КРИПТОКОМ

УТВЕРЖДЕН
СЕИУ.00009-02 33 01 - ЛУ

СРЕДСТВО КРИПТОГРАФИЧЕСКОЙ ЗАЩИТЫ ИНФОРМАЦИИ
МагПро КриптоПакет вер. 2.0

Библиотека libcrypto. Руководство программиста

СЕИУ.00009-02 33 01
Листов 114

Литера О

2011

Аннотация

Настоящий документ содержит описание интерфейса прикладных программ к библиотеке libcrypto из состава СКЗИ «МагПро Криптопакет», реализующей работу с сертификатами X509, заявками на выпуск сертификата (PKCS10) и защищенными сообщениями электронной почты (PKCS7).

Содержание

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

1 Инициализация библиотеки

Инициализация библиотеки состоит из следующих этапов:

1. Инициализация стандартных алгоритмов. Выполняется функцией **OpenSSL_add_all_algorithms** или **SSL_library_init**
2. Инициализировать таблицы сообщений об ошибках. Выполняется функцией **ERR_load_error_strings** (см раздел 18.6) или **SSL_load_error_strings** (см описание библиотеки libssl) .
3. Подгрузка модуля реализации алгоритмов ГОСТ одним из способов, описанных ниже.

В случае приложения, использующего функциональность libssl, подгрузка модуля, реализующего алгоритмы ГОСТ должна быть осуществлена до вызова фукнции **SSL_library_init**, так как если алгоритмы ГОСТ не будут доступны в момент вызова этой функции, то шифрсьюты ГОСТ в список доступных не попадут. В остальном, порядок выполнения этих этапов произвольный.

```
void OpenSSL_add_all_algorithms(void)
```

Активизирует втсроенныае криптоалгоритмы OpenSSL. Используется в приложениях, работающих с подписанными/зашифрованными документами, таймштапингом и прочей функциональностью libcrypto, но не ипольующих функциональность libssl.

```
void OPENSSL_add_all_algorithms_conf()
```

Актиизирует встренные алгоритмы и считывает файл конфигурации (что позволяет также активизировать модуль поддержки алгоритмов ГОСТ)

1.1 Инициализация с использованием конфигурационного файла

Стандартным способом инициализации библиотеки libcrypto является считывание конфигурационного файла OpenSSL. Оно выполняется с помощью функции
OPENSSL_config.

```
void OPENSSL_config(char *config_name)
```

Выполняет чтение стандартого конфигурационного файла, в частности загрузку подгружаемых модулей реализации алгоритмов, описанных в этом файле. Если в качестве параметра `config_name` указать `NULL`, будет использован стандартный алгоритм поиска файла конфигурации, реализованный функцией **CONF_get1_default_config_file**.

```
char *CONF_get1_default_config_file(void)
```

Возвращает имя стандартного файла конфигурации.
Алгоритм формирования имени стандартного файла конфигурации следующий:

1. Если установлена переменная среды `OPENSSL_CONF`, используется имя файла, указанное в этой переменной.
2. Если переменная не установлена, используется файл `openssl.cnf` находящийся в директории, заданной при компиляции как параметр openssldir скрипта `Configure`.

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

1.2 Инициализация без использования конфигурационного файла

В некоторых приложениях использование стандартного (общесистемного) файла конфигурации OpenSSL нежелательно.

В этих случаях можно осуществить подгрузку модуля реализации алгоритмов ГОСТ средствами приложения.

Для этой цели употребляется функция **ENGINE_by_id**.

```
ENGINE *ENGINE_by_id(const char *id)
```

Возвращает указатель на engine по её идентификатору. Например:
ENGINE *e=ENGINE_by_id("cryptocom");
Эта функция позволяет получить указатель на engine в следующих случаях:

1. engine уже загружена в текущий процесс и требуется получить ее идентификатор для передачи в такие функции как **ENGINE_load_private_key**, **EVP_DigestInit_ex**, **EVP_CipherInit_ex**.
2. engine вкомпилирована в библиотеку OpenSSL статически.
3. engine находится в директории, из которой OpenSSL осуществляет загрузку динамических engines.
   Директория, отличная от умолчательной может быть задана с помощью переменной среды `OPENSSL_ENGINES`.

После загрузки engine cryptocom с помощью **ENGINE_by_id** необходимо установить эту engine как умолчательную для реализации всех типов криптоаглогоритмов, на которые существуют российские стандарты.

Для этого применяется функция **ENGINE_set_default**.

```
int ENGINE_set_default(ENGINE *e, unsigned int flags)
```

В качестве первого параметра в эту функцию передается ссылка на engine, возвращенная функией **ENGINE_by_id**, а в качестве параметра комбинация из констант `ENGINE_METHOD_*`, описанных в файле `engine.h`. Рекомендуется использование константы `ENGINE_METHOD_ALL`.

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

2 Создание и обработка заявок, сертификатов и списков отзыва

Заявка на получение сертификата (Certificate Signing Request, CSR) формата PKCS10 представляется в памяти структурой `X509_REQ`.

Сертификат представляется структурой `X509`, а список отзыва (Certificate Revocation List, CRL) — `X509_CRL`.

Для всех этих структур определены функции чтения записи в форматах PEM и DER с использованием абстракци ввода-вывода `BIO`.

Интерфейс этих функций единообразен для всех типов данных и описан в разделе 18.3.

2.1 Формирование заявки

Для создания заявки на сертификат необходимо:

1. Создать пустую структуру `X509_REQ` с помощью функции **X509_REQ_new**;
2. Установить версию заявки (0 для версии 1) с помощью функции **X509_REQ_set_version**;
3. Заполнить поле subject, сформировав структуру `X509_NAME` и установить её с помощью **X509_REQ_set_subject_name** (см раздел 2.4);
4. Добавить атрибуты, если необходимо, с помощью **X509_REQ_add1_attr_by_txt** или **X509_REQ_add1_attr_by_NID**;
5. Установить открытый ключ с помощью функции **X509_REQ_set_pubkey**;
6. Добавить, если необходимо, расширения X509v3 поместив их в заявку с помощью функции **X509_REQ_add_extension**;
7. Подписать заявку закрытым ключом, парным к открытому ключу, добавленному в заявку, с помощью функции **X509_REQ_sign**.
8. Сохранить заявку в файл или блок памяти с помощью функции **PEM_write_bio_X509_REQ_NEW**, **PEM_write_bio_X509_REQ** или **i2d_X509_REQ_bio**. Функция **PEM_write_bio_X509_REQ_NEW** отличается от **PEM_write_bio_X509_REQ** тем, что создает заголовок «BEGIN NEW CERTIFICATE REQUEST» вместо «BEGIN CERTIFICATE REQUEST».
9. Освободить структуру `X509_REQ` вызовом **X509_REQ_free**.

```
X509_REQ * X509_REQ_new();
```

Размещает в памяти структуру `X509_REQ`.

```
void X509_REQ_free(X509_REQ *)
```

Освобождает структуру `X509_REQ`, размещенную с помощью **X509_REQ_new**.

```
int X509_REQ_set_version(X509_REQ *x, long version);
```

Устанавливает требуемый номер версии для заявки.

```
int X509_REQ_set_subject_name(X509_REQ *req, X509_NAME *name);
```

Устанавливает поле subject заявки. Параметр ***name*** указывает на структуру `X509_NAME`, способы создания которой описаны в разделе 2.4.

```
int X509_REQ_set_pubkey(X509_REQ *x, EVP_PKEY *pkey);
```

Устанавливает публичный ключ заявки из указанной структуры ключевой пары. См раздел 9.

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

```
int X509_REQ_add1_attr_by_OBJ(X509_REQ *req,
       const ASN1_OBJECT *obj, int type,
       const unsigned char *bytes, int len);

int X509_REQ_add1_attr_by_NID(X509_REQ *req,
       int nid, int type,
       const unsigned char *bytes, int len);

int X509_REQ_add1_attr_by_txt(X509_REQ *req,
       const char *attrname, int type,
       const unsigned char *bytes, int len);
```

Добавляют атрибут в заявку.

Параметры:

`req` — Структура `X509_REQ`, в которую добавляется атрибут;
`obj` — ASN.1 OID идентифицирующий атрибут в виде структуры `ASN1_OBJECT`;
`nid` — Числовой идентификатор (NID) OpenSSL, соответствующий OID атрибута
`attrname` — Имя атрибута. Может быть либо строковым представлением OID, либо символическим идентификатором (short name), определенным в файле `objects.dat` OpenSSL или описанным в соотвествующей секции файла конфигурации.
`type` — тип атрибута. Может быть одной из констант `MBSTRING_xxx`, определенных в файле `asn1.h`, или одной из констант `V_ASN1_BIT_STRING` или `V_ASN1_OCTET_STRING`, если содержимое атрибута бинарное.
`bytes` — буфер, содержащий значение атрибута. Содержимое буфера должно соответствовать указанному типу.
`len` — Количество байт в буфере ***bytes***.

```
int X509_REQ_add_extensions(X509_REQ *req,
       STACK_OF(X509_EXTENSION) *exts);
```

Добавляет в заявку ***req*** X509v3 расширения, содержащиеся в стеке ***exts***. См раздел 2.5.

```
int X509_REQ_sign(X509_REQ *x, EVP_PKEY *pkey, const EVP_MD *md);
```

Подписывает заявку закрытым ключом, содержащимся в объекте ключевой пары ***pkey*** (это должен быть тот же самый объект, открытый ключ которого был установлен в качестве ключа заявки с помощью **X509_REQ_set_pubkey**) с использованием алгоритма хэширования ***md***.

При создании заявок с алгоритмом ГОСТ Р 34.10-2001 в качестве алгоритма хэширования необходимо использовать ГОСТ Р 34.11-94. Структуру `EVP_MD` для этого алгоритма можно получить с помощью вызова **EVP_get_digestbyname("md_gost94")** (см. раздел 7).

2.2 Анализ заявок, сертификатов и CRL

Как правило, приложениям приходится анализировать сертификаты, полученные в процессе установления TLS-соединения или в составе PKCS7-сообщения.

Соответствующие информационные функции возвращают сертификат в виде структуры `X509`. В тех редких случаях, когда необходимо загрузить сертификат из файла, используются функции загруки PEM или DER-формата, описанные в разделе 18.3.

2.2.1 Серийный номер сертификата

```
ASN1_INTEGER * X509_get_serialNumber(X509 *x);
```

Возвращает серийный номер сертификата в виде структуры `ASN1_INTEGER`. Возвращенный указатель является частью структуры `X509`. Освобождать его нет необходимости, но при освобождении структуры `X509` он станет невалидным.

Структуры `ASN1_INTEGER` могут быть преобразованы в `long` с помощью функции

```
long ASN1_INTEGER_get(ASN1_INTEGER *a);
```

В случае если число, хранящееся в струкутре `ASN1_INTEGER`, выходит за пределы допустимого диапазона для `long`, функция возвращает `0xffffffffL`. В этом случае можно преобразовать `ASN1_INTEGER` в тип `BIGNUM` OpenSSL с помощью функции

```
BIGNUM *ASN1_INTEGER_to_BN(ASN1_INTEGER *ai, BIGNUM *bn);
```

Параметр ***bn*** может указывать на уже размещенную в памяти переменную типа `BIGNUM`, тогда значение будет помещено в неё, или быть равным `NULL`, тогда будет динамически размещена новая переменная типа `BIGNUM`, которую после использования нужно будет освободить с помощью **BN_free**.

Распечатать значение переменной типа `BIGNUM` можно с помощью функции

```
int BN_print(BIO *fp, const BIGNUM *a);
```

Кроме того, можно получить строковое представление числа в десятичном или шестнадцатиричном виде с помощью функций

```
char *BN_bn2hex(const BIGNUM *a);

char *BN_bn2dec(const BIGNUM *a);
```

Эти функции возвращают динамически размещенную в памяти строку, которая должна быть освобождена с помощью **OPENSSL_free**.

2.2.2 Сроки действия сертификата и CRL

```
ASN1_TIME *X509_get_notBefore(X509 *cert);

ASN1_TIME *X509_get_notAfter(X509 *cert);

ASN1_TIME *X509_CRL_get_lastUpdate(X509_CRL *crl);

ASN1_TIME *X509_CRL_get_nextUpdate(X509_CRL *crl);
```

Возвращают дату начала и конца срока действия сертификата, дату выпуска списка отзыва и дату выпуска следующего списка отзыва (дату устаревания списка отзыва).

Значение `ASN1_TIME` может быть выведено с помощью функции

```
int ASN1_TIME_print(BIO *fp, ASN1_TIME *a);
```

2.2.3 Наименование владельца сертификата/заявки и УЦ, выпустившего сертификат/CRL

```
X509_NAME *X509_get_subject_name(X509 *cert);

X509_NAME *X509_get_issuer(X509 *cert);

X509_NAME *X509_REQ_get_subject_name(X509_REQ *req);

X509_NAME *X509_CRL_get_issuer(X509_CRL *crl);
```

Функции позволяют получить наименование владельца (subject) сертификата или заявки и наименование УЦ (issuer), выпустившего сертификат и CRL. Функции анализа структуры `X509_NAME` описаны в разделе 2.4.

2.2.4 Расширения X509v3

```
int X509_get_ext_count(X509 *cert)

int X509_CRL_get_ext_count(X509_CRL *x);
```

Возвращает количество расширений, содержащихся в сертификате или CRL.

```
int X509_get_ext_by_NID(X509 *x, int nid, int lastpos);

int X509_get_ext_by_OBJ(X509 *x, ASN1_OBJECT *obj, int lastpos);

int X509_CRL_get_ext_by_NID(X509_CRL *x, int nid, int lastpos);

int X509_CRL_get_ext_by_OBJ(X509_CRL *x, ASN1_OBJECT *obj,
       int lastpos);
```

Производят поиск расширения по его OID, заданному структурой `ASN1_OBJECT` или NID, начиная с позиции ***lastpos***. Возвращают позицию расширения в списке или -1, если расширение не найдено.

```
X509_EXTENSION *X509_get_ext(X509 *x, int loc);

X509_EXTENSION *X509_CRL_get_ext(X509_CRL *x, int loc);
```

Возвращает расширение по позиции в списке в виде структуры `X509_EXTENSION`.

```
STACK_OF(X509_EXTENSION) *X509_REQ_get_extensions(X509_REQ *req);
```

Возвращает все расширения заявки в виде стека.

Функции, позволяющие проанализировать структуру `X509_EXTENSION`, описаны в разделе 2.5.

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

2.2.5 Информационные функции

```
STACK_OF(OPENSSL_STRING) *X509_get1_email(X509 *x)
```

Возвращает стэк адресов электронной почты, встречающихся в DN и расширениях сертификата.

Полученный стэк необходимо освободить после использования с помощью функции

```
void X509_email_free(STACK_OF(STRING) *sk)
```

```
STACK_OF(OPENSSL_STRING) *X509_get1_ocsp(X509 *x)
```

Возвращает список URL OCSP-респондеров, содержащийся в сертификате. После использования стэк необходимо освободить с помощью функции **X509_email_free**

2.2.6 Низкоуровенвые функции проверки подписи

Обычно приложения не используют низкоуровневых проверок подписи под сертификатами, CRL и заявками. Вместо этого используется объект `X509_STORE`, реализующий хранилище доверенных сертификатов и CRL.

Тем не менее для отладочных целей и диагностики ошибок может понадобиться проверить подпись под объектом на явно указанном ключе.

```
EVP_PKEY *X509_get_pubkey(X509 *cert);
```

```
EVP_PKEY *X509_REQ_get_pubkey(X509_REQ *req);
```

Извлекает открытый ключ из сертификата/заявки в виде структуры хранения ключевой пары `EVP_PKEY`.

```
int X509_verify(X509 *cert, EVP_PKEY *r);
```

```
int X509_REQ_verify(X509_REQ *req, EVP_PKEY *r);
```

```
int X509_CRL_verify(X509_CRL *crl, EVP_PKEY *r);
```

Проверяют подписи под сертификатом, заявкой и CRL соответственно, используя явно указанный открытый ключ. Возвращают 1, если проверка успешна, и 0, если подпись некорректна.

2.3 Проверка соответствия сертификата и секретного ключа

Функция

```
int X509_check_private_key(X509 *x,EVP_PKEY *k);
```

Проверяет, что указанный сертификат ***x*** содержит открытый ключ, соответствующий ключевой паре ***k***.

Возвращает 1 в случае успеха, 0 в случае несоответствия ключа, -1 в случае несоответствия алгоритма и -2 в случае несоответсвия параметров эллиптической кривой или ошибки формата.

Рекомендуется вызывать эту функцию после загрузки сертификата и секретного ключа перед выполнением операций электронной подписи, чтобы убедиться, что загруженные ключевые данные корректны.

В случае, если ключ предполагается использовать для установления TLS-соединений, вместо данной функции можно пользоваться **SSL_check_private_key** из библиотеки `libssl`.

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

2.4 Манипулирование X509 Distinguished Name.

2.4.1 Создание и освобождение

```
X509_NAME *X509_NAME_new();
```

Размещает новый объект типа `X509_NAME`.

```
void X509_NAME_free(X509_NAME *n);
```

Освобождает объект типа `X509_NAME`.

2.4.2 Добавление элементов

.

```
int X509_NAME_add_entry_by_txt(X509_NAME *name, const char *field,
      int type, const unsigned char *bytes, int len, int loc, int
      set);

int X509_NAME_add_entry_by_OBJ(X509_NAME *name, ASN1_OBJECT *obj,
      int type, unsigned char *bytes, int len, int loc, int set);

int X509_NAME_add_entry_by_NID(X509_NAME *name, int nid, int type,
      unsigned char *bytes, int len, int loc, int set);
```

Параметр ***name*** задает структуру, которая должна быть модифицирована. Параметры ***field***, ***nid*** и ***obj*** задают OID поля, которое должно быть добавлено. ***field*** может быть либо строковым представлением OID, либо строковым идентификатором (short name), определенным для данного OID в коде OpenSSL или в конфигурационном файле. ***nid*** — внутренний числовой идентификатор, присвоенный OID-у.

Параметр ***type*** задает тип строки. Это одна из констант `MBSTRING_xxx`, определенных в файле `asn1.h`. Для полей на русском языке рекомендуется использование `MBSTRING_UTF8`, для полей содержащих только символы ASCII — `MBSTRING_ASC`.

Параметр ***bytes*** — указатель на буфер, содержащий текст поля. Параметр ***len*** — длина значения в байтах. Если ***len*** равно -1, содержимое ***bytes*** интерпретируется как NUL-терминированная строка.

Параметр ***loc*** указывает позицию в списке полей, куда должно быть добавлено данное поле. Обычно используется значение -1 (добавлять в конец списка).

Параметр ***set*** может принимать значения 0, 1 или -1. Если значение равно 0, то добавляется новая запись. Если значение -1 или 1, то значение добавляется к предыдущей или последующей записи. Многозначные записи используются в X509 крайне редко, поэтому как правило, ***set*** будет иметь значение 0.

Возвращаемое значение 1 в случае удачного завершения и 0 в случае ошибки.

2.4.3 Просмотр и поиск элементов

```
int X509_NAME_entry_count(X509_NAME *name);
```

Возвращает количество полей в имени.

```
X509_NAME_ENTRY *X509_NAME_get_entry(X509_NAME *name, int loc);
```

Позволяет получить значение поля имени по его позиции.

```
X509_NAME_ENTRY *X509_NAME_delete_entry(X509_NAME *name, int loc);
```

Удаляет запись из структуры и возвращает указатель на удаленную запись (или NULL в случае ошибки).

Возвращенная запись должна быть освобождена с помощью **X509_NAME_ENTRY_free**.

```
int X509_NAME_get_index_by_NID(X509_NAME *name, int nid, int lastpos);

int X509_NAME_get_index_by_OBJ(X509_NAME *name, ASN1_OBJECT *obj, int
      lastpos);
```

Поиск поля с определенным OID, начиная с позиции ***lastpos***. Возвращают позицию поля в списке или -1, если поля не обнаружено.

2.4.4 Анализ отдельного поля имени

```
ASN1_OBJECT * X509_NAME_ENTRY_get_object(X509_NAME_ENTRY  *ne);
```

Возвращает OID имени в виде структуры ASN1_OBJECT.

Получить числовой идентификатор (NID) можно с помощью функции **OBJ_obj2nid**, а по числовому идентификатору можно получить строковый идентификатор и описательное наименование с помощью **OBJ_nid2sn** и **OBJ_nid2ln** соответственно. Для OID, не зарегистрированных в таблицах OpenSSL, строковое представление OID может быть получено с помощью **OBJ_obj2txt**.

```
ASN1_STRING *X509_NAME_ENTRY_get_data(X509_NAME_ENTRY *ne);
```

Возвращает значение расширения. Полученное значение может быть преобразовано в строку с помошью **ASN1_STRING_print_ex**, описанной в разделе 18.4.

2.4.5 Манипуляции с именем в целом

```
int X509_NAME_cmp(const X509_NAME *a, const X509_NAME *b);
```

Сравнивает две структуры X509_NAME. Возвращает 0, если они равны, и -1 или 1, если не равны. Задает некоторый порядок на множестве структур X509_NAME.

```
int X509_NAME_print_ex(BIO *out, X509_NAME *nm, int indent,
      unsigned long flags);
```

Выводит имя в соответствии с заданными флагами. Поле флагов представляет собой битовую маску (т.е. допустимо комбинировать флаги с помощью битовых операций). Параметр ***indent*** задает отступ от начала строки (используется в основном, если используемые флаги задают многострочный вывод).

Определены следующией флаги

XN_FLAG_SEP_COMMA_PLUS — Разделять компоненты имени запятой, а значения многозначной компоненты — плюсом.

XN_FLAG_SEP_CPLUS_SPC — Использовать те же разделители, но с добавлением пробелов.

XN_FLAG_SEP_SPLUS_SPC — Использовать точку с запятой с пробелом в качестве разделителя полей.

XN_FLAG_SEP_MULTILINE — Выводить каждую компоненту на отдельной строке.

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

XN_FLAG_DN_REV — Выводить компоненты в порядке, противоположном тому, в котором они хранятся в структуре.
XN_FLAG_FN_SN — Использовать строковые идентификаторы (shortname) в качестве имен полей.
XN_FLAG_FN_LN — Использовать в качестве имен полей описательные наименования (long name).
XN_FLAG_FN_OID — Использовать строковое представление OID в качестве имен полей.
XN_FLAG_FN_NONE — Выводить только значения полей, без имен.
XN_FLAG_SPC_EQ — Выодить пробелы вокруг знака '=', отделяющего имя поля от значения.
XN_FLAG_DUMP_UNKNOWN_FIELDS — Выводить неизвестные поля как шестнадцатиричный дамп ASN.1 представления.
XN_FLAG_FN_ALIGN — Выравнивать имена полей до 20 символов (имеет смысл только при многострочном выводе).

Кроме того, можно использовать все флаги, определенные для функции **ASN1_STRING_PRINT_ex** (см. раздел 18.4.

Для наиболее распространенных форматов вывода определены следующие комбинации флагов

XN_FLAG_RFC2253 — Комбинация ASN1_STRFLGS_RFC2253, XN_FLAG_SEP_COMMA_PLUS, XN_FLAG_DN_REV, XN_FLAG_FN_SN, XN_FLAG_DUMP_UNKNOWN_FIELDS.
XN_FLAG_ONELINE — Комбинация ASN1_STRFLGS_RFC2253, ASN1_STRFLGS_ESC_MSB,ASN1_STRFLGS_ESC_QUITE, XN_FLAG_SEP_CPLUS_SPC, XN_FLAG_SPC_EQ, XN_FLAG_FN_SN.
XN_FLAG_MULTILINE — Комбинация ASN1_STRFLGS_ESC_CTRL, ASN1_STRFLGS_ESC_MSB,XN_FLAG_SEP_MULTILINE, XN_FLAG_SPC_EQ, XN_FLAG_FN_LN, XN_FLAG_FN_ALIGN.

Эти стандартные комбинации мало пригодны для русскоязычных полей, так как включают флаг ASN1_STRFLGS_ESC_MSB. Можно использовать комбинацию этих флагов с & ~ASN1_STRFLGS_ESC_MSB для получения корректного представления русских букв в кодировке UTF-8.

2.5 Манипуляции с расширениями X509v3

Расширения X509v3 представляются в виде структуры X509_EXTENSION. Эта структура содержит собственно данные расширения в виде сериализованной ASN1-структуры.

API для работы с этими структурами поддерживает дополнительные возможности для ряда стандартных расширений, описанных в RFC5280.

Для этих расширений предусмотрено их формирование из внутреннего представления (специфичного для каждого расширения) и извлечение из сертификата/заявки/списка отзыва с разбором во внутреннее представление.

Кроме этого предусмотрена печать этих расширений в человеко-читаемую строку и формирование на основе строки, эквивалентной строке в конфигурационном файле openssl.cnf (см руководство оператора утилиты openssl).

OpenSSL поддерживает механизм для добавления обработчиков нестандартных расширений.

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

```
void *X509_EXT_d2i(X509_EXTENSION *ext)
```

Разбирает расширение, содержащееся в переменной ext в структуру, зависящую от типа расширения. Возвращает `NULL` если расширение неизвестно. В противном случае реальный тип возвращаемого указателя зависит от расширения. Например для расширения subjectAltName это будет `STACK_OF(GENERAL_NAME)`.

```
int     X509_EXTENSION_get_critical(X509_EXTENSION *ex);
```

Получает флаг критичности.

```
ASN1_OBJECT *   X509_EXTENSION_get_object(X509_EXTENSION
      *ex);
```

Получает OID указанного расширения.

```
X509_EXTENSION *X509_EXT_i2d(int ext_nid, int crit, void *ext_struc)
```

Создает структуру `X509_EXTENSION` по числовому идентификатору (NID) расширения, флагу критичности и специфичной для ext_nid структуре данных.

```
X509_EXTENSION *X509_EXTENSION_create_by_NID(X509_EXTENSION
      **ex,
      int nid, int crit, ASN1_OCTET_STRING *data);

X509_EXTENSION *X509_EXTENSION_create_by_OBJ(X509_EXTENSION
      **ex,
      ASN1_OBJECT *obj,int crit,ASN1_OCTET_STRING *data);
```

Создает расширение с указанным OID, флагом критичности и данными. Данные передаются в виде `ASN1_OCTET_STRING`.

```
ASN1_OCTET_STRING *X509_EXTENSION_get_data(X509_EXTENSION
      *ne);
```

Получаает данные указанного расширения.

2.6 Манипуляции с атрибутами X509v3

3 Работа с контейнерами PKCS#12

Контейнеры PKCS#12 представляют собой стандартный формат для хранения ключевой пары (т.е. закрытого ключа вместе с сертификатом) и используются разнообразными программными продуктами для резервного копирования и передачи ключевых пар.

3.1 Загрузка PKCS#12

Функции **d2i_PKCS12_bio** и **d2i_PKCS12_fp** производят загрузку файла PKCS#12 (обычно имеющего расширение .p12 или .pfx) в структуру `PKCS12`. В процессе загрузки не производится расшифрования частей структуры, зашифрованных на транспортном пароле. Для этого нужно к полученной струкуре применить функцию **PKCS12_parse**.

```
PKCS12 *d2i_PKCS12_bio(BIO *bp, PKCS12 **p12);

PKCS12 *d2i_PKCS12_fp(FILE *fp, PKCS12 **p12);
```

Обе функции имеют сходный синтаксис. Пермым параметром передается указатель на поток (структуру BIO библиотеки libcrypto или структуру FILE стандартной библиотеки языка C) откуда следует читать PKCS#12 контейнер.

Вторым параметром передается указатель на переменную, куда следует поместить указатель на прочитанную структуру. Если в качестве второго аргумента передан `NULL`, указатель будет только возвращен в качестве возвращаемого значения функции.

Если на входе в функцию указатель p12 указывает на переменную, содержащую ненулевой указатель, предполагается что он указывает на валидную структуру `PKCS12`, которую следует переиспользовать.

Указатель, возвращенный этими функциями, может быть освобожден с помощью функции **PKCS12_free**.

