УТВЕРЖДЕН

АДМГ.40002-01 33 01-ЛУ

ПРОГРАММНОЕ СРЕДСТВО КРИПТОГРАФИЧЕСКОЙ ЗАЩИТЫ ИНФОРМАЦИИ СЛЕДОПЫТSSL

Вариант исполнения № 1 (Изм. 3)

Руководство программиста

АДМГ.40002-01 33 01

Листов 56

2023

АННОТАЦИЯ

Настоящий документ является руководством программиста Программного средства криптографической защиты информации СледопытSSL (вариант исполнения № 1) АДМГ.40002-01 (далее – СКЗИ СледопытSSL).

В документе приведены основные характеристики и особенности программы, описаны процедуры вызова программы, а также представлены описания входных и выходных данных.

Перед началом работы с СКЗИ СледопытSSL необходимо ознакомиться с эксплуатационной документацией, перечень которой приведен в документе «Ведомость эксплуатационных документов» АДМГ.40002-01 20 01.

СОДЕРЖАНИЕ

1. НАЗНАЧЕНИЕ И УСЛОВИЯ ПРИМЕНЕНИЯ ПРОГРАММЫ

1.1. Назначение и функции программы

СКЗИ СледопытSSL является функционально законченным программным изделием, предназначенным для использования на мобильном устройстве (МУ), функционирующем под управлением операционной системы (ОС) Аврора, имеющей один из следующих сертификатов соответствия:

– ФСБ России «Требования к средствам защиты информации ограниченного доступа, не содержащей сведений, составляющих государственную тайну, от несанкционированного доступа» по классу защиты не ниже АК2;

– ФСТЭК России «Профиль защиты операционных систем типа «А» четвертого класса защиты» и «Требования по безопасности информации, устанавливающие уровни доверия к средствам технической защиты информации и средствам обеспечения безопасности информационных технологий» по 4 уровню доверия.

Оценка корректности встраивания СКЗИ СледопытSSL в ОС Аврора производится в соответствии с положениями документа «Методика встраивания», согласованного с в/ч 43753.

СКЗИ СледопытSSL совместно с ОС Аврора (версии 4.0.2.249 и выше) предназначено для функционирования на аппаратных платформах, подробная информация о которых приведена в документе «Формуляр» АДМГ.40002-01 30 01.

При обновлении ОС Аврора, имеющей один из перечисленных выше сертификатов соответствия, не требуется проведение дополнительных исследований СКЗИ в следующих случаях:

– неизменности аппаратных платформ и версий ядра ОС Аврора, для которых была произведена оценка корректности встраивания;

– соответствии контрольным суммам, указанным в документе АДМГ.40002-01 30 01.

ПРИМЕЧАНИЕ. Состав компонентов СКЗИ СледопытSSL приведен в документе АДМГ.40002-01 30 01.

СКЗИ СледопытSSL функционирует на МУ с установленной ОС Аврора и представляет собой программный комплекс, предназначенный для защиты информации, которой являются следующие конфиденциальные данные:

– передаваемые через сети общего пользования (например, сеть Интернет) посредством протокола TLS между СКЗИ и доверенным оконечным оборудованием;

– хранящиеся в локальном криптографическом контейнере (криптоконтейнере).

СКЗИ СледопытSSL соответствует документу «Требования к средствам криптографической защиты информации, не содержащей сведений, составляющих государственную тайну» по классу защиты КС2.

СКЗИ СледопытSSL предназначено для использования на территории Российской Федерации.

СКЗИ СледопытSSL предназначено для выполнения следующих криптографических функций:

– защиты (шифрования) информации, передаваемой по линиям связи по протоколу TLS при установлении соединения с серверами, поддерживающими данную функцию. Защищенное TLS-соединение с использованием отечественных криптоалгоритмов устанавливается автоматически при обращении пользователя к серверу, поддерживающему данную возможность.

ПРИМЕЧАНИЕ. Веб-браузер является типичным пользовательским программным средством, активирующим возможность установки защищенного TLS-соединения. В случае установления защищенного соединения по протоколу TLS с использованием криптографических алгоритмов, браузер будет распознавать веб-страницу как надежную и безопасную, на панели инструментов браузера будет отображаться значок  , при касании которого откроется вкладка с надписью «Безопасное соединение»;

– обеспечения TLS-соединения с односторонней аутентификацией (сервера).

ПРИМЕЧАНИЕ. Защищенное TLS-соединение обеспечивается при взаимодействии с сертифицированными СКЗИ, установленными на сервере;

– защиты (шифрования) информации, хранимой в криптоконтейнере;

ПРИМЕЧАНИЕ. Мобильное приложение (МП) «Криптозаметки» является графическим прикладным приложением, реализующим функционал взаимодействия с криптоконтейнером посредством вызова разрешенных функций, описание которых приведено в документе «Правила пользования СКЗИ» АДМГ.40002-01 92 01.

– генерации псевдослучайных последовательностей (посредством датчика случайных чисел (ДСЧ));

– простой или усиленной неквалифицированной электронной подписи (ЭП) (выработки и проверки) файлов с данными пользователей;

– контроля целостности файлов с данными учетной записи пользователя с использованием хеш-функции;

ПРИМЕЧАНИЕ. Выработка и проверка ЭП, контроля целостности файлов с данными учетной записи осуществляются сторонними приложениями с использованием экспортируемых функций СКЗИ СледопытSSL в соответствии с настоящим.

– выработки ключевой информации.

СКЗИ реализует следующие основные возможности:

– защищенное соединение по протоколу TLS с использованием криптографических алгоритмов: ГОСТ 28147-1989, ГОСТ Р 34.10-2012, ГОСТ Р 34.11-2012, ГОСТ Р 34.12-2015, ГОСТ Р 34.13-2015;

– защищенное соединение по протоколу TLS 1.2 в соответствии с рекомендациями по стандартизации Р 1323565.1.020-2020 «Информационная технология. Криптографическая защита информации. Использование российских

криптографических алгоритмов в протоколе безопасности транспортного уровня (TLS 1.2)»;

– выработку ключевой информации с использованием кода безопасности и защиты такой информации (Р 50.1.111-2016);

– формирование транспортных ключевых контейнеров для ключей в соответствии с ГОСТ Р 34.10-2012 (Р 50.1.112-2016);

– выработку общей ключевой информации с использованием разделяемого сторонами пароля и последующей аутентификации при взаимодействии по каналу (Р 50.1.115-2016);

– вычисление кода аутентификации HMAC на основе хеш-функции ГОСТ Р 34.11-2012 (Р 50.1.113-2016);

– порождение ключевого материала на основе функций диверсификации KDF_GOSTR3411_2012_256, KDF_TREE_GOSTR3411_2012_256 (Р 50.1.113-2016);

– создание криптографических сообщений формата CMS в соответствии с документом ТС 26.2.001-2020 (Использование алгоритмов ГОСТ 28147-89, ГОСТ Р 34.11 и ГОСТ Р 34.10 в криптографических сообщениях формата CMS, 2014 г.);

– работу с сертификатами и списками отзыва сертификатов инфраструктуры открытых ключей X.509 с поддержкой алгоритмов ГОСТ 34.10, ГОСТ 34.11 в соответствии с документом ТС 26.2.001-2020.

Задачей ОС Аврора является обеспечение поддержки графических интерфейсов. Система графического интерфейса ОС Аврора обеспечивает однооконный многозадачный графический режим работы, который разрешает одновременное использование нескольких программ: используется 1 прикладная программа, которая запускается в полноэкранном режиме, при этом остальные запущенные МП выполняются в фоновом режиме. Система графического интерфейса предоставляет возможность переключения между текущим открытым МП и МП, работающим в фоновом режиме.

1.2. Условия выполнения программы

СКЗИ СледопытSSL может быть установлено на МУ с ОС Аврора, функционирующей на аппаратных платформах, подробная информация о которых приведена в документе АДМГ.40002-01 30 01.

.

2. ОБРАЩЕНИЕ К ПРОГРАММЕ

В случае использования прикладным программным обеспечением функций СКЗИ СледопытSSL из подраздела 2.2 необходимо проводить оценку влияния по документу «Методика оценки влияния Ф1-343 ДСП», согласованному с в/ч 43753.

Проведение оценки влияния прикладного программного обеспечения, использующего функции из подраздела 2.1 или не использующего функции СКЗИ СледопытSSL, не требуется в случае использования API в соответствии с примером, приведенным в приложении (Приложение 1).

2.1. Работа с API

Для установки TLS-соединения при помощи API в МП, использующих библиотеки Qt, должны быть подключены следующие классы:

– `QSslConfiguration` — настройка параметров SSL-подключения;

– `QSslCipher` — выбор криптографического SSL-шифра;

– `QSslCertificate` — API для работы с x509-сертификатом;

– `QNetworkAccessManager` — базовый класс для отправки и получения HTTP-запросов;

– `QNetworkRequest` — настройка отправляемого HTTP-запроса;

– `QNetworkReply` — класс, содержащий данные полученного HTTP-ответа;

– `QSslSocket` — класс для работы с SSL-сокетом напрямую;

– `QSslError` — класс, содержащий ошибки при установке SSL-соединения.

2.1.1. Начало работы с API

Для настройки параметров конфигурации SSL необходимо использовать следующую команду:

```
//получение конфигурации по умолчанию
QSslConfiguration configuration =
QSslConfiguration::defaultConfiguration();
```

2.1.2. Загрузка SSL-сертификата из pem-файла

ПРИМЕЧАНИЕ. Использование самоподписанного сертификата допускается только в качестве тестового.

Допускается использование любого типа SSL-сертификата формата .pem — как подписанного официальным центром сертификации, так и самоподписанного. Загрузка SSL-сертификата из pem-файла осуществляется с помощью следующей команды:

```
// указание SSL-сертификата
const QString fileName =
QStringLiteral("/home/defaultuser/crypto.pem");
QFile certFile(fileName);
if (!certFile.exists() || !certFile.open(QFile::ReadOnly))
{
return;
}
const QByteArray certData = certFile.readAll();
certFile.close();
QSslCertificate sslCert(certData);
// выбор доверенных сертификатов
QList<QSslCertificate> certs;
certs << sslCert;
configuration.setCaCertificates(certs)
```

2.1.3. Выбор поддерживаемых шифров для подключения

Все соответствующие ГОСТ шифры могут быть выбраны с помощью следующей команды:

```
QList<QSslCipher> ciphers;
for (const auto cipher: QSslConfiguration::supportedCiphers()) {
    if (cipher.name().contains(QLatin1String("GOST"))) {
        ciphers << cipher;
    }
}
// выбор доверенных шифров
configuration.setCiphers(ciphers);
```

Примеры использования API для установки защищенного соединения приведены в приложении (Приложение 1).

