hmac 98
Резюме 98

Глава 5. Формирование ключа шифрования из пароля 100

Технические требования 100
В чем разница между паролем и ключом шифрования? 101
Что такое функция формирования ключа? 101
Обзор функций формирования ключа, поддерживаемых OpenSSL 104
Формирование ключа из пароля в командной строке 105
Формирование ключа из пароля из программы 105
Реализация программы kdf 106
Выполнение программы kdf 107
Резюме 108

Часть III. Асимметричная криптография и сертификаты 109

Глава 6. Асимметричное шифрование и расшифрование 110

Технические требования 110
Что такое асимметричное шифрование 111
Что такое атака с человеком посередине 111
Личная встреча 112
Проверка цифрового отпечатка ключа по телефону 112
Разделение ключа 113
Подписание ключа доверенной третьей стороной 113
Какие виды асимметричного шифрования доступны в OpenSSL 113
Что такое сеансовый ключ 114
Криптостойкость RSA 115
Генерирование пары ключей RSA 118
Шифрование и расшифрование с помощью RSA в командной строке 120
Шифрование с помощью RSA из программы 122
Реализация программы rsa-encrypt 123
Выполнение программы rsa-encrypt 125
Что такое очередь ошибок в OpenSSL 126
Расшифрование с помощью RSA из программы 129
Реализация программы rsa-decrypt 130
Выполнение программы rsa-decrypt 131
Резюме 131

Глава 7. Цифровые подписи и их проверка 132

Технические требования 133
Что такое цифровая подпись 133
Различие между цифровыми подписями и имитовставками 134
Обзор алгоритмов цифровых подписей, поддерживаемых OpenSSL 135
Алгоритм RSA 135
Алгоритм DSA 135
Алгоритм ECDSA 136
Алгоритм EdDSA 137
Алгоритм SM2 138
Какой алгоритм цифровой подписи выбрать? 139
Генерирование пары ключей эллиптического шифрования 139
Подписание и проверка подписи в командной строке 141
Подписание из программы 142
Реализация программы ec-sign 143
Выполнение программы ec-sign 145
Проверка подписи из программы 145
Реализация программы ec-verify 146
Выполнение программы ec-verify 147
Резюме 148

Глава 8. Сертификаты X.509 и инфраструктура открытых ключей 149

Технические требования 150
Что такое сертификат X.509? 150
Цепочки сертификатов 152
Как выпускаются сертификаты X.509 156
Что такое расширения X509v3? 159
Инфраструктура открытых ключей X.509 159
Генерирование самоподписанного сертификата 160
Генерирование несамоподписанного сертификата 162
Проверка сертификата в командной строке 165
Проверка сертификата из программы 166
  Реализация программы x509-verify 166
  Выполнение программы x509-verify 168
Резюме 169

Часть IV. TLS-подключения и безопасная связь 170

Глава 9. Установление TLS-подключений и передача по ним данных 171

Технические требования 172
Что такое протокол TLS 172
  TLS-квитирование 174
  Что происходит после TLS-квитирования? 176
История протокола TLS 177
Установление клиентского TLS-подключения в командной строке 180
Подготовка сертификатов для подключения к TLS-серверу 181
Прием запроса на подключение к TLS-серверу в командной строке 183
Базовые объекты ввода-вывода в OpenSSL 183
Установление клиентского TLS-подключения из программы 186
  Реализация программы tls-client 187
  Выполнение программы tls-client 190
Прием запроса на подключение к TLS-серверу из программы 191
  Реализация программы tls-server 192
  Выполнение программы tls-server 197
Резюме 199

Глава 10. Использование сертификатов X.509 в TLS 200

Технические требования 200
Специальная проверка сертификата другой стороны в программах на C 201
  Регистрация функции обратного вызова для проверки 203
  Реализация функции обратного вызова для проверки 204

Выполнение программы ... 207
Использование списков отозванных сертификатов в программах на C ... 208
Регистрация функции обратного вызова для поиска в CRL ... 210
Реализация функции обратного вызова для поиска в CRL ... 211
Реализация функции скачивания CRL с точки распространения ... 213
Реализация функции скачивания CRL с URL-адреса типа HTTP ... 214
Выполнение программы ... 216
Протокол онлайновой проверки состояния сертификата ... 217
Что такое протокол онлайновой проверки состояния сертификата ... 217
Использование OCSP в командной строке ... 218
Использование OCSP в программах на C ... 221
Регистрация функции обратного вызова для OCSP ... 223
Реализация функции обратного вызова для OCSP ... 223
Выполнение программы ... 227
Использование клиентских сертификатов в TLS ... 228
Генерирование клиентских сертификатов TLS ... 228
Упаковка клиентских сертификатов в контейнер PKCS #12 ... 229
Программный запрос и проверка клиентского сертификата TLS на стороне сервера ... 230
Проверка клиентского сертификата TLS ... 231
Реализация функции построения ответа ... 232
Выполнение программы ... 233
Программное установление TLS-подключения с клиентским сертификатом ... 235
Изменение кода, унаследованного от программы tls-client ... 236
Загрузка сертификата TLS-клиента ... 237
Выполнение программы ... 240
Резюме ... 240

Глава 11. Специализированные применения TLS ... 242

Технические требования ... 242
Что такое закрепление сертификатов в TLS ... 243
Использование закрепления сертификатов ... 245
Изменение функции run_tls_client() ... 246
Реализация функции cert_verify_callback() ... 248
Выполнение программы tls-cert-pinning ... 249
Что такое блокирующие и неблокирующие сокеты ... 251
Использование неблокирующих сокетов в TLS ... 252
Изменение функции run_tls_client() ... 252
Выполнение программы tls-client-non-blocking ... 257
Что такое TLS на нестандартных сокетах ... 258
Использование TLS на нестандартных сокетах ... 259
Реализация функции service_bios() ... 260
Переработанная функция run_tls_client() ... 262
Выполнение программы tls-client-memory-bio ... 267
Резюме ... 269