Функция **PKCS12_parse** позволяет разобрать наиболее распространенные разновидности контейнеров PKCS#12.

```
int PKCS12_parse(PKCS12 *p12, const char *pass, EVP_PKEY
      **pkey, X509 **cert, STACK_OF(X509) **ca)
```

Параметр `pass` должен содержать только символы ASCII. OpenSSL не поддерживает использования символов Unicode, таких как русские буквы, в паролях PKCS#12. Западноевропейские буквы с умляутами могут быть использованы, но в этом случае пароль должен быть представлен в кодировке iso8859-1, а не UTF-8. Данное ограничение касается также функции **PKCS12_create** и низкоуровневых функций зашифрования/расшифрования отдельных контейнеров, описанных ниже.

Функция выполняет расшифрование зашифрованных контейнеров с помощью пароля `pass` и помещает закрытый ключ в переменную `pkey`, сертификат этого закрытого ключа в переменную `cert`, и дополнительные сертификаты (обычно сертификаты промежуточных УЦ), содержащиеся в контейнере — в переменную `ca`.

Параметры `cert` и `pkey` не могут быть `NULL`. Параметр `ca`, может быть `NULL`, в этом случае дополнительные сертификаты, содержащиеся в контейнере, игнорируются. Переменная `*ca` может быть указателем на корректный стэк. Тогда сертификаты будут добавлены в этот стэк. Если переменная `*ca` равна NULL, будет размещен в памяти новый стек.

Если сертификатам в контейнере приписаны атрибуты `friendlyName` и `localKeyId`, они сохраняются в полях `alias` и `keyid` структуры `X509`. Прочие атрибуты сертификатов и атрибуты закрытого ключа игнорируются.

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

Функция возвращает 1, если контейнер был разобран успешно, и 0 в случае ошибки.

Данная функция позволяет разобрать только наиболее распространенный случай контейнера PKCS#12 — содержащий ровно один закрытый ключ, его сертификат и ноль или более сертификатов промежуточных УЦ.

Для разбора PKCS#12 контейнеров общего вида следует пользоваться более низкоуровневыми функциями описанными ниже.

3.2 Низкоуровневые функции разбора PKCS#12

Функция **PKCS12_verify_mac** выполняет проверку кода аутентификации (Message Authentication Code) контейнера PKCS#12. Функция **PKCS12_parse** вызывает эту проверку, а в случае использования низкоуровневых функций эта проверка должна быть произведена прежде чем будет выполняться извлечение данных из структуры `PKCS12`.

```
PKCS12_verify_mac(PKCS12 *p12, char *pass, int passlen)
```

В случае, если пароль является NUL-terminated строкой, можно использовать значение `passlen`, равное -1.

Функция **PKCS12_unpack_authsafes** позволяет разобрать структуру `PKCS12` на набор структур `PKCS7`, которые могут быть как зашифрованными, так и не зашифрованными.

```
STACK_OF(PKCS7) PKCS12_unpack_authsafes(PKCS12 *p12);
```

Возвращает стэк `PKCS7` структур или `NULL`, если произошла ошибка.

У зашифрованных структур поле `type` будет равно `NID_pkcs7_encrypted`, у незашифрованных `NID_pkcs7_data`. (Незашифрованные PKCS7 структуры могут содержать внутри себя зашифрованный в формате PKCS#8 закрытый ключ).

Освободить стек можно вызовом **sk_PKCS7_pop_free**.

Распаковка этих структур производится с помощью функций **PKCS12_unpack_p7data** и **PKCS12_unpack_p7encdata**.

```
STACK_OF(PKCS12_SAFEBAG) PKCS12_unpack_p7data(PKCS7 *p7);

STACK_OF(PKCS12_SAFEBAG) PKCS12_unpack_p7encdata(PKCS7 *p7,
        char *pass, int passlen);
```

Определение типа отдельного safebag производится с помощью макроса **M_PKCS12_bag_type**. Он может возвращать следующие константы:

`NID_keyBag` — незашифрованный закрытый ключ. В этом случае поле `value.keybag` содержит структуру `PKCS8_PRIV_KEY_INFO`, которая может быть преобразована в `EVP_PKEY` с помощью функции **EVP_PKCS82PKEY**.

`NID_pkcs8ShroudedKeyBag` — Содержит зашифрованную PKCS#8 структуру. Структура `PKCS8_PRIV_KEY_INFO` может быть расшифрована с помощью функции **PKCS12_decrypt_skey**.

`NID_certBag` — Содержит сертификат. Макрос **M_PKCS12_cert_bag_type** должен возвращать `NID_x509Certificate`. Сертификат может быть извлечен посредством функции **PKCS12_certbag2x509**.

`NID_crlBag` — Содержит список отзыва. Макрос **M_PKCS12_cert_bag_type** должен возвращать `NID_x509Crl`. Список отзыва может быть извлечен с помощью функции **PKCS12_certbag2x509crl**.

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

NID_safeContentsBag — Содержит вложенные контейнеры. Поле value.safes содержит STACK_OF(PKCS12_SAFEBAG).

```
int M_PKCS12_bag_type(PKCS12_SAFEBAG *bag);

int M_PKCS12_cert_bag_type(PKCS12_SAFEBAG *bag);
```

Возвращают численный идентификатор (NID) ASN1-объекта, задающего тип контейнера или подтип для контейнера, содержащего сертификаты.

```
PKCS8_PRIV_KEY_INFO *PKCS12_decrypt_skey(PKCS1_SAFEBAG *bag,
      const char *pass, int passlen);
```

Расшифровывает секретный ключ, содержащийся в контейнере типа NID_pkcs8ShroudedKeyBag с помощью пароля pass длины passlen и возвращает его в виде PKCS8_PRIV_KEY_INFO.

```
X509 *PKCS12_certbag2x509(PKCS12_SAFEBAG *bag);
```

Извлекает из контейнера сертификат. В случае если тип контейнера не соответствует, возвращает NULL.

```
X509_CRL *PKCS12_certbag2x509crl(PKCS12_SAFEBAG *bag);
```

Извлекает из контейнера список отзыва. Если тип контейнера не соответствует, возвращает NULL.

Атрибуты контейнера, такие как friendlyName, CSPname и localKeyId, находятся в поле attrib структуры PKCS12_SAFEBAG в виде STACK_OF(X509_ATTRIBUTE). Рекомендуется использовать для работы с ними функции, описанные в разделе 2.6.

3.3 Создание контейнеров PKCS#12

Функция **PKCS12_create** обепечивает создание PKCS#12 во всех наиболее распространенных случаях.

```
PKCS12 *PKCS12_create(char *pass, char *name, EVP_PKEY *pkey,
      X509 *cert, STACK_OF(X509) *ca, int nid_key, int nid_cert,
      int iter, int mac_iter, int keytype)
```

Создает и возвращает структуру PKCS12, содерщающую закрытый ключ pkey, его сертификат cert и дополнительные сертификаты ca. Использует алгоритм шифрования, заданный парамтером nid_key для шифрования закрытого ключа и nid_cert для шифрования сертификатов. Параметр iter задает количество итераций для вывода ключа шифрования из пароля (если 0, используется умолчательное значение 2048). Параметр mac_iter задает количство итераций для алгоритма вычисления аутентификационного кода (MAC).

Параметр name задает «дружественное» имя для сертификата и закрытого ключа. (это имя можно также задать с помощью поля alias в структуре X509).

Данная функция не позволяет указать алгоритм хэширования, используемый в алгоритме вычисления кода аутентификации. Поэтому при создании PKCS#12 контейнеров, содержащих ключи российских алгоритмов, следует указывать -1 в качестве значения mac_iter и вычислять код аутентификации отдельно, явным вызовом функции **PKCS12_set_mac**.

В качестве значения параметра nid_key следует указывать NID_id_Gost28147_89 или

```
EVP_CIPHER_type(EVP_get_cipherbyname("gost89")
```

В качестве `cert_nid` можно указывать либо то же самое значение, либо -1 (использование последнего приведет к тому, что сертификаты будут храниться незашифрованными).

Параметр `keytype` задает тип ключа. Это нестандартное расширение, которое может быть использовано только ПО от Microsoft. Значение представляет собой комбинацию битовых флагов `KEY_EX` и `KEY_SIG`. Если параметр `keytype` равен нулю, данное расширение не создается.

```
PKCS12_set_mac(PKCS12 *p12, char *mpass, int passlen,
       unsigned char *salt, int saltlen, int iter, const EVP_MD
       *md_type)
```

Вычисляет и сохраняет в структуре `PKCS12` код аутентификации на базе алгоритма хэширования `md_type` и пароля `mpass`.

В качестве salt можно указывать `NULL` и `saltlen` 0. В этом случае в качестве «соли» будут использованы псевдослучайные байты, снятые с ДСЧ. Для совместимости с МагПро CSP 2.0 рекомендуется указывать количество итераций MAC, отличное от 1.

Для сохранения полученной струкуры `PKCS12` в файл следует использовать функции

```
int i2d_PKCS12_bio(BIO *out, PKCS12 *p12)
```

```
int i2d_PKCS12_fp(FILE *out, PKCS12 *p12)
```

Функции возвращают положительное значение в случае успеха и 0 в случае ошибки.

4 Создание и обработка сообщений S/MIME

Пример `smime_test.c` содержит несколько функций, демонстрирующих типичные действия по созданию и обработке сообщений S/MIME. Обработка ошибок, диагностика и использование дополнительных возможностей минимизированы с тем, чтобы подчеркнуть основные действия. Дополнительные возможности изложены в подробных описаниях задействованных функций. static-функции в примере являются служебными и демонстрируют, как правило, один из нескольких существенно различных способов получить требуемый результат, наиболее простой для целей данного примера. Другие способы изложены в других разделах документа.

```
#include <stdio.h>
#include <openssl/x509.h>

#define END fprintf(stderr, "%s:%d: %s failed\n", __FILE__, __LINE__, __FUNCTION__); goto end

static X509 *load_cert_pem(const char *filename)
        {
        X509 *x = NULL;
        BIO *cert_bio = NULL;
        if ((cert_bio = BIO_new_file(filename, "r")) == NULL)
                {END;}
        x = PEM_read_bio_X509_AUX(cert_bio, NULL, NULL, NULL);
        end:
        if (cert_bio) BIO_free(cert_bio);
        return x;
        }

static EVP_PKEY *load_private_key_pem (const char *filename)
        {
        EVP_PKEY *pkey = NULL;
        BIO *key_bio = NULL;
        key_bio = BIO_new_file(filename, "r");
        if (!key_bio) {END;}
        pkey = PEM_read_bio_PrivateKey(key_bio, NULL, NULL, NULL);
        end:
        if (key_bio) BIO_free(key_bio);
        return pkey;
        }

static X509_STORE *load_cert_store (const char *filename)
        {
        X509_STORE *store = NULL;
        X509_LOOKUP *lookup;
        store = X509_STORE_new();
        if (!store) {END;}
        lookup = X509_STORE_add_lookup(store, X509_LOOKUP_file());
        if (!lookup) {END;}
        if (!X509_LOOKUP_load_file(lookup,filename,X509_FILETYPE_PEM)) {END;}
        ERR_clear_error();
        return store;
        end:
```

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

```
        X509_STORE_free(store);
        return NULL;
        }

int encrypt ()
        {
        int rc = 1;
        const EVP_CIPHER *cipher = NULL;
        STACK_OF(X509) *rcp_certs = NULL;
        X509 *cert = NULL;
        BIO *in = NULL, *out = NULL;
        PKCS7 *p7 = NULL;
        cipher = EVP_get_cipherbyname("gost89");
        if (!cipher) {END;}
        rcp_certs = sk_X509_new_null();
        if (!rcp_certs) {END;}
        cert = load_cert_pem("smime_test_recipient_A/cert.pem");
        if (!cert) {END;}
        sk_X509_push(rcp_certs, cert);
        cert = load_cert_pem("smime_test_recipient_B/cert.pem");
        if (!cert) {END;}
        sk_X509_push(rcp_certs, cert);
        in = BIO_new_file("smime_test.c", "r");
        if (!in) {END;}
        out = BIO_new_file("smime_test.enc", "w");
        if (!out) {END;}
        p7 = PKCS7_encrypt(rcp_certs, in, cipher, 0);
        if (!p7) {END;}
        if (!SMIME_write_PKCS7(out, p7, NULL, 0)) {END;}
        rc = 0;
        end:
        if (rcp_certs) sk_X509_pop_free(rcp_certs, X509_free);
        if (in) BIO_free(in);
        if (out) BIO_free(out);
        if (p7) PKCS7_free(p7);
        return rc;
        }

int decrypt ()
        {
        int rc = 1;
        EVP_PKEY *pkey = NULL;
        X509 *cert = NULL;
        BIO *in = NULL, *out = NULL;
        PKCS7 *p7 = NULL;
        pkey = load_private_key_pem("smime_test_recipient_B/seckey.pem");
        if (!pkey) {END;}
        cert = load_cert_pem("smime_test_recipient_B/cert.pem");
        if (!cert) {END;}
        in = BIO_new_file("smime_test.enc", "r");
```

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

```
        if (!in) {END;}
        out = BIO_new_file("smime_test.dec", "w");
        if (!out) {END;}
        p7 = SMIME_read_PKCS7(in, NULL);
        if (!p7) {END;}
        if (!PKCS7_decrypt(p7, pkey, cert, out, 0)) {END;}
        rc = 0;
        end:
        if (pkey) EVP_PKEY_free(pkey);
        if (in) BIO_free(in);
        if (out) BIO_free(out);
        if (p7) PKCS7_free(p7);
        if (cert) X509_free(cert);
        return rc;
        }

int sign ()
        {
        int rc = 1;
        EVP_PKEY *pkey = NULL;
        X509 *cert = NULL;
        BIO *in = NULL, *out = NULL;
        PKCS7 *p7 = NULL;
        pkey = load_private_key_pem("smime_test_sender_A/seckey.pem");
        if (!pkey) {END;}
        cert = load_cert_pem("smime_test_sender_A/cert.pem");
        if (!cert) {END;}
        in = BIO_new_file("smime_test.c", "r");
        if (!in) {END;}
        out = BIO_new_file("smime_test.sgn", "w");
        if (!out) {END;}
        int flags = PKCS7_DETACHED|PKCS7_STREAM|PKCS7_TEXT;
        p7 = PKCS7_sign(cert, pkey, NULL, in, flags);
        if (!p7) {END;}
        if (!SMIME_write_PKCS7(out, p7, in, flags)) {END;}
        rc = 0;
        end:
        if (p7) PKCS7_free(p7);
        if (in) BIO_free(in);
        if (out) BIO_free(out);
        if (cert) X509_free(cert);
        if (pkey) EVP_PKEY_free(pkey);
        return rc;
        }

int verify ()
        {
        int rc = 1;
        BIO *in = NULL, *out = NULL, *indata = NULL;
        X509_STORE *store = NULL;
```

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

```
        PKCS7 *p7 = NULL;
        in = BIO_new_file("smime_test.sgn", "r");
        if (!in) {END;}
        out = BIO_new_file("smime_test.vrf", "w");
        if (!out) {END;}
        store = load_cert_store("smime_test_CA/cacert.pem");
        if (!store) {END;}
        p7 = SMIME_read_PKCS7(in, &indata);
        if (!p7) {END;}
        if (!PKCS7_verify(p7, NULL, store, indata, out, 0)) {END;}
        rc = 0;
        end:
        if (p7) PKCS7_free(p7);
        if (store) X509_STORE_free(store);
        if (in) BIO_free(in);
        if (indata) BIO_free(indata);
        if (out) BIO_free(out);
        return rc;
        }

static int usage (const char *name)
        {
        fprintf(stderr, "Usage: %s {enc|dec|sgn}\n", name);
        return 1;
        }

int main (int argc, char **argv)
        {
        int rc;
        if (argc < 2) return usage(argv[0]);
        OPENSSL_config(NULL);
        ERR_load_crypto_strings();
        if (!strcmp(argv[1], "enc"))
                rc = encrypt();
        else if (!strcmp(argv[1], "dec"))
                rc = decrypt();
        else if (!strcmp(argv[1], "sgn"))
                rc = sign();
        else if (!strcmp(argv[1], "vrf"))
                rc = verify();
        else
                return usage(argv[0]);
        if (rc)
                ERR_print_errors_fp(stderr);
        return rc;
        }
```

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

4.1 Создание зашифрованного сообщения S/MIME

Типичные действия при создании зашифрованного, но не аутентифицированного сообщения демонстрирует функция **encrypt**. Стандарт S/MIME предусматривает зашифрованное аутентифицированное сообщение, но существующие реализации его в приложениях не поддерживают эту возможность. В существующих реализациях для зашифрования сообщения наличие у отправителя ключевой пары не требуется, и использовать ее, даже при ее наличии, невозможно. Последовательность действий:

1. Получить указатель на описание алгоритма шифрования (структуру `EVP_CIPHER`). В типичном случае указатель получается по имени вызовом функции **EVP_get_cipherbyname**. Для алгоритма ГОСТ 28147-89 следует использовать имя `"gost89"`.
2. Сформировать набор сертификатов получателей. Набор сертификатов необходимо сформировать в виде стека указателей на структуры, представляющие сертификаты — `STACK_OF(X509)`. Для этого этот стек создается вызовом **sk_X509_new_null**, и затем пополняется вызовами **sk_X509_push**. Каждый сертификат в примере загружается из PEM-файла с помощью вызова **PEM_read_bio_X509_AUX**. Загрузка сертификата из PEM-файла выделена в служебную функцию **load_cert_pem**.
3. Создать объекты ввода-вывода (`BIO`) для входных и выходных данных. В примере оба объекта построены на файлах.
4. Вызвать функцию **PKCS7_encrypt**. Ей передаются набор сертификатов получателей, описание алгоритма шифрования и входные данные. Она возвращает указатель на структуру `PKCS7`, подтип `envelopedData`, содержащую зашифрованные данные и всю необходимую служебную информацию.
5. Вывести в выходной объект `BIO` полученную структуру вызовом **SMIME_write_PKCS7**.

4.2 Расшифрование зашифрованного сообщения S/MIME

Типичные действия при расшифрованиии зашифрованного сообщения S/MIME:

1. Загрузить свои закрытый ключ и сертификат. Сертификат требуется для выбора «своей» служебной информации из содержащихся в сообщении, так как сообщение может быть зашифровано сразу для нескольких получателей, и получатель определяется идентификатором его сертификата. В примере сертификат и закрытый ключ загружаются из PEM-файлов. Сертификат нужен в виде структуры `X509`, а закрытый ключ — в виде структуры `EVP_PKEY`.
2. Создать объекты ввода-вывода (`BIO`) для входных и выходных данных. В примере оба объекта построены на файлах.
3. Преобразовать входные данные в структуру `PKCS7` вызовом функции **SMIME_read_PKCS7**.
4. Вызвать функцию **PKCS7_decrypt**. Ей передаются указатели на структуру `PKCS7`, закрытый ключ, сертификат и объект вывода. Расшифрованное сообщение выводится в объект вывода, а функция возвращает факт успешности или неудачи расшифрования.

4.3 Создание подписанного сообщения S/MIME

Типичные действия при создании подписанного сообщения S/MIME:

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

1. Загрузить свои закрытый ключ и сертификат. При необходимости следует также загрузить необходимые дополнительные сертификаты, которые необходимо включить в письмо (например, сертификаты промежуточных УЦ). Закрытый ключ нужен в виде структуры `EVP_PKEY`, сертификат — в виде структуры `X509`, а дополнительные сертификаты — в виде `STACK_OF(X509)`.
2. Создать объекты ввода-вывода (`BIO`) для входных и выходных данных. В примере оба объекта построены на файлах.
3. Вызвать функцию **PKCS7_sign**. Ей передаются указатели на сертификат, закрытый ключ, дополнительные сертификаты, если они есть, и объект ввода. Она возвращает сформированную структуру `PKCS7`.
4. Преобразовать полученную структуру `PKCS7` в сообщение S/MIME и вывести его в объект вывода вызовом **SMIME_write_PKCS7**.

Флаги в функции **PKCS7_sign** и **SMIME_write_PKCS7** следует передавать одни и те же. В примере использована комбинация флагов, позволяющая создать сообщение типа `multipart/signed`, содержащее в двух своих частях сами данные и отделенную (detached) подпись. При такой комбинации флагов структура `PKCS7`, возвращенная функцией **PKCS7_sign**, будет еще не заполнена реальными данными, и заполнит ее лишь вызов **SMIME_write_PKCS7**.

4.4 Проверка подписи под сообщением S/MIME

Типичные действия при проверке подписи:

1. Загрузить хранилище доверенных сертификатов УЦ. В примере хранилище загружается из файла, содержащего единственный сертификат. Хранилище требуется в виде структуры `X509_STORE`. Возможно также загрузить набор дополнительных сертификатов, которые можно использовать в дополнение к или вместо содержащихся в сообщении.
2. Создать объекты ввода-вывода (`BIO`) для входного сообщения и выходных данных. В примере оба объекта построены на файлах.
3. Подготовить указатель на `BIO` для входных (тех, подпись под которыми проверяется) данных.
4. Вызовом **SMIME_read_PKCS7** разобрать входное сообщение на подпись (указатель на структуру `PKCS7` функция возвращает) и входные данные (в случае отделенной подписи функция создает объект `BIO` для данных и возвращает его адрес в переданном указателе).
5. Вызовом **PKCS7_verify** проверить подпись. Ей передаются указатели на подпись, набор дополнительных сертификатов, хранилище доверенных сертификатов, объект входных данных, объект выходных данных. Функция проверяет подпись и выводит подписанные данные в объект выходных данных.

5 Функции, предназначенные для создания и обработки сообщений S/MIME

5.1 Флаги в параметрах функций

Аргумент ***flags*** используется для указания особенностей режимов работы. Иногда флаг меняет работу функции незначительно, иногда — радикально. Это битовая маска, содержащая следующие биты, определенные в `pkcs7.h`: `PKCS7_TEXT`, `PKCS7_NOCERTS`, `PKCS7_NOSIGS`, `PKCS7_NOCHAIN`, `PKCS7_NOINTERN`, `PKCS7_NOVERIFY`, `PKCS7_DETACHED`, `PKCS7_BINARY`, `PKCS7_NOATTR`, `PKCS7_NOSMIMECAP`, `PKCS7_NOOLDMIMETYPE`, `PKCS7_CRLFEOL`, `PKCS7_STREAM`, `PKCS7_NOCRL`.

Некоторые из этих битов имеют разное, но согласованное значение для разных функций. Т.е. передавать один и тот же набор флагов, например, в функции **PKCS7_sign** и **SMIME_write_PKCS7** допустимо, а иногда и необходимо. Все функции безболезненно игнорируют «чужие» флаги (т.е. флаги, которым у них нет применения).

5.2 Зашифрование данных: **PKCS7_encrypt**

```
PKCS7 *PKCS7_encrypt(STACK_OF(X509) *certs, BIO *in,
       const EVP_CIPHER *cipher, int flags);
```

Функция зашифровывает входные данные, содержащиеся в объекте ввода ***in***, симметричным алгоритмом ***cipher*** для получателей, указанных набором сертификатов ***certs***. Она формирует структуру `PKCS7`, подтип `EnvelopedData`, и возвращает указатель на нее в случае успеха, и `NULL` в случае неудачи. Код ошибки можно получить вызовом **ERR_get_error**.

Флаги:

`PKCS7_TEXT` — данные будут зашифрованы вместе с MIME-заголовком для типа `text/plain`.

`PKCS7_BINARY` — если этот флаг указан, данные не будут перед зашифрованием преобразованы в соответствии со стандартом MIME (с заменой всех концов строк на последовательность `"\r\n"`). Флаг `PKCS7_TEXT` в этом случае будет проигнорирован.

Эта функция не использует закрытого ключа отправителя, так как подавляющее большинство приложений, отправляющих зашифрованные сообщения, не способно ей его предоставить.

Алгоритм транспорта ключа симметричного шифрования определяется алгоритмом открытого ключа из сертификата получателя. При использовании алгоритмов ГОСТ формирование служебной информации и транспорт сеансового ключа симметричного шифрования происходят согласно RFC 4490.

5.3 Расшифрование сообщения: **PKCS7_decrypt**

```
int PKCS7_decrypt(PKCS7 *p7, EVP_PKEY *pkey, X509 *cert, BIO *data,
       int flags);
```

Функция расшифровывает данные, содержащиеся в структуре ***p7***, с помощью закрытого ключа получателя ***pkey***. Сертификат ***cert*** используется для выбора «своей» служебной информации из набора имеющихся и обязан соответствовать ключу ***pkey***. Расшифрованные данные

выводятся в объект вывода ***data***. Функция возвращает 1 в случае успешной расшифровки и 0 в случае неудачи. Код ошибки можно получить вызовом **ERR_get_error**.

Флаги:

`PKCS7_TEXT` — если этот флаг указан, то из расшифрованных данных удаляется MIME-заголовок для типа `text/plain` (предположительно добавленный туда вызовом **PKCS7_encrypt**). Отсутствие этого заголовка или наличие иного считается ошибкой.

5.4 Подпись данных: PKCS7_sign

```
PKCS7 *PKCS7_sign(X509 *signcert, EVP_PKEY *pkey,
       STACK_OF(X509) *certs, BIO *data, int flags);
```

Функция подписывает данные, получаемые из объекта ввода ***data***, закрытым ключом ***pkey***, создавая структуру `PKCS7`, подтип `SignedData`. Сертификат ***signcert*** записывается в структуру в качестве сертификата подписавшего, и должен соответствовать ключу ***pkey***. Сертификаты ***certs*** (например, сертификаты промежуточных УЦ), если имеются, записываются в структуру в качестве дополнительных. Функция возвращает указатель на созданную структуру в случае успеха, и `NULL` в случае неудачи. Код ошибки можно получить вызовом **ERR_get_error**.

Флаги:

`PKCS7_TEXT` — данные будут подписаны вместе с MIME-заголовком для типа `text/plain`.

`PKCS7_NOCERTS` — сертификат отправителя (равно как и дополнительные) не будет включен в сообщение.

`PKCS7_DETACHED` — подписываемые данные не будут включены в сообщение (будет создана отделенная подпись).

`PKCS7_BINARY` — подписываемые данные перед подписью не будут подготовлены в соответствии со стандартом MIME.

`PKCS7_NOATTR` — не формировать authenticatedAttributes.

`PKCS7_NOSMIMECAP` — не формировать атрибут SMIMECapabilities.

`PKCS7_STREAM` — только сформировать структуру `PKCS7`, готовую для подписи, но не подписывать данные. Операция подписи будет произведена последующей записью в объект `BIO`, получаемый вызовом **PKCS7_dataInit**. Вкупе с особенностями `BIO` это позволяет создавать сообщение S/MIME в один проход. Запись должна быть затем корректно завершена. Из функций API корректно завершать ее умеет только **SMIME_write_PKCS7**, вызванная с тем же флагом. В текущей версии с этим флагом обязательно указывать `PKCS7_DETACHED`.

5.5 Добавление второй подписи: PKCS7_add_sign

```
int PKCS7_add_sign(PKCS7 *p7, X509 *signcert, EVP_PKEY *pkey,
       STACK_OF(X509) *certs, BIO *indata, int flags);
```

Функция добавляет подпись к уже существующей структуре `PKCS7`. Остальные параметры имеют то же значение, что для **PKCS7_sign**. Функция возвращает 1 в случае успеха, 0 в случае неудачи. Код ошибки можно получить вызовом **ERR_get_error**. Данная функциональность отсутствует в оригинальном OpenSSL, и добавлена в МагПро КриптоПакет. Функция поддерживает не все режимы работы **PKCS7_sign**.