2.2. Работа с СКЗИ СледопытSSL

Подробное описание СКЗИ СледопытSSL приведено в документе АДМГ.40002-01 92 01.

2.2.1. Инициализация СКЗИ СледопытSSL

Для инициализации СКЗИ СледопытSSL в самом начале работы программы должен присутствовать вызов следующих функций:

– `SSL_load_error_strings`;

– `OpenSSL_add_ssl_algorithms`;

– `OpenSSL_add_all_algorithms`.

Для генерации ключевого материала, векторов инициализации, одноразовых параметров подписи и других криптографических артефактов МП и библиотеками используется ДСЧ.

ДСЧ входит в состав СКЗИ СледопытSSL, разработан в соответствии с документом Р 1323565.1.012-2017 «Информационная технология. Криптографическая защита информации. Принципы разработки и модернизации шифровальных (криптографических) средств защиты информации» и состоит из следующих компонентов:

– биологический датчик случайных чисел – является графическим компонентом ДСЧ и используется для выработки относительно короткой (несколько сотен бит) начальной последовательности для программного датчика случайных чисел (ПДСЧ);

– ПДСЧ – является программным компонентом ДСЧ и представляет собой основной датчик, используемый для выработки криптографических ключей и других артефактов, а также для быстрого получения больших объемов псевдослучайной информации на основе начального заполнения.

Инициализация ДСЧ осуществляется администратором при первоначальной настройке МУ. В процессе инициализации ПДСЧ устанавливает таймер сроком на 3 года, при срабатывании которого запускается МП «ДСЧ ОС» и начинается процесс переинициализации ДСЧ, если МУ включено и сессия пользователя активна.

ПРИМЕЧАНИЕ. Подробное описание процессов инициализации, переинициализации, а также работы МП «ДСЧ ОС» приведено в документе АДМГ.40002-01 92 01;

Информация о поддерживаемых МУ приведена в документе «Операционная система Аврора. Формуляр» АДМГ.10034-02 30 01 на ОС Аврора, для совместного использования с которой предназначено СКЗИ СледопытSSL.

2.2.2. Реализация протокола TLS

Для установления соединения с сервером по протоколу TLS клиентская программа может выполнить вызов следующих функций СКЗИ:

– `SSL_library_init` – инициализация библиотеки;

– `SSL_CTX_new` – создание контекста TLS-соединения. В качестве аргумента в данную функцию передается указатель на объект библиотеки, определяющий версию используемого протокола;

– `SSL_CTX_set_options` – установка дополнительных опций протокола;

– `SSL_CTX_load_verify_locations` – загрузка цепочки корневых сертификатов для проверки сертификата сервера;

– `BIO_new_ssl_connect` – создание контекста сетевого соединения по протоколу TCP;

– `BIO_set_conn_hostname` – установка сетевого адреса сервера;

– `BIO_get_ssl` – связывание контекста сетевого соединения TCP с контекстом TLS-соединения;

– SSL_set_cipher_list – установка списка поддерживаемых режимов (режим описывает алгоритм согласования ключей, алгоритм шифрования, алгоритм выработки MAC). Для криптоалгоритмов семейства ГОСТ в библиотеке существует 4 режима: «Совместимости 1», «Совместимости 2», «Кузнечик», «Магма». Предпочитаемые режимы передаются в данную функцию в виде текстовой строки, например «kGOST2012256»;

– BIO_do_handshake – выполнение процедуры TLS Handshake;

– SSL_get_peer_certificate и SSL_get_verify_result – проверка переданного сервером сертификата на валидность.

В дальнейшем для обмена с сервером по защищенному каналу клиентская программа может использовать функции BIO_read (принять данные от сервера), BIO_write (передать данные серверу).

Для использования криптоалгоритмов семейства ГОСТ в функцию SSL_set_cipher_list необходимо передавать одну из следующих строк либо их конкатенацию через символ «:»:

– kGOST – режим «Совместимости 1»;

– kGOST2012256 – режим «Совместимости 2», «Кузнечик», «Магма»;

– aGOST01 – режим «Совместимости 1»;

– aGOST2012256 – режим «Совместимости 2», «Кузнечик», «Магма»;

– eGOST201564 – режим «Магма»;

– eGOST2015128 – режим «Кузнечик»;

– eGOST28147CNT – режим «Совместимости 1»;

– eGOST28147CNTZ – режим «Совместимости 2»;

– GOST2015128MAC – режим «Кузнечик»;

– GOST201564MAC – режим «Магма»;

– GOST89MAC – режим «Совместимости 1»;

– GOST89MACZ – режим «Совместимости 2».

При обработке команды `EVP_CTRL_TLS1_2_TLSTREE` выполняется модификация ключа и IV по номеру последовательности (`tlstree`). Для ГОСТ-криптонаборов используется режим «MacThenEncrypt», при этом в OpenSSL поле «sequence» инкрементируется сразу после операции mac (т.к. предполагается, что для шифрования номер последовательности не требуется), поэтому в ГОСТ-модуле при шифровании последовательность корректируется на «-1». Т.к. в OpenSSL для tls12 не предусмотрена установка IV и ключа для каждой операции шифрования, модификация и установка и ключа, и IV выполняется в ГОСТ-модуле по команде `EVP_CTRL_TLS1_2_TLSTREE`.

Выполняется контроль максимального количества записей в соединении. Контроль осуществляется в функции `tls1_mac()` сразу после инкрементирования поля «seq». Если все байты поля после инкрементирования имеют значение 0 (4 байта [4..7] для «Магмы»), то текущая запись имеет максимальный номер 0xffffffffffffffff (xffffffff для «Магмы»). Функция возвращает значение 0, что приводит к завершению соединения с ошибкой «Bad MAC». Запись с этим номером не будет отправлена.

Поддерживается «Renegotiation_info» – расширение TLS по рекомендации RFC 5746. Данное расширение используется клиентом и сервером для связи начального `handshake` с последующим пересогласованием. Клиент сообщает серверу о поддержке данного режима, посылая в «ClientHello» либо пустое расширение «Renegotiation_info», либо фиктивный криптонабор «empty_reneg_info_scsv». Сервер всегда посылает в ответ в «ServerHello» пустое расширение.

Несмотря на то, что для OpenSSL наиболее «совместимым» решением является отправка фиктивного криптонабора (т.к. старые серверы могут разрывать соединение при получении не поддерживаемого расширения), рекомендация требует от клиента посылать пустое расширение. Реализовано в соответствии с рекомендацией, изменены функции `tls_construct_ctos_renegotiate()` и `ssl_cipher_list_to_bytes()`.

Расширение «encrypt_then_mac». Режим «EncryptThenMac» является расширением tls12, которое всегда посылает клиент. Если сервер также посылает данное расширение, то устанавливается признак «s->ext.use_etm = 1» и используется данный режим. Для того, чтобы для ГОСТ-криптонаборов использовался только режим «MacThenEncrypt», добавлена проверка поля «s >s3->tmp.new_cipher->id» в функции `tls_parse_stoc_etm()` при получении расширения от сервера.

Длина поля «verify_data» сообщения «Finished» по умолчанию для tls12 – 12, по рекомендации – 32. В функции `tls1_final_finish_mac()` реализуется проверка: в случае использования ГОСТ-криптонабора используется длина 32.

О кодировании открытых ключей в сообщении «ClientKeyExchange». Согласно рекомендации, поле «SubjectPublicKeyInfo» должно кодироваться в соответствии с ГОСТ Р ИСО/МЭК 9594-8, раздел 8. При этом параметр дайджеста в составе поля «AlgorithmIdentifier» является опциональным. Способы кодирования ключей для разных типов эллиптических кривых (ЭК) и необходимость наличия поля дайджеста определяются стандартом Р 1323565.1.023-2018, в соответствии с которым поле дайджеста для новых типов ЭК (tc26) использовать не рекомендуется (а для `tc256_paramsetBCD` - запрещается). В целях совместимости дайджест отсутствует только в `tc26_256_paramsetBCD`. Выполняется в функции `encode_gost_algor_params() (gost_ameth.c)`.

О версии протокола в заголовке записи сообщения ClientHello. OpenSSL для совместимости посылает значение 0x301, что соответствует TLS1. Согласно рекомендации следует использовать 0x303. Изменения внесены в функцию `do_ssl3_write()` файл `rec_layer_s3.c`.

В настоящее время приняты обновленные международные идентификаторы (`https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml`), которые также указаны в проекте обновления рекомендации. В текущей версии использованы обновленные значения.

Криптонаборы:

– 0xC1,0x00 TLS_GOSTR341112_256_WITH_KUZNYECHIK_CTR_OMAC;

– 0xC1,0x01 TLS_GOSTR341112_256_WITH_MAGMA_CTR_OMAC.

Алгоритмы подписи:

– 64 (0x40) gostr34102012_256;

– 65 (0x41) gostr34102012_512.

Типы сертификатов:

– 67 gost_sign256;

– 68 gost_sign512.

2.2.3. Генерация псевдослучайной последовательности

Получение последовательностей случайных чисел осуществляется из библиотеки СКЗИ СледопытSSL посредством ее обращения к `randseed.service` по сокету и вызову функции `RAND_bytes(rand_buffer, length)`, выполняющей выработку псевдослучайных чисел с использованием криптоалгоритмов семейства ГОСТ. Получать последовательности случайных чисел может МП, работающее с любыми правами.

Пример кода:

```
unsigned char psp_buf[32]
RAND_bytes(psp_buf, 32)
```

Поскольку функция `RAND_bytes` способна вернуть ошибку, не выдав случайные данные, необходимо использовать `RAND_pseudo_bytes` в качестве функции, которая всегда выдает данные.

2.2.4. Работа с алгоритмами блочного шифрования семейства ГОСТ

Доступ к функциям криптоалгоритмов блочного шифрования реализуется через объект структуры типа: `EVP_CIPHER` – симметричный криптоалгоритм.