Часть V. Мини-УЦ 270

Глава 12. Эксплуатация мини-УЦ 271

Технические требования 271
Подкоманда openssl ca 271
Генерирование сертификата корневого УЦ 273
Генерирование сертификата промежуточного УЦ 278
Генерирование сертификата для веб-сервера 282
Генерирование сертификата веб-клиента и почтового клиента 283
Отзыв сертификатов и генерирование CRL 285
Информирование о состоянии отзыва сертификатов по протоколу OCSP 288
Резюме 291

Предметный указатель 292

Вступительное слово

Больше тридцати лет занимаясь кодированием, я много раз встречал на своем пути ребят, которые могли бы написать книгу, а то и десять книг о распространенных, но от того не менее сложных технологиях. Но редко доводилось сталкиваться с людьми, которые умеют исследовать тему так глубоко, как Алексей Хлебников.

Я имею удовольствие работать и дружить с ним на протяжении вот уже нескольких лет и с нетерпением жду возможности с головой погрузиться в эту книгу. Вне всяких сомнений, эта тема заслуживает самого пристального внимания.

В этой книге мы совершим путешествие по основам OpenSSL, криптографии вообще, криптографическим режимам, «прелестям» сертификатов и установлению TLS-подключений. И с мельчайшими деталями, если я правильно понимаю, что делает Алексей, а я уверен, что понимаю правильно.

Это важная книга, в ней подробно рассматриваются технологии, которые мы принимаем как само собой разумеющиеся и к которым прибегаем постоянно, чтобы обезопасить свое присутствие в сети.

В книге достаточно примеров и пошаговых объяснений основополагающих концепций, чтобы вы могли продвигаться без затруднений. Прочитав ее до конца, вы сможете использовать наиболее популярные средства OpenSSL в своих программах как для настольных компьютеров, так и для веба.

Книга, безусловно, представляет интерес для практиков, но будет небезынтересна также руководителям и всем прочим, кто считает безопасность важным вопросом, но мало знает о ней. Не волнуйтесь: чтобы прочитать и понять ее, глубокие знания математики не понадобятся.

Как технарь с длинным послужным списком, я считаю, что узнавать новое или глубже погружаться в темы, которые знаешь поверхностно, – дело полезное и благодарное, поддерживающее нас в форме!

Твой выход, Алексей!

– Ярле Адольфсен
Серийный предприниматель, технический директор bspoke,
бывший технический директор Link Mobility
и пионер компьютерной графики в конце 1980-х – начале 1990-х

Об авторе

Алексей Хлебников более 20 лет профессионально занимается ИТ, где ему довелось попробовать себя в разных ролях: разработчик ПО, системный администратор, инженер DevOps, технический руководитель, архитектор и руководитель проекта. За эти годы Алексей поработал с разными технологиями: безопасность, искусственный интеллект, веб-разработка, встраиваемые приложения, мобильные приложения и робототехника. В частности, Алексей работал в компании Opera Software над знаменитым браузером Opera. Алексей всегда интересовался безопасностью. Он был одним из тех, кто сопровождал модули браузера Opera, относящиеся к безопасности, отвечал за криптографию, SSL/TLS и интеграцию с OpenSSL. Он также был членом архитектурной группы по безопасности, отвечавшей за безопасность Opera. Ныне Алексей живет в Осло, Норвегия, и работает старшим консультантом в компании bspoke AS. Он также возглавляет архитектурную группу у своего нынешнего работодателя.

Прежде всего я хочу поблагодарить свою любимую жену Ларису и нашего сына Дмитрия за постоянную любовь и поддержку, в том числе во время написания этой книги, когда я уделял им меньше внимания. Я также благодарен всем редакторам, менеджерам и другим сотрудникам издательства Packt Publishing, работавшим над этой книгой, и ее техническому рецензенту Крису. Их помощь и ценные советы помогли мне улучшить книгу во благо читателей

О рецензенте

Кшиштоф Квятковский – инженер-криптограф, в фокусе интересов которого находятся исследования по криптографии и их реализация. Он имеет ученую степень магистра математики по вычислительным методам. На протяжении своей 15-летней карьеры Крис работал над различными вопросами криптографии, связи и безопасности программного обеспечения – от встраиваемых до больших распределенных систем. В настоящее время его интересует реализация современных криптографических схем, стойких относительно квантовых технологий, а также помощь организациям, желающим перейти на них.

Я хочу поблагодарить свою чудесную любящую семью, которая понимает мою загруженность и всегда занимает мою сторону

Предисловие

В наши дни безопасность и использование сетей – неотъемлемая особенность программного обеспечения. Современный интернет кишит червями, троянцами, незаконными посредниками и другими угрозами. Поэтому обеспечение безопасности важнее, чем когда-либо в прошлом.

OpenSSL – один из самых широко используемых и важных проектов с открытым исходным кодом в интернете – предназначен именно для этой цели. Если вы разработчик ПО, системный администратор, инженер по сетевой безопасности или специалист по DevOps, то, скорее всего, сталкивались с этим комплектом инструментов в прошлом. Но как использовать его с наибольшей пользой? Из этой книги вы узнаете о самых важных возможностях OpenSSL и получите представление обо всем его потенциале.

В книге приводятся пошаговые объяснения основ криптографии и сетевой безопасности, а также практические примеры, иллюстрирующие эти идеи. Мы начнем с простого – как выполнить симметричное шифрование и вычислить хеш-значение (digest) сообщения. Затем пойдем дальше и поговорим о MAC и HMAC, открытых и закрытых ключах и цифровых подписях. По ходу дела вы узнаете о сертификатах стандарта X.509, инфраструктуре открытых ключей и TLS-подключениях.