5.6 Проверка подписей: **PKCS7_verify**

```
int PKCS7_verify(PKCS7 *p7, STACK_OF(X509) *certs,
      X509_STORE *store, BIO *indata, BIO *out, int flags);
```

Функция проверяет подписи под сообщением. ***p7*** указывает на служебную структуру сообщения, ***indata*** — на подписанные данные, независимо от того, содержались они в структуре или это была отделенная подпись (если данные содержатся в структуре, этот параметр может быть `NULL`). Параметр ***certs*** указывает набор дополнительных сертификатов, среди которых следует искать сертификат подписавшего, параметр ***store*** — на хранилище доверенных сертификатов для проверки цепочек доверия. В объект вывода ***out*** выводятся подписанные данные в случае успеха проверки. Проверка считается успешной, если:

- ***p7*** указывает на структуру `PKCS7` подтипа `SignedData`;
- в этой структуре содержится хотя бы одна подпись;
- сертификаты всех, подписавших сообщение, обнаружены либо в наборе ***certs***, либо в сообщении (набор ***certs*** имеет преимущество);
- все эти сертификаты успешно проверены по цепочкам доверия, восходящим к ***store*** и пригодны для подписи сообщений S/MIME (обладают назначением **smimesign**); о работе с хранилищем сертификатов см. раздел 13;
- все подписи успешно проверены.

Функция возвращает 1 в случае успеха, и 0 в случае неудачи. Код ошибки можно получить вызовом **ERR_get_error**.

Флаги:

`PKCS7_NOINTERN` — запретить поиск сертификатов в сообщении. Этот параметр позволяет требовать подписей с использованием только ограниченного набора сертификатов.

`PKCS7_TEXT` — если этот флаг указан, то из данных после проверки подписи удаляется MIME-заголовок для типа `text/plain` (предположительно добавленный туда вызовом **PKCS7_encrypt**). Отсутствие этого заголовка или наличие иного считается ошибкой.

`PKCS7_NOVERIFY` — не производить проверку цепочки доверия сертификатов.

`PKCS7_NOCHAIN` — требовать наличия всех сертификатов промежуточных УЦ в хранилище ***store***, игнорируя содержащиеся в сообщении.

`PKCS7_NOSIGS` — не проверять подписи под сообщением.

Изменять стандартное поведение функции следует с осторожностью. Так, комбинация флагов `PKCS7_NOVERIFY|PKCS7_NOSIGS` полностью отключает все проверки. Она может быть использована, например, если требуется увидеть содержимое сообщения независимо от успеха проверки.

```
STACK_OF(X509) *PKCS7_get0_signers(PKCS7 *p7, STACK_OF(X509) *certs,
      int flags);
```

Функция возвращает набор сертификатов абонентов, подписавших сообщение, не производя никаких проверок. Параметры ***certs*** и ***flags*** имеют то же значение, что для **PKCS7_verify** (из флагов учитывается только `PKCS7_NOINTERN`).

5.7 Запись сообщения S/MIME: **SMIME_write_PKCS7**

```
int SMIME_write_PKCS7(BIO *out, PKCS7 *p7, BIO *data, int flags);
```

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

Функция формирует сообщение S/MIME из структуры ***p7*** и (в случае отделенной подписи) данных ***data*** и записывает его в объект вывода ***out***. Функция возвращает 1 в случае успеха и 0 в случае неудачи. Код ошибки можно получить вызовом **ERR_get_error**.

Флаги:

`PKCS7_DETACHED` — сигнализирует о том, что подпись отделенная. Этот же флаг должен был быть указан при вызове **PKCS7_sign**, сформировавшем структуру ***p7***.

`PKCS7_TEXT` — добавить заголовки для типа `text/plain` к подписываемому содержимому. Флаг имеет смысл только при указании `PKCS7_DETACHED`.

`PKCS7_STREAM` — произвести отложенную подпись. Этот же флаг должне быть указан при вызове **PKCS7_sign**, сформировавшем структуру ***p7***.

5.8 Чтение сообщения S/MIME: **SMIME_read_PKCS7**

```
PKCS7 *SMIME_read_PKCS7(BIO *in, BIO **bcont);
```

Функция разбирает сообщение S/MIME из объекта ввода ***in***. Если сообщение — подписанное с отделенной подписью, то содержимое будет записано в память и по завершении функции доступно для чтения из ****bcont*** (если `bcont!=NULL`, в противном случае ****bcont*** будет содержать `NULL`. Функция возвращает указатель на сформированную структуру `PKCS7` в случае успеха, и `NULL` в случае неудачи. Код ошибки можно получить вызовом **ERR_get_error**.

Если по возвращении из функции ****bcont*** не `NULL`, этот указатель следует передать в качестве параметра ***indata*** в функцию **PKCS7_verify** для проверки подписи (из этого автоматически следует, что тип полученной структуры `PKCS7` — `SignedData`). В противном случае тип полученной структуры можно выяснить вызовом **PKCS7_type**.

Для корректной поддержки будущей функциональности, если параметр ***bcont*** не равен `NULL`, ****bcont*** следует инициализировать `NULL` прежде чем передавать в функцию **SMIME_read_PKCS7**.

Функция предполагает, что структура `PKCS7` в сообщении закодирована в base64, а не binary или quoted-printable. Функция также может не справиться со сложной структурой сообщения S/MIME.

5.9 Другие способы чтения и записи структур `PKCS7`

Функции **SMIME_read_PKCS7** и **SMIME_write_PKCS7** соответствуют аргументу `SMIME` параметров `-inform` и `-outform` команды `openssl smime`.

Четыре нижеследующих функции соответствуют аргументу `PEM` этих параметров и работают с форматом PEM — DER-представлением структуры ASN.1, закодированным в Base64 и окруженным соответствующими строками заголовка.

```
PKCS7 *PEM_read_bio_PKCS7(BIO *bp, PKCS7 **x, pem_password_cb *cb,
       void *u);

PKCS7 *PEM_read_PKCS7(FILE *fp, PKCS7 **x, pem_password_cb *cb,
       void *u);
```

Функции читают PEM-представление структуры `PKCS7`. Значение параметров и возвращаемого значения см. в разделе 18.3.

```
int PEM_write_bio_PKCS7(BIO *bp, PKCS7 *x);
```

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

```
int PEM_write_PKCS7(FILE *fp, PKCS7 *x);
```

Функции записывают PEM-представление структуры PKCS7. Значение параметров и возвращаемого значения см. в разделе 18.3.

Следующие четыре функции работают с DER-представлением и соответствуют аргументу DER параметров -inform и -outform команды openssl smime.

```
PKCS7 *d2i_PKCS7_bio(BIO *bp, PKCS7 **p7);

PKCS7 *d2i_PKCS7_fp(FILE *fp, PKCS7 **p7);
```

Функции читают DER-представление структуры PKCS7 из объекта BIO или FILE соответственно и возвращают указатель на заполненную структуру. Аргумент ***p7*** имеет тот же смысл, что и аргумент ***x*** функций **PEM_read_bio_PKCS7** и **PEM_read_PKCS7**. В случае ошибки функции возвращают NULL.

```
int i2d_PKCS7_bio (BIO *bp, PKCS7 *p7);

int i2d_PKCS7_fp (FILE *fp, PKCS7 *p7);
```

Функции записывают DER-представление структуры ***p7*** в ***bp*** или ***fp*** соответственно. Они возвращают 1 в случае успеха и 0 в случае неудачи.

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

6 Работа с CMS

В версию OpenSSL 1.0 включен набор функций для работы с сообщениями CMS (Cryptographic Message Syntax), предоставляющий гораздо больше возможностей, чем API для PKCS#7 сообщений, описанный в предыдущем разделе. Во-первых, этот API поддерживает ряд новых возможностей формата, описанных в RFC 3852, во-вторых, этот API поддерживает потоковую работу, что позволяет подписывать и шифровать файлы за один проход и работать с файлами, размер которых превышает доступную память.

Этот API включает в себя следующие функции:

```
int CMS_add0_cert(CMS_ContentInfo *cms, X509 *cert);

int CMS_add1_cert(CMS_ContentInfo *cms, X509 *cert);

STACK_OF(X509) *CMS_get1_certs(CMS_ContentInfo *cms);

int CMS_add0_crl(CMS_ContentInfo *cms, X509_CRL *crl);

int CMS_add1_crl(CMS_ContentInfo *cms, X509_CRL *crl);

STACK_OF(X509_CRL) *CMS_get1_crls(CMS_ContentInfo *cms);
```

Функции **CMS_add0_cert()** и **CMS_add1_cert()** добавляют сертификат cert к cms. Должны быть типа signed data или enveloped data.

Функция **CMS_get1_certs()** возвращает все сертификаты в cms.

Функции **CMS_add0_crl()** и **CMS_add1_crl()** добавляют CRL crl к cms. Функция **CMS_get1_crls()** возвращает все CRL в cms.

Структура `CMS_ContentInfo` должна быть типа signed data или enveloped data, в противном случае будет возвращена ошибка.

Для структур типа signed data сертификаты и CRL добавляются в поля структуры certificates и crls. Для структур типа enveloped data они добавляются в OriginatorInfo.

Как указывает 0 в имени функции **CMS_add0_cert()**, она добавляет сертификат cert внутрь структуры cms, не создавая его копии, и сертификат не должен освобождаться после вызова функции, в противовес функции **CMS_add1_cert()**, которая создает копию сертификата, и он освобождается после ее вызова.

Один и тот же сертификат или CRL не должен добавляться к одной и той же cms-структуре более одного раза.

CMS_add0_cert(), **CMS_add1_cert()**, **CMS_add0_crl()** и **CMS_add1_crl()** возвращают 1 при успешном завершении и 0 при неудачном завершении.

CMS_get1_certs() и **CMS_get1_crls()** возвращают STACK сертификатов или CRL или NULL, если сертификатов или CRL в структуре нет, или если происходит ошибка. Единственная ошибка, возможная на практике — неверный тип структуры cms.

```
CMS_RecipientInfo *CMS_add1_recipient_cert(CMS_ContentInfo *cms,
      X509 *recip, unsigned int flags);

CMS_RecipientInfo *CMS_add0_recipient_key(CMS_ContentInfo *cms, int
      nid,
      unsigned char *key, size_t keylen, unsigned char *id, size_t
      idlen,
      ASN1_GENERA LIZEDTIME *date, ASN1_OBJECT *otherTypeId,
```

```
        ASN1_TYPE *otherType);
```

Функция **CMS_add1_recipient_cert()** добавляет получателя recip в cms-структуру CMS_ContentInfo типа enveloped data в виде структуры KeyTransRecipientInfo.

Функция **CMS_add0_recipient_key()** добавляет симметричный ключ key длины keylen, используя идентификатор алгоритма шифрования nid, идентификатор получателя id длиной idlen и опциональные величины date, otherTypeId и otherType в cms-структуру CMS_ContentInfo типа enveloped data в виде структуры KEKRecipientInfo.

Структуру CMS_ContentInfo следует получать через начальный вызов функции **CMS_encrypt()** с флагом CMS_PARTIAL set.

Главная цель этих функций — предоставить более тонкий контроль над cms-структурой типа enveloped data в тех случаях, где не подходят умолчания более простой функции **CMS_encrypt()**. Например, если необходимо добавить одну или больше структур типа KEKRecipientInfo. Можно также добавлять новые атрибуты с помощью возвращенной структуры CMS_RecipientInfo и функций утилиты атрибутов CMS.

OpenSSL по умолчанию будет идентифицировать сертификаты получателей, используя поля issuer name и serial number. Если установлена CMS_USE_KEYID, она будет использовать вместо этого значение subject key identifier. Если ни один сертификат получателя не имеет расширения subject key identifier, происходит ошибка.

В настоящее время идентификатор nid поддерживается только для алгоритмов шифрования на основе AES, а именно: NID_id_aes128_wrap, NID_id_aes192_wrap и NID_id_aes256_wrap. Если значение nid установлено в NID_undef, будет использоваться алгоритм шифрования AES, соответствующий значению keylen.

Функции **CMS_add1_recipient_cert()** и **CMS_add0_recipient_key()** возвращают внутренний указатель на только что добавленную структуру CMS_RecipientInfo или NULL, если происходит ошибка.

```
CMS_ContentInfo *CMS_compress(BIO *in, int comp_nid, unsigned int
        flags);
```

Функция **CMS_compress()** создает и возвращает CMS-структуру типа CompressedData.

comp_nid — алгоритм сжатия, который следует использовать, или NID_undef, если следует использовать умолчательный алгоритм (сжатие zlib). in — данные, которые следует сжать. flags — опциональный набор флагов.

Единственный поддерживаемый в настоящее время алгоритм сжатия — zlib, исплользующий NID NID_zlib_compression.

Если поддержка zlib не была включена при компиляции OpenSSL, то функция **CMS_compress()** вернет ошибку.

Если установлен флаг CMS_TEXT, перед данными добавляются MIME-заголовки для типа text/plain.

Как правило, предоставленные данные переводятся в канонический MIME-формат (как требуется по спецификациям S/MIME). Если установлен флаг CMS_BINARY, никакого перевода не происходит. Эту опцию следует использовать, если данные предоставлены в двоичном формате, и перевод в другой формат их исказит. Если установлен флаг CMS_BINARY, то флаг CMS_TEXT игнорируется.

Если установлен флаг CMS_STREAM, возвращается частичная структура типа CMS_ContentInfo, подходящая для потока ввода-вывода: из BIO in данные не читаются.

Сжатые данные включаются в структуру типа CMS_ContentInfo, если не установлен фла CMS_DETACHED; в этом случае они опускаются. На практике это редко используется и

не поддерживается функцией **SMIME_write_CMS()**.

Если установлен флаг `CMS_STREAM`, возвращенная структура типа `CMS_ContentInfo` не полна, и вывод содержащихся в ней данных через функцию, не производящую корректное финализирование структуры типа `CMS_ContentInfo`, даст непредсказуемые результаты.

Структуру финализируют несколько функций, в том числе **SMIME_write_CMS()**, **i2d_CMS_bio_stream()**, **PEM_write_bio_CMS_stream()**. Финализация также может быть выполнена получением потокового ASN1 BIO при непосредственном использовании функции **BIO_new_CMS()**.

Дополнительные параметры сжатия, такие, как уровень сжатия zlib, в настоящее время не могут быть установлены.

Функция **CMS_compress()** возвращает или структуру типа `CMS_ContentInfo`, или NULL, если произошла ошибка. Выяснить природу ошибки можно с помощью функции **ERR_get_error()**.

```
int CMS_decrypt(CMS_ContentInfo *cms, EVP_PKEY *pkey, X509 *cert,
      BIO *dcont, BIO *out, unsigned int flags);
```

Функция **CMS_decrypt()** извлекает и расшифровывает данные, содержащиеся в CMS-структуре типа EnvelopedData. pkey — закрытый ключ получателя, cert — сертификат получателя, out — `BIO`, в которое следует записывать данные, flags — опциональный набор флагов.

Параметр dcont используется в том редком случае, когда зашифрованные данные являются внешними по отношению к структуре. Как правило, он устанавливается в NULL.

Перед использованием этой функции следует вызвать функцию **OpenSSL_add_all_algorithms()** или эквивалентную ей, иначе случатся ошибки, связанные с неизвестными алгоритмами.

Хотя сертификат получателя не является необходимым при расшифровании данных, он нужен, чтобы определить подходящих (из нескольких возможных) получателей в CMS-структуре. Если значение cert установлено в NULL, то пробуются все возможные получатели.

Возможно определить корректные ключи получателей другими способами (например, найти в базе данных) и установить их в CMS-структуре заранее, используя функции утилиты CMS, например **CMS_set1_pkey()**. В этом случае значения `cert` и `pkey` следует установить в NULL.

Чтобы обработать типы `KEKRecipientInfo`, следует перед вызовом **CMS_decrypt()** вызвать функции **CMS_set1_key()** или **CMS_RecipientInfo_set0_key()** и **CMS_ReceipientInfo_decrypt()** и установить значения cert и pkey в NULL.

В параметр flags могут быть переданы следующие флаги:

Если флаг `CMS_TEXT` установлен, MIME-заголовки для типа text/plain удаляются из данных. Если данные не имеют тип text/plain, возвращается ошибка.

Функция **CMS_decrypt()** возвращает 1 в случае успешного завершения и 0 в случае ошибки. Природу ошибки можно узнать с использованием функции **ERR_get_error()**.

```
CMS_ContentInfo *CMS_encrypt(STACK_OF(X509) *certs, BIO *in,
      const EVP_CIPHER *cipher, unsigned int flags);
```

Функция **CMS_encrypt()** создает и возвращает CMS-структуру типа `Enveloped Data`. certs — список сертификатов получателей. in — данные, которые необходимо зашифровать. cipher — симметричный шифр, который необходимо использовать. flags — опциональный набор флагов.

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

Поддерживаются только сертификаты, содержащие ключи RSA и ГОСТ, поэтому сертификаты реципиентов, переданные в эту функцию, все должны содержать открытые ключи RSA или ГОСТ, хотя и необязательно должны быть подписаны с использованием этих алгоритмов.

Алгоритм, переданный в параметре cipher, должен поддерживать ASN1-кодировку его параметров.

Многие браузеры включают опцию «подписать и зашифровать», которая представляет собой просто S/MIME-структуру типа `EnvelopedData`, которая содержит подписанное S/MIME-сообщение. Этого можно легко добиться, сохраняя подписанное S/MIME-сообщение в памяти BIO и передавая его в функцию **CMS_encrypt()**.

В параметр flags можно передать следующие флаги:

Если установлен флаг `CMS_TEXT`, к началу данных будут добавлены MIME-заголовки для типа text/plain.

Как правило, предоставленные данные переводится в канонический MIME-формат (как требуется по спецификациям S/MIME). Если установлен флаг `CMS_BINARY`, никакого перевода не происходит. Эту опцию следует использовать, если данные предоставлены в двоичном формате, и перевод в другой формат их исказит. Если установлен флаг `CMS_BINARY`, то флаг `CMS_TEXT` игнорируется.

OpenSSL по умолчанию будет идентифицировать сертификаты получателей, используя поля issuer name и serial number. Если установлена `CMS_USE_KEYID`, она будет использовать вместо этого значение subject key identifier. Если ни один сертификат получателя не имеет расширения subject key identifier, происходит ошибка.

Если установлен флаг `CMS_STREAM`, возвращается частичная структура типа `CMS_ContentInfo`, подходящая для потока ввода-вывода: из BIO in данные не читаются.

Если установлен флаг `CMS_PARTIAL`, возвращается частичная структура типа `CMS_ContentInfo`, к которой перед финализацией могут быть добавлены дополнительные получатели и атрибуты.

Сжатые данные включаются в структуру типа `CMS_ContentInfo`, если не установлен фла `CMS_DETACHED`; в этом случае они опускаются. На практике это редко используется и не поддерживается функцией **SMIME_write_CMS()**.

Если установлен флаг `CMS_STREAM`, возвращенная структура типа `CMS_ContentInfo` не полна, и вывод содержащихся в ней данных через функцию, не производящую корректное финализирование структуры типа `CMS_ContentInfo`, даст непредсказуемые результаты.

Структуру финализируют несколько функций, в том числе **SMIME_write_CMS()**, **i2d_CMS_bio_stream()**, **PEM_write_bio_CMS_stream()**. Финализация также может быть выполнена получением потокового ASN1 BIO при непосредственном использовании функции **BIO_new_CMS()**.

Получатели, указанные в параметре certs, используют информационную CMS-структуру типа `KeyTransRecipientInfo`. Этот тип также поддерживается при использовании флага `CMS_PARTIAL` и функции **CMS_add0_recipient_key()**.

Параметр certs может иметь значение NULL, если установлен флаг `CMS_PARTIAL`, и получатели добавляются позже с помощью функций **CMS_add1_recipient_cert()** или **CMS_add0_recipient_key()**.

Функция **CMS_encrypt()** возвращает либо структуру типа `CMS_ContentInfo`, или NULL, если произошла ошибка. Ошибку можно получить с помощью функции **ERR_get_error()**.

```
int CMS_final(CMS_ContentInfo *cms, BIO *data, BIO *dcont,
      unsigned int flags);
```

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

Функция **CMS_final()** финализирует CMS-структуру. Ее цель — выполнить все операции, необходимые для CMS (например, вычислить дайджест) и установить соответствующие поля. Параметр data содержит обрабатываемые данные. Параметр dcont содержит BIO, в которое следует записать данные после обработки; он используется только с данными, внешними по отношению к структуре, и обычно устанавливается в NULL.

Эта функция, как правило, вызывается, когда используется флаг CMS_PARTIAL. Ее следует использовать только тогда, когда потоковый ввод-вывод не выполняется, потому что функции потокового ввода-вывода выполняют операции финализации сами.

Функция **CMS_final()** возвращает 1 в случае успешного завершения или 0 в случае неудачи.

```
STACK_OF(CMS_RecipientInfo) *CMS_get0_RecipientInfos(CMS_ContentInfo
      *cms);

int CMS_RecipientInfo_type(CMS_RecipientInfo *ri);

int CMS_RecipientInfo_ktri_get0_signer_id(CMS_RecipientInfo *ri,
      ASN1_OC
      TET_STRING **keyid, X509_NAME **issuer, ASN1_INTEGER **sno);

int CMS_RecipientInfo_ktri_cert_cmp(CMS_RecipientInfo *ri, X509
      *cert);

int CMS_RecipientInfo_set0_pkey(CMS_RecipientInfo *ri, EVP_PKEY
      *pkey);

int CMS_RecipientInfo_kekri_get0_id(CMS_RecipientInfo *ri,
      X509_ALGOR *palg, ASN1_OCTET_STRING **pid, ASN1_GENERALIZEDTIME
      **pdate,
      ASN1_OBJECT **potherid, ASN1_TYPE **pothertype);

int CMS_RecipientInfo_kekri_id_cmp(CMS_RecipientInfo *ri,
      const unsigned char *id, size_t idlen);

int CMS_RecipientInfo_set0_key(CMS_RecipientInfo *ri,
      unsigned char *key, size_t keylen);

int CMS_RecipientInfo_decrypt(CMS_ContentInfo *cms, CMS_RecipientInfo
      *ri);
```

Функция **CMS_get0_RecipientInfos()** возвращает все структуры типа CMS_RecipientInfo, связанные с CMS-структурой типа EnvelopedData.

Функция **CMS_RecipientInfo_type()** возвращает CMS-тип структуры ri типа CMS_RecipientInfo. В настоящее время она возвращает CMS-типы CMS_RECIPINFO_TRANS, CMS_RECIPINFO_AGREE, CMS_RECIPINFO_KEK, CMS_RECIPINFO_PASS или CMS_RECIPINFO_OTHER.

Функция **CMS_RecipientInfo_ktri_get0_signer_id()** получает идентификатор получателя сертификата, связанный с конкретной структурой ri типа CMS_RecipientInfo, которая должна быть CMS-типа CMS_RECIPINFO_TRANS. Значение keyidentifier может быть записано в параметр keyid или же в параметр issuer записывается DN удостоверяющего центра, выпустившего сертификат, а в параметр sno записывается серийный номер этого сертификата.

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

Функция **CMS_RecipientInfo_ktri_cert_cmp()** сравнивает сертификат cert со структурой ri типа `CMS_RecipientInfo`, которая должна быть CMS-типа `CMS_RECIPINFO_TRANS`. Она возвращает ноль, если сравнение успешно, и не ноль, если сравнение не успешно.

Функция **CMS_RecipientInfo_set0_pkey()** связывает закрытый ключ pkey со структурой ri типа `CMS_RecipientInfo`, которая должна быть CMS-типа `CMS_RECIPINFO_TRANS`.

Функция **CMS_RecipientInfo_kekri_get0_id()** извлекает ключевую информацию из структуры ri типа `CMS_RecipientInfo`, которая должна быть CMS-типа `CMS_RECIPINFO_KEK`. Любой из оставшихся параметров может быть установлен в NULL, если приложение не заинтересовано в значении соответствующего поля. Там, где поле опционально и отсутствует, в соответствующий параметр будет записано значение NULL. Значение поля keyEncryptionAlgorithm записывается в параметр palg, значение поля keyIdentifier записывается в параметр pid, значение поля date, если таковое присутствует, записывается в параметр pdate, если присутствует поле other, компоненты keyAttrId и keyAttr записываются в параметры potherid и pothertype.

Функция **CMS_RecipientInfo_kekri_id_cmp()** сравнивает ID в параметрах id и idlen со структурой ri типа `CMS_RecipientInfo`, которая должна быть CMS-типа `CMS_RECIPINFO_KEK`. Она возвращает ноль, если сравнение успешно, и не ноль, если сравнение не успешно.

Функция **CMS_RecipientInfo_set0_key()** связывает симметричный ключ key длины keylen со структурой ri типа `CMS_RecipientInfo`, которая должна быть CMS-типа `CMS_RECIPINFO_KEK`.

Функция **CMS_RecipientInfo_decrypt()** пытается расшифровать структуру ri типа `CMS_RecipientInfo` в CMS-структуру. Предварительно со структурой необходимо связать ключ.

Главная цель этих функций — предоставить приложению возможность просмотра ключей получателей с использованием любого подходящего метода, когда более простой способ использования функции **CMS_decrypt()** не подходит.

В типичном случае приложение получит все структуры типа `CMS_RecipientInfo` с помощью функции **CMS_get0_RecipientInfos()** и проверит CMS-тип каждой структуры с помощью функции **CMS_RecpientInfo_type()**. В зависимости от CMS-типа структура типа `CMS_RecipientInfo` может быть проигнорирована, или же данные, идентифицирующие ключ, будут получены из нее с помощью соответствующей функции. Затем, если можно каким-то легальным способом получить соответствующий закрытый ключ, его можно потом связать со структурой и вызвать функцию **CMS_RecpientInfo_decrypt()**. При удачном завершении можно вызвать функцию **CMS_decrypt()** с ключом NULL, чтобы расшифровать внутренние данные.

Функция **CMS_get0_RecipientInfos()** возвращает все структуры типа `CMS_RecipientInfo`, или же NULL в случае ошибки.

Функции **CMS_RecipientInfo_ktri_get0_signer_id()**, **CMS_RecipientInfo_set0_pkey()**, **CMS_RecipientInfo_kekri_get0_id()**, **CMS_RecipientInfo_set0_key()** и **CMS_RecipientInfo_decrypt()** возвращают 1 в случае удачного завершения или 0 в случае неудачи.

Функции **CMS_RecipientInfo_ktri_cert_cmp()** и **CMS_RecipientInfo_kekri_cmp()** возвращают 0 в случае удачного сравнения и ненулевую величину в случае неудачи.

Любую ошибку можно получить с помощью функции **ERR_get_error()**.

```
STACK_OF(CMS_SignerInfo) *CMS_get0_SignerInfos(CMS_ContentInfo *cms);

int CMS_SignerInfo_get0_signer_id(CMS_SignerInfo *si,
```

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

```
        ASN1_OCTET_STRING
        **keyid, X509_NAME **issuer, ASN1_INTEGER **sno);

int CMS_SignerInfo_cert_cmp(CMS_SignerInfo *si, X509 *cert);

void CMS_SignerInfo_set1_signer_cert(CMS_SignerInfo *si, X509
        *signer);
```

Функция **CMS_get0_SignerInfos()** возвращает все структуры типа `CMS_SignerInfo`, связанные с CMS-структурой типа `SignedData`.

Функция **CMS_SignerInfo_get0_signer_id()** извлекает идентификатор сертификата, содержащийся в структуре si, с конкретной структурой si типа `CMS_SignerInfo`. Значение keyidentifier может быть записано в параметр keyid или же в параметр issuer записывается DN удостоверяющего центра, выпустившего сертификат, а в параметр sno записывается серийный номер этого сертификата.

Функция **CMS_SignerInfo_cert_cmp()** сравнивает сертификат cert с идентификатором сертификата, содержащимся в структуре si. Она возвращает ноль, если сравнение успешно, и ненулевое значение в противном случае.

Функция **CMS_SignerInfo_set1_signer_cert()** устанавливает в поле сертификата подписавшего в структуре si копию сертификата, переданного ей в параметре signer.

Главная цель этих функций — предоставить приложению возможность просматривать сертификаты подписавших с использованием любого подходящего способа, когда более простой способ с помощью функции **CMS_verify()** не подходит.

В типичном случае приложение получит все структуры типа `CMS_SignerInfo` с помощью функции **CMS_get0_SignerInfo()** и получит информацию об идентификаторах с помощью CMS. Затем оно каким-то способом получит сертификат подписавшего (или вернет ошибку, если не сможет его найти) и установит его с помощью функции **CMS_SignerInfo_set1_signer_cert()**.