В СКЗИ СледопытSSL определены следующие объекты типа `EVP_CIPHER` для криптоалгоритмов семейства ГОСТ:

– `cipher_gost` – алгоритм ГОСТ 28147-1989 в режиме CFB (идентификатор `NID_id_Gost28147_89`);

– `cipher_gost_cbc` – алгоритм ГОСТ 28147-1989 в режиме CBC (идентификатор `NID_gost89_cbc`);

– `cipher_gost_cpacnt` – алгоритм ГОСТ 28147-1989 в режиме OFB (идентификатор `NID_gost89_cnt`);

– `cipher_gost_cpcnt_12` – алгоритм ГОСТ 28147-1989 в режиме OFB с таблицей подстановок `Gost28147_TC26ParamSetZ` (идентификатор `NID_gost89_cnt_12`);

– `grasshopper_ecb` – алгоритм ГОСТ 34.12-2015 128 бит в режиме ECB (идентификатор `NID_grasshopper_ecb`);

– `grasshopper_ctr` – алгоритм ГОСТ 34.12-2015 128 бит в режиме CTR (идентификатор `NID_grasshopper_ctr`);

– `grasshopper_ofb` – алгоритм ГОСТ 34.12-2015 128 бит в режиме OFB (идентификатор `NID_grasshopper_ofb`);

– `grasshopper_cbc` – алгоритм ГОСТ 34.12-2015 128 бит в режиме CBC (идентификатор `NID_grasshopper_cbc`);

– `grasshopper_cfb` – алгоритм ГОСТ 34.12-2015 128 бит в режиме CFB (идентификатор `NID_grasshopper_cfb`);

– `magma_ecb` – алгоритм ГОСТ 34.12-2015 64 бит в режиме ECB (идентификатор `NID_magma_ecb`);

– `magma_ctr` – алгоритм ГОСТ 34.12-2015 64 бит в режиме CTR (идентификатор `NID_magma_ctr`);

– `magma_ofb` – алгоритм ГОСТ 34.12-2015 64 бит в режиме OFB (идентификатор `NID_magma_ofb`);

– magma_cbc – алгоритм ГОСТ 34.12-2015 64 бит в режиме CBC (идентификатор NID_magma_cbc);

– magma_cfb – алгоритм ГОСТ 34.12-2015 64 бит в режиме CFB (идентификатор NID_magma_cfb).

Для выполнения шифрования программа может выполнить вызов следующих функций СКЗИ:

– ENGINE_by_id с параметром «gost» - для получения объекта типа «engine», предоставляющего прикладным программам набор функций по выполнению криптографических операций семейства ГОСТ;

– ENGINE_get_cipher – для получения объекта типа EVP_CIPHER, соответствующего заданному алгоритму шифрования;

– EVP_CIPHER_CTX_new – для создания контекста шифрования;

– EVP_CipherInit – для установки ключа, начального заполнения и задания режима работы шифратора (зашифрование или расшифрование);

– EVP_CIPHER_CTX_ctrl – для установки узлов замены шифратора;

– EVP_CipherUpdate – для выполнения непосредственной операции шифрования данных.

Пример кода зашифрования данных:

```
ENGINE *engine;
const EVP_CIPHER *cipher;
EVP_CIPHER_CTX *ctx;
unsigned char cr_text[16];
int cr_size;

engine = ENGINE_by_id("gost");
cipher = ENGINE_get_cipher(engine, NID_id_Gost28147_89);

ctx = EVP_CIPHER_CTX_new();
EVP_CipherInit(ctx, cipher, Test_Key_GOST_1989, Test_Iv_GOST_1989, 1);
EVP_CIPHER_CTX_ctrl(ctx, EVP_CTRL_INIT,
NID_id_Gost28147_89_TestParamSet, NULL);
EVP_CipherUpdate(ctx, cr_text, &cr_size, Test_Data_GOST_1989,
sizeof(Test_Data_GOST_1989));
```

Следует иметь в виду, что способ передачи параметра `num` в функцию `EVP_CIPHER_CTX_ctrl (ctx, EVP_CTRL_GOST_SET_M, num, pointer)` зависит от режима шифрования: иногда это длина вектора инициализации в байтах, а иногда – в блоках шифра.

Длина блока шифра «Магма» составляет 8 байт, а шифра «Кузнечик» - 16 байт. Функция `EVP_CIPHER_CTX_block_size` возвращает другие значения.

В функцию `EVP_CipherInit` параметр `int enc` необходимо передавать равным 1 для зашифрования, и 0 для расшифрования.

Шифровать в режимах ECB и CBC допускается только данные, размер которых кратен длине блока шифра (8 байт для «Магмы», 16 байт для «Кузнечика»). Результатом зашифрования функциями `C_CipherUpdate` и `C_CipherFinal` является шифротекст (длины, равной длине открытого текста) и незначащее дополнение длиной в 1 блок. При расшифровании необходимо передать данный шифротекст и 1 блок произвольных данных.

2.2.5. Работа с алгоритмами вычисления аутентификационных кодов (MAC) семейства ГОСТ

Доступ к функциям криптоалгоритмов вычисления аутентификационных кодов реализуется через объект структуры типа: `EVP_MD` – криптоалгоритм вычисления MAC.

В СКЗИ СледопытSSL определены следующие объекты типа `EVP_MD` для криптоалгоритмов семейства ГОСТ:

– `imit_gost_cpa` – алгоритм ГОСТ 28147-1989 в режиме выработки имитовставки с таблицей подстановок `Gost28147_CryptoProParamSetA` (идентификатор `NID_id_Gost28147_89_MAC`);

– `imit_gost_cp_12` – алгоритм ГОСТ 28147-1989 в режиме выработки имитовставки с таблицей подстановок `Gost28147_TC26ParamSetZ` (идентификатор `NID_gost_mac_12`);

– digest_gost – алгоритм хеширования ГОСТ 34.11-1994 (идентификатор NID_id_GostR3411_94);

– digest_gost2012_256 – алгоритм хеширования ГОСТ 34.11-2012 256 бит (идентификатор NID_id_GostR3411_2012_256);

– digest_gost2012_512 – алгоритм хеширования ГОСТ 34.11-2012 512 бит (идентификатор NID_id_GostR3411_2012_512);

– grasshopper_mac – алгоритм ГОСТ 34.12-2015 128 бит в режиме выработки имитовставки (идентификатор NID_grasshopper_mac);

– magma_mac – алгоритм ГОСТ 34.12-2015 64 бит в режиме выработки имитовставки (идентификатор NID_magma_mac).

Для вычисления аутентификационного кода сообщения программа может выполнить вызов следующих функций СКЗИ:

– ENGINE_by_id с параметром «gost» - для получения объекта типа «engine», предоставляющего прикладным программам набор функций по выполнению криптографических операций семейства ГОСТ;

– ENGINE_get_digest – для получения объекта типа EVP_MD, соответствующего заданному алгоритму вычисления аутентификационного кода;

– EVP_DigestInit – для инициализации объекта типа EVP_MD;

– EVP_DigestUpdate – для подсчета аутентификационного кода на очередной блок сообщения;

– EVP_DigestFinal – для завершения процесса подсчета аутентификационного кода и его получения.

Пример кода вычисления MAC:

```
ENGINE *engine;
const EVP_MD *digest;
EVP_MD_CTX ctx;
uint8_t hash[32];
unsigned int len;

engine = ENGINE_by_id("gost");
digest = ENGINE_get_digest(engine, NID_id_GostR3411_2012_256);
```

```
EVP_DigestInit(&ctx, digest);
EVP_DigestUpdate(&ctx, Test_Data_1_GOST_3411_2012,
sizeof(Test_Data_1_GOST_3411_2012));
EVP_DigestFinal(&ctx, hash, &len);
```

2.2.6. Формирование ключей ЭП

Ключи ЭП, формируемые при помощи функций, могут использоваться только в качестве тестовых.

Доступ к функциям формирования ключей ЭП реализуется через объекты структур следующего типа:

– `EVP_PKEY`– объект для работы с ключами ЭП;

– `EC_KEY`– ключ ЭП;

– `EC_GROUP` – параметры ЭК;

– `EC_POINT` – точка ЭК (открытый ключ ЭП).

Для формирования пары закрытого/открытого ключей ЭП программа может выполнить вызов следующих функций СКЗИ:

– `ENGINE_by_id` с параметром «gost» - для получения объекта типа «engine», предоставляющего прикладным программам набор функций по выполнению криптографических операций семейства ГОСТ;

– `EC_GROUP_new_by_curve_name` – для получения ЭК;

– `EC_KEY_new` – создание объекта ключа ЭП;

– `EC_KEY_set_group` – привязка к ключу выбранной ЭК;

– `EVP_PKEY_new` – создание объекта для работы с ключами ЭП;

– `EVP_PKEY_assign` – привязка объекта `EC_KEY` к объекту `EVP_PKEY`;

– `EVP_PKEY_CTX_new` – создание контекста для работы с ключами ЭП;

– `EVP_PKEY_keygen_init` – инициализация контекста генератора ключей;

– `EVP_PKEY_keygen` – генерация закрытого ключа ЭП и формирование соответствующего ему открытого ключа. Выработка ключа выполняется с помощью генератора случайных чисел СКЗИ СледопытSSL с использованием криптоалгоритмов семейства ГОСТ;

– EC_KEY_get0_private_key – получение значения закрытого ключа;

– EC_KEY_get0_public_key – получение значения открытого ключа (в виде точки ЭК);

– EC_POINT_get_affine_coordinates_GFp – преобразование точки ЭК в значение ее координат (извлечение непосредственного значения открытого ключа ЭП).

Пример кода формирования ключей ЭП:

```
EVP_PKEY_CTX *ctx = NULL;
EVP_PKEY *pkey = NULL;
EC_KEY *eckey = NULL;
EC_POINT *ecpoint = NULL;
const BIGNUM *priv = NULL;
const BIGNUM *pubkey_x = NULL;
const BIGNUM *pubkey_y = NULL;
EC_GROUP *ecgroup = NULL;
BN_CTX *bnctx = NULL;
ENGINE *engine;

engine = ENGINE_by_id("gost");

ecgroup =
EC_GROUP_new_by_curve_name(NID_id_tc26_gost_3410_2012_256_paramSetA);
eckey = EC_KEY_new();
EC_KEY_set_group(eckey, ecgroup);

pkey = EVP_PKEY_new();
EVP_PKEY_assign(pkey, NID_id_GostR3410_2012_256, eckey);

ctx = EVP_PKEY_CTX_new(pkey, engine);

EVP_PKEY_keygen_init(ctx);
EVP_PKEY_keygen(ctx, &pkey);

priv = EC_KEY_get0_private_key(EVP_PKEY_get0(pkey));
_BN_bn2binpad(priv, out_priv_key,
OPENSSL_HELPER_GOST_SIGNATURE_KEY_SIZE);

pubkey = EC_KEY_get0_public_key(EVP_PKEY_get0(pkey));
bnctx = BN_CTX_new();
pubkey_x = BN_CTX_get(bnctx);
pubkey_y = BN_CTX_get(bnctx);
EC_POINT_get_affine_coordinates_GFp(EC_KEY_get0_group(EVP_PKEY_get0(pk
ey)), pubkey, pubkey_x, pubkey_y, bnctx);

_BN_bn2binpad(pubkey_x, out_pub,
OPENSSL_HELPER_GOST_SIGNATURE_PUBLIC_KEY_SIZE >> 1);
```

```
_BN_bn2binpad(pubkey_y, ((char*)out_pub) +
(OPENSSL_HELPER_GOST_SIGNATURE_PUBLIC_KEY_SIZE >> 1),
OPENSSL_HELPER_GOST_SIGNATURE_PUBLIC_KEY_SIZE >> 1);
```

Для получения объекта `EC_KEY` из объекта `EVP_PKEY` необходимо использовать метод `EVP_PKEY_get0` с приведением типов.