Прочитав книгу, вы сможете использовать наиболее популярные средства OpenSSL, что позволит реализовать криптографическую защиту и TLS в своих приложениях и сетевой инфраструктуре.

Целевая аудитория

Эта книга ориентирована на разработчиков ПО, системных администраторов, специалистов по DevOps, инженеров по сетевой безопасности и аналитиков, которые хотят обезопасить свои приложения и инфраструктуру. Разработчики узнают, как использовать библиотеку OpenSSL, чтобы включить в свои программы криптографические средства и TLS. Специалисты по DevOps и системные администраторы научатся работать с криптографическими ключами и сертификатами из командной строки и узнают, как организовать миниатюрный удостоверяющий центр для своей организации. Предполагается знакомство с основами безопасности и сетевых технологий.

Структура книги

В главе 1 «OpenSSL и другие библиотеки SSL/TLS» в общих чертах описывается, что такое OpenSSL и его сильные стороны, рассматривается история

OpenSSL и новые возможности, появившиеся в OpenSSL 3.0. Мы также сравним OpenSSL с другими библиотеками SSL/TLS.

В главе 2 «Симметричное шифрование и расшифрование» рассматриваются важные понятия симметричного шифрования: шифры, режимы шифрования и дополнение. Мы поговорим о современных шифрах, режимах шифрования и типах дополнения и дадим совет, какую технологию использовать в конкретной ситуации. Применение этих технологий иллюстрируется с помощью команд и кода на C.

Глава 3 «Хеш-значения сообщений» посвящена хеш-значениям сообщений, или, как их еще называют, криптографическим хешам: зачем они нужны и как используются. Мы дадим обзор современных криптографических функций хеширования, которые вычисляют хеш-значения сообщений, и порекомендуем, какие функции в каких ситуациях использовать. Вычисление хеш-значений иллюстрируется с помощью команд и кода на C.

В главе 4 «MAC и HMAC» объясняется, зачем нужен код аутентичности сообщений (имитовставка, *англ.* Message Authentication Codes – MAC) и где он используется. Затем обсуждается популярный тип MAC – имитовставка на основе односторонних хеш-функций (*англ.* hash-based MAC – HMAC). Мы также узнаем о том, как сочетать HMAC с шифрованием, и о принципе криптографической обреченности. Вычисление HMAC иллюстрируется примером кода.

В главе 5 «Формирование ключа шифрования из пароля» показано, почему сам пароль нельзя использовать для шифрования, а необходимо вывести из него ключ. Мы дадим обзор современных функций формирования ключа и порекомендуем, в каких ситуациях какую использовать. Формирование ключа шифрования иллюстрируется с помощью команд и кода на C.

В главе 6 «Асимметричное шифрование и расшифрование» речь пойдет о том, зачем нужно асимметричное шифрование, как оно работает и какую роль в шифровании и расшифровании играют закрытые и открытые ключи. Шифрование и расшифрование с применением RSA иллюстрируются с помощью команд и кода на C.

В главе 7 «Цифровые подписи и их проверка» мы расскажем, для чего нужны цифровые подписи и как они используются. Мы дадим обзор современных алгоритмов цифровой подписи, в частности RSA, ECDSA и EdDSA, и порекомендуем, какую схему в какой ситуации использовать. Формирование и проверка цифровой подписи иллюстрируются с помощью команд и кода на C.

В главе 9 «Сертификаты X.509 и инфраструктура открытых ключей» подробно описывается, что такое сертификаты X.509, зачем они нужны и где используются. Мы объясним, как одни сертификаты применяются для подписания других, как формируются цепочки сертификатов, а также что такое **инфраструктура открытых ключей** (Public Key Infrastructure – PKI) и как проверка сертификатов используется для проверки идентификаторов, например идентификаторов сайтов. Применение этих методов иллюстрируется с помощью команд и кода на C.

В главе 9 «Установление TLS-подключений и передача по ним данных» подробно рассматривается, что такое протокол TLS, зачем он нужен и по-

чему так широко используется. Мы также узнаем, в чем разница между SSL и TLS. Затем научимся устанавливать и разрывать TLS-подключение, а также передавать и принимать по нему данные. Работа с TLS иллюстрируются с помощью команд и кода на C.

Глава 10 «Использование сертификатов X.509 в TLS» рассматривается, как работать с сертификатами X.509 в TLS и почему сертификаты так важны для TLS. Мы также узнаем, как проверить удаленный сертификат. Затем сделаем еще один шаг и научимся проверять действительность сертификата с помощью CRL и OCSP. Наконец, мы покажем, как использовать клиентский сертификат. Работа с сертификатами иллюстрируется с помощью команд и кода на C.

В главе 11 «Специализированные применения TLS» мы познакомимся с такими вопросами, как закрепление сертификатов, использование неблокирующего режима сетевого взаимодействия и установление TLS-подключений по нестандартным сокетам с помощью базовых объектов ввода-вывода OpenSSL (Basic Input-Output Object – BIO). Применение описанных методов иллюстрируется с помощью кода на C.

В главе 12 «Эксплуатация мини-УЦ» приводятся инструкции, как обустроить собственный миниатюрный удостоверяющий центр (УЦ) для управления сертификатами и построения инфраструктуры открытых ключей в организации. Эксплуатация мини-УЦ иллюстрируется с помощью команд и кода на C.