Когда все сертификаты подписавших будут установлены, можно вызывать функцию **CMS_verify()**.

Хотя функция **CMS_get0_SignerInfos()** может возвращать NULL при возникновении ошибки или если нет ни одного сертификата подписавшего, на практике это не составляет проблемы, потому что единственная возможная ошибка — если CMS-структура не является структурой типа `signedData` из-за ошибки приложения.

Функция **CMS_get0_SignerInfos()** возвращает все структуры типа `CMS_SignerInfo`, или же NULL, если нет сертификатов подписавших или если происходит ошибка.

Функция **CMS_SignerInfo_get0_signer_id()** возвращает 1 в случае успеха и 0 в случае ошибки.

Функция **CMS_SignerInfo_cert_cmp()** возвращает 0 в случае успешного сравнения и ненулевое значение в противном случае.

Функция **CMS_SignerInfo_set1_signer_cert()** ничего не возвращает.

Любая ошибка может быть получена с помощью функции **ERR_get_error()**.

```
const ASN1_OBJECT *CMS_get0_type(CMS_ContentInfo *cms);

int CMS_set1_eContentType(CMS_ContentInfo *cms, const ASN1_OBJECT
        *oid);

const ASN1_OBJECT *CMS_get0_eContentType(CMS_ContentInfo *cms);
```

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

Функция **CMS_get0_type()** возвращает content-type структуры типа CMS_ContentInfo в виде указателя типа ASN1_OBJECT. Приложение может решать само, как обрабатывать структуру типа CMS_ContentInfo на основе этого значения.

Функция **CMS_set1_eContentType()** устанавливает встроенный content-type структуры типа CMS_ContentInfo. Ее следует вызывать вместе с CMS-функциями с флагом CMS_PARTIAL и до того, как структура будет финализирована, иначе результаты будут неопределенными.

Функция **ASN1_OBJECT *CMS_get0_eContentType()** возвращает указатель на встроенный content-type.

Как указывает 0 в названии функций **CMS_get0_type()** и **CMS_get0_eContentType()**, они возвращают внутренние указатели, которые не следует освобождать. Функция **CMS_set1_eContentType()** копирует предоставленный OID, который следует освобождать после использования.

Возвращенные величины типа ASN1_OBJECT могут быть конвертированы в целочисленное значение NID с помощью функции **OBJ_obj2nid()**. Для поддерживаемых в настоящее время content-type возвращаются следующие величины:

```
NID_pkcs7_data
NID_pkcs7_signed
NID_pkcs7_digest
NID_id_smime_ct_compressedData:
NID_pkcs7_encrypted
NID_pkcs7_enveloped
```

Функции **CMS_get0_type()** и **CMS_get0_eContentType()** возвращают структуру типа ASN1_OBJECT.

Функция **CMS_set1_eContentType()** возвращает 1 при успешном завершении или 0, если произошла ошибка. Ошибку можно определить с помощью функции **ERR_get_error()**.

```
CMS_ReceiptRequest *CMS_ReceiptRequest_create0(unsigned char *id,
       int idlen, int allorfirst, STACK_OF(GENERAL_NAMES)
       *receiptList,
       STACK_OF(GENERAL_NAME S) *receiptsTo);

int CMS_add1_ReceiptRequest(CMS_SignerInfo *si, CMS_ReceiptRequest
       *rr);

int CMS_get1_ReceiptRequest(CMS_SignerInfo *si, CMS_ReceiptRequest
       **prr);

void CMS_ReceiptRequest_get0_values(CMS_ReceiptRequest *rr,
       ASN1_STRING
       **pcid, int *pallorfirst, STACK_OF(GENERAL_NAMES) **plist,
       STACK_OF(GENERAL_NAMES) **prto);
```

Функция **CMS_ReceiptRequest_create0()** создает структуру подписанного запроса квитанции. Поле signedContentIdentifier устанавливается с помощью параметров id и idlen или же устанавливается в 32 байта псевдослучайных данных, если значение id равно NULL. Если значение параметра receiptlist установлено в NULL, используется опция allOrFirstTier в receiptsFrom и устанавливается в значение параметра allorfirst. Если значение параметра

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

receiptList не равно нулю, используется опция receiptList в receiptsFrom. Параметр receiptsTo указывает значение поля receiptsTo.

Функция **CMS_add1_ReceiptRequest()** добавляет подписанный запрос на квитанцию rr к структуре si типа `SignerInfo`.

Функция **CMS_get1_ReceiptRequest()** ищет подписанный запрос на квитанцию в si, если находит, то декодирует и записывает в prr.

Функция **CMS_ReceiptRequest_get0_values()** извлекает значения запроса на квитанцию. signedContentIdentifier копируется в pcid. Если используется опция allOrFirstTier из receiptsFrom, ее значение копируется в pallorfirst, в противном случае поле receiptList копируется в plist. Параметр receiptsTo копируется в prto.

Дополнительные подробности о значениях полей указаны в RFC2634.

Содержание подписанного запроса должно считаться имеющим значение только в том случае, если соответствующая структура типа `CMS_ContentInfo` может быть успешно проверена с помощью функции **CMS_verify()**.

Функция **CMS_ReceiptRequest_create0()** возвращает структуру подписанного запроса на квитанцию, или же NULL, если произошла ошибка.

Функция **CMS_add1_ReceiptRequest()** возвращает 1 в случае успешного завершения или 0 в случае ошибки.

Функция **CMS_get1_ReceiptRequest()** возвращает 1, если подписанный запрос на квитанцию обнаружен и декодирован. Она возвращает 0, если подписанный запрос на квитанцию не присутствует, и -1, если он присутствует, но искажен.

```
CMS_SignerInfo *CMS_sign_add1_signer(CMS_ContentInfo *cms,
      X509 *signcert, EVP_PKEY *pkey, const EVP_MD *md, unsigned int
      flags);

int CMS_SignerInfo_sign(CMS_SignerInfo *si);
```

Функция **CMS_sign_add1_signer()** добавляет сертификат подписавшего вместе с сертификатом signcert и закрытым ключом pkey, пользуясь дайджестом сообщения md к CMS-структуре типа `CMS_ContentInfo` типа `SignedData`.

Структуру типа `CMS_ContentInfo` следует получить из начального вызова функции **CMS_sign()** с установленным флагом `CMS_PARTIAL` или в случае переподписывания валидной структуры типа `CMS_ContentInfo` типа `SignedData`.

Если значение параметра md установлено в NULL, будет применен умолчательный дайджест для алгоритма открытого ключа.

Если флаг `CMS_REUSE_DIGEST` не установлен, возвращаемая структура типа `CMS_ContentInfo` не полна и должна быть финализирована или поточно (если возможно) или вызовом функции **CMS_final()**.

CMS_SignerInfo_sign() явным образом подписывает структуру `CMS_SignerInfo`, главным образом она используется, когда установлены оба флага `CMS_REUSE_DIGEST` и `CMS_PARTIAL`.

Основное предназначение функции **CMS_sign_add1_signer()** — предоставить более тонкий контроль над подписанной CMS-структурой данных там, где не подходят умолчания более простой функции **CMS_sign()**. Например, если имеется несколько сертификатов подписавших или нужны неумолчательные алгоритмы дайджеста. Можно также добавлять новые атрибуты с помощью возвращенной структуры типа `CMS_SignerInfo` и функций утилиты атрибутов CMS или функций подписанного CMS-запроса на квитанцию.

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

Любые из нижеследующих флагов (вместе и по отдельности) могут быть переданы в параметр flags.

Если установлен флаг `CMS_REUSE_DIGEST`, делается попытка скопировать значение дайджеста данных из структуры типа `CMS_ContentInfo`: добавить сертификат подписавшего к существующей структуре. Если невозможно найти соответствующее значение дайджеста, происходит ошибка. Если установить этот флаг, возвращенная структура типа `CMS_ContentInfo` будет корректной и финализированной.

Если вместе с флагом `CMS_REUSE_DIGEST` установлен флаг `CMS_PARTIAL`, то структура типа `CMS_SignerInfo` не будет финализированной, так что можно добавлять необходимые атрибуты. В этом случае необходим явный вызов функции **CMS_SignerInfo_sign()**, чтобы финализировать структуру.

Если установлен флаг `CMS_NOCERTS`, то сертификат подписавшего не будет включен в структуру типа `CMS_ContentInfo`, но сертификат подписавшего все равно необходимо передать в параметр signcert. Это может уменьшить размер структуры, если сертификат подписавшего может быть получен другими способами, например из ранее подписанного сообщения.

Структура типа `SignedData` включает несколько атрибутов `signedAttributes`, включая время подписи, content-type и лист поддерживаемых шифров в атрибуте `SMIMECapabilities`. Если установлен флаг `CMS_NOATTR`, то никакие signedAttributes использованы не будут. Если установлен флаг `CMS_NOSMIMECAP`, опускаются только `SMIMECapabilities`

OpenSSL по умолчанию будет определять сертификаты подписавших с помощью DN удостоверяющего центра и серийному номеру. Если установлен флаг `CMS_USE_KEYID`, вместо них будет использовано значение subject key identifier. Если сертификат подписавшего не имеет расширения subject key identifier, происходит ошибка.

Если присутствует атрибуют `SMIMECapabilities`, он указывает поддержку следующих алгоритмов в порядке предпочтения: 256-битный AES, ГОСТ Р 34.11-94, ГОСТ 28147-89, 192-битный AES, 128-битный AES, triple DES, 128-битный RC2, 64-битный RC2, DES и 40-битный RC2. Если какой-либо из этих алгоритмов недоступен, он не будет включен: например, алгоритмы ГОСТ не будут включены, если не загружен ГОСТ ENGINE.

Функция **CMS_sign1_add_signers()** возвращает внутренний указатель на только что доавбленную структуру, или же NULL в случае ошибки. Это может быть использовано для установки дополнительных атрибутов перед ее финализацией.

```
CMS_ContentInfo *CMS_sign(X509 *signcert, EVP_PKEY *pkey,
      STACK_OF(X509) *certs, BIO *data, unsigned int flags);
```

Функция **CMS_sign()** создает и возвращает CMS-структуру типа `SignedData`. signcert — сертификат, на котором следует подписывать структуру, pkey — соответствующий закрытый ключ. certs — опциональный дополнительный набор сертификатов для включения в CMS-структуру (например, все сертификаты промежуточных УЦ в цепочке). Любой из этих параметров может быть установлен в NULL, подробнее см. ниже.

Данные, которые необходимо подписать, берутся из BIO.

flags — опциональный набор флагов.

Любой из нижеперечисленных флагов (вместе или по отдельности) может быть передан в параметр flags.

Многие клиенты S/MIME ожидают, что подписанные данные будут содержать корректные MIME-заголовки. Если установлен флаг `CMS_TEXT`, к данным вначале добавляются MIME-заголовки типа text/plain.

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

Если установлен флаг CMS_NOCERTS, то сертификат подписавшего не будет включен в структуру типа CMS_ContentInfo, хотя сертификат подписавшего все равно необходимо передавать в параметре signcert. Это может уменьшить размер структуры, если сертификат подписавшего может быть получен другими способами, например из ранее подписанного сообщения.

Подписываемые данные включаются в структуру типа CMS_ContentInfo, если не указан флаг CMS_DETACHED. Если такой флаг установлен, подписанные данные опускаются. Это используется для получения отдельных подписей типа CMS_ContentInfo, которые используются, например, в текстовых подписанных S/MIME-сообщениях.

Как правило, предоставленные данные переводится в канонический MIME-формат (как требуется по спецификациям S/MIME). Если установлен флаг CMS_BINARY, никакого перевода не происходит. Эту опцию следует использовать, если данные предоставлены в двоичном формате, и перевод в другой формат их исказит. Если установлен флаг CMS_BINARY, то флаг CMS_TEXT игнорируется.

Структура типа SignedData включает несколько атрибутов signedAttributes, включая время подписи, content-type и лист поддерживаемых шифров в атрибуте SMIMECapabilities. Если установлен флаг CMS_NOATTR, то никакие signedAttributes использованы не будут. Если установлен флаг CMS_NOSMIMECAP, опускаются только SMIMECapabilities

Если присутствует атрибуют SMIMECapabilities, он указывает поддержку следующих алгоритмов в порядке предпочтения: 256-битный AES, ГОСТ Р 34.11-94, ГОСТ 28147-89, 192-битный AES, 128-битный AES, triple DES, 128-битный RC2, 64-битный RC2, DES и 40-битный RC2. Если какой-либо из этих алгоритмов недоступен, он не будет включен: например, алгоритмы ГОСТ не будут включены, если не загружен ГОСТ ENGINE.

OpenSSL по умолчанию будет определять сертификаты подписавших с помощью DN удостоверяющего центра и серийному номеру. Если установлен флаг CMS_USE_KEYID, вместо них будет использовано значение subject key identifier. Если сертификат подписавшего не имеет расширения subject key identifier, происходит ошибка.

Если установлен флаг CMS_STREAM, то возвращенная структура типа CMS_ContentInfo только инициализируется и подготавливается к операции подписи. Но само подписывание не выполняется, и данные, которые нужно подписать, не считываются из параметра data. Подписывание откладывается до тех пор, пока данные не будут записаны. Таким образом данные могут быть подписаны за один проход.

Если установлен флаг CMS_PARTIAL, возвращается частичная структура типа CMS_ContentInfo, к которой перед финализацией могут быть добавлены дополнительные сертификаты подписи и возможности.

Если установлен флаг CMS_STREAM, то возвращенная структура типа CMS_ContentInfo не полна, и вывод содержащихся в ней данных через функцию, не производящую корректное финализирование структуры типа CMS_ContentInfo, даст непредсказуемые результаты.

Структуру финализируют несколько функций, в том числе **SMIME_write_CMS()**, **i2d_CMS_bio_stream()**, **PEM_write_bio_CMS_stream()**. Финализация также может быть выполнена получением потокового ASN1 BIO при непосредственном использовании функции **BIO_new_CMS()**.

Если указан сертификат подписи, он будет использовать умолчательный алгоритм дайджеста для алгоритма подписи, т.е. SHA1 и для ключей RSA, и для ключей DSA.

Если параметры signcert и pkey установлены в NULL, получается CMS-структура, содержащая только сертификаты.

Функция **CMS_sign()** — базовая функция подписи, и ее вывод может подходить для многих целей. Для более тонкого контроля над форматом вывода параметры certs, signcert и pkey могут быть установлены в NULL, и установлен флаг `CMS_PARTIAL`. Тогда можно добавить один или несколько сертификатов подписи с помощью функции **CMS_sign_add1_signer()**, использовать неумолчательные алгоритмы дайджеста и добавлять атрибуты по выбору. Затем необходимо вызвать функцию **CMS_final()**, чтобы финализировать структуру, если не происходит потокового ввода-вывода.

Некоторые атрибуты, такие, как `counter signatures`, не поддерживаются.

Функция **CMS_sign()** возвращает либо корректную структуру типа `CMS_ContentInfo`, либо NULL, если произошла ошибка. Ошибку можно получить с помощью функции **ERR_get_error()**.

```
CMS_ContentInfo *CMS_sign_receipt(CMS_SignerInfo *si, X509 *signcert,
      EVP_PKEY *pkey, STACK_OF(X509) *certs, unsigned int flags);
```

Функция **CMS_sign_receipt()** создает и возвращает CMS-структуру подписанной квитанции. si — структура типа `CMS_SignerInfo`, содержащая подписанный запрос на квитанцию. signcert — сертификат, на котором следует вырабатывать подпись под квитанцией, pkey — соответствующий закрытый ключ. certs — опциональный дополнительный набор сертификатов для включения в CMS-структуру (например, все сертификаты промежуточных УЦ в цепочке).

flags — опциональный набор флагов.

Эта функция ведет себя подобно функции **CMS_sign()** за исключением того, что значения флагов `CMS_DETACHED`, `CMS_BINARY`, `CMS_NOATTR`, `CMS_TEXT` и `CMS_STREAM` не поддерживаются, потому что они не имеют смысла в контексте подписанных квитанций.

Функция **CMS_sign_receipt()** возвращает либо корректную структуру типа `CMS_ContentInfo`, либо NULL, если произошла ошибка. Ошибку можно получить с помощью функции **ERR_get_error()**.

```
int CMS_uncompress(CMS_ContentInfo *cms, BIO *dcont, BIO *out,
      unsigned int flags);
```

Функция **CMS_uncompress()** извлекает и распаковывает данные из CMS-структуры cms типа CompressedData. data — BIO, в которое следует записывать данные, flags — опциональный набор флагов.

Параметр dcont используется в тех редких случаях, когда сжатые данные не включены в структуру. Обычно он устанавливается в NULL.

Единственный поддерживаемый сейчас алгоритм сжатия — это zlib: если структура указывает на применение любого другого алгоритма, возвращается ошибка.

Если поддержка алгоритма zlib не включена в OpenSSL при компиляции, функция **CMS_uncompress()** будет всегда возвращать ошибку.

В параметр flags могут быть переданы следующие флаги:

Если установлен флаг `CMS_TEXT`, из данных будут удалены MIME-заголовки типа text/plain. Если данные не являются данными типа text/plain, возвращается ошибка.

Функция **CMS_uncompress()** возвращает 1 в случае успешного завершения и 0 в случае завершения с ошибкой. Ошибку можно получить с помощью функции **ERR_get_error()**.

```
int CMS_verify(CMS_ContentInfo *cms, STACK_OF(X509) *certs,
      X509_STORE *store, BIO *indata, BIO *out, unsigned int flags);

STACK_OF(X509) *CMS_get0_signers(CMS_ContentInfo *cms);
```

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

Функция **CMS_verify()** проверяет CMS-структуру типа `SignedData`. cms — структура типа `CMS_ContentInfo`, которую следует проверить. certs — набор сертификатов, в котором следует искать сертификат (сертификаты) подписи. store — хранилище доверенных сертификатов, используемое для проверки цепочки доверия. indata — внешние данные, если данные не присутствуют в cms. Данные записываются в параметр out, если он не установлен в NULL.

flags — опциональный набор флагов, который можно использовать для модификации операции проверки.

Функция **CMS_get0_signers()** возвращает сертификат (сертификаты) подписи из структуры cms, ее необходимо вызывать после успешного выполнения функции **CMS_verify()**.

Как правило, процесс проверки происходит следующим образом:

Сначала проводятся несколько проверок разумности структуры cms. Тип структуры cms должен быть `SignedData`. Должна присутствовать хотя бы одна подпись под данными, и, если данные являются внешними, параметр indata не должен быть установлен в NULL.

Затем делается попытка найти все сертификаты подписи, сначала в параметре certs (если он не установлен в NULL), затем среди любых сертификатов, содержащихся в самой структуре cms. Если нельзя найти сертификат подписи, работа функции завершается с ошибкой.

Каждый сертификат подписи проверяется по цепочке сертификатов с использованием цели smimesign и предоставленного хранилища доверенных сертификатов. Любвые сертификаты, включенные в сообщение, используются как сертификаты промежуточных УЦ. Если включена проверка списков отзыва сертификатов (CRL), функция также пытается найти их в хранилище, кроме того, используются все CRL, включенные в сообщение. Если какая-либо проверка цепочки доверия не удается, возвращается код ошибки.

Наконец, читаются подписанные данные (и записываются в параметр out, если он не установлен в NULL) и проверяется подпись.

Если все подписи оказываются корректными, функция успешно завершается.

Любой из нижеуказанных флагов (вместе или по отдельности) может быть передан в параметр flags для модификации умолчательной процедуры проверки.

Если установлен флаг `CMS_NOINTERN`, сертификаты подписи не ищутся среди сертификатов, включенных в сообщение. Это означает, что все сертификаты подписи должны быть в параметре certs.

Если установлен флаг `CMS_NOCRL`, а проверка CRL включена в хранилище, то все CRL в самом сообщении игнорируются.

Если установлен флаг `CMS_TEXT`, из данных будут удалены MIME-заголовки типа text/plain. Если данные не являются данными типа text/plain, возвращается ошибка.

Если установлен флаг `CMS_NO_SIGNER_CERT_VERIFY`, то сертификаты подписи не проверяются.

Если установлен флаг `CMS_NO_ATTR_VERIFY`, то подпись под подписанными атрибутами не проверяется.

Если установлен флаг `CMS_NO_CONTENT_VERIFY`, дайджест данных не проверяется.

Одно из приложений флага `CMS_NOINTERN` — принимать только сообщения, подписанные на небольшом количестве сертификатов. Приемлемые сертификаты должны передаваться в параметре certs. В этом случае, если среди сертификатов в параметре certs нет сертификата подписи, проверка будет неудачной, потому что сертификат подписи найти не удастся.

В некоторых случаях стандартные методы поиска и проверки сертификатов не подходят: например, если приложение может пожелать просмотреть сертификаты в базе данных или выполнить проверку по своему сценарию. Это может быть достигнуто, если устанавливать и проверять сертификаты подписи вручную с помощью функций утилиты подписанных данных.

Следует с осторожностью модифицировать умолчательную процедуру проверки, например установка флага `CMS_NO_CONTENT_VERIFY` полностью отключает всю проверку данных, и любые модифицированные данные будут сочтеные корректными. Это сочетание, однако, полезно, если нужно только записать данные в параметр out, а их корректность не считается важной.

Проверку цепочки сертификатов, вероятно, следует производить с использованием времени подписи, а не текущего времени. Однако, поскольку время подписи указывается сертификатом подписи, ему нельзя доверять без дополнительной проверки (такой, как доверенная метка времени).

Функция **CMS_verify()** возвращает 1 при успешной проверке и ноль, если произошла ошибка.

Функция **CMS_get0_signers()** возвращает все сертификаты подписи, или же NULL, если произошла ошибка. Ошибку можно определить с помощью функции **ERR_get_error()**.

```
int CMS_verify_receipt(CMS_ContentInfo *rcms, CMS_ContentInfo *ocms,
      STACK_OF(X509) *certs, X509_STORE *store, unsigned int flags);
```

Функция **CMS_verify_receipt()** проверяет подписанную CMS-квитанцию. rcms — подписанная квитанция, которую нужно проверить. ocms — исходная структура типа `SignedData`, содержащия запрос на квитанцию. certs — набор сертификатов, в котором надо искать сертификат подписи. store — хранилище доверенных сертификатов (используется для проверки цепочки доверия).

flags — опциональный набор флагов, который можно использовать для модификации операции проверки.

Эта функция ведет себя подобно функции **CMS_verify()** за исключением того, что флаги `CMS_DETACHED`, `CMS_BINARY`, `CMS_TEXT` и `CMS_STREAM` не поддерживаются, поскольку они не имеют смысла в контексте подписанных квитанций.

Функция **CMS_verify_receipt()** возвращает 1 в случае успешной проверки и 0, если происходит ошибка.

7 Подсчет хэш-сумм

Функция **EVP_Digest** вычисляет хэш-сумму бфуера данных `data` длины `count` байт по алгоритму `type` и возвращает ее в буфере `md`. Количество байт, использованных в буфере `md` помещается в переменную `size`. Гарантируется что количество записываемых данных не превосходит `EVP_MAX_MD_SIZE`.

Параметр `impl` см ниже в описании функции **EVP_DigestInit_ex**.

```
int EVP_Digest(const void *data, size_t count,
       unsigned char *md, unsigned int *size, const EVP_MD *type,
       ENGINE
       *impl)
```

В случае, если все данные не доступны одновременно (а, например, поблочно считаются из файла), для подсчета хэш-сумм необходимо создать и проинициализировать структуру `EVP_MD_CTX`, затем прохэшировать необходимые данные и финализировать `EVP_MD_CTX`, получив значение хэш-суммы.

После финализации дальнейшее хэширование данных в данную структуру невозможно. Поэтому если необходимо получить значение хэш-суммы и продолжить хэширование данных, необходимо создать копию `EVP_MD_CTX`, финализировать копию, и продолжить работу с исходной структурой.

```
EVP_MD_CTX *EVP_MD_CTX_create(void)
```

Создает и размещает в динамической памяти структуру `EVP_MD_CTX`. Выполняет предварительную инициализацию созданной структуры.

```
void EVP_MD_CTX_init(EVP_MD_CTX *ctx)
```

Производит предварительную инициализацию структуры `EVP_MD_CTX`.

```
int EVP_DigestInit_ex(EVP_MD_CTX *ctx, const EVP_MD *md, ENGINE *impl)
```

Инициализирует структуру для начала работы с определенным алгоритмом хэширования. Алгоритм задается структурой `EVP_MD`. В случае если требуется использовать специфическую реализацию данного алгоритма, например, аппаратную, реализованную в подгружаемом модуле `ENGINE`, нужно указать ссылку на соответствующую engine.

Если единственная доступная реализация алгоритма преодставляется подгружаемым модулем, то можно указаывать `NULL` в качестве параметра `impl`, так как ссылка на необходимую engine уже содержится в структуре `EVP_MD`.

Структура должна быть предварительно инициализирована вызовом **EVP_MD_CTX_init**, если только она не была динамически создана вызовом **EVP_MD_CTX_create**.

```
int EVP_DigestInit(EVP_MD_CTX *ctx, const EVP_MD *md)
```

Функция **EVP_DigestInit** отличается от **EVP_DigestInit_ex** тем что автоматически инициализирует контекст, т.е. не требует предварительного вызова **EVP_MD_CTX_init** и всегда использует умолчательную реализацию алгоритма.

Структуры `EVP_MD`, описывающие алгоритмы хэширования зарегистрированы в специальной таблице OpenSSL, где возможен поиск:

- По строковому идентификатору алгоритма — с помощью функции **EVP_get_digestbyname**;

– По ASN.1 OID (представленному в виде структуры `ASN1_OBJECT`) — с помощью функции **EVP_get_digestbyobj**;
– По числовому идентификатору (NID) алгоритма — с помощью функции **EVP_get_digestbynid**.

Все эти функции возвращают ссылку на константную структуру, хранящуюся в таблице, которая может быть непосредственно передана в функцию **EVP_DigestInit_ex** и не нуждается в последующем освобождении.

Функция возвращает положительное значение в случае успеха и нулевое или отрицательное в случае ошибки.

```
int EVP_DigestUpdate(EVP_MD_CTX *ctx, const void *d, size_t cnt)
```

Хэширует ***cnt*** байт данных из буфера ***d*** с использованием инициализированной структуры ***ctx***.

```
int EVP_DigestFinal_ex(EVP_MD_CTX *ctx, unsigned char *md,
       unsigned int *s)
```

Финализирует структуру ***ctx***, помещая значение хэш-суммы в буфер ***md***. Если параметр ***s*** не равен `NULL`, то в переменную, на которую указывает этот параметр, будет помещено количество байт данных, записанных в ***md***. Количество записываемых байт не более `EVP_MAX_MD_SIZE`.

После вызова этой функции нельзя использовать структуру ***ctx*** в дальнейших вызовах **EVP_DigestUpdate**, но можно её повторно переинициализировать с помощью **EVP_DigestInit_ex**.

```
int EVP_DigestFinal(EVP_MD_CTX *ctx, unsigned char *md,
       unsigned int *s)
```

Отличается от **EVP_DigestFinal_ex** тем что производит очистку вызовом **EVP_MD_CTX_cleanup**.

```
int EVP_MD_CTX_cleanup(EVP_MD_CTX *ctx)
```

Очищает использованную структру `EVP_MD_CTX`, освобождая память, динамически размещенную **EVP_DigestInit_ex**. Эту функцию следует вызывать по завершении работы со статическими структурами `EVP_MD_CTX`.

```
int EVP_DigestFinal(EVP_MD_CTX *ctx, unsigned char *md,
       unsigned int *s)
```

Аналогично **EVP_DigestFinal_ex**, контекста после получения значения хэш-суммы выполняет очистку вызовом **EVP_MD_CTX_cleanup**.

```
void EVP_MD_CTX_destroy(EVP_MD_CTX *ctx)
```

Очищает и освобождает динамически размещенную с помощью функции **EVP_MD_CTX_create** структуру `EVP_MD_CTX`.

```
int EVP_MD_CTX_copy_ex(EVP_MD_CTX *out,const EVP_MD_CTX *in);
```

Копирует состояние алгоритма хэширования из ***in*** в ***out***. Структура ***out*** должна быть инициализирована до копирования.

```
int EVP_MD_CTX_copy(EVP_MD_CTX *out,const EVP_MD_CTX *in)
```

Аналогично **EVP_MD_CTX_copy_ex**, но не требует предварительной инициализации.