2.2.7. Работа с алгоритмами вычисления и проверки ЭП семейства ГОСТ

Доступ к функциям криптоалгоритмов вычисления ЭП реализуется через объекты структур следующего типа:

– `EVP_PKEY` – объект для работы с ключами ЭП;

– `EC_KEY` – ключ ЭП;

– `EC_GROUP` – параметры ЭК.

Для формирования ЭП сообщения программа может выполнить вызов следующих функций СКЗИ:

– `ENGINE_by_id` с параметром «gost» - для получения объекта типа «engine», предоставляющего прикладным программам набор функций по выполнению криптографических операций семейства ГОСТ;

– вычислить хеш-код сообщения (как было показано в пункте выше);

– `EC_GROUP_new_by_curve_name` – для получения ЭК. В СКЗИ СледопытSSL встроены следующие кривые семейства ГОСТ:

- OID 1.2.643.2.2.35.0 (идентификатор `NID_id_GostR3410_2001_TestParamSet`);
- OID 1.2.643.2.2.35.1 (идентификатор `NID_id_GostR3410_2001_CryptoPro_A_ParamSet`);
- OID 1.2.643.2.2.35.2 (идентификатор `NID_id_GostR3410_2001_CryptoPro_B_ParamSet`);
- OID 1.2.643.2.2.35.3 (идентификатор `NID_id_GostR3410_2001_CryptoPro_C_ParamSet`);

- OID 1.2.643.2.2.36.0 (идентификатор `NID_id_GostR3410_2001_CryptoPro_XchA_ParamSet`);
- OID 1.2.643.2.2.36.1 (идентификатор `NID_id_GostR3410_2001_CryptoPro_XchB_ParamSet`);
- OID 1.2.643.7.1.2.1.1.1 (идентификатор `NID_id_tc26_gost_3410_2012_256_paramSetA`);
- OID 1.2.643.7.1.2.1.1.2 (идентификатор `NID_id_tc26_gost_3410_2012_256_paramSetB`);
- OID 1.2.643.7.1.2.1.1.3 (идентификатор `NID_id_tc26_gost_3410_2012_256_paramSetC`);
- OID 1.2.643.7.1.2.1.1.4 (идентификатор `NID_id_tc26_gost_3410_2012_256_paramSetD`);
- OID 1.2.643.7.1.2.1.2.0 (идентификатор `NID_id_tc26_gost_3410_2012_512_paramSetTest`);
- OID 1.2.643.7.1.2.1.2.1 (идентификатор `NID_id_tc26_gost_3410_2012_512_paramSetA`);
- OID 1.2.643.7.1.2.1.2.2 (идентификатор `NID_id_tc26_gost_3410_2012_512_paramSetB`).
- OID 1.2.643.7.1.2.1.2.3 (идентификатор `NID_id_tc26_gost_3410_2012_512_paramSetC`);

– `EC_KEY_new` – создание объекта ключа ЭП;

– `EC_KEY_set_group` – привязка к ключу выбранной ЭК;

– `EVP_PKEY_new` – создание объекта для работы с ключами ЭП;

– `EVP_PKEY_assign` – привязка объекта `EC_KEY` к объекту `EVP_PKEY`;

– `EVP_PKEY_CTX_new` – создание контекста для работы с ключами ЭП;

– `EC_KEY_set_private_key` – установка закрытого ключа ЭП;

– `EVP_PKEY_sign_init` – инициализация контекста создания подписи;

– EVP_PKEY_sign – вычисление ЭП на хеш-код.

Для проверки ЭП сообщения программа может выполнить вызов следующих функций СКЗИ:

– ENGINE_by_id с параметром «gost» - для получения объекта типа «engine», предоставляющего прикладным программам набор функций по выполнению криптографических операций семейства ГОСТ;

– вычислить хеш-код сообщения (как показано в пункте выше);

– EC_GROUP_new_by_curve_name – для получения ЭК;

– EC_KEY_new – создание объекта ключа ЭП;

– EC_KEY_set_group – привязка к ключу выбранной ЭК;

– EVP_PKEY_new – создание объекта для работы с ключами ЭП;

– EVP_PKEY_assign – привязка объекта EC_KEY к объекту EVP_PKEY;

– EVP_PKEY_CTX_new – создание контекста для работы с ключами ЭП;

– EC_KEY_set_public_key – установка открытого ключа ЭП;

– EVP_PKEY_verify_init – инициализация контекста проверки подписи;

– EVP_PKEY_verify – проверка ЭП на хеш-код.

Пример кода вычисления ЭП:

```
ENGINE *engine;
EVP_PKEY_CTX *ctx;
EVP_PKEY *ds_key;
EC_KEY *ec_key;
EC_GROUP *ec_group;
uint8_t sign[128];
size_t sign_len;

engine = ENGINE_by_id("gost");

ec_group =
EC_GROUP_new_by_curve_name(NID_id_tc26_gost_3410_2012_512_paramSetTest
);

ec_key = EC_KEY_new();
EC_KEY_set_group(ec_key, ec_group);

ds_key = EVP_PKEY_new();
EVP_PKEY_assign(ds_key, NID_id_GostR3410_2012_512, ec_key);
```

```
ctx = EVP_PKEY_CTX_new(ds_key, engine);

ec_key = EVP_PKEY_get0(ds_key);
EC_KEY_set_private_key(ec_key, priv_key);

EVP_PKEY_sign_init(ctx);
EVP_PKEY_sign(ctx, sign, &sign_len, (const unsigned
char*)Test_Data_GOST_3410_2012_512,
sizeof(Test_Data_GOST_3410_2012_512));
```

Пример кода проверки ЭП:

```
ENGINE *engine;
EVP_PKEY_CTX *ctx;
EVP_PKEY *ds_key;
EC_POINT *ec_point;
EC_KEY *ec_key;
EC_GROUP *ec_group;
int res;

engine = ENGINE_by_id("gost");

ec_group =
EC_GROUP_new_by_curve_name(NID_id_tc26_gost_3410_2012_512_paramSetTest
);

ec_key = EC_KEY_new();
EC_KEY_set_group(ec_key, ec_group);

ds_key = EVP_PKEY_new();
EVP_PKEY_assign(ds_key, NID_id_GostR3410_2012_512, ec_key);

ctx = EVP_PKEY_CTX_new(ds_key, engine);
ec_key = EVP_PKEY_get0(ds_key);

ec_point = EC_POINT_new(EC_KEY_get0_group(EVP_PKEY_get0(ds_key)));
EC_POINT_set_affine_coordinates_GFp(EC_KEY_get0_group(EVP_PKEY_get0((E
VP_PKEY *)ds_key)), ec_point, pub_key_x, pub_key_y, NULL);

EC_KEY_set_public_key(ec_key, ec_point);

EVP_PKEY_verify_init(ctx);

if((res = EVP_PKEY_verify(ctx, sign, 128, (const unsigned
char*)Test_Data_GOST_3410_2012_512,
sizeof(Test_Data_GOST_3410_2012_512))) <= 0)
{
        printf("verify error\n");
}
```

2.2.8. Работа с алгоритмами вычисления аутентификационных кодов на базе хеш-функций (HMAC) семейства ГОСТ

Доступ к функциям криптоалгоритмов вычисления аутентификационных кодов HMAC реализуется через объект структуры типа: `EVP_MD` – криптоалгоритм вычисления MAC.

В СКЗИ СледопытSSL определены следующие объекты типа `EVP_MD` криптоалгоритмов семейства ГОСТ для использования в функциях вычисления HMAC:

– `digest_gost` – алгоритм хеширования ГОСТ 34.11-1994 (идентификатор `NID_id_GostR3411_94`);

– `digest_gost2012_256` – алгоритм хеширования ГОСТ 34.11-2012 256 бит (идентификатор `NID_id_GostR3411_2012_256`);

– `digest_gost2012_512` – алгоритм хеширования ГОСТ 34.11-2012 512 бит (идентификатор `NID_id_GostR3411_2012_512`).

Для вычисления аутентификационного кода HMAC сообщения программа может выполнить вызов следующих функций СКЗИ:

– `ENGINE_by_id` с параметром «gost» - для получения объекта типа «engine», предоставляющего прикладным программам набор функций по выполнению криптографических операций семейства ГОСТ;

– `ENGINE_get_digest` – для получения объекта типа `EVP_MD`, соответствующего заданному алгоритму вычисления аутентификационного кода;

– `HMAC_CTX_init` – инициализация контекста вычисления HMAC;

– `HMAC_Init` – привязка к контексту HMAC объекта `EVP_MD` и установка ключа;

– `HMAC_Update` – для подсчета аутентификационного кода на очередной блок сообщения;

– `HMAC_Final` – для завершения процесса подсчета аутентификационного кода и его получения.

Пример кода вычисления HMAC:

```
ENGINE *engine;
const EVP_MD *digest;
HMAC_CTX ctx;
uint8_t hash[64];
unsigned int len;

engine = ENGINE_by_id("gost");
digest = ENGINE_get_digest(engine, NID_id_GostR3411_2012_512);

HMAC_CTX_init(&ctx);
HMAC_Init(&ctx, Test_Key_GOST_3411_2012_HMAC,
sizeof(Test_Key_GOST_3411_2012_HMAC), digest);

HMAC_Update(&ctx, Test_Data_GOST_3411_2012_HMAC,
sizeof(Test_Data_GOST_3411_2012_HMAC));

HMAC_Final(&ctx, hash, &len);
```

2.2.9. Формирование ключа из пароля в соответствии с Р 50.1.111-2016

Для формирования ключа из пароля программа может выполнить вызов следующих функций СКЗИ:

– `ENGINE_by_id` с параметром «gost» - для получения объекта типа «engine», предоставляющего прикладным программам набор функций по выполнению криптографических операций семейства ГОСТ;

– `ENGINE_get_digest` с параметром `NID_id_GostR3411_2012_512` – для получения объекта типа `EVP_MD`, соответствующего алгоритму вычисления хеш-кода ГОСТ 34.11-2012 512 бит;

– `PKCS5_PBKDF2_HMAC` – для формирования ключа из пароля с использованием соответствующего объекта `EVP_MD`.