Как извлечь максимум пользы из этой книги

Вам понадобится установить OpenSSL на свой компьютер, чтобы выполнять примеры команд и кода на C. Если вы этого еще не сделали, то в главе 2 найдете подробные инструкции. Для сборки примеров потребуются совместимый со стандартом C11 компилятор C и компоновщик. Эти программы следует установить в соответствии с инструкциями в документации. Все примеры были протестированы на Kubuntu Linux 22.04 с использованием компилятора GNU C, компоновщика GNU (LD) и программы сборки GNU Make из вышеупомянутого дистрибутива Linux. Другие инструменты разработки, например LLVM Clang или Microsoft Visual C++, также должны быть совместимы с приведенными примерами кода.

ПО, рассматриваемое в этой книге	Системные требования
OpenSSL 3.0	• Linux, FreeBSD, macOS, Windows или любая другая ОС, поддерживаемая OpenSSL • Компилятор C, например GNU C Compiler • Компоновщик, например GNU Linker (LD) • Среда разработки на C/C++ или редактор кода (по собственному усмотрению) • Средство сборки, например GNU Make (факультативно)

Если вы пользуетесь цифровой версией этой книги, рекомендуем набирать код самостоятельно или брать его из репозитория книги на

GitHub (ссылка приведена в следующем разделе). Это позволит избежать потенциальных ошибок, связанных с копированием и вставкой кода.

Хотя объяснения средств OpenSSL и примеры кода иногда весьма детальны, эта книга задумана как руководство, а не замена документации OpenSSL. Если вас заинтересует какая-то функциональность OpenSSL, не рассмотренная в книге, обращайтесь к документации или к исходному коду OpenSSL либо просто экспериментируйте в своем коде!

Графические выделения

В этой книге применяется ряд соглашений о наборе текста.

`CodeInText`: код в тексте, имена таблиц базы данных, папок и файлов, расширения имен файлов, пути к файлам, данные, вводимые пользователем, и адреса в Твиттере. Например: «Открытые ключи пользователя SSH хранятся на сервере в файле `authorized_keys`».

Блок кода выглядит так:

```
if (pinned_server_cert)
    X509_free(pinned_server_cert);
if (pinned_server_cert_file)
    fclose(pinned_server_cert_file);
```

Ввод и вывод команд печатаются следующим образом:

```
$ ./tls-server 4433 server_keypair.pem server_cert.pem
*** Listening on port 4433
```

Полужирный: новые термины и важные слова, а также части пользовательского интерфейса. Так выделяются команды меню и текст в диалоговых окнах, например: «Уменьшается потребность в обслуживании, потому что не нужно создавать **запрос на подписание сертификата (CSR)** и обращаться в УЦ. Можно даже использовать самоподписанный сертификат».

Часть I

ВВЕДЕНИЕ

Любое путешествие начинается первым шагом. Наш первый шаг – понять, что такое OpenSSL, познакомиться с историей и узнать о том, что нового появилось в версии OpenSSL 3. Мы также дадим обзор других библиотек SSL/TLS, конкурирующих с OpenSSL.

Эта часть состоит всего из одной главы:

- главы 1 «OpenSSL и другие библиотеки SSL/TLS».

Глава 1

OpenSSL и другие библиотеки SSL/TLS

В настоящее время разработчикам, которые хотят включить в свои программы криптографические средства или SSL/TLS, доступно несколько библиотек. В этой главе мы сравним их и расскажем о сильных сторонах OpenSSL. Будут также рассмотрены история OpenSSL и новые возможности, появившиеся в версии OpenSSL 3.0.

В этой главе разбираются следующие темы:

- что такое OpenSSL;
- история OpenSSL;
- что нового в OpenSSL 3.0;
- сравнение OpenSSL с GnuTLS;
- сравнение OpenSSL с NSS;
- сравнение OpenSSL с Botan;
- сравнение OpenSSL с облегченными библиотеками TLS;
- сравнение OpenSSL с LibreSSL;
- сравнение OpenSSL с BoringSSL.

Что такое OpenSSL?

OpenSSL – комплект программ с открытым исходным кодом, включающий библиотеку **криптографических** средств и **SSL/TLS**, а также командные утилиты, которые пользуются этой библиотекой и предоставляют полезную функциональность в форме командной строки, например генерирование ключей шифрования и **сертификатов X.509**. Основная часть OpenSSL – именно библиотека, поэтому комплект ориентирован прежде всего на разработчиков ПО. Однако системные администраторы и инженеры DevOps также найдут командные утилиты OpenSSL весьма полезными.

Аббревиатура **SSL** означает **Secure Sockets Layer** (уровень безопасных сокетов). Это протокол, призванный обеспечить безопасное взаимодействие по небезопасным компьютерным сетям. Под небезопасной компьютерной сетью понимается сеть, в которой передаваемые данные можно прочитать или даже изменить, воспользовавшись вредоносным промежуточным сетевым узлом. Примером такой небезопасной сети является интернет. Безопасным называется взаимодействие, при котором передаваемые данные нельзя ни прочитать, ни изменить. SSL обеспечивает безопасность, применяя средства **симметричной** и **асимметричной** криптографии. Протокол SSL был изобретен в 1995 году корпорацией *Netscape Communications*, а в 2015-м его сменил протокол TLS. Аббревиатура **TLS** означает **Transport Layer Security** (безопасность транспортного уровня).

Комплект инструментов OpenSSL был создан еще до того, как протокол SSL был объявлен нерекомендуемым, поэтому в названии упоминается «SSL», а не «TLS».

Исторически OpenSSL распространялся на условиях *лицензии типа BSD*, но начиная с версии 3.0 распространяется на условиях *лицензии Apache 2.0*, которая также принадлежит семейству BSD. Эта лицензия разрешает использовать OpenSSL в приложениях как с открытым, так и с закрытым исходным кодом.

OpenSSL поддерживает великое множество **криптографических алгоритмов**, т. е. алгоритмов **симметричного** и **асимметричного шифрования**, **цифровой подписи, вычисления хеш-значений сообщений** и **обмена ключами**. OpenSSL поддерживает **сертификаты X.509**, протоколы **SSL**, **TLS** и **DTLS**, а также другие, менее распространенные криптографические технологии.