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

```
int EVP_MD_size(const EVP_MD *md);

int EVP_MD_CTX_size(EVP_MD_CTX *ctx);
```

Возвращают размер хэш-суммы в байтах для заданного алгоритма.

```
int EVP_MD_type(const EVP_MD *md);

int EVP_MD_CTX_type(EVP_MD_CTX *ctx);
```

Возвращают числовой идентификатор (NID) алгоритма хэширования.

```
const EVP_MD EVP_MD_CTX_md(EVP_MD_CTX *ctx);
```

Возвращает структуру `EVP_MD`, описывающую использованную при инициализации данной структуры `EV_MD_CTX`.

7.1 Пример хэширования данных

```
void hashfd(int fd)
{
        EVP_MD_CTX *ctx;
        ssize_t blocksize;
        short int md_size=0;
        int i;
        unsigned char buffer [4096];

        EVP_MD_CTX_init(&ctx,EVP_get_digestbyname("md_gost94"),NULL);

        while ((blocksize=read(fd, buffer, sizeof(buffer)))>=0)
                {
                        EVP_DigestUpdate(&ctx, buffer, blocksize);
                }
        EVP_DigestFinal_ex(&ctx,buffer,&md_size);
        EVP_MD_CTX_cleanup(&ctx);
        /* ГОСТ Р 34.11-94 специфицирует порядок байтов справа налево */
        for (i=md_size-1;i>=0;i--) {
                printf("%02x",buffer[i]);
        }
        printf("\n");
}
```

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

8 Выработка и проверка электроной подписи

Электронная подпись (с использованием асимметричных алгоритмов) и имитозащита (с использованием симметричного ключа) в OpenSSL используют единый интерфейс.

Тип используемого алгоритма определяется передаваемым параметром `pkey` имеющим тип `EVP_PKEY`.

Существует два набора высокоуровневых функций для выработки и проверки электронной подписи. В функциях, использующихся в первом наборе, закрытый ключ передается при инициализации контекста, что дает возможность использовать их для создания имитовставки. В функциях второго набора закрытый ключ передается при финализации. Этот набор проще в использовании, но не всегда является достаточным по функциональности. Пользователь может выбирать тот набор функций, который лучше подходит для его целей.

8.1 Первый набор функций

```
int EVP_DigestSignInit(EVP_MD_CTX *ctx, EVP_PKEY_CTX **pctx,
       const EVP_MD *type, ENGINE *impl, EVP_PKEY *pkey)
```

Инициализирует контекст алгоритма хэширования для операции выработки подписи с использованием энджина `impl` и закрытого ключа `pkey`. Контекст должен быть предварительно инициализирован с помощью функции **EVP_MD_CTX_init**.

Адрес контекста операции с асимметричными ключами, создаваемый в процессе инициализации, помещается в переменную `pctx`, если этот параметр не равен `NULL`. Это позволяет после инициализации контекста установить его определенные параметры.

Параметр `type` задает тип алгоритма хэширования, используемый при выработке подписи. Для ряда алгоритмов подписи, таких как ГОСТ Р 34.10 или DSA, тип алгоритма хэширования жестко специфицирован в стандарте на алгоритм выработки подписи. В этих случаях параметр `type` можно указывать как `NULL`. Нужный алгоритм хеширования будет выбран автоматически на основании алгоритма подписи. Если `type` равен `NULL`, а параметр `pkey` содержит ключ алгоритма, для которого не определен стандартный алгоритм хеширования, будет использован алгоритм хэширования SHA1.

Функция возвращает 1 в случае успеха, и 0 или отрицательное число в случае ошибки.

```
int EVP_DigestVerifyInit(EVP_MD_CTX *ctx, EVP_PKEY_CTX **pctx,
       const EVP_MD *type, ENGINE *e, EVP_PKEY
       *pkey)
```

Инициализирует контекст проверки подписи. В этом случае в параметре `pkey` может содержаться только открытый ключ, без закрытого. Такой ключ может быть получен, например, из сертификата X.509 с помощью функции **X509_get_pubkey**.

```
int EVP_DigestSignUpdate(EVP_MD_CTX *ctx, const void *d,
       unsigned int cnt)

int EVP_DigestVerifyUpdate(EVP_MD_CTX *ctx, const void *d,
       unsigned int cnt)
```

Обрабатывает `cnt` байт данных для выработки подписи. Возвращает положительное число в случае успеха, и 0 в случае ошибки.

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

```
int EVP_DigestSignFinal(EVP_MD_CTX *ctx, unsigned char *sig, size_t
      *siglen)
```

Вырабатывает подпись под данными, захэшированными в контексте `ctx`, и помещает результирующую подпись в буфер `sig`. Длину подписи помещает в переменную `siglen`.

На момент вызова функции в `siglen` должна содержаться длина доступной памяти в буфере `sig`.

```
int EVP_DigestVerifyFinal(EVP_MD_CTX *ctx, unsigned char *sig,
      size_t siglen)
```

Проверяет, что электронная подпись `sig`, длина которой `siglen`, соответствует данным, обработанным контекстом `ctx`.

Возвращает 1 в случае успеха, 0, если подпись некорректна, и отрицательное значение в случае ошибки.

Операции финализации не очищают контекста подписи/проверки. Контекст должен быть очищен с помощью **EVP_MD_CTX_cleanup**.

8.2 Второй набор функций

Для выработки подписи используются функции:

```
int EVP_SignInit_ex(EVP_MD_CTX *ctx, const EVP_MD *type,
      ENGINE *impl)

int EVP_SignUpdate(EVP_MD_CTX *ctx, const void *d, unsigned
      int cnt)

int EVP_SignFinal(EVP_MD_CTX *ctx,unsigned char *sig,unsigned
      int *s, EVP_PKEY *pkey)

void EVP_SignInit(EVP_MD_CTX *ctx, const EVP_MD *type)

int EVP_PKEY_size(EVP_PKEY *pkey)
```

.

Эти функции представляют собой высокоуровневый интерфейс к цифровым подписям.

Функция **EVP_SignInit_ex()** настраивает контекст подписи `ctx` на использование типа дайджеста из модуля engine `impl`. Контекст `ctx` перед вызовом данной функции необходимо инициализировать с помощью функции **EVP_MD_CTX_init()**. При удачном завершении данная функция возвращает 1, при неудаче 0.

Функция **EVP_SignUpdate()** хэширует `cnt` байтов данных в позиции `d` в контекст подписи `ctx`. Эту функцию можно вызывать несколько раз для одного и того же `ctx` для включения дополнительных данных. При удачном завершении данная функция возвращает 1, при неудаче 0.

Функция **EVP_SignFinal()** подписывает данные, содержащиеся в `ctx`, с использованием закрытого ключа `pkey`, и помещает результат в `sig`. Количество байт в записанных данных (т.е. длина подписи) будет записана в виде целого числа в позицию `s`, причем максимальным значением этого числа будет значение функции **EVP_PKEY_size(pkey)**. При удачном завершении данная функция возвращает 1, при неудаче 0.

Функция **EVP_SignInit()** инициализирует контекст подписи `ctx` для использования умолчательной реализации типа дайджеста.

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

Функция **EVP_PKEY_size()** возвращает максимальный размер подписи в байтах. Реальная подпись, полученная с помощью функции **EVP_SignFinal()**, может быть меньше.

Интерфейс EVP к цифровым подписям почти всегда предпочтительнее низкоуровневых интерфейсов, потому что в случае его использования код оказывается прозрачным для используемых алгоритмов и намного более гибким.

Из-за связи между дайджестами сообщений и алгоритмами открытого ключа необходимо использовать алгоритм дайджеста, соответствующий типу алгоритма открытого ключа.

При выработке подписи с помощью закрытых ключей ГОСТ Р 34.10 генератор случайных чисел необходимо инициализировать.

Вызов функции **EVP_SignFinal()** финализирует копию контекста дайджеста. Это означает, что впоследствии можно повторно обращаться к функциям **EVP_SignUpdate()** и **EVP_SignFinal()** для хэширования и подписи дополнительных данных.

Поскольку финализируется только копия контекста дайджеста, контекст после использования необходимо очищать с помощью функции **EVP_MD_CTX_cleanup()**, в противном случае произойдет утечка памяти.

Поскольку закрытый ключ передается в вызове функции **EVP_SignFinal()**, то любая ошибка, связанная с закрытым ключом (например, неподходящая комбинация алгоритмов ключа и дайджеста) может не быть указана до тех пор, пока через функцию **EVP_SignUpdate()** не пройдет достаточно большое количество данных.

С помощью этих функций невозможно изменить параметры подписи.

Для проверки подписи используются функции:

```
int EVP_VerifyInit_ex(EVP_MD_CTX *ctx, const EVP_MD *type,
       ENGINE *impl)

int EVP_VerifyUpdate(EVP_MD_CTX *ctx, const void *d, unsigned
       int cnt)

int EVP_VerifyFinal(EVP_MD_CTX *ctx,unsigned char *sigbuf,
       unsigned int siglen,EVP_PKEY *pkey)

int EVP_VerifyInit(EVP_MD_CTX *ctx, const EVP_MD *type)
```

Эти функции представляют собой высокоуровневый интерфейс к цифровым подписям.

Функция **EVP_VerifyInit_ex()** настраивает контекст проверки подписи `ctx` на использование типа дайджеста из модуля engine `impl`. Контекст `ctx` перед вызовом данной функции необходимо инициализировать с помощью функции **EVP_MD_CTX_init()**. При удачном завершении данная функция возвращает 1, при неудаче 0.

Функция **EVP_VerifyUpdate()** хэширует `cnt` байтов данных в позиции `d` в контекст проверки подписи `ctx`. Эту функцию можно вызывать несколько раз для одного и того же `ctx` для включения дополнительных данных. При удачном завершении данная функция возвращает 1, при неудаче 0.

Функция **EVP_VerifyFinal()** проверяет данные, содержащиеся в `ctx`, с использованием открытого ключа `pkey`, опираясь на siglen байт в позиции sigbuf. При удачном завершении данная функция возвращает 1, при неудаче 0, при любой другой ошибке -1.

Функция **EVP_VerifyInit()** инициализирует контекст проверки подписи `ctx` для использования умолчательной реализации типа дайджеста.

Интерфейс EVP к цифровым подписям почти всегда предпочтительнее низкоуровневых интерфейсов, потому что в случае его использования код оказывается прозрачным для используемых алгоритмов и намного более гибким.

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

Из-за связи между дайджестами сообщений и алгоритмами открытого ключа необходимо использовать алгоритм дайджеста, соответствующий типу алгоритма открытого ключа.

Вызов функции **EVP_SignFinal()** финализирует копию контекста дайджеста. Это означает, что впоследствии можно повторно обращаться к функциям **EVP_SignUpdate()** и **EVP_SignFinal()** для хэширования и проверки дополнительных данных.

Поскольку финализируется только копия контекста дайджеста, контекст после использования необходимо очищать с помощью функции **EVP_MD_CTX_cleanup()**, в противном случае произойдет утечка памяти.

9 Операции с ключевыми парами

Ключевые пары ассиметричных криптоалгоритмов в OpenSSL представляются в виде структуры `EVP_PKEY`. Эта структура содержит либо только открытый ключ, либо и закрытый, и открытый. При загрузке закрытого ключа автоматически вычисляется соответствующий открытый.

Эта же структура используется для хранения симметричного ключа алгоритмов имитозащиты.

Операции со ключевой парой, как правило, предполагают создание контекста операции `EVP_PKEY_CTX`, инициализацию его для конкретной операции и собственно выполнение операции.

9.1 Операции с EVP_PKEY

`EVP_PKEY_new()`

Создает пустую структуру `EVP_PKEY`, не содержащую ни параметров, ни открытого ключа, ни закрытого. Как правило, в приложениях не используется

```
int EVP_PKEY_set_type(EVP_PKEY *pkey, int id)
```

Устанавливает тип алгоритма по его числовому коду (nid).

```
int EVP_PKEY_set_type_str( EVP_PKEY *pkey,const char *str, int len)
```

Устаннавливает тип ключа по названию алгоритма.

```
EVP_PKEY_free(EVP_PKEY *pkey)
```

Уменьшает счетчик ссылок на данную структуру `EVP_PKEY` и освобождает её, если этот счетчик ссылок стал равным 0.

```
int EVP_PKEY_missing_parameters(const EVP_PKEY *pkey)
```

Возвращает единицу, если в данной структуре `EVP_PKEY` не установлены параметры алгоритма.

```
int EVP_PKEY_copy_parameters(EVP_PKEY to, const EVP_PKEY *from)
```

Копирует параметры алгоритма из структуры `from` в структуру `to`.

```
int EVP_PKEY_cmp_parameters(const EVP_PKEY *a, const EVP_PKEY *b)
```

Сравнивает наборы параметров двух ключей. Возвращает 1, если они совпадают, и 0 если не совпадают, -1 если ключи разных типов и -2, если операция не поддерживается данным типом ключей.

```
int EVP_PKEY_cmp(const EVP_PKEY *a,const EVP_PKEY *b)
```

Сравнивает параметры и открытые ключи в двух структурах. Поскольку структура, содержащая закрытый ключ, всегда содержит и открытый, эта функция может использоваться для проверки соответствия открытого ключа закрытому.

Возвращаемые значения аналогичны **EVP_PKEY_cmp_parameters**.

```
int EVP_PKEY_id(const EVP_PKEY *pkey)
```

Возвращает тип алгоритма соответствующего данному ключу.

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

9.2 Создание контекста

```
EVP_PKEY_CTX *EVP_PKEY_CTX_new(EVP_PKEY *pkey)
```

Создает контекст операции с ключевой парой на основе ключа `pkey`

```
EVP_PKEY_CTX *EVP_PKEY_CTX_new_id(int id, ENGINE *e)
```

Создает контекст без использования предварительно загруженного ключа. Используется, например, при операции генерации ключа.

Параметр `id` представляет собой числовой идентификатор (NID) соответствующего алгоритма.

```
int EVP_PKEY_CTX_ctrl(EVP_PKEY_CTX *ctx, int keytype, int optype,
       int cmd, int p1, void *p2)
```

Передает реализации алгоритма алгоритм-специфичную управляющую команду. Параметр `keytype` указывает тип (NID) ключа, `optype` тип выполняемой операции, `cmd` алгоритм-специфичный код команды, `p1` и `p2` - параметры команды.

Как правило, эта функция не используется разработчиками приложений явно. Используется либо функция **EVP_PKEY_CTX_ctrl_str**, либо макросы, подобные:

```
int EVP_PKEY_get_default_digest_nid(EVP_PKEY *pkey, int *pnid)
```

Запрашивает идентификатор алгоритма хэширования, который используется по умолчанию с данным асимметричным алгоритмом. Значение записывается в переменную, указатель на которую передается в параметре `pnid`.

```
int EVP_PKEY_CTX_ctrl_str(EVP_PKEY_CTX *ctx, const char *type,
       const char *value)
```

передает в реализацию алгоритма команду по строковому названию.

Алгоритм ГОСТ Р 34.10-2001 поддерживает только команду paramset, аргументом которой является наименование набора параметров в соответствии с RFC 4357. («A»,«B»,«C»,«XA»,«XB» или «test»). Кроме того, можно использовть точечное десятичное представление ASN.1 OID-ов и короткие имена, соответствующие этим OID-ам в базе данных объектов

9.3 Генерация ключевой пары

```
int EVP_PKEY_keygen_init(EVP_PKEY_CTX *ctx)
```

Инициализирует контекст для операции создания ключей. После этой операции можно использовать команды функций **EVP_PKEY_CTX_ctrl** и **EVP_PKEY_CTX_ctrl_str**, специфичные для операции создания ключей.

```
int EVP_PKEY_keygen(EVP_PKEY_CTX *ctx, EVP_PKEY **ppkey)
```

Выполняет собственно генерацию ключевой пары. Создает новую структуру `EVP_PKEY` и помещает указатель на нее в переменную `ppkey`.

Для создания ключевой пары ГОСТ Р 34.10-2001 необходимо перед генерацией установить набор параметров. Это можно сделать либо создав контекст с помощью **EVP_PKEY_CTX_new**, передав ей ключ с требуемым набором параметров, либо, если контекст создан с помощью функции **EVP_PKEY_CTX_new_id**, передав название набора параметров с помощью **EVP_PKEY_CTX_ctrl_str("paramset",имя-набора-параметров)**.

Пример генерации ключевой пары ГОСТ Р 34.10-2001

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

```
EVP_PKEY *create_gost_keypair() {
        EVP_PKEY *key1 = EVP_PKEY_new(),*newkey=NULL;
        EVP_PKEY_CTX *ctx;

        EVP_PKEY_set_type_str(key1,"GOST2001",8);
        ctx=EVP_PKEY_CTX_new(key1);
        EVP_PKEY_keygen_init(CTX);
        EVP_PKEY_CTX_ctrl_str(ctx,"paramset","A");
        EVP_PKEY_keygen(ctx,&newkey);
        EVP_PKEY_CTX_free(ctx);
        EVP_PKEY_free(key1);
        return newkey;
}
```

9.4 Выработка и проверка подписи

Описанные в данном разделе низкоуровневые функции выработки и проверки подписи работают не с данными, а с предварительно вычисленной хэш-суммой от данных. Для выработки электронной подписи под данными следует использовать более высокоуровневые функции, описанные в разделе 8.

Контекст для операции подписи должен быть создан на основе структуры `EVP_PKEY`, содержащей закрытый ключ подписывающего.

```
int EVP_PKEY_sign_init(EVP_PKEY_CTX *ctx)
```

Инициализирует контекст для операции выработки подписи.

```
int EVP_PKEY_sign(EVP_PKEY_CTX *ctx,
       unsigned char *sig, size_t *siglen, const unsigned char *tbs,
       size_t tbs_len)
```

Вырабатывает подпись под хэшем `tbs` длины `tbs_len`, помещает результат в буфер `sig`. Если `sig` равна `NULL`, вычисляет длину подписи и помещает вычисленное значение в `siglen`.

Если `sig` не `NULL`, то `siglen` на входе должна содержать количество доступных байт в буфере `sig`.

```
int EVP_PKEY_verify_init(EVP_PKEY_CTX *ctx)
```

Инициализирует контекст для операции проверки подписи. Контекст должен быть создан на основе структуры `EVP_PKEY`, содержащей открытый ключ подписавшего.

```
int EVP_PKEY_verify(EVP_PKEY_CTX *ctx,
       const unsigned char *sig, size_t siglen,
       const unsigned char *tbs, size_t tbslen)
```

Проверяет, что подпись `sig` соответствует значению хэша `tbs`.

Возвращает 1 если подпись соответствует, 0, если не соответствует и отрицательное чилло в случае ошибки.

9.5 Загрузка закрытого ключа из файла.

OpenSSL поддерживает закртые ключи в формате PKCS#8, которые могут храниться в файлах как в бинарной кодировке DER, так и в ASCII-кодировке PEM.

Для загрузки закрытого ключа из файла PEM используется функция **PEM_read_bio_PrivateKey**, аналогичная по интерфейсу остальным функциям чтения, описанным в разделе 18.3.

Для загрузки ключ из файла в формате DER испоьзуются функции

```
EVP_PKEY *d2i_PKCS8PrivateKey_bio(BIO *bp, EVP_PKEY **x,
       pem_password_cb
       *cb, void *u)

EVP_PKEY *d2i_PKCS8PrivateKey_fp(FILE *fp, EVP_PKEY **x,
       pem_password_cb
       *cb, void *u)
```

интерфейс которых аналогичен функциям чтения PEM.

9.6 Загрузка закрытого ключа из аппаратного хранилища.

```
EVP_PKEY *ENGINE_load_private_key(ENGINE *e, char *key_id,
       UI_METHOD *ui_method, void *callback_data)
```

Выполняет загрузку закрытого ключа из аппаратного токена, поддерживаемого соответствующей engine.

e ссылка на engine. Должна быть предварительно получена с помощью функции **ENGINE_by_id** (см раздел 1.2).

key_id идентификатор ключа на аппаратном токене. См раздел 10

ui_method указатель на набор фукций, используемых для взаимодействия с пользователем. Crtyptocom engine использует его только для запроса пароля, на котором зашифрован ключ. См раздел 11.

callback_data указатель, передаваемый в функции взаимодействия с пользователем.

Неготовность устройства (например неприслоненная таблетка в устройстве «АККОРД») приводит к завершению функции с ошибкой и возвращению NULL.

9.7 Запись закрытых ключей в файл

Для записи используются функци

```
int PEM_write_bio_PKCS8PrivateKey(BIO *bp, EVP_PKEY *x,
       const EVP_CIPHER *enc,  char *kstr, int klen,
       pem_password_cb *cb, void *u)

int PEM_write_PKCS8PrivateKey(FILE *fp, EVP_PKEY *x,
       const EVP_CIPHER *enc, char *kstr, int klen, pem_password_cb
       *cb,
       void *u)

int i2d_PKCS8PrivateKey_bio(BIO *bp, EVP_PKEY *x,
```

```
        const EVP_CIPHER *enc, char *kstr, int klen,
        pem_password_cb *cb, void *u);

int i2d_PKCS8PrivateKey_fp(FILE *fp, EVP_PKEY *x,
        const EVP_CIPHER *enc, char *kstr, int klen,
        pem_password_cb *cb, void *u)
```

В случае, если параметр `enc` не равен NULL, эти функции выполняют зашифрование ключа на пассфразе с использованием указанного алгоритма. Ключ зашифрования выводится из пассфразы по алгоритму PKCS#5v2.

Пассфраза может быть передан с помощью параметра `kstr` и длины `klen`, либо может быть предоставлена функция обратного вызова `cb`, которая запросит пассфразу у пользователя. Параметр `u` будет передан в функцию `cb` в качестве параметра `userdata`. Если `kstr` и `cb` равны `NULL`, а `u` не `NULL`, то `u` интерпретируется как строка символов C, оканчивающаяся нулевым байтом и используется в качестве пассфразы.

В соответствии со стандартом PKCS#5v2, если пассфраза содержит русские буквы, она должна быть конвертирована в кодировку UTF-8.

9.8 Запись закрытых ключей на аппаратный носитель

В OpenSSL не существует стандартного API для записи ключей на аппаратный носитель. Поэтому в продукте СКЗИ «МагПро Криптопакет» для записи ключей на поддерживаемые аппаратные носители используется функция **ENGINE_ctrl_cmd**, позволяющая передать произвольную управляющую команду энджину.

```
int ENGINE_ctrl_cmd(ENGINE *e, const char *cmd_name,
        long i, void *p, void (*f)(void), int cmd_optional)
```

Имя команды записи ключа на носитель `"ce write private key"`.

В данном случае в качестве параметра `p` передается указатель на следующую структуру:

```
struct write_key_args_st {
    const EVP_PKEY *pkey;
    const char *key_id;
    unsigned int flags;
    UI_METHOD *ui_method;
    void *callback_data;
};
```

Где

`pkey` — записываемый ключ

`key_id` — строковый идентификатор носителя и ключа на носителе (см. 10)

`flags` — флаги операции записи.

Поддерживаемые флаги:

`CE_WRITE_CTRL_PASSWORD_ENCRYPT` — указывает, что функция должна при записи запросить пароль и зашифровать ключ на пароле.

`CE_WRITE_CTRL_OVERWRITE` — производить запись ключа на место существующего

`ui` — указатель на набор функций, используемый для взаимодействия с пользователем. См. раздел 11.

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

`callback_data` — указатель, передаваемый в функции взаимодействия с пользователем.

Вызов команды из приложения выглядит так:

```
struct write_key_args_st r;
ENGINE *e;
EVP_PKEY *key;
UI_METHOD *my_method;
void *my_data;
....

r.pkey = key;
r.flags = CE_WRITE_CTRL_PASSWORD_ENCRYPT;
r.key_id = "ACCORD.S";
r.ui_method = my_method;
r.callback_data = my_data;

e = ENGINE_by_id("cryptocom");

if (!ENGINE_ctrl_cmd(e, "ce write private key", 0, &r, NULL, 0))
{
        ERR_print_errors_fp(stderr);
}
```

Разработчикам модулей поддержки ключевых носителей, предназначенных для работы совместно с СКЗИ «МагПро Криптопакет», рекомендуется поддерживать этот интерфейс.

9.9 Удаление закрытых ключей с аппаратного носителя

Удаление также производится с помощью управляющей команды энджина.

Имя команды `"ce wipe key container"`.

Структура параметров:

```
struct wipe_container_args_st {
    const char *container_id;
    UI_METHOD *ui_method;
    void *callback_data;
};
```

Поскольку для носителей, поддерживаемых cryptocom engine, реализована только операция удаления контейнера целиком, при этой операции указывается только спецификация контейнера, без суффикса, указывающего с каким из ключей в контейнере производится работа.

9.10 Работа с форматом PKCS#8

В некоторых случаях необходимо преобразовывать закрытые ключи из формата PKCS#8 во внутреннее представление `EVP_PKEY` и обратно без использования вышеописанных функций записи и чтения закрытых ключей в файл.

Незашифрованный PKCS8 контейнер представляется в виде структуры `PKCS8_PRIV_KEY_INFO`. Эта структура может быть прочитана с помощью соответствующих **d2i** и **PEM_read_bio_** функций.

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

Зашифрованный PKCS8 контейнер представляетс в виде структуры `X509_SIG` (так как аналогично подписи состоит из идентификатора алогритма и бинарных данных).

10 Формат идентификаторов ключа на аппаратных носителях, поддерживаемых энджином cryptocom

Строка идентификатор ключа для устройств, поддерживаемых cryptocom engine имеет следующий формат:

ТИП-УСТРОЙСТВА[=*ID*][:*имя-контейнера*].{X|S}

Где ТИП-УСТРОЙСТВА может быть ACCORD или VJUGA, ID- специфичный для устройства аппаратный идентификатор устройства, имя контейнера — имя ключевого контейнера, заданное при его создании. Идентификатор устройства и имя контейнера могут быть пропущены.

X или S — идентификатор одного из двух ключей в контейнере. X — ключ обмена ключами, S — ключ подписи.

Попытка обращения к ключу с именем контейнера, отличным от того, который имеется на доступном в данный момент устройстве, приводит к ошибочному завершению операции.

При операции загрузки или записи ключа спецификация ключа указывается полностью. При операции очистки ключевого контейнера суффикс .X или .S не указывается, так как удаляется контейнер целиком.

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

11 Создание методов взаимодействия с пользователем

Командно-строчные приложения могут использовать стандартный метод взаимодействия с пользователем, указатель на который возвращается функцией **UI_OpenSSL**. Графические приложения или приложения использующие полноэкранный текстовый интерфейс, должны определить свой собственный метод.

```
UI_METHOD *UI_create_method(char *name);
```

Создает структуру метода взаимодействия с пользователем.

```
void UI_destroy_method(UI_method *method);
```

Освобождает структуру.

```
int UI_method_set_opener(UI_METHOD *method, int (*opener)(UI *ui));
```

Устанавливает функцию opener, вызываемую при инициализации взаимодействия с пользователем. Эта функция может, например, разместить структуру данных диалогового окна.

Указатель на специфические для метода данные должен быть установлен в переданную структуру UI с помощью функции **UI_add_user_data**. Другие функции смогут получить этот указатель с помощью **UI_get0_user_data**.

```
void *UI_add_user_data(UI *ui, void *user_data);

void *UI_get0_user_data(UI *ui);

int UI_method_set_writer(UI_METHOD *method, int (*writer)(UI *ui,
       UI_STRING *uis));
```

Устанавливает функцию writer, формирующую элемент интерфейса.

```
int UI_method_set_flusher(UI_METHOD *method,
       int (*flusher)(UI *ui));
```

Устанавливает функцию flusher, которая вызвыается, когда формирование интерфейса завершено и нужно предоставить пользователю возможность ввести необходимые данные.

```
int UI_method_set_reader(UI_METHOD *method,
       int (*reader)(UI *ui, UI_STRING *uis));
```

Устанавливает функцию reader, которая должна получить данные у соответствующих элементов интерфейса и поместить их в структуру `UI_STRING`. Эта функция вызывается для всех элементов управления, добавленных в интерфейс, в том числе и для тех, которые не предполагают никакого ввода.

```
int UI_method_set_closer(UI_METHOD *method, int (*closer)(UI
       *ui));
```

Устанавливает функцию closer, которая должна завершить выполнение и освободить все ресурсы, задействованные в функции opener.