Пример кода формирования ключа из пароля:

```
ENGINE *engine;
const EVP_MD *digest;
unsigned char DK[100];
char* pas1 = "password";
unsigned char* slt1 = "salt";

engine = ENGINE_by_id("gost");
digest = ENGINE_get_digest(engine, NID_id_GostR3411_2012_512);

PKCS5_PBKDF2_HMAC(pas1, 8, slt1, 4, 1, digest, 64, DK);
```

2.2.10. Порядок использования СКЗИ СледопытSSL для формирования транспортного ключевого контейнера в соответствии с Р 50.1.112-2016

Для формирования транспортного ключевого контейнера в формате `EncryptedData` программа может выполнить вызов следующих функций СКЗИ:

– `ENGINE_by_id` с параметром «gost» - для получения объекта типа «engine», предоставляющего прикладным программам набор функций по выполнению криптографических операций семейства ГОСТ;

– `ENGINE_get_digest` с параметром `NID_id_GostR3411_2012_512` – для получения объекта типа `EVP_MD`, соответствующего алгоритму вычисления хеш-кода ГОСТ 34.11-2012 512 бит;

– `PKCS12_create` – непосредственное формирование ключевого контейнера. В качестве алгоритмов шифрования ключа и сертификата в функцию должны передаваться идентификаторы `NID_id_Gost28147_89`;

– `PKCS12_set_mac` – формирование кода аутентификации контейнера с использованием соответствующего объекта `EVP_MD`.

Пример кода формирования транспортного ключевого контейнера в формате `EncryptedData`:

```
ENGINE *engine;
EVP_MD *digest;
EVP_PKEY *privateKey;
PKCS12 *p12;
X509 *testCert;
FILE *fp;
BIO *certBio = NULL;
BIO *privateKeyBio = NULL;

engine = ENGINE_by_id("gost");
digest = ENGINE_get_digest(engine, NID_id_GostR3411_2012_512);

certBio = BIO_new(BIO_s_mem());
BIO_puts(certBio, PKCS12_TestCert);

privateKeyBio = BIO_new(BIO_s_mem());
BIO_puts(privateKeyBio, PKCS12_TestPrivateKey);

testCert = PEM_read_bio_X509(certBio, NULL, NULL, NULL);
```

```
privateKey = PEM_read_bio_PrivateKey(privateKeyBio, NULL, NULL, NULL);

p12Bio = BIO_new(BIO_s_mem());

p12 = PKCS12_create(PKCS12_Password, NULL, privateKey, testCert, NULL,
NID_id_Gost28147_89, NID_id_Gost28147_89, 2000, -1, 0);

PKCS12_set_mac(p12, PKCS12_Password, 23, PKCS12_MacSalt,
sizeof(PKCS12_MacSalt), 2000, digest);

fp = fopen("pkcs12_enc.pfx", "wb");
i2d_PKCS12_fp(fp, p12);
fclose(fp);
```

Для разбора транспортного ключевого контейнера в формате `EncryptedData` программа может выполнить вызов следующих функций СКЗИ:

– `PKCS12_verify_mac` – проверка кода аутентификации контейнера;

– `PKCS12_parse` – разбор и расшифрование контейнера.

Пример кода разбора транспортного ключевого контейнера в формате `EncryptedData`:

```
BIO *p12Bio;
PKCS12 *p12;
EVP_PKEY *parsedPrivateKey;
X509 *parsedCert;

p12Bio = BIO_new(BIO_s_mem());
BIO_write(p12Bio, contBuf, contSize);

p12 = PKCS12_new();
d2i_PKCS12_bio(p12Bio, &p12);

if (!PKCS12_verify_mac(p12, PKCS12_Password, 23))
{
    printf("PKCS12 verify mac error\n");
}

PKCS12_parse(p12, PKCS12_Password, &parsedPrivateKey, &parsedCert,
NULL)
```

Для формирования транспортного ключевого контейнера в формате `EnvelopedData` программа может выполнить вызов следующих функций СКЗИ:

– ENGINE_by_id с параметром «gost» - для получения объекта типа «engine», предоставляющего прикладным программам набор функций по выполнению криптографических операций семейства ГОСТ;

– ENGINE_ctrl с параметрами GOST_CTRL_CMS_RI_TYPE и CMS_RECIPINFO_AGREE – для задания режима выработки ключа по протоколу Диффи-Хелмана;

– PKCS12_create – непосредственное формирование ключевого контейнера. В качестве алгоритмов шифрования ключа и сертификата в функцию должны передаваться идентификаторы NID_id_Gost28147_89.

Пример кода формирования транспортного ключевого контейнера в формате EnvelopedData:

```
ENGINE *engine;
EVP_PKEY *privateKey;
PKCS12 *p12;
X509 *testCert;
FILE *fp;
BIO *certBio = NULL;
BIO *privateKeyBio = NULL;

engine = ENGINE_by_id("gost");

ENGINE_ctrl(engine, GOST_CTRL_CMS_RI_TYPE, CMS_RECIPINFO_AGREE, NULL,
NULL);

certBio = BIO_new(BIO_s_mem());
BIO_puts(certBio, PKCS12_TestCert);

privateKeyBio = BIO_new(BIO_s_mem());
BIO_puts(privateKeyBio, PKCS12_TestPrivateKey);

testCert = PEM_read_bio_X509(certBio, NULL, NULL, NULL);
privateKey = PEM_read_bio_PrivateKey(privateKeyBio, NULL, NULL, NULL);

p12 = PKCS12_create(NULL, NULL, privateKey, testCert, NULL,
NID_id_Gost28147_89, NID_id_Gost28147_89, 2000, -1, 0)

fp = fopen("pkcs12_env.pfx", "wb");
i2d_PKCS12_fp(fp, p12);
fclose(fp);
```

Для разбора транспортного ключевого контейнера в формате EnvelopedData программа может выполнить вызов функций СКЗИ:

– PKCS12_parse_with_pkey – проверка кода аутентификации контейнера, его разбор и расшифрование.

Пример кода разбора транспортного ключевого контейнера в формате EnvelopedData:

```
BIO *p12Bio;
EVP_PKEY *privateKey;
PKCS12 *p12;
X509 *testCert;
EVP_PKEY *parsedPrivateKey;
X509 *parsedCert;
BIO *certBio = NULL;
BIO *privateKeyBio = NULL;

p12Bio = BIO_new(BIO_s_mem());
BIO_write(p12Bio, contBuf, contSize);

p12 = PKCS12_new();
d2i_PKCS12_bio(p12Bio, &p12);

certBio = BIO_new(BIO_s_mem());
BIO_puts(certBio, PKCS12_TestCert);

privateKeyBio = BIO_new(BIO_s_mem());
BIO_puts(privateKeyBio, PKCS12_TestPrivateKey);

testCert = PEM_read_bio_X509(certBio, NULL, NULL, NULL);
privateKey = PEM_read_bio_PrivateKey(privateKeyBio, NULL, NULL, NULL);

PKCS12_parse_with_pkey(p12, NULL, &parsedPrivateKey, &parsedCert,
NULL, privateKey, testCert);
```

2.2.11. Диверсификация ключевого материала в соответствии с Р 50.1.113-2016

В СКЗИ СледопытSSL реализовано 2 алгоритма диверсификации ключевого материала:

– KDF_GOSTR3411_2012_256;

– KDF_TREE_GOSTR3411_2012_256.

Для диверсификации ключевого материала программа должна выполнить вызов следующих функций СКЗИ:

– ENGINE_by_id с параметром «gost» - для получения объекта типа «engine», предоставляющего прикладным программам набор функций по выполнению криптографических операций семейства ГОСТ;

– ENGINE_ctrl – вызов функции диверсификации из объекта «engine». Требуемый алгоритм передается в данную функцию вторым параметром (GOST_CTRL_KDF - KDF_GOSTR3411_2012_256, GOST_CTRL_KDF_TREE - KDF_TREE_GOSTR3411_2012_256). Четвертым параметром передается структура GOST_KDF_PARAMS или GOST_KDF_TREE_PARAMS, описывающая параметры алгоритма.

Пример кода диверсификации по алгоритму KDF_GOSTR3411_2012_256:

```
unsigned char out[128];
GOST_KDF_PARAMS params;
ENGINE *engine;

engine = ENGINE_by_id("gost");

params.kdf_key = Key_KDF_GOST_3411_2012_256;
params.kdf_key_len = 32;
params.label = Label_KDF_GOST_3411_2012_256;
params.label_len = 4;
params.seed = Seed_KDF_GOST_3411_2012_256;
params.seed_len = 8;
params.out = out;

ENGINE_ctrl(engine, GOST_CTRL_KDF, 0, &params, NULL);
```

Пример кода диверсификации по алгоритму KDF_TREE_GOSTR3411_2012_256:

```
unsigned char out[128];
GOST_KDF_TREE_PARAMS params;
ENGINE *engine;

engine = ENGINE_by_id("gost");

params.kdf_key = Key_KDF_GOST_3411_2012_256;
params.kdf_key_len = 32;
params.label = Label_KDF_GOST_3411_2012_256;
params.label_len = 4;
params.seed = Seed_KDF_GOST_3411_2012_256;
params.seed_len = 8;
params.R = 1;
params.out_size = 64;
params.out = out;
```

```
ENGINE_ctrl(engine, GOST_CTRL_KDF_TREE, 0, &params, NULL);
```

2.2.12. Работа с алгоритмами PRF (псевдослучайные функции протоколов TLS) в соответствии с Р 50.1.113-2016

В СКЗИ реализованы следующие алгоритмы PRF:

- `PRF_TLS_GOSTR3411_2012_256`;
- `PRF_TLS_GOSTR3411_2012_512`;
- `PRF_IPSEC_KEYMAT_GOSTR3411_2012_256`;
- `PRF_IPSEC_KEYMAT_GOSTR3411_2012_512`;
- `PRF_IPSEC_PRFPLUS_GOSTR3411_2012_256`;
- `PRF_IPSEC_PRFPLUS_GOSTR3411_2012_512`.