OpenSSL существует уже довольно давно и за годы разработки стал поддерживать много операционных систем. Первоначально OpenSSL разрабатывался для Unix-подобных ОС и до сих поддерживает различные варианты Unix, включая *GNU/Linux*, *BSD*, а также старые и новые коммерческие версии Unix, например *IBM AIX* и *macOS*. OpenSSL также поддерживает другие популярные операционные системы, в частности *Microsoft Windows*, мобильные ОС типа *Android* и *iOS* и даже такие старые и экзотические ОС, как MS-DOS и VMS.

За долгие годы разработки в библиотеку были включены многочисленные оптимизации, например *ассемблерные вставки* для большинства популярных процессорных архитектур, в т. ч. **x86**, **x86_64** и **ARM**. В настоящее время OpenSSL является одной из самых быстрых библиотек криптографии и TLS.

В силу своей универсальности, поддержки большого числа алгоритмов и операционных систем, а также быстродействия OpenSSL стала промышленным стандартом де-факто. OpenSSL так популярна, что другие библиотеки TLS включают так называемые **уровни совместимости с** OpenSSL, чтобы с ними можно было работать, применяя **интерфейсы прикладного программирования** (**API**) OpenSSL.

OpenSSL – очень популярная библиотека, но как она пришла к столь единодушной поддержке? Разберемся – и для этого поговорим об истории OpenSSL.

История OpenSSL

Библиотека OpenSSL основана на **библиотеке SSLeay**, написанной Эриком Эндрю Янгом. Суффикс *eay* в названии SSLeay как раз и означает *Eric Andrew Young*. Разработка SSLeay началась в 1995 году как реализация с открытым исходным кодом библиотеки SSL. В то время библиотека *NSS* еще не была доступна. Впоследствии к команде разработчиков присоединился Тим Хадсон. Но в 1998 году Эрика и Тима наняла на работу компания *RSA Corporation*, и у них не стало хватать времени на дальнейшую разработку SSLeay.

В 1998 году от SSLeay ответвился проект OpenSSL, т. е. OpenSSL стала преемником SSLeay. Членами-основателями стали Марк Кокс, Ральф Энгельшалл, Стивен Хенсон, Бен Лаури и Пол Саттон. Первая версия OpenSSL, с номером 0.9.1, была выпущена 23 декабря 1998 года, всего через неделю после перехода Эрика и Тима в RSA Corporation, когда работа над SSLeay фактически прекратилась.

За прошедшие годы многие люди и компании дописывали код и выполняли другие работы, связанные с OpenSSL. Список компаний, внесших свой вклад, впечатляет: Oracle, Siemens, Akamai, Red Hat, IBM, VMware, Intel и Arm, – и это далеко не все.

В настоящее время разработку OpenSSL координирует управляющий комитет OpenSSL, насчитывающий семь членов. Команда разработчиков ядра, имеющая право фиксации изменений, состоит примерно из 20 человек. И только двое посвящают все свое время работе над OpenSSL. Остальные занимаются этим либо в свободное время, либо по поручению компаний-работодателей.

История OpenSSL интересна, но какое будущее ей уготовано? Рассмотрим последние изменения, внесенные в комплект.

Что нового в OpenSSL 3.0?

Одним из главных изменений в **OpenSSL 3.0** стал порядок лицензирования. Лицензия программного проекта редко изменяется за время его существования. До выхода версии 3.0 проект OpenSSL распространялся на условиях лицензии *в духе BSD*, а начиная с версии 3.0 – на условиях лицензии *Apache License 2.0*.

В версии OpenSSL 3.0 произошли значительные изменения во внутренней архитектуре библиотеки. Архитектурные изменения не закончились и будут продолжены в OpenSSL 4.0. Введена концепция **поставщиков реализации операций OpenSSL**. **Поставщиком** называется часть кода, предоставляющая реализацию криптографических алгоритмов. Существующий криптографический код в OpenSSL доступен в основном через поставщиков *Default* и *Legacy*. **Движки** по-прежнему поддерживаются, но предпочтение отдается поставщикам. В OpenSSL 4.0 поддержка API движков может быть прекращена. Поддерживаются также сторонние поставщики, что позволяет

независимым разработчикам включать свои криптографические алгоритмы в OpenSSL.

Еще одна интересная возможность OpenSSL 3.0 – **ядерный TLS** (Kernel TLS – **KTLS**). Приложение, пользующееся KTLS, может создать специальный **TLS-сокет**, похожий на TCP-сокет. Затем OpenSSL выполняет процедуру **TLS-квитирования** и передает согласованный ключ шифрования и другие данные **ядру операционной системы** в виде **параметров TLS-сокета**. После этого передачей данных по протоколу TLS занимается код KTLS. Такой перенос нагрузки TLS на ядро может ускорить передачу данных в высоконагруженных системах, где производительность очень важна, особенно если ядро может воспользоваться аппаратными средствами ускорения шифра **Advanced Encryption Standard** (**AES**) и, следовательно, разгрузить главный процессор. Конечно, поддержка KTLS необходима как в самой библиотеке TLS, так и в ядре операционной системы. На момент написания этой книги KTLS поддерживали только Linux и FreeBSD.