Все функции, входящие в состав метода, возвращают 1 в случае успешного завершения, 0 в случае ошибки, и -1 в случае возникновения специальной ситации, такой как прерывание консольного приложения по Ctrl-C или закрытия окна средствами оконной системы.

Структура `UI_STRING` передаваемая в функции reader и writer, является непрозрачной для приложения структурой. Полуить доступ к содержащейся в ней информации можно с помощью следующих функций:

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

```
enum UI_string_types UI_get_string_type(UI_STRING *uis);
```

Возвращает тип данного элемента управления. Возможные значения

`UIT_PROMPT` — Строка ввода строкового значения.
`UIT_VERIFY` — Строка ввода строкового значения с верификацией (например при вводе пароля).
`UIT_BOOLEAN` — Вопрос, предполагающй ответ да/нет.
`UIT_INFO` — Информационная строка для пользователя.
`UIT_ERROR` — Сообщение об ошибке.

```
UI_get_input_flags(UI_STRING *uis);
```

Возвращает флаги, связанные с данным элементом управления. Возможные биты:

`UI_INPUT_FLAG_ECHO` — Установлен, если вводимые данные должны отображаться.
`UI_INPUT_FLAG_DEFAULT_PWD` — Использовать умолчательное значение пароля (например, установленное приложением в специфическую для приложения структуру, доступную через **UI_get_user_data**). Рекомендуется, чтобы не более одного элемента управления в каждом интерфейсе имело этот флаг установленным.

```
const char *UI_get0_output_string(UI_STRING *uis);
```

Получить строку для вывода.

```
const char *UI_get0_action_string(UI_STRING *uis);
```

Получить строку с рекомендациями для пользователя (для элементов управления предполагающих ввод да/нет).

```
const char *UI_get0_result_string(UI_STRING *uis);
```

Получить строку результата ввода.

```
const char *UI_get0_test_string(UI_STRING *uis);
```

Получить строку с которой сравнивается результат. Используется для проверки корректности ввода пароля, который не отображается на экране.

```
int UI_get_result_minsize(UI_STRING *uis);
```

Получить минимальный размер вводимых данных.

```
int UI_get_result_maxsize(UI_STRING *uis);
```

Получить максимальный размер вводимых данных.

```
UI_set_result(UI *ui, UI_STRING *uis, const char *result);
```

Установить результат, полученный от пользователя.

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

12 Реализация поддержки аппаратных ключевых контейнеров

Для реализации поддержки аппаратных ключевых контейнеров необходимо реализовать модуль engine, реализующий функцию **ENGINE_load_private_key**, возвращающую структуру `EVP_PKEY`, содержащую готовый к использованию закрытый ключ.

Для этого необходимо реализовать собственно функцию со следующим прототипом:

```
EVP_PKEY *load_key_func(ENGINE *e, const char *key_id,
      UI_METHOD *ui_method,void *callback_data);
```

и реализовать ряд служебных функций, обеспечивающих регистрацию этой engine в OpenSSL.

После чего этот модуль собирается в динамическую библиотеку, и помещается в директорию, откуда OpenSSL подгружает свои engines.

Рекомендуется также реализовать управляющую функцию, поддерживающую команды записи ключа и очистки ключевого контейнера, описанные в разделах 9.8 и 9.9.

12.1 Реализация функции загрузки ключа

12.1.1 Использование функций взаимодействия с пользователем

Функция загрузки ключей получает два параметра: `ui_method`, представляющий собой непрозрачную для приложения структуру с указателями на функции взаимодействия с пользователем, и `callback_data`, представляющую собой указатель, который должен быть передан в эту функцию.

Для того чтобы организовать взаимодействие с пользователем, необходимо сначала создать структуру `UI` и установить в неё этот указатель.

```
UI *UI_new_method(UI_METHOD *method);
```

Создает экземпляр структуры взаимодействия с пользоателем, использующий указанный метод.

```
void UI_add_user_data(UI *ui, void *callback_data);
```

Устанавливает указатель на пользовательские данные в экземпляр структуры взаимодействия с пользователем.

После этого необходимо создать в структуре `UI` необходимые поля ввода:

```
int UI_add_input_string(UI *ui, const char *prompt, int flags,
      char *result_buf, int minsize, int maxsize);
```

Добавляет в интерфейс пользователя строку ввода с приглашением `prompt`, результат ввода в которую будет помещен в `result_buf`. Параметры `minsize` и `maxsize` задают минимальную и максимальную длину строки, которую можно вводить в это поле. Т.е. например, если указать `minsize` равным 6 в строке ввода пароля, то ввод более короткого пароля будет отвергнут на уровне обработки пользовательского интерфейса.

Параметр `flags` представляет собой битовую маску флагов. Возможные значения:

`UI_INPUT_FLAG_ECHO` — указывает, что вводимые пользователем символы должны отображаться (т.е. данная строка ввода не является сенситивной информацией, такой как пароль или PIN-код).

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

`UI_INPUT_FLAG_DEFAULT_PWD` — указывает что в качестве значения этого поля реализация `UI_METHOD` имеет право вернуть значение пароля по умолчанию (например, взяв его из какого-то поля внутри структуры, на которую указывает `callback_data`).

```
int UI_add_verify_string(UI *ui, const char *prompt, int flags,
       char *result_buf, int minsize, int maxsize,
       const char *test_buf);
```

Добавляет строку повторного ввода пароля. В отличие от **UI_add_input_string** эта функция получает дополнительный константный буфер (обычно совпадающий с буфером одной из предыдущих строк ввода), и функции обработки пользовательского ввода считают ввод успешным, только если строка, введенная в данное поле, совпадает со строкой, содержащейся в `test_buf`.

```
int UI_add_input_boolean(UI *ui, const char *propmt,
       const char *action_desc, const char *ok_chars,
       const char *cancel_chars, int flags, char *result_buf);
```

Добавляет поле для запроса да/нет. Параметр `prompt` задает описание поля, параметр `action_desc` — дополнительные указания пользователю. `UI_METHOD` может игнорировать строку `action_desc`, например, если данное поле представляется в диалоговом окне в качестве чек-бокса, и действия пользователя по выбору ответа очевидны. Параметры `ok_chars` и `cancel_chars` задают наборы символов, которые воспринимаются как корректные ответы «да» и «нет» соответственно. В `result_buf` помещается результат ввода.

```
int UI_add_info_string(UI *ui, const char *text);
```

Добавляет в интерфейс информационное сообщение для пользователя.

```
int UI_add_error_string(UI *ui, const char *text);
```

Добавляет сообщение об ошибке.

Все функции добавления полей ввода возвращают в случае успеха некоторое положительное значение, индекс поля в интерфейсе, которое может быть затем использовано для получения результата ввода с помощью функции **UI_get0_result**. Обычно эта функция не используется, и после завершения взаимодействия с пользователем результаты ввода анализируются непосредственно в тех буферах, которые были переданы функциям добавления в качестве `result_buf`.

После добавления всех необходимых полей, нужно вызывать функцию

```
int UI_process(UI *ui);
```

Эта функция возвращает отрицательное значение, если ввод был прерван или произошла ошибка. Если она вернула 0 или положительное значение, необходимые данные были введены.

После завершения этой функции можно обращаться к введенным данным либо посредством непосредственного доступа к буферам, переданным в функции добавления полей, либо с помощью

```
const char *UI_get0_result(UI *ui, int index);
```

Эта функция возвращает указатель на буфер поля ввода с соответствующим индексом.

Как правило, можно вызывать **UI_process** несколько раз. Например, если не удалось расшифровать ключ на введенном пароле, можно добавить в интерфейс сообщение об ошибке с помощью **UI_add_error_string** и вызывать **UI_process** еще раз с тем же интерфейсом.

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

Но некоторые реализации `UI_METHOD` могут требовать пересоздания интерфейса. Поэтому рекомендуется проверить возможность повторного использования интерфейса с помощью вызова

UI_ctrl(ui,UI_CTRL_IS_REDOABLE,0,NULL,NULL);

Этот вызов возвращает 1 если повторое использование интерфейса возможно, и 0 если нет.

После окончания использования интерфейса его необходимо удалить с помощью

```
void UI_free(UI *ui)
```

12.1.2 Формирование структуры EVP_PKEY

Если формат хранения данных на носителе представляет собой PKCS#8, для разбора этих структур можно использовать функции чтения закрытого ключа из файла.

Для этого нужно содзать объект `BIO` в буфере памяти и читать ключ из него.

Если используется какой-то формат хранения, то в какой-то момент в результате разбора этого формата будет получен OID параметров ключа и набор байт, представляющих собой двоичное представление закрытого ключа.

Из этих данных нужно сформировать незашифрованную PKCS#8 структуру, после чего разобрать её вышеописанным способом.

После разбора нужно немедленно очистить созданный буфер с помощью функции **OPENSSL_cleanse**.

12.2 Использование системы обработки ошибок OpenSSL

Для выдачи из engine сообщений об ошибках, которые могут быть корректно использованы приложениями, использующими функции обработки ошибок, описанные в разделе 18.6 используется препроцессор `mkerr.pl`, входящий в состав исходных текстов OpenSSL.

Для этого препроцессора необходимо создать конфигурационный файл с расширением .ec, содержащий как минимум строку

L ID filename.h filename.c

где ID — идентификатор engine, из которого будут сформированы имена функций загрузки списока сообщений об ошибках (ERR_load_ID_strings) и выдачи сообщения об ошибках (IDerr).

Кроме этого там могут содержаться дополнительные строки вида

L NONE filename.h NONE
L NONE filename.h filename.c
L NONE NONE filename.c

указывающие на внутренние для реализации данной engine файлы, которые следует просматривать на предмет поиска прототипов функций.

Препроцессор запускается командой

```
mkerr.pl -conf filename.ec -nostatic -write filename.c filename2.c ...
```

Т.е. ему передаются имя файла конфигурации и список всех файлов исходного текста, в которых вызывается функция выдачи сообщения об ошибок.

Далее, во все файлы проекта, в которых необходимо выдавать сообщения об ошибках, необходимо включить директивой #include заголовочный файл, сгенерированный этим препроцессором, в котором объявлена специфичная для данной engine функция *ID*`err`.

Эта функция вызывается с двумя параметрами:

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

1. Код вызвавшей функции. Представляет собой имя функции, переведенное в верхний регистр с добавленным префиксом *ID*_F_.
   Например если в файле .ec описан идентификатор MYENGINE, то при вызове из функции **load_key** следует использовать идентификатор MYENGINE_F_LOAD_KEY.
2. Причина ошибки. Также константа, написанная большими буквами, описывающая причину ошибки с префиксом *ID*_R_.
   Например, в случае причины ошибки «файл не найден», можно использовать идентификатор MYENGINE_R_FILE_NOT_FOUND.

Допустимо использовать также коды причин ошибки, определенные в заголовочных файлах OpenSSL, включаемых в файлы вашего проекта, например коды с префиксом EVP_R_.

Вызов функции выдачи сообщения об ошибке будет выглядеть следующим образом:
MYENGINEerr(MYENGINE_F_LOAD_KEY,MYENGINE_R_FILE_NOT_FOUND)

После формирования сообщения об ошибке можно добавить к нему дополнительные данные с помощью функции **ERR_add_error_data**.

```
ERR_add_error_data(int num, ...)
```

Функция добавляет к сообщению об ошибке дополнительную информацию. Ее первый параметр задает число последующих аргументов, а последующие аргументы являются указателями на строки символов, добавляемых к сообщению.

Например: ERR_add_error_data(3,filename,":",strerr(errno)); добавит к вышеприведенному сообщению об ошибке имя файла и сообщение об ошибке от операционной системы, разделенные двоеточием.

После каждого добавления нового кода функции или нового кода причины, необходимо запустить препроцессор `mkerr.pl` для того чтобы он сгенерировал определения соответствующих макросов.

12.3 Реализация control-функции

Для реализации control функции необходимо написать фукнцию с прототипом

```
int func(ENGINE *e,int cmd, long intparam, void *ptrparam,
      void (*funcparam)(void));
```

Кроме этого следует создать массив структур ENGINE_CMD_DEFN (обычно в виде глобальной статической переменной) описывающих эти команды.

Структура ENGINE_CMD_DEFN имеет следующее определение:

typedef struct ENGINE_CMD_DEFN_st
{
unsigned int cmd_num; /* *The command number* */
const char *cmd_name; /* *The command name itself* */
const char *cmd_desc; /* *A short description of the command* */
unsigned int cmd_flags; /* *The input the command expects* */
} ENGINE_CMD_DEFN;

В поле cmd_num помещается код команды, который будет передан вашей control-функции в качестве параметра cmd, в поле cmd_name строковое имя команды, которое может быть использовано при вызове **ENGINE_ctrl_cmd** или **ENGINE_CTRL_cmd_string**, а также, если флаги в поле cmd_flags это позволяют, в конфигурационном файле openssl.cnf. В поле cmd_desc помещается человеко-читаемое краткое описание команды.

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

В поле `cmd_flags` возможны следующие флаги:

`ENGINE_CMD_FLAG_NUMERIC` — команда имеет числовой аргумент. Т.е. при вызове команды при чтении конфигурационного файла следует сконвертировать параметр директивы конфигурации в целое число и передать его в качестве `intparam`.

`ENGINE_CMD_FLAG_STRING` — команда имеет строковый аргумент. Если команда вызывается при чтении конфигурационного файла, в параметр `ptrparam` передается указатель на константную строку значения, указанного в этом файле.

`ENGINE_CMD_FLAG_NO_INPUT` — команда не имеет параметров.

`ENGINE_CMD_FLAG_INTERNAL` — Команда не предназначена для использования в конфигурационном файле, и может быть вызвана только из приложения с помощью функции **ENGINE_ctrl_cmd**. Команды `"ce write private key"` и `"ce wipe key container"`, если они поддерживаются вашей engine, должны иметь этот флаг.

Значения кодов команд менее `ENGINE_CMD_BASE` зарезервированы для использования OpenSSL и engine, поставляемыми в ее составе. Поэтому команды, специфичные для вашей engine, должны быть определены как (ENGINE_CMD_BASE+1), (ENGINE_CMD_BASE+2) и так далее.

control-функция должна проанализировать значения команды и выполнить соответствующие действия. Если переданная команда не поддерживается engine, функция должна возвратить −1. В случае успешной обработки команды следует возвратить 1, в случае ошибки — 0.

12.4 Реализация служебных функций инициализации engine

Для корректной регистрации engine должны быть реализованы следующие функции:

12.4.1 Функция bind

Обычно называется `bind_`*идентификатор engine* Выполняет регистрацию всех функций, предоставляемых данной engine.

Прототип:

```
int bind_something(ENGINE *e, const char *id);
```

Вызывается при загрузке engine в OpenSSL и получает указатель на структуру данных engine, которую требуется инициализировать и идентификатор engine, который необходимо проверить на совпадение с идентификатором текущей engine.

Возвращает 0 в случае ошибки и 1 в случае успеха.

Обычно эта функция декларируется как static.

Операции которые необходимо выполнить в этой функции:

1. Установить идентификатор с помощью **ENGINE_set_id**.
2. Установить имя engine с помощью **ENGINE_set_name**.
3. Зарегистрировать функцию загрузки ключа с помощью **ENGINE_set_load_privkey_function**.
4. Зарегистрировать control-функцию (выполняющую операции записи ключа, очистки ключевого контейнера и, возможно, какие-то другие специфичные для вашей engine операции) с помощью **ENGINE_set_ctrl_function**.

5. Зарегистрировать таблицу команд с помощью функции **ENGINE_set_cmd_defns**.
6. Загрузить набор сообщений об ошибках с помощью функции **ERR_load_XXX_strings**, сгенерированной препроцессором сообщений об ошибках.
7. Добавить engine в список активных с помощью **ENGINE_add**.

```
int ENGINE_set_id(ENGINE *e, const char *id);
```

Устанавливает в структуре ENGINE идентификатор.

```
int ENGINE_set_name(ENGINE *e const char *name);
```

Устанавливает человеко-читаемое наименование engine.

```
int ENGINE_set_ctrl_function(ENGINE *e,
       int (*f)(ENGINE *, int, long, void *, void (*f)(void)));
```

Устанавливает control-функцию.

```
int ENGINE_set_cmd_defns(ENGINE *e, const ENGINE_CMD_DEFN *defns);
```

Регистрирует таблицу команд.

```
int ENGINE_set_load_privkey_function(ENGINE *e,
       EVP_PKEY *(*load_priv_f)(ENGINE *, const char *, UI_METHOD *,
       void*));
```

Устанавливает функцию загрузки ключа.

```
int ENGINE_add(ENGINE *e);
```

Добавляет engine во внутренний список OpenSSL. Эта операция необходима для того, чтобы функция **ENGINE_load_private_key** нашла функцию загрузки, предоставляемую данной engine.

Для динамической загрузки engine после определения bind-функции используются макросы `IMPLEMENT_DYNAMIC_BIND_FN(bind_func)` и `IMPLEMENT_DYNAMIC_CHECK_FN()`.

Для возможности статической компиляции engine в приложение, использующее статическую libcrypto, необходимо также реализовать функцию

```
void ENGINE_load_engineid(void);
```

Эта функция должна выполнить следующие операции:

1. Создать экземпляр структуры данных `ENGINE` с помощью **ENGINE_new**.
2. Вызвать bind-функцию.

За исключением функций, сгенерированных макросами `IMPLEMENT_DYNAMIC_BIND_FN` и `IMPLEMENT_DYNAMIC_CHECK_FN` эта функция является единственной функцией в engine, доступ к котрой осуществляется по имени, а не по указателю, предварительно зарегистрированному с помощью функций API. Т.е. единственная функция, которая не может быть объявлена static.

13 Работа с цепочками сертификатов и хранилищем доверенных сертификатов УЦ

Для проверки валидности сертификатов в OpenSSL применяется структура `X509_STORE`. Эта структура поддерживает каталог сертификатов доверенных и промежуточных УЦ и их списков отзыва, позволяет выстраивать цепочки сертификатов от проверяемого до доверенного корневого.

Функции проверки подписи под сообщением PKCS7 (**PKCS7_verify**) и сертификатами (**X509_verify_cert**) получают эту структуру в качестве параметра и используют её для построения необходимых цепочек доверия.

Клиенты и серверы TLS могут также использовать явным образом сформированную структуру `X509_STORE` подключив её к контексту с помощью функции `SSL_CTX_set_cert_store`. Хотя в libssl существует ряд функций для упрощенной инициализации структуры `X509_STORE`.

13.1 Создание хранилища сертификатов

Для создания структуры `X509_STORE` используется функция

```
X509_STORE *X509_STORE_new(void);
```

После этого необходимо добавить в него метод поиска сертификатов, `X509_LOOKUP_METHOD`.

Добавление метода производится с помощью функции

```
X509_LOOKUP *X509_STORE_add_lookup(X509_STORE *v, X509_LOOKUP_METHOD
      *m);
```

Эта функция возвращает объект типа `X509_LOOKUP`, который может использоваться для настройки параметров поиска сертификатов.

OpenSSL предоставляет два стандартных метода поиска сертификатов, указатели на которые возвращаются функциями

```
X509_LOOKUP_METHOD  *X509_LOOKUP_file(void);
```

(см. раздел 13.6) и

```
X509_LOOKUP_METHOD *X508_LOOKUP_hash_dir(void);
```

(см. раздел 13.7).

Кроме того, приложение может реализовать собственный метод поиска сертификатов, позволяющий хранить доверенные корневые сертификаты и списки отзыва, например, в базе данных (см. раздел 13.8).

Кроме метода поиска сертификата может быть установлена функция обратного вызова при помощи макроса

```
X509_STORE_set_verify_cb_func(X509_STORE ctx,int (*func)(int ok,
      X509_STORE_CTX *ctx));
```

Описание поведения этой функции см в разделе 13.5.

Также можно установить ряд параметров поиска и верификации:

– Флаги, указывающие на набор выполняемых проверок (функция **X509_STORE_set_flags**).

– Область применения сертификата, которая требуется в данном случае (функция **X509_STORE_set_purpose**).
– Признак доверенности (функция **X509_STORE_set_trust**).

13.2 Параметры проверки сертификатов

```
int X509_STORE_set_flags(X509_STORE *ctx, unsigned long flags);
```

Позволяет задать следующие флаги:

`X509_V_FLAG_CB_ISSUER_CHECK` — передать проверки имени владельца и выпустившего УЦ в функцию обратного вызова
`X509_V_FLAG_USE_CHECK_TIME` — использовать указанное в параметрах хранилища время вместо текущего при проверках актуальности
`X509_V_FLAG_CRL_CHECK` — Выполнять проверку списков отзыва для проверяемого сертификата
`X509_V_FLAG_CRL_CHECK_ALL` — Выполнять проверку списков отзыва для всех сертификатов в цепочке.
`X509_V_FLAG_IGNORE_CRITICAL` — игнорировать неизвестные расширения, помеченные как критичные
`X509_V_FLAG_X509_STRICT` — использовать более жесткую проверку на соответствие сертификатов формату X.509
`X509_V_FLAG_ALLOW_PROXY_CERTS` — разрешить использование прокси-сертификатов.

```
int X509_STORE_set_purpose(X509_STORE *ctx, int purpose);
```

Устанавливает область применения сертификата, который следует признать валидным при проверке. Облать применения в X509 задается с помощью ASN.1 OID-а. В OpenSSL под областью применения сертификатов понимается некоторая комбинация расширений X509v3. Каждой области применения присвоен числовой идентификатор, символьный идентификатор (short name) и название.

В данную функцию передается числовой идентификатор.

Стандартные области применения задаются следующими константами:

`X509_PURPOSE_SSL_CLIENT` — Идентификация TLS-клиента
`X509_PURPOSE_SSL_SERVER` — Идентификация TLS-сервера
`X509_PURPOSE_NS_SSL_SERVER` — Идентификация SSL-сервера Netscape (устаревший тип)
`X509_PURPOSE_SMIME_SIGN` — Подпись сообщений SMIME
`X509_PURPOSE_SMIME_ENCRYPT` — Шифрование сообщений SMIME
`X509_PURPOSE_CRL_SIGN` — Подпись списков отзыва
`X509_PURPOSE_ANY` — Произвольная (явно не указанная) область применения
`X509_PURPOSE_OCSP_HELPER` — Идентификация сервера протокола online-валидации сертификатов (OCSP).

Приложение может также определять свои собственные области применения с помощью функции **X509_PURPOSE_add**.

Преобразование имени, полученного из командной строки в идентификатор можно произвести с помощью функции

```
int X509_PURPOSE_get_by_sname(char *sname)
```

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

Эта функция возвращает индекс области применения во внутреней таблице.

Получить структуру X509_PURPOSE по идентификатору можно с помощью

```
X509_PURPOSE *X509_PURPOSE_get0(int idx)
```

Эта функция, как и все функции, содержащие get0 в названии, возвращает указатель на существующий объект. Освобождать полученный объект не нужно.

Если известен только идентфикатор, нужно сначала преобразовать его в индекс с помощью

```
int X509_PURPOSE_get_by_id(int id)
```

Получение из объекта параметров производится с помощью функций

```
char *X509_PURPOSE_get0_name(X509_PURPOSE *xp);

char *X509_PURPOSE_get0_sname(X509_PURPOSE *xp);

int *X509_PURPOSE_get_id(X509_PURPOSE *xp);

int *X509_PURPOSE_get_trust(X509_PURPOSE *xp);

int X509_STORE_set_depth(X509_STORE *store, int depth);
```

Устанавливает максимальную длину цепочки, которая должна привести к доверенному корневом сертификату, для того, чтобы сертификат был признан валидным.

Если валидны только сертификаты, выпущенные непосредственно корневым УЦ, длину цепочки следует установить в 2.

Если возможно использование одного уровня промежуточных УЦ, то в 3 и так далее.

13.3 Процедура проверки сертификата

Процедура проверки сертификатов с помощью хранилища сертификатов выглядит следующим образом:

1. На базе структуры X509_STORE с помощью функции **X509_STORE_CTX_init** создается структура проверки индивидуального сертификата X509_STORE_CTX.
2. Если необходимо, для неё устанавливаются параметры, аналогичные описанным выше параметрам хранилища. Эти параметры будут влиять только на проверку данного сертификата. Не установленные на данном этапе параметры наследуются от хранилища.
3. Вызывается функция **X509_verify_cert**.
4. Если она вернула нулевое значение, следует проанализировать значение поля error структуры X509_STORE_CTX.
5. струкутура X509_STORE_CTX очищается посредством **X509_STORE_CTX_cleanup** (если она была размещена статически) или освобождается посредством **X509_STORE_CTX_free**.

```
X509_STORE_CTX *X509_STORE_CTX_new();
```

Размещает динамически в памяти структуру X509_STORE_CTX и возвращает указатель на неё. Структуру требуется в дальнейшем инициализировать с помощью **X509_STORE_CTX_init** и в конце концов освободить с помощью **X509_STORE_CTX_free**.

```
int X509_STORE_CTX_init(X509_STORE_CTX *ctx, X509_STORE
```

```
*store, X509 *cert, STACK_OF(X509) *chain);
```

Инициализирует структуру проверки сертификата. Параметр ***ctx*** может быть либо адресом переменной типа `X509_STORE_CTX`, либо указателем, инициализированным с помощью **X509_STORE_CTX_new**.

Параметр ***store*** содержит инициализированное хранилище корневых сертификатов.

Параметр ***cert*** — проверяемый сертификат. Может быть `NULL`, если предполагается использовать данную структуру не для проверки сертификата, а для поиска необходимого доверенного сертификата (см. раздел 13.4.

Параметр ***chain*** — дополнительный набор сертификатов, которые можно использовать при проверке. Сертификаты из этого набора считаются недоверенными и проверяются так же, как и проверяемый сертификат.

Например, это может быть цепочка сертификатов промежуточных УЦ, включенная в сообщение или выданная другой стороной TLS-соединения.

```
int X509_verify_cert(X509_STORE_CTX *ctx)
```

Выполняет проверку сертификата с помощью ранее созданной структуры `X509_STORE_CTX`. Возвращает ненулевое значение, если проверка завершилась успешно, и нулевое если произошла ошибка.

```
void X509_STORE_CTX_cleanup(X509_STORE_CTX *ctx)
```

Очищает структуру `X509_STORE_CTX`. Явный вызов требуется в случае статической структуры. Не освобождает хранилище, проверяемый сертификат и дополнительные сертификаты.

```
void X509_STORE_CTX_free(X509_STORE_CTX *ctx)
```

Освобождает структуру `X509_STORE_CTX`, размещенную с помощью функции **X509_STORE_CTX_new**. Перед освобождением памяти, занимаемой собственно структурой, вызывает **X509_STORE_CTX_cleanup**.

13.4 Процедура поиска нужного сертификата

В некоторых случаях бывает нужно получить конкретный сертификат по его subject, например для проверки подписи под CRL, так как структура `X509_STORE_CTX` позволяет проверять только сертификаты, но не CRL.

Для этой цели используется функция

```
int X509_STORE_get_by_subject(X509_STORE_CTX *ctx, int type,
      X509_NAME *name, X509_OBJECT *ret);
```

Параметр `ctx` содержит подготовленную структуру `X509_STORE_CTX`.

Параметр `type` тип требуемого объекта, т.е. одну из констант `X509_LU_X509` или `X509_LU_CRL`.

Параметр `name` содержит структуру `X509_NAME`, идентифицирующую запрашиваемый объект. Например, эта структура может быть получена посредством вызова функции **X509_get_name** или **X509_CRL_get_issuer**.