Для выработки псевдослучайной последовательности по алгоритмам `PRF_TLS_GOSTR3411_2012_256`, `PRF_TLS_GOSTR3411_2012_512` программа должна выполнить вызов следующих функций СКЗИ:

- `ENGINE_by_id` с параметром «gost» - для получения объекта типа «engine», предоставляющего прикладным программам набор функций по выполнению криптографических операций семейства ГОСТ;
- `ENGINE_get_digest` – для получения объекта типа `EVP_MD`, соответствующего заданному алгоритму вычисления функции PRF (алгоритм хеширования ГОСТ 34.11-2012 256 бит или алгоритм хеширования ГОСТ 34.11-2012 512 бит);
- `OPENSSL_tls1_PRF` – вычисление значения функции PRF.

Для выработки псевдослучайной последовательности по следующим алгоритмам:

- `PRF_IPSEC_KEYMAT_GOSTR3411_2012_256`;
- `PRF_IPSEC_KEYMAT_GOSTR3411_2012_512`;
- `PRF_IPSEC_PRFPLUS_GOSTR3411_2012_256`;
- `PRF_IPSEC_PRFPLUS_GOSTR3411_2012_512`.

Программа должна выполнить вызов следующих функций СКЗИ:

– ENGINE_by_id с параметром «gost» - для получения объекта типа «engine», предоставляющего прикладным программам набор функций по выполнению криптографических операций семейства ГОСТ;

– ENGINE_ctrl – вызов функции PRF из объекта «engine».

Требуемый алгоритм передается в данную функцию вторым параметром:

– OST_CTRL_PRF_IPSEC_KEYMAT_256 - PRF_IPSEC_KEYMAT_GOSTR3411_2012_256;

– GOST_CTRL_PRF_IPSEC_KEYMAT_512 - PRF_IPSEC_KEYMAT_GOSTR3411_2012_512;

– GOST_CTRL_PRF_IPSEC_PLUS_256 - PRF_IPSEC_PRFPLUS_GOSTR3411_2012_256;

– GOST_CTRL_PRF_IPSEC_PLUS_512 - PRF_IPSEC_PRFPLUS_GOSTR3411_2012_512.

Четверым параметром передается структура GOST_PRF_IPSEC_PARAMS, описывающая параметры алгоритма.

Пример кода выработки псевдослучайной последовательности по алгоритму PRF_TLS_GOSTR3411_2012_256:

```
ENGINE *engine;
const EVP_MD *digest;
unsigned char out_buf[128];

engine = ENGINE_by_id("gost");
digest = ENGINE_get_digest(engine, NID_id_GostR3411_2012_256);

OPENSSL_tls1_PRF(digest, PRF_GOST_3411_2012_Key,
sizeof(PRF_GOST_3411_2012_Key),
                     PRF_GOST_3411_2012_Label,
sizeof(PRF_GOST_3411_2012_Label),
                     PRF_GOST_3411_2012_Seed,
sizeof(PRF_GOST_3411_2012_Seed),
                     NULL, 0, NULL, 0, NULL, 0, out_buf, 128);
```

Пример кода выработки псевдослучайной последовательности по алгоритму PRF_IPSEC_KEYMAT_GOSTR3411_2012_256:

```
unsigned char out[64];
GOST_PRF_IPSEC_PARAMS params;
ENGINE *engine;

engine = ENGINE_by_id("gost");

params.key = Key_PRF_IPSEC_KEYMAT_GOSTR3411_2012_256;
params.key_len = sizeof(Key_PRF_IPSEC_KEYMAT_GOSTR3411_2012_256);
params.seed = S_PRF_IPSEC_KEYMAT_GOSTR3411_2012_256;
params.seed_len = sizeof(S_PRF_IPSEC_KEYMAT_GOSTR3411_2012_256);
params.out = out;
params.out_len = 64;

ENGINE_ctrl(engine, GOST_CTRL_PRF_IPSEC_KEYMAT_256, 0, &params, NULL);
```

2.2.13. Формирование сообщений CMS с использованием алгоритмов семейства ГОСТ

Для доступа к функциям формирования сообщений CMS используются объекты структур следующего типа:

– `BIO` – работа с потоками ввода/вывода;

– `X509` – сертификаты `X509`;

– `EVP_PKEY`– объект для работы с ключами ЭП;

– `EVP_CIPHER` – симметричный криптоалгоритм.

Для формирования подписанных ЭП сообщений CMS программа может использовать вызов следующих функций СКЗИ:

– `BIO_new, BIO_s_mem` – формирование потоков ввода/вывода;

– `BIO_puts` – запись в поток сертификата, закрытого ключа подписи;

– `PEM_read_bio_X509` – разбор сертификата;

– `PEM_read_bio_PrivateKey` – чтение закрытого ключа из потока в структуру `EVP_PKEY`;

– `CMS_sign` – формирование структуры подписанного сообщения CMS;

– `CMS_final` – окончательное формирование сообщения CMS;

– `i2d_CMS_bio_stream` – перевод сообщения CMS в формат DER;

– `BIO_get_mem_data, BIO_read` – чтение данных из потока.

Для формирования зашифрованных сообщений CMS программа может использовать вызов следующих функций СКЗИ:

– ENGINE_by_id с параметром «gost» - для получения объекта типа «engine», предоставляющего прикладным программам набор функций по выполнению криптографических операций семейства ГОСТ;

– ENGINE_get_cipher – для получения объекта типа EVP_CIPHER, соответствующего заданному алгоритму зашифрования сообщения;

– BIO_new, BIO_s_mem – формирование потоков ввода/вывода;

– BIO_puts – запись в поток сертификата закрытого ключа подписи;

– PEM_read_bio_X509 – разбор сертификата;

– PEM_read_bio_PrivateKey – чтение закрытого ключа из потока в структуру EVP_PKEY;

– CMS_encrypt либо CMS_encrypt_with_own_key – для формирования зашифрованного сообщения CMS (в первом случае в процедуре согласования ключей используется эфемерный ключ, во втором – собственный закрытый ключ ЭП);

– CMS_final – окончательное формирование сообщения CMS;

– i2d_CMS_bio_stream – перевод сообщения CMS в формат DER;

– BIO_get_mem_data, BIO_read – чтение данных из потока.

Для расшифрования сообщений CMS программа дополнительно может использовать вызов следующих функций СКЗИ:

– d2i_CMS_bio – перевод сообщения CMS из формата DER во внутреннее представление;

– CMS_decrypt – расшифрование сообщения.

При формировании зашифрованных сообщений CMS для управления типом алгоритма согласования ключей (KeyAgreeRecipientInfo или KeyTransRecipientInfo) может использоваться вызов функции ENGINE_ctrl со вторым параметром GOST_CTRL_CMS_RI_TYPE. Третьим параметром задается режим согласования ключей.

Пример кода формирования подписанного CMS:

```
EVP_PKEY *privateKey=NULL;
X509 *testCert=NULL;
FILE *fp;
BIO *certBio = NULL;
BIO *privateKeyBio = NULL;
BIO *dataBio = NULL;
BIO *outBio = NULL;
unsigned char buf[2000];
int size;
CMS_ContentInfo *cms;
int flags;
STACK_OF(X509) *certs = sk_X509_new_null();

certBio = BIO_new(BIO_s_mem());
BIO_puts(certBio, PKCS12_TestCert);
testCert = PEM_read_bio_X509(certBio, NULL, NULL, NULL);

privateKeyBio = BIO_new(BIO_s_mem());
BIO_puts(privateKeyBio, PKCS12_TestPrivateKey);
privateKey = PEM_read_bio_PrivateKey(privateKeyBio, NULL, NULL, NULL);

outBio = BIO_new(BIO_s_mem());

dataBio = BIO_new(BIO_s_mem());
BIO_puts(dataBio, "Test String");

flags = 0;
cms = CMS_sign(testCert, privateKey, NULL, dataBio, flags);
CMS_final(cms, dataBio, NULL, flags);
i2d_CMS_bio_stream(outBio, cms, dataBio, flags);

size = BIO_get_mem_data(outBio, NULL);
BIO_read(outBio, buf, size);

fp = fopen("CMS_SIGNED_256.bin", "wb");
fwrite(buf, 1, size, fp);
fclose(fp);
```

Пример кода формирования зашифрованного CMS и его расшифрования:

```
EVP_PKEY *privateKey=NULL;
X509 *testCert=NULL;
FILE *fp;
BIO *certBio = NULL;
BIO *privateKeyBio = NULL;
BIO *dataBio = NULL;
BIO *outBio = NULL;
unsigned char buf[2000];
int size;
```

```
CMS_ContentInfo *cms;
const EVP_CIPHER *cipher = NULL;
int flags;
STACK_OF(X509) *certs = sk_X509_new_null();
ENGINE *engine;

engine = ENGINE_by_id("gost");

certBio = BIO_new(BIO_s_mem());
BIO_puts(certBio, certStr);
testCert = PEM_read_bio_X509(certBio, NULL, NULL, NULL);

privateKeyBio = BIO_new(BIO_s_mem());
BIO_puts(privateKeyBio, privateKeyStr);
privateKey = PEM_read_bio_PrivateKey(privateKeyBio, NULL, NULL, NULL);

dataBio = BIO_new(BIO_s_mem());
BIO_puts(dataBio, "Test String");

outBio = BIO_new(BIO_s_mem());
flags = CMS_NOCERTS | CMS_NOATTR | CMS_NOSMIMECAP | SMIME_BINARY;

cipher = ENGINE_get_cipher(engine, NID_id_Gost28147_89);

sk_X509_push(certs, testCert);

cms = CMS_encrypt(certs, dataBio, cipher, flags);
CMS_final(cms, dataBio, NULL, flags);
i2d_CMS_bio_stream(outBio, cms, dataBio, flags);

size = BIO_get_mem_data(outBio, NULL);
BIO_read(outBio, buf, size);

fp = fopen(resFilename, "wb");
fwrite(buf, 1, size, fp);
fclose(fp);

//РАСШИФРОВАНИЕ

BIO_write(outBio, buf, size);
cms = d2i_CMS_bio(outBio, NULL);

CMS_decrypt(cms, privateKey, testCert, NULL, outBio, flags);

size = BIO_get_mem_data(outBio, NULL);
BIO_read(outBio, buf, size);

CMS_ContentInfo_free(cms);

buf[size] = 0;
printf("\tRes string:%s\n",buf);
```