Из других существенных изменений в OpenSSL 3.0 отметим следующие:

- поддержка **протокола управления сертификатами**;
- простейший HTTP/HTTPS-клиент;
- **режим счетчика с аутентификацией Галуа с синтетическим вектором инициализации** (**AES-GCM-SIV**) для шифра AES;
- новые алгоритмы вычисления **имитовставки** (**MAC**), в частности **GMAC** и **KMAC**;
- новые алгоритмы **функций формирования ключа** (**KDF**), в частности **SSKDF** и **SSHKDF**;
- новые высокоуровневые API, например **EVP_MAC**, **EVP_KDF** и **EVP_RAND**;
- низкоуровневые API объявлены нерекомендуемыми, предпочтение отдается новым высокоуровневым API;
- почищен код;
- изменен механизм обработки ошибок;
- старые небезопасные алгоритмы недоступны, если задан уровень безопасности по умолчанию;
- из командной программы `openssl` исключен интерактивный режим.

OpenSSL – зрелый пакет программ, т. е. самые важные средства в нем уже реализованы. Поэтому последние изменения не обещают пользователям много новой функциональности. В последней версии основное внимание уделено архитектурным улучшениям библиотеки.

Хотя OpenSSL – самая популярная библиотека криптографии и TLS, она не единственная. В следующих разделах мы сравним OpenSSL с конкурентами.

Сравнение OpenSSL с GnuTLS

GnuTLS – свободная библиотека TLS, созданная для нужд **проекта GNU**. Когда создавалась GnuTLS, большинство приложений в проекте GNU распространялись на условиях *лицензии GPL 2.0*, несовместимой со старой лицен-

зией OpenSSL. Авторы ПО, лицензированного на условиях GPL 2.0, должны были оговаривать исключения из лицензии, если хотели компоновать свой продукт с OpenSSL. GnuTLS изначально лицензировалась на условиях *LGPL 2.0*, поэтому не требовала таких исключений.

В настоящее время GnuTLS лицензируется на условиях *LGPL 2.1*. Эта лицензия разрешает использовать библиотеку в **свободном программном обеспечении с открытым исходным кодом** (free and open source software – **FOSS**). Также разрешено использовать библиотеку в проектах с закрытым кодом, но на определенных условиях, в частности только с **динамической компоновкой**.

GnuTLS не включает криптографических средств, арифметики больших чисел и некоторой другой функциональности, имеющейся в OpenSSL. Вместо этого GnuTLS *использует другие библиотеки*, входящие в **экосистему GNU**, которые предоставляют необходимую функциональность: **Nettle** для криптографии, **GMP** для арифметики больших чисел, **Libtasn1** для **ASN.1** (**Abstract Syntax Notation One** – абстрактная синтаксическая нотация версии 1) и т. д.

У GnuTLS есть интересная особенность – она поддерживает не только сертификаты X.509, но и сертификаты OpenPGP. В отличие от сертификата X.509, который подписывается выпускающей стороной в момент выпуска, сертификат OpenPGP поддерживает так называемую *сеть доверия*, может иметь несколько подписей, которые могут добавляться даже после выпуска сертификата. К сожалению, сертификаты OpenPGP не стали распространенными для **TLS-подключений**.

Если не считать поддержки сертификатов OpenPGP, GnuTLS и ее криптографическая библиотека, Nettle, поддерживают меньше криптографических алгоритмов, чем OpenSSL, а с точки зрения производительности работают чуть медленнее. Впрочем, все популярные алгоритмы GnuTLS и Nettle поддерживают.

Что выбрать, OpenSSL или GnuTLS? Я рекомендую остановиться на GnuTLS, если вы разрабатываете программу, распространяемую на условиях лицензии GPL; в противном случае выбирайте OpenSSL.

Еще одним конкурентом OpenSSL является библиотека NSS, которую мы обсудим в следующем разделе.

Сравнение OpenSSL с NSS

Network Security Services (**NSS**) – исторически первая библиотека SSL/TLS. Начиналась она как код обеспечения безопасности, встроенный в браузер *Netscape Navigator 1.0*, который был выпущен в 1994 году компанией Netscape Communications Corporation. Netscape также изобрела протокол SSL, предшественник TLS. Netscape Navigator сменил Netscape Communicator, за которым последовал браузер Mozilla и, наконец, Firefox. Вскоре после выпуска Netscape Communicator 4.0 в 1997 году код безопасности Netscape был выпущен в виде отдельной библиотеки, получившей название **Hard Core Library** (**HCL**). Впоследствии HCL была переименована в NSS.

Библиотека NSS распространялась на условиях лицензии *Mozilla Public License 2.0*, позволявшей разработчикам приложений использовать ее в программах с открытым и закрытым исходным кодом.

Известные приложения, в которых применяется библиотека NSS, – это приложения компании Mozilla Foundation, в частности Firefox и Thunderbird, некоторые приложения Oracle и Red Hat, а также некоторые приложения, входящие в открытые офисные пакеты LibreOffice и Evolution. Несмотря на то что все это – солидные и уважаемые компании и приложения, а сама библиотека NSS имеет долгую историю, она не снискала такой же популярности, как OpenSSL. Не знаю, почему. Некоторые говорят, что OpenSSL проще пользоваться, а документация у нее лучше.

Что выбрать, NSS или OpenSSL? Я рекомендую OpenSSL из-за лучшей документации и большего сообщества разработчиков и пользователей.

Как для NSS, так и для GnuTLS основной API написан на **C**. С библиотекой **Botan**, которую мы рассмотрим в следующем разделе, дело обстоит иначе.

Сравнение OpenSSL с Botan

Большинство библиотек TLS написаны на **C**, и основной API также рассчитан на C. Библиотека же Botan написана на **C++11** и в версии 3.0 переработана под стандарт **C++17**. Основной API Botan рассчитан на **C++**, но имеются также **привязки** к **C** и **Python**. Сторонние проекты предлагают еще привязки к Ruby, Rust и Haskell. Имеются экспериментальные привязки API к Java и Ocaml.

Стоит отметить, что библиотека Botan хорошо документирована. Botan распространяется на условиях **лицензии BSD с двумя оговорками**, которая разрешает использовать библиотеку в приложениях с открытым и закрытым исходным кодом.