Результат поиска помещается в структуру `X509_OBJECT`, указатель на которую передан в параметре `ret`.

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

```
struct x509_object_st
    {
    int type;
    union
        {
        char *ptr;
        X509 *x509;
        X509_CRL *crl;
        EVP_PKEY *pkey;
        } data;
    } X509_OBJECT;
```

Поле `type` устанавливается в соответствии с типом найденного объекта.
Функция возвращает 1, если подходящий объект найден, и 0 в противном случае.

13.5 Функция обратного вызова проверки сертификатов

```
X509_STORE_set_verify_cb_func(X509_STORE ctx,int (*func)(int ok,
      X509_STORE_CTX *ctx));
```

Функция обратного вызова получает два аргумента:

`ok` — имеет значение 1 если стандартная процедура проверки сертификата (включая построение цепочки доверия) завершилась успешно, и 0, если нет.
`ctx` — объект `X509_STORE_CTX` libcrypto, использованный для проверки сертификата (включает и ссылку на проверяемый сертификат).

Эта функция предназначена для выполнения дополнительных, нестандартных проверок сертификатов.

В случае, если цепочка доверия включает сертификаты промежуточных CA, функция обратного вызова вызывается для каждого из сертификатов в цепочке.

В случае, если параметр ***ok*** имеет значение 0, код ошибки, вызвавшей такое значение, находится в поле `ctx->error` и представляет собой одну из констант `X509_V_ERR_*`, определенных в файле `x509_vfy.h`.

Функция обратного вызова может, проанализировав код ошибки, принять решение, что данная ошибка несущественна, и возвратить единицу, что является указанием на то что ошибку следует проигнорировать и продолжить проверку.

С другой стороны, если полученное значение ok равно 1, функция может провести дополнительные проверки и вернуть 0, что является основанием отвергнуть сертификат, даже если по результатам стандартных проверок он признан корректным. В этом случае после окончания проверки в поле `error` контекста проверки будет содержаться значение `X509_V_ERR_APPLICATION_VERIFICATION`

13.6 Метод поиска сертификатов в файле

Метод поиска сертификатов в файле, указатель на который возвращается функцией **X509_LOOKUP_file**, используется для поиска сертификатов в текстовом файле, содержащем последовательность сертификатов и CRL в формате PEM.

Имя файла может быть задано с помощью макроса

```
X509_LOOKUP_load_file(X509_LOOKUP *l,char *filename,int
      filetype)
```

В качестве значения параметра ***filetype*** при загрузке явно указанного файла следует всегда передавать константу `X509_FILETYPE_PEM`, определенную в файле `x509.h`.

С помощью этой же функции может быть также загружен стандартный файл сертификатов. Для этого нужно вызвать эту функцию с `NULL` в качестве имени файла и типом `X509_FILETYPE_DEFAULT`.

Стандартный файл сертификатов располагается в корневой директории openssl (заданной при компиляции и возвращаемой командой `openssl version -d`) и имеет имя `cert.pem`.

Если установлена переменная среды `SSL_CERT_FILE` то в качестве стандартного используется файл, имя которого содержится в этой переменной.

Недостатком этого метода является то, что все сертификаты, содержащиеся в файле, загружаются в память процесса сразу же при загрузке файла. Поэтому если необходимо использовать большое количество доверенных сертификатов, следует использвоать метод поиска сертификатов в хэшированной директории.

13.7 Метод поиска сертификатов в директории

Метод поиска сертификатов в директории, указатель на который возвращается функцией **X509_LOOKUP_hash_dir** производит поиск сертификата в хранилище, представляющем собой директорию в файловой системе, в которой содержатся файлы сертификатов и списков отзыва в формате PEM по одному сертификату или CRL в файле.

Имена файлов должны представлять собой хэш-суммы subject-а сертификата или crl, сгенерированые опцией -hash команды openssl x509 или openssl crl соответственно. К именам файлов CRL следует добавить маленькую латинскую букву 'r'. Если в хранилище имеется несколько сертификатов или CRL с одинаковой хэш-суммой, то к имени файла добавляется точка и порядковый номер, начиная с 0.

В комплект OpenSSL входит утилита `c_rehash`, которая создает в директории с файлами сертификатов и CRL символические ссылки с именами, понимаемыми данным методом поиска.

Добавление директории в список поиска производится макросом

```
X509_LOOKUP_add_dir(X509_LOOKUP ctx,char *dir,int type)
```

Вызов с именем директории, равным `NULL` и значением `type`, равным `X509_FILETYPE_DEFAULT`, добавляет стандартную директорию (поддиректорию certs директории OpenSSL или директорию, указанную в переменной среды `SSL_CERT_DIR`).

Стандартная директория всегда должна содержать сертификаты и CRL в формате PEM. При явном добавлении директории можно использовать сертификаты в формате DER, при этом следует указывать в параметре type значение `X509_FILETYPE_ASN1`.

13.8 Создание собственных методов поиска сертификатов

Для создания собственных методов поиска сертификатов следует определить фунции, реализующие функциональность метода и поместить их в соответствующие поля структуры `X509_LOOKUP_METHOD`. Эта структура обычно определяется как статическая. Поля, соответствующие функциям, не реализованным в данном методе, должны быть инициализированы в `NULL`.

Структура `X509_LOOKUP_METHOD` содержит следующие поля:

```
int new_item (X509_LOOKUP *ctx)
```

Вызывается при создании соответствующего X509_LOOKUP. Создает специфическую для данного метода структуру данных и размещает указатель на неё в поле meth_data переданной структуры X509_LOOKUP. Может отсутствовать, если данный метод не нуждается в специфической структуре данных, например, если этот метод просто загружает сертификаты в стандартный кэш X509_STORE, подобно стандартному методу поиска в файле.

```
void free(X509_LOOKUP *ctx)
```

Освобождает структуру, размещенную функцией **new_item**. Должна присутствовать, если присутствует функция **new_item**.

```
int init (X509_LOOKUP *ctx)
```

Выполняет необходимые действия по инициализации метода поиска (например, открывает соединение с базой данных). Вызывается при явном вызове приложением функции **X509_LOOKUP_init**.

```
int shutdown (X509_LOOKUP *ctx)
```

Выполняет необходимые действия по завершению работы метода (например, закрывает соединение с базой данных). Вызывается при явном вызове приложением функции **X509_LOOKUP_shutown)**

```
int ctrl(X509_LOOKUP *ctx,int cmd, const char *argc, long
      argl, char **ret)
```

Используется для специфичного для данного метода управления объектом X509_LOOKUP. Первый параметр — код команды, второй и третий используются для передачи специфических данных в функцию, четвертый — для возвращения данных из функции.

Может модифицировать данные, специфичные для метода или добавлять сертификаты в кэш структуры X509_STORE, указатель на которую содержится в поле store_ctx структуры X509_LOOKUP с помощью функций **X509_STORE_add_cert** и **X509_STORE_add_crl**.

```
int get_by_subject(X509_LOOKUP *ctx, int type, X509_NAME
      *name, X509_OBJECT *ret);
```

Выполняет операцию поиска сертификата или CRL по subject.

Переменная type может принимать значения X509_LU_X509 (сертификат) и X509_LU_CRL (CRL).

Переменная name содержит требуемый subject.

Указатель на найденный сертификат (динамически размещенный в памяти в виде структуры X509) или crl (динамически размещенный в виде структуры X509_CRL) должен быть размещен в поле data структуры, указатель на которую передан в параметре ret. Поле type этой структуры должно быть заполнено значением type.

Функция должна возвращать 1, если поиск завершился успешно и 0, если произошла ошибка.

```
int get_by_issuer_serial(X509_LOOKUP *ctx, int type,
      X509_NAME *name, ASN1_INTEGER *serial, X509_OBECT *ret)
```

Выполняет поиск сертификата по серийному номеру и имени выпустившего УЦ. Не используется при процедуре построения цепочек доверия. Вызывается только при явном вызове функции **X509_LOOKUP_by_issuer_serial** и должна быть реализована только если приложение нуждается в данной функциональности.

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

В стандартных методах не реализвана.

```
int get_by_fingerprint(X509_LOOKUP *ctx, int type, unsigned
      char *bytes, int len, X509_OBJECT *ret);
```

Осуществляет поиск сертификата по цифровому отпечатку (хэш-сумме) открытого ключа. Используется только при явном вызове **X509_LOOKUP_by_fingerprint**.

В стандартных методах не реализована.

```
int get_by_alias(X509_LOOKUP *ctx, int type, char *str, int
      len, X509_OBJECT *ret);
```

Поиск сертификата по строковому алиасу. Используется только при явном вызове **X509_LOOKUP_by_alias**.

В стандартных методах не реализована.

14 Работа с протоколом онлайн проверки статус сертификатов

Протокол онлайн проверки статуса сертификатов (On-line Certificate Status Protocol, OCSP) описанный в RFC 2560 позволяет приложению запросить текущий статус сертификата непосредственно в момент проверки.

OpenSSL содержит необходимую функциональность как для реализации клиента OCSP, так и для реализации сервера. RFC 2560 позволяет использовать различные виды сетевых протоколов для взаимодействия с OCSP-сервером, хотя наиболее распространенным является использование проткола HTTP.

В данном документе мы описываем только формирование и анализ криптографически защищенных сообщений протокола, не касаясь вопросов передачи и приема их по сети. Реализация транспорта, как правило, выполняется средствами, не входящими в состав libcrypto.

14.1 Формирование OCSP-запроса

```
OCSP_REQUEST* OCSP_REQUEST_new()
```

Создает пустую струкутур `OCSP_REQUEST`. В эту структуру необходимо добавить информацию об одном или нескольких сертификатах, статус которых запрашивается. Кроме этого, структуру можно подписать электронной подписью запрашивающего, если это необходимо.

```
OCSP_REQUEST_free(OCSP_REQUEST *req)
```

Освобождает предварительно размещенную структуру `OCSP_REQUEST`.

Добавление в запрос информации о сертификате производится с помощью функции

```
OCSP_ONEREQ *OCSP_request_add0_id(OCSP_REQUEST *req,
      OCSP_CERTID *cid)
```

Эта функция получает структуру `OCSP_REQUEST` и структуру идентификации сертификата `OCSP_CERTID` и возвращает указатель на фрагмент модифицированной структуры `OCSP_REQUEST` куда помещена информация о сертификате или `NULL` если произошла ошибка.

Если информация о сертификате была успешно помещена в запрос, освобождать структуру `OCSP_CERTID` не требуется. Она будет освобождена при освобождении структуры запроса.

Для формирования струкутры идентификации сертификата обычно используется функция:

```
OCSP_CERT_ID *OCSP_cert_to_id(const EVP_MD *dgst,X509 *cert,
      X509 *issuer);
```

Эта функция получает указатель на алгоритм хэширования, который необходимо использовать (для сертификатов с алгоритмами ГОСТ следует использовать алгоритм ГОСТ Р 34.11-94. Указатель на соответствующую структуру можно получить с помощью `EVP_get_digestbyname("md_gost94")`), указатель на проверяемый сертификат и указатель на сертификат выпустившего удостоверяющего центра.

Функция возвращает структуру `OCSP_CERTID` или `NULL` если произошла ошибка.

Структура `OCSP_CERTID` может быть сформирована и при отсутствии запрашиваемого сертификата. Для её формирования достаточно иметь сертификат выпустившего удостверяющего центра и серийный номер сертификата.

```
OCSP_CERTID *OCSP_cert_id_new(const EVP_MD *dgst, X509_NAME
      *issuerName,
```

```
ASN1_BIT_STRING* issuerKey, ASN1_INTEGER *serialNumber);
```

Позволяет сформировать OCSP_CERTID по серийному номеру, наименованию и открытому ключу УЦ. Получение наименования и открытого ключа из сертификата удостоверяющего центра производится с помощью информационных функций, описанных в разделе 2.2.

Если какая-нибудь из структур, передаваемыъ в данную функцию не является частью другой структуры (например X509), она должна быть явным образом освобождена после вызова.

RFC 2560 предусматривает необязательное добавление в сертификат случайного числа (nonce) для предотвращения атак с повторением данных. Для добавления nonce используется функция:

```
int OCSP_request_add1_nonce(OCSP_REQUEST *req, unsigned char
      *val, int len);
```

Где val случайная строка байтов, и len её длина. Если в качестве val передано NULL будет сгенерирована случайная nonce. Если в качестве длины передано −1, в качестве длины будет использована константа OCSP_DEFAULT_NONCE_LENGTH. Проверка nonce впоследствии осуществляется путем сравнения структуры запроса со структурой полученного ответа с помощью функции **OCSP_check_nonce** (см раздел 14.2).

Необязательная подпись под запросом может быть выработана с помощью функции:

```
int OCSP_request_sign(OCSP_REQUEST *req,
      X509 *signer, EVP_PKEY *key, const EVP_MD *dgst, STACK_OF(X509)
      *certs, unsigned long flags);
```

Эта функция получает запрос, сертификат и секретный ключ подписывающего, указатель на алгоритм хэширования, используемый при подписи, набор дополнительных сертификатов, включаемых в запрос (например, сертификатов промежуточных удостоверяющих центров, необходимых для проверки подписи OCSP-сервером) и поле флагов.

В настоящий момент поддерживается единственный флаг OCSP_NOCERTS, указывающий что сертификат подписавшего и любые сертификаты, содержащиеся в параметре certs не следует включать в запрос.

Если параметр dgst имеет значение NULL для выработки подписи будет использован алгоритм хэширования, соответствующий алгоритму ключа подписи.

После формирования и, возможно подписи, запрос должен быть сериализован с помощью функции

i2d_OCSP_REQUEST или **i2d_OCSP_REQUEST_bio**, аналогичный прочим i2d-функциям.

14.2 Анализ OCSP-ответа

Разбор полученного от сервера OCSP-ответа производится с помощью функций **d2i_OCSP_RESPONSE**, **d2i_OCSP_RESPONSE_bio**, **d2i_OCSP_RESPONSE_fp**. Полученная структура OCSP_RESPONSE должна быть впоследствии освобождена вызовом **OCSP_RESPONSE_free**. При этом будут также освобождены все данные, на которые указывают указатели типов OCSP_BASICRES, OCSP_SINGLERESP и ASN1_GENERALIZEDTIME получаемые с помощью функций, описанных в данном разделе.

```
int OCSP_response_status(OCSP_RESPONSE *resp)
```

Возвращает статус — одну из констант:

OCSP_RESPONSE_STATUS_SUCCESSFUL — Запрос обработан успешно
OCSP_RESPONSE_STATUS_MALFORMEDREQUEST — Ошибка в формате запроса
OCSP_RESPONSE_STATUS_INTERNALERROR — Внутренняя ошибка сервера
OCSP_RESPONSE_STATUS_TRYLATER — Необходимо повторить запрос позже
OCSP_RESPONSE_STATUS_SIGNREQUIRED — В данном случае требуется, чтобы запрос был подписан
OCSP_RESPONSE_STATUS_UNAUTHORIZED — Запрос неавторизован

Получить строку, описывающую значение статуса в человеко-читаемом виде (на английском языке) можно с помощью функции

```
const char *OCSP_response_status_str(int status)
```

Дальнейший анализ ответа производится путем извлечения из структуры OCSP_RESPONSE структуры OCSP_BASICRESP

```
OCSP_BASICRESP *OCSP_response_get1_basic(OCSP_RESPONSE *response)
```

Функция возвращает указатель на структуру OCSP_BASICRESP или NULL в случае ошибки.

```
int OCSP_check_nonce(OCSP_REQUEST *req, OCSP_BASICRESP *resp)
```

Проверяет что расширение nonce в ответе соответствует аналогичному расширению в отправленном запросе. Возвращает положительное значение в случае успеха, и нулевое или отрицательное в случае ошибки. Значение -1 соответствует отсутствию расширения nonce в ответе. Эта ситуация не всегда должна рассматриваться как ошибка.

```
int OCSP_basic_verify(OCSP_BASIC_RESP *resp, STACK_OF(X509)
      *certs, X509_STORE *st, unsigned long flags);
```

Получает хранилище сертификатов удостоверяющих центров, набор дополнительных сертификатов, которым следует доверять и набор флагов.

Поддерживаются следующией флаги:

OCSP_NOINTERN — Игнорировать сертификаты, содержащиеся в ответе сервера, использовать только сертификаты, доступные в хранилище или списке явным образом переданных сертификатов.
OCSP_NOSIGS — Не проверять подпись под ответом (использовать только для целей тестирования)
OCSP_NOCHAIN — Игнорировать цепочку доверия, переданную сервером.
OCSP_NOVERIFY — Не проверять сертификат открытого ключа, на котором выработана подпись
OCSP_NOEXPLICIT — Не проверять наличие расширение явного указания доверия в сертификате сервера
OCSP_NOCHECKS — Не выполнять проверок на соответствие области применения сертификата подписи.
OCSP_TRUSTOTHER — Если открытый ключ подписи сообщения обнаружен в наборе сертификатов, переданных в параметре certs не выполнять дальнейших проверок этого сертификата.

Функция выполняет проверку подписи под ответом сервера и соответствие сертификата, использованного при выработке этой подписи, требованиям, сформулированным в RFC 2560,

а также соответствие идентификатора ответчика, содержащегося в ответе, сертификату использованному для подписи ответа.

Узнать статус конкретного сертификата можно с помощью функции

```
int OCSP_resp_find_status(OCSP_BASICRESP *resp, OCSP_CERTID *id,
       int *status,
       int *reason,
       ASN1_GENERALIZEDTIME **revtime,
       ASN1_GENERALIZEDTIME **thisupd,
       ASN1_GENERALIZEDTIME **nextupd);
```

Эта функция получает указатель на структуру `OCSP_BASICRESP` и указатель на идентификатор сертификата `OCSP_CERTID` и заполняет указатели `status`, `reason`, `revtime`,`thisupd`, `nextupd`.

Функция возвращает положительное число, если соответствующий сертификат найден, 0 если не найден и отрицательное число в случае ошибки.

Если какая-либо информация возвращаемая через параметры, не требуется, в качестве соответствующего параметра можно передать `NULL`.

В переменную `status` помещается одна из констант `V_OCSP_CERTSTATUS_GOOD`, `V_OCSP_CERTSTATUS_REVOKED` или `V_OCSP_CERTSTATUS_UNKNOWN`.

Если статус равен `V_OCSP_CERTSTATUS_REVOKED`, то в переменную `reason` помещается код причины отзыва:

`OCSP_REVOKED_STATUS_NOSTATUS` — Нет причины (статус сертификата good или unknown)
`OCSP_REVOKED_STATUS_UNSPECIFIED` — Причина отзыва не указана
`OCSP_REVOKED_STATUS_KEYCOMPROMISE` — Компрометация ключа
`OCSP_REVOKED_STATUS_CACOMPROMISE` — Компрометация ключа удостоверяющего центра
`OCSP_REVOKED_STATUS_AFFILIATIONCHANGED` — Изменилось отношение владельца ключа с организацией (уволен, переведен на другую должность требующую сертификата с другими свойствами)
`OCSP_REVOKED_STATUS_SUPERSEDED` — Заменен на более новый
`OCSP_REVOKED_STATUS_CESSATIONOFOPERATION` — Организция прекратила деятельность
`OCSP_REVOKED_STATUS_CERTIFICATEHOLD` — Сертификат временно заблокирован
`OCSP_REVOKED_STATUS_REMOVEFROMCRL` — Сертификат удален из списка отзыва (восстановлен в правах).

В переменную `revtime` помещается указатель на структуру `ASN1_GENERALIZEDTIME`, содержащую время отзыва сертификата.

В переменные `thisupd` и `nextupd` записывается информация о времени обновления списка отзыва удостоверяющего центра, выпустившего данный сертификат.

Кроме этой функции приложение может использовать последовательный перебор статусов содержащихся в ответе.

Получить количество статусов сертификатов, содержащихся в ответе, можно с помощью функции

```
int OCSP_resp_count(OCSP_BASICRESP *resp)
```

Каждому из этих статусов соответствует структура `OCSP_SINGLE_RESP`.

Её можно получить либо по порядковому номеру с помощью функции

```
OCSP_SINGLERESP *OCSP_resp_get0(OCSP_BASICRESP *bs, int
        idx);
```

Где idx целое число от 0 до количества статусов в ответе.

Получить информацию из структуры OCSP_SINGLERSP можно с помощью функции

```
int OCSP_single_get0_status(OCSP_SINGLERESP *single, int *reason,
        ASN1_GENERALIZEDTIME **revtime,
        ASN1_GENERALIZEDTIME **thisupd,
        ASN1_GENERALIZEDTIME
        **nextupd);
```

Эта функция возвращает значение статуса сертификата V_OCSP_CERTSTATUS_GOOD, V_OCSP_CERTSTATUS_REVOKED или V_OCSP_CERTSTATUS_UNKNOWN и заполняет переданные указатели аналогично функции **OCSP_find_status**.

В случае необходимости получения идентификатора сертификата из структуры OCSP_SINGLERESP необходимо обращаться непосредственно к полю certId этой структуры. API функция для этой цели не предоставляется.

Сравнение идентификаторов сертификатов может быть выполнено с помощью функции

```
int OCSP_id_cmp(OCSP_CERTID *a, OCSP_CERTID *b);
```

Эта функция возвращает 0 если два указанных идентификатора сертификата совпадают, и ненулевое значение, если они различаются.

14.3 API для реализации сервера OCSP

При обработке запроса OCSP-сервер должен произвести разбор запроса с помощью функций **d2i_OCSP_REQUEST**, **d2i_OCSP_REQUEST_bio** или **d2i_OCSP_REQUEST_fp** и сформировать ответ. После окончания формирования работы структуры OCSP_REQUEST и OCSP_RESPONSE должны быть освобождены с помощью **OCSP_REQUEST_free** и OCSP_RESPONSE_free соответственно.

```
OCSP_BASESP *OCSP_BASESP_new()
```

Создает структуру OCSP_BASERESP в которой формируется ответ. После создания OCSP_RESPONSE с помощью **OCSP_RESPONSE_create** эта структура должна быть освобождена с помощью **OCSP_BASERESP_free**.

Перебор запросов статуса, содержащихся в запросе осуществляется с помощью функций:

```
int CSP_request_onereq_count(OCSP_REQUEST *req)
```

возвращает количество запросов статуса в запросе.

```
OCSP_ONEREQ *OCSP_request_onereq_get0(OCSP_REQUEST *req,int
        idx);
```

Возвращает индивидуальный запрос с указанным порядковым номером

```
OCSP_CERTID  *OCSP_onereq_get0_id(OCSP_ONEREQ *r)
```

Возвращает индентификатор сертификата, содержащийся в запросе.

```
int OCSP_id_get0_info(ASN1_OCTET_STRING **piNameHash,
        ASN1_OBJECT **pmd,
        ASN1_OCTET_STRING **pikeyHash,
```

```
        ASN1_INTEGER **pserial, OCSP_CERTID *cid);
```

Возвращает информацию о идентификаторе сертификата, в частности OID использованного алгоритма хеширования и серийный номер сертификата.

Для определения соответствующего сертификата удостоверяющего центра, OCSP-сервер должен подготовить структуры `OCSP_CERTID` с использованием того же алгоритма хэширования с помощью функции **OCSP_cert_to_id** используя `NULL` в качестве идентифицируемого сертификата, и сравнить их с содержащимся в запросе запроса с помощью функции

```
int OCSP_id_issuer_cmp(OCSP_CERTID *a, OCSP_CERTID *b)
```

Эта функция возвращает 0 если в двух указанных идентификаторах сертификата совпадает информация об удостоверяющем центре, и ненулевое значение, если она различается.

После того как определен сертификат CA и серийный номер, сервер может определить статус сертификата. (Если сертификат CA не найден, статус сертификата устанавливается в `V_OCSP_CERTSTATUS_UNKNOWN`).

Информация о статусе сертификата добавляется в структуру `OCSP_BASIC_RESP` с помощью функции

```
OCSP_SINGLERESP *OCSP_basic_add1_status(OCSP_BASICRESP *rsp,
        OCSP_CERTID *cid,
        int status, int reason,
        ASN1_TIME *revtime,
        ASN1_TIME *thisupd,
        ASN1_TIME *nextupd);
```

Параметры `status` и `reason` должны содержать те же значения, что и соответствующие параметры, возвращаемые функцией **OCSP_single_get0_status**. Параметр `revtime` заполняется только в случае если сертификат отозван. Параметры `thisupd` и `nextupd` заполняются всегда.

Если в запросе содержится расширение nonce, оно должно быть скопировано в ответ с помошью функции

```
int OCSP_copy_nonce(OCSP_BASICRESP *resp, OCSP_REQUEST *req);
```

После этого ответ подписывается с помощью фукнции:

```
int OCSP_basic_sign(OCSP_BASICRESP *brsp,
        X509 *signer, EVP_PKEY *key, const EVP_MD *dgst,
        STACK_OF(X509) *certs, unsigned long flags);
```

аналогичной **OCSP_request_sign**. Эта функция поддерживает следующие флаги:

`OCSP_NOCERTS` — Не включать в ответ сертификат ключа, которым выработана подпись и сертификаты, указыанные в параметре `certs`.

`OCSP_RESPID_KEY` — Использовать в качестве идентификатора сервера, помещаемого в подписанную часть запроса не subject сертификата, а SHA-1 хэш открытого ключа. Использовать этот режим при работе с сертификатами ГОСТ не следует, так как тут используется нестандартный для России алгоритм хэширования, и использование именно этого алгоритма обязательно согласно RFC 2560.

и преобразуется в `OCSP_RESPONSE` с помощью функции

```
OCSP_RESPONSE *OCSP_RESPONCE_create(int status, OCSP_BASICRESP *bs);
```

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

При создании ответа со статусом, отличным от `OCSP_RESPONSE_STATUS_SUCCESSFUL` в качестве параметра `bs` передается `NULL`.

Затем `OCSP_RESPONSE` сериализуется с помощью **i2d_OCSP_RESPONSE** или **i2d_OCSP_RESPONSE_bio** и отправляется клиенту.

15 Работа с метками времени

15.1 Формирование запроса к timestamping authority

`TS_REQ* TS_REQ_new()`

Формирует пустую структуру запроса на метку времени.

`int TS_REQ_set_version(TS_REQ *req,int version)`

Устанавливает номер версии.

```
int TS_REQ_set_msg_impirint(TS_REQ *req, TS_MSG_IMPRINT
      *imprint)
```

Устанавливает отпечаток сообщения

`TS_MSG_IMPRINT *TS_MSG_IMPRINT_new()`

Создает струкуруру `TS_MSG_IMPRINT`.

```
int TS_MSG_IMPRINT_set_algo(TS_MSG_IMPRINT *imprint,
      X509_ALGOR *algo)
```

Устанавливает в структуру `TS_MSG_IMPRINT` информацию об алгоритме хэширования.

```
int TS_MSG_IMPRINT_set_msg(TS_MSG_IMPRINT *imprint, unsigned
      char *data, int len)
```

Устанавливает в структуру `TS_MSG_IMPRINT` значение хэш-суммы сообщения (см раздел 7)

```
int TS_MSG_REQ_set_policy_id(TS_REQ *req,ASN1_OBJECT
      *policy)
```

Устанавливает идентификатор запрашиваемой полиси.

`int TS_REQ_set_nonce(TS_REQ *req,ASN1_INTEGER *nonce)`

Устанавливает одноразовый идентификатор запроса (nonce). Для создания идентификатора запроса нужно сформировать 64-битное случайное число, например, получив 8 случайных байт с помощью функции **RAND_bytes** и преобразовать его в структуру `ASN1_INTEGER`, например используя функцию **BN_TO_ASN1_INTEGER**.

`int TS_REQ_set_cert_req(TS_REQ *req,int certflag)`

Устанавливает флаг запроса сертифката (1 - запрашивать сертификат, 0 - не запрашивать).

`int i2d_TS_REQ_bio(BIO *b,TS_REQ *query)`

Сериализует запрос в `BIO b`.

`void TS_REQ_free(TS_REQ *req`

Освобождает структуру `TS_REQ`.