2.2.14. Дополнительные функции, которые могут применяться при работе с СКЗИ

– `ASN1_STRING_to_UTF8` – преобразует строку из кодировки ASN1 в кодировку UTF8;

– `BIO_free` – освобождает ресурсы, выделенные объектом BIO;

– `BIO_new_bio_pair` – выполняет создание и последующее объединение 2 объектов BIO;

– `BIO_new_mem_buf` – создает объект BIO, выполняющий работу с памятью;

– `BIO_pending` – возвращает количество ожидающих символов в буферах чтения и записи объектов BIO;

– `BIO_should_retry` – проверяет необходимость выполнения повторного вызова функций `BIO_write, BIO_read`;

– `CRYPTO_get_locking_callback` – возвращает указатель на функцию выполнения блокировки доступа к ресурсам библиотеки СКЗИ;

– `CRYPTO_set_locking_callback` – устанавливает функцию выполнения блокировки доступа к ресурсам библиотеки СКЗИ;

– `CRYPTO_num_locks` – возвращает количество блокировок доступа к ресурсам библиотеки СКЗИ;

– `CRYPTO_set_id_callback` – устанавливает функцию получения идентификатора потока;

– `d2i_SSL_SESSION` – преобразует внешние данные формата ASN1 TLS-сессии в объект `SSL_SESSION`;

– `EC_KEY_new_by_curve_name` – создает новый объект `EC_KEY`, используя идентификатор ЭК;

– `EC_KEY_free` – освобождает память, выделенную под объект `EC_KEY`;

– `EVP_PKEY_free` – освобождает память, выделенную под объект `EVP_PKEY`;

– `ERR_clear_error` – очищает очередь ошибок;

– ERR_get_error – возвращает первый код ошибки из очереди ошибок;

– ERR_error_string_n – возвращает строку, содержащую описание ошибки библиотеки СКЗИ с заданным кодом;

– i2d_SSL_SESSION – преобразует объект SSL_SESSION в формат ASN1;

– OPENSSL_free – освобождает память по указанному адресу;

– PEM_write_bio_X509 – записывает содержимое BIO в структуру формата X509;

– PEM_read_bio_X509_AUX – выполняет чтение сертификата X509 в формате .pem;

– sk_GENERAL_NAME_value – возвращает заданный объект типа GENERAL_NAME из списка;

– sk_GENERAL_NAME_num – возвращает количество объектов типа GENERAL_NAME в списке;

– sk_GENERAL_NAME_pop_free – выполняет освобождение списка объектов типа GENERAL_NAME;

– sk_X509_NAME_new_null – выполняет создание пустого списка для хранения объектов типа X509_NAME;

– sk_X509_NAME_pop_free – выполняет освобождение списка объектов типа X509_NAME;

– sk_X509_NAME_push – выполняет добавление в список объекта типа X509_NAME;

– SSL_do_handshake – выполняет процедуру handshake протокола TLS;

– SSL_free – освобождает память, выделенную под структуру SSL;

– SSL_get_error – возвращает код ошибки после вызова функции ввода/вывода TLS-сессии;

– SSL_get_servername – возвращает имя сервера TLS-сессии;

– SSL_get_SSL_CTX – возвращает указатель на объект SSL_CTX;

– SSL_get0_next_proto_negotiated – возвращает указатель на поле с требуемым протоколом верхнего уровня (next protocol negotiation) TLS-сессии;

– SSL_is_init_finished – проверяет, закончилась ли установка TLS-сессии;

– SSL_new – создает новый объект типа SSL;

– SSL_read – выполняет чтение заданного количества байт переданных данных из TLS-сессии;

– SSL_set_accept_state – задает использование TLS-сессии в режиме сервера;

– SSL_set_bio – соединяет объект SSL с заданным объектом типа BIO;

– SSL_set_connect_state – задает использование TLS-сессии в режиме клиента;

– SSL_set_info_callback – устанавливает функцию, которую можно использовать для получения информации о состоянии TLS-сессии;

– SSL_set_session – устанавливает объект SSL_SESSION, который будет использоваться в заданной TLS-сессии;

– SSL_set_SSL_CTX – устанавливает объект SSL_CTX, который будет использоваться в заданной TLS-сессии;

– SSL_set_tlsext_host_name – устанавливает имя сервера в TLS-сессии;

– SSL_state_string_long – возвращает строку, указывающую на состояние объекта SSL;

– SSL_state_string – возвращает строку из 6 символов, указывающую на состояние объекта SSL;

– SSL_write – выполняет запись заданного количества байт передаваемых данных в TLS-сессию;

– SSL_CTX_add_extra_chain_cert – добавляет сертификат X509 в TLS-сессию;

– SSL_CTX_check_private_key – проверяет соответствие закрытого ключа заданному сертификату;

– SSL_CTX_free – освобождает память, выделенную под объект SSL_CTX;

– SSL_CTX_get_cert_store – возвращает указатель на массив корневых сертификатов, использующихся в TLS-сессии;

– SSL_CTX_get_ex_data – возвращает указатель на пользовательские данные объекта SSL_CTX;

– SSL_CTX_get_ex_new_index – выполняет регистрацию нового индекса для данных конкретного МП в TLS-сессии;

– SSL_CTX_sess_set_new_cb – устанавливает указатель на функцию, которая автоматически вызывается при установлении нового TLS-соединения;

– SSL_CTX_set_cipher_list – устанавливает список доступных криптонаборов для установки TLS-соединения;

– SSL_CTX_set_client_CA_list – устанавливает список корневых сертификатов, отправляемых клиенту при запросе сертификата в рамках TLS-сессии;

– SSL_CTX_set_ex_data – устанавливает указатель на пользовательские данные объекта SSL_CTX;

– SSL_CTX_set_next_proto_select_cb – устанавливает функцию, которая вызывается клиентом для выбора протокола из предоставленного сервером списка в рамках TLS-сессии;

– SSL_CTX_set_next_protos_advertised_cb – устанавливает функцию, которая используется для получения списка поддерживаемых протоколов для согласования следующего в рамках TLS-сессии;

– SSL_CTX_set_options – устанавливает опции объекта SSL_CTX;

– SSL_CTX_set_session_cache_mode – включает/отключает кэширование сеанса TLS-сессии;

– SSL_CTX_set_tlsext_servername_arg – устанавливает аргумент для последующей передачи в функцию получения имени сервера TLS-сессии;

– SSL_CTX_set_tlsext_servername_callback – устанавливает функцию, которая вызывается клиентом для проверки корректности имени сервера TLS-сессии;

– SSL_CTX_set_tlsext_ticket_keys – устанавливает ticket-ключи TLS-сессии;

– SSL_CTX_set_tmp_ecdh – устанавливает параметры протокола ECDH TLS-сессии;

– SSL_CTX_set_verify – устанавливает функцию, которая вызывается для проверки корректности сертификата TLS-сессии;

– SSL_CTX_use_certificate – устанавливает сертификат для использования в TLS-сессии;

– SSL_CTX_use_PrivateKey – устанавливает закрытый ключ для использования в TLS-сессии;

– TLSv1_2_method – возвращает указатель на функции для работы с протоколом TLSv1.2;

– X509_free – освобождает память, выделенную под объект X509;

– X509_get_ext_d2i – выполняет декодирование расширения сертификата из кодировки ASN1;

– X509_get_subject_name – возвращает имя субъекта сертификата X509;

– X509_NAME_dup – выполняет копирование объекта X509_NAME;

– X509_NAME_free – освобождает память, выделенную под объект X509_NAME;

– X509_NAME_get_index_by_NID – выполняет поиск в структуре X509_NAME объекта с заданным идентификатором;

– X509_NAME_get_entry – возвращает объект с заданным идентификатором из структуры X509_NAME;

– X509_NAME_ENTRY_get_data – возвращает значение структуры ASN1_STRING поля X509_NAME_ENTRY;

– X509_STORE_add_cert – добавляет объект X509 в структуру X509_STORE;

– X509_STORE_free – освобождает память, выделенную под объект X509_ STORE;

– X509_STORE_new – создает новый объект типа X509_STORE.

2.3. Криптоконтейнер

2.3.1. Общая информация

В ОС Аврора, для совместного использования с которой предназначено СКЗИ СледопытSSL, реализован криптоконтейнер, предоставляющий возможность защищенного хранения конфиденциальных данных любого МП с обеспечением конфиденциальности и целостности информации.

Криптоконтейнер позволяет пользовательскому МП хранить информацию в защищенном виде в постоянном хранилище.

ПРИМЕЧАНИЕ. Криптоконтейнер представляет собой шифрованную папку, в которой могут храниться различные файлы и папки.

Программный продукт состоит из 2 модулей:

– модуля диспетчеризации, который позволяет обращаться к модулю обработки и управляет контейнерами: создание/удаление, открытие/закрытие, управление паролем;

– модуля обработки, который исполняет операции открытия, закрытия контейнеров, файловые операции внутри контейнера, шифрование/расшифрование данных, авторизацию и аутентификацию.

В настоящее время целевой платформой являются МУ, функционирующие под управлением ОС Аврора (и основанные на ней).

Авторизация доступа к сохраненной информации осуществляется по паролю, который используется для защиты доступа к шифрованным данным, хранящимся в криптоконтейнере.

ПРИМЕЧАНИЯ:

1. Подробная информация о требованиях к паролю, а также о смене и сбросе пароля приведена в документе АДМГ.40002-01 92 01;

2. Примером МП, использующего криптоконтейнер, является МП «Криптозаметки», т.к. обрабатываемые им данные хранятся в шифрованном виде с использованием СКЗИ СледопытSSL. Подробное описание СКЗИ СледопытSSL и МП «Криптозаметки» приведено в документе АДМГ.40002-01 92 01.

2.3.2. Требования к окружению

Программа предназначена для работы на МУ под управлением ОС Аврора, которая функционирует:

– с запущенной системной службой межпрограммного взаимодействия D-Bus;

– с установленным СКЗИ СледопытSSL;

– под управлением службы начального запуска `systemd`;

– и сконфигурирована с поддержкой FUSE - файловой системы (ФС) в пространстве пользователя.

ПРИМЕЧАНИЕ. Требуется системная служба запроса пароля пользователя.

2.3.3. Описание работы

При первом обращении к модулю диспетчеризации либо при ручном запуске происходит запуск сервиса диспетчеризации, который занимает на системной шине `D-Bus` имя `ru.omprussia.sstore`. Данный сервис предоставляет на системной шине программный интерфейс.