Я рекомендую Botan разработчикам, которые хотят использовать C++ API и готовы смириться с менее популярной библиотекой, имеющей более узкое сообщество разработчиков. Если вам нужен C API, более производительная библиотека и широкое сообщество разработчиков, лучше выбрать OpenSSL.

Библиотека Botan потребляет больше памяти, чем OpenSSL, в отличие от облегченных библиотек TLS, которые мы обсудим далее.

Сравнение OpenSSL с облегченными библиотеками TLS

Есть несколько **облегченных библиотек TLS**. Они ориентированы на рынки встраиваемых устройств, в частности для **интернета вещей** и других небольших устройств, которыми люди пользуются, не считая их компьютерами. Сюда входят платежные терминалы в магазинчиках у дома, умные

часы, умные лампочки и разнообразные промышленные датчики. Обычно такие устройства оснащены *слабыми процессорами*, *небольшой оперативной памятью* и *небольшой внешней памятью*.

Своей облегченностью такие библиотеки TLS чаще всего обязаны *модульности*, т. е. возможности компилировать только необходимые модули, поддерживать меньшее число криптографических алгоритмов и протоколов и раскрывать менее развитый API; то есть они предоставляют разработчикам приложений меньше возможностей и настроек.

Из облегченных библиотек TLS лучше других известны **wolfSSL** (раньше yaSSL, или Yet Another SSL), **Mbed TLS** (раньше PolarSSL) и **MatrixSSL**.

Самой развитой из трех этих библиотек, похоже, является wolfSSL. Она поддерживает много криптографических алгоритмов, в т. ч. такие новые, как **ChaCha20** и **Poly1305**.

В последние версии wolfSSL включены *ассемблерные вставки*, поэтому она работает очень быстро. Согласно *тестам производительности* на сайте wolfSSL, производительность чистого шифрования часто не уступает, а в некоторых случаях значительно превосходит производительность OpenSSL – иногда аж на 50 %. Из другой имеющейся в интернете информации следует, что старые версии wolfSSL заметно медленнее OpenSSL. Если вас интересует производительность, то я рекомендую прогнать тесты производительности на целевом оборудовании, взяв последние версии библиотек. Объясняется это тем, что библиотеки постоянно развиваются и последние версии могут содержать больше оптимизаций.

Mbed TLS и MatrixSSL поддерживают заметно меньше криптографических алгоритмов.

WolfSSL и MatrixSSL имеют две лицензии: *GPL 2.0* и коммерческую. Лицензия GPL 2.0 позволяет использовать библиотеку только в GPL-совместимых FOSS-приложениях. Коммерческая лицензия позволяет применять библиотеку в приложениях *с закрытым исходным кодом*.

Mbed TLS распространяется на условиях лицензии *Apache License 2.0*, позволяющей использовать библиотеку в приложениях как с открытым, так с закрытым исходным кодом.

Хотя авторы облегченных библиотек TLS заявляют, что их творения значительно, примерно в 20 раз, меньше OpenSSL, в конфигурациях по умолчанию они вовсе не так малы. Ниже приведены размеры библиотек, откомпилированных на машине с архитектурой x86_64 и ОС Ubuntu 22.04:

- wolfSSL (версия 5.2.0, libwolfssl.so): 1768 КиБ;
- Mbed TLS (версия 2.28.0, libmbedtls.so + libmbedcrypto.so): 664 КиБ;
- MatrixSSL (версия 4.5.1, libssl_s.a + libcrypt_s.a + libcore_s.a): 1772 КиБ;
- OpenSSL (версия 3.0.2, libssl.so + libcrypto.so): 5000 КиБ.

Чтобы уменьшить размер, пользователь должен сам откомпилировать библиотеку и отключить все ненужные ему модули.

OpenSSL также имеет модульную структуру и позволяет исключать ненужные модули при компиляции. Однако это труднее, чем в случае облегченных библиотек. Какие-то модули OpenSSL трудно исключить, потому что от них зависят другие, хотя и не должны бы. А некоторые модули OpenSSL попросту

очень велики, особенно модуль `x509`, содержащий код для работы с сертификатами X.509. Таким образом, уменьшить размер откомпилированных файлов OpenSSL можно, но не так сильно, как в случае облегченных библиотек.

Используйте облегченную библиотеку TLS, если OpenSSL не помещается в память целевого устройства. В противном случае пользуйтесь полномасштабной библиотекой TLS, например OpenSSL.

Последние две библиотеки, которые мы рассмотрим, ответвились от самой OpenSSL. Разберемся, почему первозданная OpenSSL показалась кому-то неподходящей.

Сравнение OpenSSL с LibreSSL

LibreSSL – ответвление от OpenSSL, созданное в 2014 году в рамках проекта OpenBSD Project как реакция на печально знаменитую уязвимость Heartbleed, обнаруженную в OpenSSL. При разработке LibreSSL ставилась цель повысить *безопасность* и *удобство сопровождения* библиотеки ценой удаления старых, непопулярных и больше не считающихся стойкими криптографических алгоритмов и других средств.

OpenBSD – операционная система на базе Unix, одна из семейства систем BSD, ориентированная на безопасность. Создатели OpenBSD разрабатывают не только ядро операционной системы и утилиты. Еще один известный их проект – OpenSSH. Другие проекты, запущенные как приложения для OpenBSD, – OpenNTPD, OpenSMTPD и т. д. Сейчас к ним добавилась и библиотека LibreSSL.