При получении запроса от МП сервис диспетчеризации проверяет существование контейнера для МП. В случае если криптоконтейнер не предоставлен, сервис диспетчеризации выдает сообщение об ошибке, которое позволяет определить причину возникновения ошибки. Сервис проверяет, разрешено ли МП запрашивать криптоконтейнер. Запросы криптоконтейнера удовлетворяются только для пользовательских МП, исполняемые файлы которых находятся в ФС по пути `/bin`, `/usr/bin`, `/sbin`, `/usr/sbin`. Далее сервис диспетчеризации вызывает системное окно для запроса пароля авторизации.

Служба диспетчеризации запускает службу обработки запросов доступа к криптоконтейнеру. В случае успеха служба диспетчеризации возвращает МП ответ на запрос в предоставлении криптоконтейнера с указанием пути, по которому он предоставлен.

Служба диспетчеризации передает службе обработки запросов информацию о МП, запросившем криптоконтейнер, и пароль авторизации. На каждое пользовательское МП, использующее криптоконтейнер, запускается собственная служба обработки запросов доступа.

Службой обработки запросов доступа пароль преобразуется в ключевую информацию, которая используется для расшифрования хранимого зашифрованного ключа криптоконтейнера. Правильность расшифрования проверяется с помощью хранимой имитовставки. Служба обеспечивает доступ к криптоконтейнеру только того МП, которое его запросило. Другие МП при попытке доступа к криптоконтейнеру открытому иным МП, получают ошибку доступа. Далее МП использует криптоконтейнер как обычный участок ФС со стандартными доступными файловыми операциями.

При отсутствии необходимости в дальнейшем использовании криптоконтейнера МП отправляет запрос на его закрытие службе диспетчеризации. Служба диспетчеризации завершает службу обработки запросов для данного криптоконтейнера и в случае успеха возвращает успешный код возврата в ответ на запрос МП.

2.3.4. Интерфейс взаимодействия

Программный интерфейс предоставляется на системной шине `D-Bus` по адресу `ru.omprussia.sstore`.

У интерфейса сервиса имеются методы — `Exists`, `Create`, `Open`, `Close`, `Delete` и `Recrypt`, проверяющие существование, создающие, открывающие, закрывающие, удаляющие и защищающие криптоконтейнер новым паролем соответственно.

При первом открытии МП с помощью сервиса должно определить, создан ли криптоконтейнер для данного МП (клиента). Если криптоконтейнер создан, то МП запрашивает пароль доступа к нему. Если криптоконтейнер не создан, то МП, обращаясь к сервису, создает его с помощью метода `Create` и запрашивает пароль для его защиты, который будет использован также для его открытия.

2.3.5. Идентификация МП

Разграничение доступа МП к криптоконтейнеру происходит на основании идентификаторов МП. Идентификатором МП считается последний компонент имени исполняемого файла МП. Идентификаторы МП достоверно определяются по системным идентификаторам экземпляров МП (`pid`), осуществляющих запросы к сервису.

Соответствие системного идентификатора экземпляра идентификатору МП гарантируется одним из 2 механизмов.

Первый - механизм `named cgroup`. Данным механизмом управляет `booster`, создавая `cgroup` по пути, соответствующему пути исполняемого файла МП при старте экземпляра МП, и помещая экземпляр в созданную `cgroup`.

Второй механизм использует `sailjail` для МП, запущенных в песочнице.

2.3.6. Ограничения использования и известные проблемы

Криптоконтейнер поддерживает ограниченный набор файловых операций:

- создание и удаление файлов и папок;
- запрос файловой статистики;
- перемещение (смена имени) файлов и директорий.

3. ВХОДНЫЕ И ВЫХОДНЫЕ ДАННЫЕ

3.1. Входные данные

Входными данными для СКЗИ СледопытSSL являются:

- открытая информация;
- сеансовые ключевые данные;
- сертификаты ключей проверки.

3.2. Выходные данные

Выходными данными для СКЗИ СледопытSSL являются результаты выполнения криптографических операций.

ПЕРЕЧЕНЬ ТЕРМИНОВ И СОКРАЩЕНИЙ

Используемые в настоящем документе термины и сокращения приведены в таблице (Таблица 1).

Таблица 1

Термин/ Сокращение	Расшифровка
Доверенное оконечное оборудование	Автоматизированная информационная система, обеспечивающая защищенное взаимодействие по протоколу TLS и реализующая криптографические алгоритмы: ГОСТ Р 34.10-2012, ГОСТ Р 34.11-2012, ГОСТ Р 34.12-2015, ГОСТ Р 34.13-2015
ДСЧ	Датчик случайных чисел
Криптоконтейнер	Криптографический контейнер
МП	Мобильное приложение
МУ	Мобильное устройство
ОС	Операционная система
ПДСЧ	Программный датчик случайных чисел
СКЗИ	Средство криптографической защиты информации
СКЗИ СледопытSSL	Программное средство криптографической защиты информации СледопытSSL (вариант исполнения № 1)
ФС	Файловая система
ФСБ России	Федеральная служба безопасности Российской Федерации
ФСТЭК России	Федеральная служба по техническому и экспортному контролю Российской Федерации
ЭК	Эллиптические кривые
ЭП	Электронная подпись
API	Application Programming Interface - описание способов (набор классов, процедур, функций, структур или констант), которыми одна компьютерная программа может взаимодействовать с другой программой
CMS	Cryptographic Message Syntax – синтаксис криптографических сообщений

Термин/ Сокращение	Расшифровка
D-Bus	Система межпроцессного взаимодействия, которая позволяет МП в ОС сообщаться друг с другом
ECDH	Elliptic curve Diffie–Hellman – протокол Диффи-Хеллмана на эллиптических кривых: криптографический протокол, позволяющий двум сторонам, имеющим пары открытый/закрытый ключ на эллиптических кривых, получить общий секретный ключ, используя незащищённый от прослушивания канал связи
HMAC	Hash-Based Message Authentication Code – код аутентификации (проверки подлинности) сообщений, использующий хеш-функции. Один из механизмов проверки целостности информации, позволяющий гарантировать то, что данные, передаваемые или хранящиеся в ненадежной среде, не были изменены посторонними лицами
HTTP	HyperText Transfer Protocol – протокол прикладного уровня передачи данных. Основой HTTP является технология «клиент-сервер», т.е. предполагается существование потребителей (клиентов), которые инициируют соединение и посылают запрос, и поставщиков (серверов), которые ожидают соединения для получения запроса, производят необходимые действия и возвращают обратно сообщение с результатом
PRF	Pseudorandom Function – псевдослучайная функция: отображение, позволяющее вырабатывать псевдослучайные последовательности байтов
Qt	Кроссплатформенная библиотека разработки программного обеспечения
SSL	Secure Sockets Layer – криптографический протокол, который обеспечивает установление безопасного соединения между клиентом и сервером
TCP	Transmission Control Protocol – протокол транспортного уровня, гарантирующий целостность передаваемых данных и уведомление отправителя о результатах передачи
TLS	Transport Layer Security – криптографический протокол, обеспечивающий защищенную передачу данных между узлами в сети Интернет

Термин/ Сокращение	Расшифровка
UTF-8	Unicode Transformation Format, 8-bit – распространённый стандарт кодирования символов, позволяющий более компактно хранить и передавать символы Юникода, используя переменное количество байт (от 1 до 4), и обеспечивающий полную обратную совместимость с 7-битной кодировкой ASCII

ПРИЛОЖЕНИЕ 1

Примеры использования API для установки защищенного соединения

1. Подключение к удаленному серверу с использованием SSL-сокета:

```
// добавление информации о сервере
const QString hostname = QStringLiteral("gost.security.ru");
const int port = 443;
// создание SSL-сокета
QSslSocket *socket = new QSslSocket(this);
socket->setSslConfiguration(configuration);
// подключение обработчиков сигналов
connect(socket, &QSslSocket::encrypted, this,
&Connector::onEncrypted);
connect(socket, static_cast<void (QSslSocket::*)
(const QList<QSslError> &errors)>(&QSslSocket::sslErrors), this,
&Connector::onSslErrors);
connect(socket, &QSslSocket::peerVerifyError, this,
&Connector::onPeerVerifyError);
// подключение к серверу
socket->connectToHostEncrypted(hostname, port);
```

Необходимо настроить обработку сигналов, где с помощью проверки значения `keyExchangeMethod` подтверждается, что подключение к серверу было выполнено с помощью шифрования согласно ГОСТ:

```
void Connector::onEncrypted()
{
    qDebug() << Q_FUNC_INFO << "keyExchangeMethod:"
      << socket->sessionCipher().keyExchangeMethod();
}
```

Обработка сигнала о возникновении ошибки SSL-соединения выполняется с помощью следующей команды:

```
void Connector::onPeerVerifyError(const QSslError &error)
{
    qDebug() << Q_FUNC_INFO << error;
}
```

Значения ошибок, которые могут возникнуть во время установления связи SSL, приведены в описании класса `QSslError`.

2. Отправка HTTP-запроса методом GET на удаленный сервер:

```
// создание менеджера для управления HTTP-запросами
QNetworkAccessManager *manager = new QNetworkAccessManager(this);
// подключение обработчиков сигналов
connect(m_nam, &QNetworkAccessManager::encrypted, this,
&Connector::onRequestEncrypted);
connect(m_nam, &QNetworkAccessManager::sslErrors, this,
&Connector::onRequestSslErrors);

const QString hostname = QStringLiteral("https://gost.security.ru");
QUrl url(hostname);

// создание HTTP-запроса
QNetworkRequest request;
request.setSslConfiguration(configuration);
request.setUrl(url);

// отправка HTTP-запроса
manager->get(request);
```

Необходимо настроить обработку сигналов, где с помощью проверки значения `keyExchangeMethod` подтверждается, что подключение к серверу было выполнено с помощью шифрования согласно ГОСТ:

```
void Connector::onRequestEncrypted()
{
    qDebug() << Q_FUNC_INFO << "keyExchangeMethod:"
      << reply->
sslConfiguration().sessionCipher().keyExchangeMethod();
}
```

Обработка сигнала при возникновении ошибки SSL-соединения выполняется с помощью следующей команды:

```
void Connector::onRequestSslErrors(QNetworkReply *reply,
const QList<QSslError> &errors)
{
    qDebug() << Q_FUNC_INFO << reply << errors;
}
```

Значения ошибок, которые могут возникнуть во время установления связи SSL, приведены в описании класса `QSslError`.

ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

Изм.	Номера листов (страниц)				Всего листов (страниц) в докум.	№ документа	Входящий № сопроводительного докум. и дата	Подп.	Дата
	измененных	замененных	новых	аннулированных					