После разветвления разработчики LibreSSL исключили из OpenSSL много оригинального кода, который сочли устаревшим или небезопасным. Они заявляли, что уже в первую неделю сократили код OpenSSL примерно наполовину. Они также добавили несколько новых криптографических алгоритмов, в частности Advanced Encryption Standard в режиме счетчика с аутентификацией Галуа (AES-GCM) и ChaCha-Poly1305. Кроме добавления и удаления, существующий код был частично переработан и укреплен.

Несмотря на всю нацеленность LibreSSL на повышение безопасности, в ней самой обнаружены некоторые уязвимости, не затронувшие OpenSSL, например CVE-2017-8301.

За последние несколько лет команда OpenSSL также проделала работу над повышением безопасности и пригодности к сопровождению, а еще удалила или объявила нерекомендуемой часть кода – хотя и не так много, как в LibreSSL. Однако много кода и новых возможностей было и добавлено.

У OpenSSL гораздо больше ресурсов для развития, чем у LibreSSL, а это означает, что темп продвижения выше. Например, с конца 2018 по начало 2021 года в LibreSSL было включено примерно *1500 исправлений и дополнений от 36 разработчиков*. За то же время в OpenSSL было включено более *5000 исправлений и дополнений от 276 разработчиков*. Помимо разработчиков ядра, OpenSSL получает код от больших компаний, таких как Oracle, Red Hat, IBM и др.

Хотя основная задача LibreSSL – получить библиотеку TLS для проекта OpenBSD, она также ставит целью поддерживать другие платформы и сохранять совместимость на уровне API с OpenSSL. На момент написания книги API совместим с OpenSSL 1.0.1, но включает не все новейшие API, появившиеся начиная с версии OpenSSL 1.0.2.

Сразу после ответвления от OpenSSL библиотека LibreSSL стала библиотекой TLS по умолчанию в OpenBSD, но на большинстве других платформ готовность принять ее невелика. В большинстве дистрибутивов Linux попрежнему используется OpenSSL, некоторые все же решили попробовать LibreSSL в качестве общесистемной библиотеки, но впоследствии отказались от этого. Большинство поставщиков коммерческих приложений, которые традиционно использовали OpenSSL, решили держаться этого курса и дальше.

В конкуренции с LibreSSL выигрывает OpenSSL. Я рекомендую выбирать OpenSSL, если только вы не пишете программы специально для сообщества OpenBSD, а в таком случае следует предпочесть LibreSSL.

В следующем разделе мы рассмотрим еще одно ответвление от OpenSSL, созданное могущественной корпорацией Google.

Сравнение OpenSSL с BoringSSL

BoringSSL – еще одно разветвление OpenSSL, о котором было объявлено с 2014 года. BoringSSL была создана для нужд корпорации Google. В течение многих лет Google сопровождала собственные дополнения для OpenSSL, используемые в различных продуктах Google, в частности *Chrome*, *Android* и серверной инфраструктуре. Наконец, они решили *разветвить* OpenSSL и поддерживать свою ветвь в качестве отдельной библиотеки.

Как и в случае LibreSSL, Google удалила из BoringSSL значительную часть оригинального кода OpenSSL, отвечавшего за поддержку старых и непопулярных алгоритмов и средств. Google также добавила функциональность, отсутствовавшую в OpenSSL. Например, **CRYPTO_BUFFER** позволяет избавиться от дубликатов сертификатов X.509 в памяти, сократив тем самым ее потребление. Кроме того, она дает возможность исключить входящий в OpenSSL код X.509 и ASN.1 из приложения, если OpenSSL компонуется с приложением статически. А надо сказать, что код X.509 составляет заметную часть OpenSSL.

В отличие от LibreSSL, BoringSSL не ставит себе целью добиться **совместимости на уровне API** с OpenSSL или даже с предыдущими версиями BoringSSL. Google хочет оставить за собой свободу менять API библиотеки по собственному усмотрению. Это разумно, потому что Google контролирует как саму BoringSSL, так и основные программные проекты, в которых она используется, поэтому нетрудно синхронизировать изменения API в BoringSSL и этих проектах. Если не требовать стабильности API, то можно освободить ресурсы, которые иначе пришлось бы зарезервировать для сопровождения старых API.

Но это также означает, что если кто-то за пределами Google захочет использовать BoringSSL, то должен быть готов к *несовместимым изменениям* в API библиотеки, причем в самый неподходящий момент. Это очень неудобно для разработчиков, использующих библиотеку. Понимая это, Google заявляет, что хотя BoringSSL – проект с открытым исходным кодом, он не предназначен для общего пользования.

На мой взгляд, код BoringSSL открыт в основном для сторонних соразработчиков таких проектов Google, как Chrome и Android.

Я не рекомендую использовать BoringSSL в своих приложениях по причине **нестабильности API**. Кроме того, OpenSSL может похвастаться более широкой функциональностью, лучшей документацией и значительно более крупным сообществом.

На этом мы заканчиваем обзор некоторых конкурентов OpenSSL. Теперь вы понимаете основные различия между популярными библиотеками TLS и знаете, какую библиотеку в какой ситуации лучше использовать.

Резюме

В этой главе мы узнали, что такое библиотека OpenSSL и зачем ее использовать. Мы бегло рассмотрели историю OpenSSL и заглянули в ее будущее. Мы поговорили о других библиотеках TLS, конкурирующих с OpenSSL, и описали их сильные и слабые стороны.

Теперь вы должны лучше понимать, какую библиотеку TLS использовать в конкретной ситуации. Если ситуация стандартная, то я рекомендую выбирать OpenSSL, потому что многие другие поступили так же, а это значит, что OpenSSL – самая популярная библиотека и промышленный стандарт де-факто.

Зная, почему следует выбирать OpenSSL, пора узнать, как ей пользоваться. В следующих нескольких главах, носящих практический характер, мы научимся использовать мощные средства OpenSSL – от симметричного шифрования до создания TLS-подключений с сертификатами X.509.