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

15.2 Анализ и визуализация запросов

```
TS_REQ *d2i_TS_REQ_bio(BIO *b, TS_REQ* req)
```

Читает из потока b структуру TS_REQ.

```
int TS_REQ_print_bio(BIO *out,TS_REQ *req)
```

Выводить в BIO текстовое представление запроса.

15.3 Формирование метки времени

Для формированиия метки времени необходимо сформировать контекст timestamping authority, установив необходимые параметры.

После этого можно воспользоваться высокоуровневой функцией

```
TS_RESP *TS_RESP_create_response(TS_RESP_CTX *ctx, BIO
      *req_bio)
```

Функция читает из req_bio запрос, анализирует его в соответствии с выставленными параметрами контекста и генерирует ответ, используя для подписи сертификат и секретный ключ, установленный в контекст.

Функция может быть вызвана много раз с одним и тем же контекстом.

15.4 Создание и конфигурирование контекста timestamping authority

```
TS_RESP_CTX *TS_RESP_CTX_new()
```

Создает структуру TS_RESP_CTX.

```
void TS_RESP_CTX_free(TS_RESP_CTX *ctx)
```

Освобождает стркутуру TS_RESP_CTX.

```
int TS_RESP_CTX_set_signer_cert(TS_RESP_CTX *ctx, X509
      *signer)
```

Устанавливает сертификат, которым подписываются метки времени. Обязательно.

```
int TS_RESP_CTX_set_signer_key(TS_RESP_CTX *ctx, EVP_PKEY
      *key)
```

Устанавливает секретный ключ, которым подписываются метки времени. Обязательно.

```
int TS_RESP_CTX_set_def_policy(TS_RESP_CTX *ctx, ASN1_OBJECT
      *def_policy)
```

Устанавливает политику по умолчанию. Обязательно.

```
int TS_RESP_CTX_set_certs(TS_RESP_CTX *ctx, STACK_OF(X509)
      *certs)
```

Устанавливает цепочку дополнительных сертификатов (например сертификатов промежуточных УЦ, включаемую в метки времени при наличии флага запроса сертификата.

```
int TS_RESP_CTX_add_policy(TS_RESP_CTX *ctx, ASN1_OBJECT
      *policy)
```

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

Устанавливает дополнительную разрешенную политику (по умолчанию принимаются только запросы не запрашивающие политику явно, или запросы, запрашивающие политику, совпадающую с политикой по умолчанию).

```
int TS_RESP_CTX_add_md(TS_RESP_CTX *ctx, const EVP_MD *md)
```

Добавляет алгоритм хэширования в список допустимых. Должна быть вызвана как минимум один раз. По умолчанию список допустимых алгоритмов хэширования пуст.

```
int TS_RESP_CTX_set_accuracy(TS_RESP_CTX *ctx,
      int secs, int millis, int micros)
```

Устанавливает точность метки времени.

```
int TS_RESP_CTX_set_clock_precision_digits(TS_RESP_CTX *ctx,
      unsigned clock_precision_digits)
```

Устанавливает число значимых цифр после запятой в метке времени (0 точность до секунды, 3 до миллисекунды). Максимальное допустимое число знаков после запятой определяется константой `TS_MAX_CLOCK_PRECISION_DIGITS`, равной 6 (микросекунды).

```
void TS_RESP_CTX_add_flags(TS_RESP_CTX *ctx, int flags)
```

Устанавливает флаги. Допустимые флаги:

`TS_TSA_NAME` — Не включать наименование timestamping autrority в метку времени
`TS_ORDERING` — Установить флаг ordering в true.
`TS_ESS_CERT_ID_CHAIN` — Включать сертификат подписи и прочие сертификаты установленные в контекст в атрибут ESS signing certificate. (по умолчанию включается только сертификат подписи).

```
void TS_RESP_CTX_set_serial_cb(TS_RESP_CTX *ctx, ASN1_INTEGER
      *(*callback)(TS_RESP_CTX *ctx, void *data), void *data)
```

Устанавливает функцию обратного вызова, возвращающую серийный номер для создаваемой метки времени. Параметр `data` будет передан в неизменном виде в функцию.

```
void TS_RESP_CTX_set_time_cb(TS_RESP_CTX *ctx, int
      (*cb)(TS_RESP_CTX *ctx, void *data,long *sec,long *usec),
      void *data)
```

Устанавливает коллбэк функцию, возвращающую время в виде двух переменных — секунды и микросекунды. Можно использовать для получения более точных значений времени, чем возвращается системными часами.

15.5 Проверка меток времени

Для чтения и анализа ответов от TSA применяется функции

```
TS_RESP *d2i_TS_RESP_bio(BIO *b,TS_RESP*resp)
```

Читает из `BIO` структуру `TS_RESP` помещая её в структуру `resp` или, если параметр `resp` равен `NULL`, размещая новую структуру с помощью **TS_RESP_new**.

```
TS_RESP_print_bio(BIO *out_bio, TS_RESP *response)
```

Выводит ответ от TSA в человеко-читаемом формате.

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

```
PKCS7 *TS_RES_get_token(TS_RESP *response)
```

Извлекает из ответа TSA собственно метку времени в виде сообщения PKCS#7.

Для проверки метки времени следует создать контекст проверки — структуру `TS_VERIFY_CTX`.

```
TS_VERIFY_CTX *TS_VERIFY_CTX_new(void)
```

Создает контекст.

```
void TS_VERIFY_CTX_free(TS_VERIFY_CTX *ctx)
```

Освобождает контекст.

```
TS_VERIFY_CTX *TS_REQ_to_TS_VERIFY_CTX(TS_REQ* request,
      TS_VERIFY_CTX *ctx)
```

Создает контекст проверки на базе запроса, т.е. для проверки ответа на конкретный запрос.

Для настройки `TS_VERIFY_CTX` не предусмотрено таких функций, как для `TS_RESP_CTX`, поэтому необходимые параметры надо устанавливать непосредственно в поля структуры.

Поле `flags` содержит битовую маску флагов

`TS_VFY_SIGNATURE` — проверять подпись TSA и сертификат. Необходиом создать объект хранилища сертификатов (см раздел 13 и поместить указатель на него в поле `store`. Можно также поместить набор дополнительных сертификатов в поле `certs`.

`TS_VFY_VERSION` — проверять версию.

`TS_VFY_IMPRINT` — проверять хэш-сумму. Необхоимо поместить указатель на данные хэша в поле `imprint` и его длину в `imprint_len`. Можно также поместить желаемый алгоритм хэширования в поле `md_alg`. Если этого не сделать, будет использован алгоритм, указанный в проверяемой метке времени.

`TS_VFY_POLICY` — проверять политику. Необходимо поместить OID политики в поле `policy`.

`TS_VFY_DATA` — Вычислить хэш данных, с использованием алгоритма, указанного в метке времени по данным, и сравнивать его с хэшом, содержащимся в метке времени. Необходимо помстить в поле `data` указатель на `BIO`, откуда читаются данные. Этот флаг несовместим с флагом `TS_VFY_IMPRINT`.

`TS_VFY_NONCE` — проверять соответствие одноразового индентификатора со значением, помещенным в поле `nonce` контекста.

`TS_VFY_SIGNER` — Проверить что поле TSA name метки времени соответствует владельцу сертификата, которым подписана заявка.

`TS_VFY_TSA_NAME` — проверить что поле TSA name метки времени соответствует полю `tsa_name` контекста.

`TS_VFY_ALL_IMPRINT` — OR всех флагов кроме `TS_VFY_DATA`

`TS_VFY_ALL_DATA` — OR всех флагов кроме `TS_VFY_IMPRINT`

```
int TS_RESP_verify_response(TS_VERIFY_CTX *ctx, TS_RESP
      *response)
```

Проверяет ответ от TSA

```
int TS_RESP_verify_token(TS_VERIFY_CTX *ctx, PKCS7 *token)
```

Проверяет собственно метку времени.

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

16 Работа с датчиками случайных чисел

16.1 Выбор датчика случайных чисел, поддерживаемого СКЗИ «МагПро Криптопакет»

СКЗИ «МагПро Криптопакет» поддерживает работу с несколькими типами сертифицированных датчиков случайных чисел.

В норме, используемый датчик случайных чисел задается в конфигурационном файле в виде параметра `RNG` в секции, описывающей параметры cryptocom engine.

Можно переопределить тип используемого датчика с помощью задания переменной среды `RNG`, что имеет приоритет перед значением из конфигурационного файла, так как, как правило, пользователю легче изменить среду текущего процесса, чем конфигурационный файл.

Если необходимо изменить тип датчика из приложения, с помощью **ENGINE_ctrl_cmd**, то перед этим необходимо очистить переменные среды `RNG` и `RNG_PARAMS` с помощью функции **unsetenv**.

Датчик случайных чисел `PROGRAM` не может быть использован для генерации долговременных ключей. Если нет возможности использовать аппаратный датчик, то приложения выполняющие операции подписи, шифрования, TLS соединения и т.д., могут использовать датчик `PROGRAM`, а приложения, генерирующие долговременные ключи, должны использовать датчик `KEYBOARD`.

16.2 Функция обратного вызова клавиатурного ДСЧ

Для работы датчика `KEYBOARD` необходимо установить коллбэк функцию взаимодействия с пользователем.

Это выполняется с помощью управляющей команды `"ce set rds callbacks"`.

Приложениям, использующим аппаратные ДСЧ рекомендуется также устанавливать функции обратного вызова с помощью данной команды, так как это позволит более дружественным для пользователя способом обрабаывать ошибки аппаратного датчика.

В качестве параметра этой команде передается указатель на структуру `cc_rds_callbacks`, определенную следующим образом:

```
typedef void *cc_app_data_t;
typedef int (__cdecl *cc_rds_keyboard_cb_ft) (cc_app_data_t app_data,
    int character, int scored, int total);
typedef int (__cdecl *cc_rds_cb_ft) (cc_app_data_t app_data, const char *type,
    const char *ident, const char *container_name, int opcode, int stage, int done,
    int total);

typedef struct cc_rds_cb_st {
    cc_rds_keyboard_cb_ft kbd_cb;
    cc_rds_cb_ft rds_cb;
    cc_app_data_t kbd_client_data;
    cc_app_data_t rds_client_data;
    } cc_rds_callbacks, *cc_rds_cb_h;
```

Поле `kbd_cb` указывает на функцию обратного вызова клавиатурного датчика. Поле `rds_cb` на функцию обратного вызова аппаратных датчиков. Поля `kbd_client_data`

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

`rds_client_data` это непрозрачные для библиотеки указатели, передаваемые в эти функции.

```
int kbd_cb(cc_app_data_t app_data,
       int character, int scored, int total)
```

Функция должна сообщить пользователю символ `character`, дождаться ввода символа с клавиатуры и вернуть этот символ.

Поля `scored` и `total` указывают текущее количество использованных при инициализации клавиатурного ДСЧ символов, и общее количество символов, необходимых для инициализации. Эти два поля могут быть использованы для отображения индикатора прогресса.

Следует учитывать, что при неправильном вводе пользователем символов, значение `scored` может уменьшаться.

В случае если `scored==total`, инициализация датчика завершена, и функция должна убрать интерфейс пользователя с экрана и вернуть значение `character`.

```
int rds_cb(cc_app_data_t app_data, cc_cstr_t type,
       cc_cstr_t ident, cc_cstr_t container_name, int opcode, int
       stage, int done,
       int total)
```

Вызывается как при проблемах с аппаратным датчиком или ключевым носителем, так и для отображения хода выполнения операции.

Функция должна возвратить управление немедленно, произведя, возможно необходимые действия по отображению/скрытию интерфейсных элементов.

Функция возвращает один из трех кодов

`cc_rds_cb_rc_continue` — продолжить операцию
`cc_rds_cb_rc_cancel` — прервать операцию
`cc_rds_cb_rc_timeout` — закончилось время ожидания действий пользователя

Параметры функции:

`app_data` — Указатель, переданный библиотеке в составе структуры функций обратного вызова.
`type` — тип устройства, в виде строки символов
`ident` — идентификатор устройства в виде строки символов
`container_name` — имя ключевого контейнера на устройстве (если имеет отношение к даноной операции)
`opcode` — код выполняемой операции
`stage` — стадия выполняемой операции
`done` — количество выполненных условных единиц работы
`total` — общее количество условных единиц в данной операции

Возможные значения параметра `opcode`

`cc_rds_cb_op_write` — операция записи данных на устройство
`cc_rds_cb_op_read` — операция чтения данных на устройство
`cc_rds_cb_op_random` — операция получения случайных числе
`cc_rds_cb_op_write_key` — операция чтения ключа
`cc_rds_cb_op_read_key` — операция записи ключа

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

`cc_rds_cb_op_has_key` — операция проверки наличия ключа
`cc_rds_cb_op_gen_key` — операция генерации ключа
`cc_rds_cb_op_init_storage` — инициализация ключевого контейнера
`cc_rds_cb_op_wipe_storage` — затирание ключевого контейнера

Возможные значения параметра `stage`

`cc_rds_cb_stage_end` — Завершение ожидания действий пользователя. При вызове с этим значением функция должна спрятать ранее выведенный интерфейс сообщения пользователю о необходимых действиях
`cc_rds_cb_stage_query` — Запрос действий пользователя (например «Вставьте устройство в USB-порт», «Прислоните таблетку к считывателю»). Получив это значение функция должна отобразить соответствующее сообщение и вернуть управление, чтобы код библиотеки мог проверить выполнение пользователем этой операции.
`cc_rds_cb_stage_error` — Запрос действий пользователя после неудачи разовой попытки
`cc_rds_cb_stage_inner_lock` —Запрос действий пользователя при блокировке тем же процессом
`cc_rds_cb_stage_outer_lock` —Запрос действий пользователя при блокировке другим процессом
`cc_rds_cb_stage_wrong` — Не тот носитель, запрос на замену
`cc_rds_cb_stage_progress` — сообщение о ходе операции (индикация прогресса)
`cc_rds_cb_stage_wait` — Требуется подождать готовности неопределенное время

16.3 Получение случайных чисел

```
int RAND_bytes(unsigned char *buf, int num)
```

Записывает `num` криптографически сильных псевдослучайных чисел в буфер `buf`. Возвращает 1 в случае успеха, 0 в случае ошибки.

```
int RAND_pseudo_bytes(unsigned char *buf, int num)
```

Записывает `num` псевдослучайных чисел в буфер `buf`. Достаточная для генерации долговременных ключей непредсказуемость этих чисел не гарантируется. В случае если полученные случайные данные криптографически сильные, возвращает 1, в случае если они недостаточно сильные, 0 и -1 в случае ошибки.

17 Низкоуровневые функции работы с криптоалгоритамами

Функции, описанные в данном разделе, как правило, не используются приложениями напрямую. Приложения, использующие стандартные протоколы и форматы защищенных данных могут пользоваться более высокоуровневыми функциями.

Их использование необходимо только в случае приложений, реализующих свои собственные криптопротоколы или форматы защищенных данных.

Данные функции получают или возвращают сессионные ключи в виде массивов байт, что может привести к компрометации сессионного ключа в случае ошибок в приложениях.

Поэтому приложения использующие данные API должны проходить экспертизу корректности встраивания.

17.1 Использование функций подписи для имитозащиты

При использовании данных функций для выработки имитовставки, требуется сформировать структуру `EVP_PKEY`, содержащую ключ имитозащиты.

Для этого используется функция

```
EVP_PKEY *EVP_PKEY_new_mac_key(int type, ENGINE *e,
      unsigned char *key, int keylen)
```

Параметр `type` этой функции содержит числовой идентификатор алгоритма имитозащиты (может быть `EVP_PKEY_HMAC` или `NID_id_Gost28147_89_MAC`).

Параметр `e` указывает энджин, содержащий предпочитаемую реализацию этого алгоритм (может быть `NULL`).

Параметр `key` содержит буфер с ключевым материалом,

Параметр `keylen` указывает длину этого буфера.

Для алгоритма ГОСТ 28147-89 в режиме имитовставки буфер должен содержать ровно 32 байта ключевого материала.

Для алгоритма HMAC ключ может быть любой длины. В этом случае, если ключ представляет собой строку символов C, оканчивающуюся нулевым байтом, допустимо не подсчитывать её длину, а передать -1 в качестве `keylen`.

Не предусмотрено использование ключей имитозащиты в функциях **EVP_DigestVerify***. Для проверки имитовставки необходимо выработать имитовставку с тем же ключом, и побайтово сравнить выработанную имитовставку с исходной.

17.2 Выработка общего ключа

```
int EVP_PKEY_derive_init(EVP_PKEY_CTX *ctx)
```

Инициализирует контекст для операции выработки общего ключа (например, при помощи алгоритма VKO 34.10-2001). Контекст должен быть создан на основе своего закрытого ключа.

После инициализации контекста для операции верификации необходимо установить в контекст открытый ключ другой стороны соединения с помощью функции

```
int EVP_PKEY_derive_set_peer(EVP_PKEY_CTX *ctx, EVP_PKEY
      *peer)

int EVP_PKEY_derive(EVP_PKEY_CTX *ctx, unsigned char *key, size_t
      *keylen)
```

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

Вычисляет собственно общий ключ. Если параметр `key` равен `NULL`, то только помещает в `keylen` небходимую длину буфера. Если `key` не `NULL`, то `keylen` на входе должна содержать доступный размер буфера.

Данная операция поддерживается не всеми асимметричными алгоритмами.

17.3 Зашифрование сессионного ключа

В случае алгоритма ГОСТ 34.10-2001, при операции зашифрования на открытом ключе создается ASN.1 структура, описанная в RFC 4490. Если в контекст не был установлен закрытый ключ отправителя с помощью функции **EVP_PKEY_derive_set_peer**, то генерируется эфемерная ключевая пара, открытая часть которой помещается в эту структуру, а закрытая уничтожается при завершении операции.

Если закрытый ключ был установлен, то генерируется структура в которой остсутствует эфемерный открытый ключ.

```
int EVP_PKEY_encrypt_init(EVP_PKEY_CTX *ctx)
```

Инициализирует операцию зашифрования на открытом ключе. Контекст должен быть создан на основе открытого ключа адресата.

```
int EVP_PKEY_encrypt(EVP_PKEY_CTX *ctx,
      unsigned char *out, size_t *outlen, const unsigned char *in,
      size_t inlen)
```

Зашифровывает данные в буфере `in` (обычно сессионный ключ) длины `inlen` байт и помещает созданное зашифрованное значение в буфер `out` и количество записанных байт в `outlen`.

Если `out` равна `NULL`, то только вычисляет необходимую длину буфера и помещает её в `outlen`. Если `out` не `NULL`, на входе `outlen` должна содержать доступный размер буфера.

```
int EVP_PKEY_decrypt_init(EVP_PKEY_CTX *ctx)
```

Инициализирует контекст для операции расшифрования (в случае ГОСТ - для расшифрования структуры GOST KEY TRANSPORT, описанной в RFC 4490).

Контекст должен быть создан на основе закрытого ключа получателя.

Если структура не содержит эфемерного открытого ключа, то необходимо после вызова **EVP_PKEY_decrypt_init**, установить в контекст долговременный закрытый ключ отправителя с помощью функции **EVP_PKEY_derive_set_peer**.

```
int EVP_PKEY_decrypt(EVP_PKEY_CTX *ctx, unsigned char *out,
      size_t *outlen, const unsigned char *in, size_t inlen)
```

Расшифровывает структуру, содержащуюся в буфере `in` длины `inlen` и помещает результат в буфер `out` а длину записанных данных в `outlen`.

17.4 Симметричное шифрование

Для зашифрования или расшифрования данных с использованием симметричного криптоалгоритма необходимо создать и проинициализировать структуру `EVP_CIPHER_CTX`, указав симметричный ключ, и, если необходимо, вектор инициализации. Затем открытый текст (при зашифровании) или шифртекст передается в функцию **EVP_EncryptUpdate**, которая генерирует соответственно шифртекст или открытый текст.

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

В случае если алгоритм использует длину блока, отличную от единицы (например режим простой замены или режим CBC), последний блок шифртекста или открытого текста может быть получен только при финализации алгоритма.

В случае использования алгоритмов с длиной блока равной единице (ГОСТ 28147-89 в режиме гаммирования и в режиме гаммирования с обратной связью относится именно к этой категории), функции финализации всегда возвращают пустой блок.

Для инициализации алгоритма необходимо получить структуру `EVP_CIPHER`, соответствующую данному алгоритму и содержащую информацию о его параметрах, а также указатели на функции, его реализующие.

Для получения этой структуры используются функции:

```
const EVP_CIPHER *EVP_get_cipherbyname(const char *name)

const EVP_CIPHER *EVP_get_cipherbynid(int nid)

const EVP_CIPHER *EVP_get_cipherbyobj(const ASN1_OBJECT *obj)
```

Эти функции возвращают константную структуру информации об алгоритме шифрования. Данная структура может быть передана в функцию инициализации контекста шифрования либо непосредственно, либо через какое-либо высокоуровневое API (например функцию **PKCS7_encrypt**).

Кроме того, существует ряд информационных фукнций, возвращающих информаию об алгоритме:

```
int EVP_CIPHER_nid(const EVP_CIPHER *e)

int EVP_CIPHER_type(const EVP_CIPHER *e)
```

Возвращает числовой идентификатор (nid) алгоритма по которому может быть получен ASN.1 OID или название алгоритма с использованием функций работы с базой ASN.1 объектов (см раздел 18.5).

```
int EVP_CIPHER_block_size(const EVP_CIPHER *e)
```

Возвращает размер блока алгоритма шифрования.

```
int EVP_CIPHER_key_length(const EVP_CIPHER *e)
```

Возвращает требуемую длину ключа в байтах.

```
int EVP_CIPHER_iv_length(const EVP_CIPHER *e)
```

Возвращает требуемую длину вектора инициализации в байтах.

```
int EVP_CIPHER_mode(const EVP_CIPHER *e)
```

Возвращает режим шифра. Одну из следующих констант:

`EVP_CIPH_STREAM_CIPHER` — потоковый шифр
`EVP_CIPH_ECB_MODE` — режим простой замены (ECB)
`EVP_CIPH_CBC_MODE` — режим сцепления блоков шифртекста (CBC)
`EVP_CIPH_CFB_MODE` — режим гаммирования с обратной связью (CFB)
`EVP_CIPH_OFB_MODE` — режим обратной связи по выходу (OFB)

```
int EVP_CIPHER_flags(const EVP_CIPHER *e)
```

Возвращает битовую маску флагов шифра. Младшие три бита этой маски содержат режим шифра (см выше), а остальные представляют собой or-комбинацию следующих констант:

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

EVP_CIPH_VARIABLE_LENGTH — шифр переменной длины
EVP_CIPH_CUSTOM_IV — шифр требует пользовательского вектора инициализации
EVP_CIPH_ALWAYS_CALL_INIT — указывает что вызов алгоритм-специфичной функции инициализации должен производиться и в том случае, если инициализация производится без указания ключа (т.е. предполагается задать ключ позднее, что приведет к повторному вызову этой функции).
EVP_CIPH_CTRL_INIT — Требуется инициализация параметров посредством **EVP_CIPHER_CTX_ctrl**
EVP_CIPH_CUSTOM_KEY_LENGTH — Используется нестандартный способ определения длины ключа.
EVP_CIPH_NO_PADDING — не используется стандартный способ дополнения до полной длины блока.
EVP_CIPH_RAND_KEY — реализация алгоритма шифрования поддерживает генерацию случайного ключа.

Существует аналогичный набор информационных функций с префиксом EVP_CIPHER_CTX, возвращающих информацию о шифре, используемом инициализированной структурой структурой контекста шифрования.

```
void EVP_CIPHER_CTX_init(EVP_CIPHER_CTX *a)
```

Производит первоначальную инициализацию структуры EVP_CIPHER_CTX, необходимую перед вызовом функций **EVP_EncryptInit_ex** или **EVP_DecryptInit_ex**. Используется в тех случаях, когда переменная типа EVP_CIPHER_CTX размещается в статической памяти или в стеке. Если структура размещается в динамической памяти, то следует использовать функцию:

```
EVP_CIPHER_CTX *EVP_CIPHER_CTX_new()
```

выделяет память под структуру EVP_CIPHER_CTX, выполняет первоначальную инициализацию с помощью **EVP_CIPHER_CTX_init** и возвращает указатель на структуру.

```
int EVP_EncryptInit_ex(EVP_CIPHER_CTX *ctx, const EVP_CIPHER
      *type, ENGINE *impl, unsigned char *key, unsigned char *iv)

int EVP_DecryptInit_ex(EVP_CIPHER_CTX *ctx, const EVP_CIPHER
      *type, ENGINE *impl, unsigned char *key, unsignec char *iv)

int EVP_CipherInit_ex(EVP_CIPHER_CTX *ctx, const EVP_CIPHER
      *type, ENGINE *impl, unsigned char *key, unsigned char *iv, int
      env)
```

Эти функции выполняют подготовку контекста к операциям расшифрования и зашифрования. В функции **EVP_CipherInit_ex** направление операции задается при помощи параметра enc, который должен быть равен единице при зашифровании и нулю при зашифровании.

Параметр impl задает модуль в котором следует искать реализацию алгоритма. Если этот параметр равен NULL, используется умолчательная реализация.

Допустимо вызывать эти функции несколько раз, например сначала указав тип шифра, и оставив ключ и вектор инициализации равными NULL, а потом указав тип шифра равным NULL и указав ключ и вектор инициализации.

Функции **EVP_EncryptInit**, **EVP_CipherInit** и **EVP_DecryptInit** отличаются от соответствующих функций с суффиксом _ex тем, что не получают параметра impl и всегда используют умолчательную реализацию алгоритма.

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

Кроме того если параметр `cipher` не равен `NULL`, они производят очистку контекста вызовом **EVP_CIPHER_CTX_init**.

Поэтому при использовании этих фунцкий необходимо задавать алгоритм шифрования первым вызовом функции, а ключ и вектор инициализации последующим (либо все три одновременно). При использовании функций с суффиксом `_ex` можно задавать алгоритм, ключ и вектор инициализации в любом порядке.

Разработчики OpenSSL рекомендуют **всегда** использовать функции с суффиксом `_ex`.

Кроме того, при расшифровании данных, хранящихся в ASN.1 структурах (например PCS7#7 сообщениях) можно использовать функцию

```
int EVP_CIPHER_asn1_to_param(EVP_CIPHER_CTX *c, ASN1_TYPE
      *type)
```

Эта функция должна быть вызвана после того, как с помощью функций семейства Init установлен тип шифра, и до того как установлен ключ.

Функции возвращают 1 в случае успеха и 0 в случае ошибки.

```
int EVP_CIPHER_param_to_asn1(EVP_CIPHER_CTX *c, ASN1_TYPE
      *type)
```

Устанавливает значения в структуре type в соответствии с параметрами шифра, инициализированными в контексте. Применяется для формирования форматов зашифрованных данных, использующих структуру ASN.1.

```
EVP_CIPHER_CTX_set_key_length(EVP_CIPHER_CTX *x, int keylen)
```

Устанавливает размер используемый ключа для шифров, поддерживающих переменный размер ключа. В случае алгоритма ГОСТ 28147-89, размер ключа которого фиксирован, не используется.

```
int EVP_EncryptUpdate(EVP_CIPHER_CTX *ctx, unsigned char
      *out, int *outl, unsigned char *in, int inl)

int EVP_DecryptUpdate(EVP_CIPHER_CTX *ctx, unsigned char
      *out, int *outl, unsigned char *in, int inl)

int EVP_CipherUpdate(EVP_CIPHER_CTX *ctx, unsigned char *out,
      int *outl, unsigned char *n, int inl)
```

Обрабатывают блок данных, находящийся в буфере, на который указывает параметр `in` длиной `inl`. Помещают результа зашифрования/расшифрования в буфер `out` и длину полученного шифртекста/открытого текста в `outl`.

В ряде режимов шифров объем результата может отличаться от объема исходных данных, например в случае блочных шифоров последние байты данных, не составляющие целого блока не будут обрадотаны при данном вызове функции Update, но могут быть обработаны при следующем, соответственно, функция может вернуть как меньше, так и больше данных чем было получено.

Функция **EVP_CipherUpdate** определяет направление действия (зашифрование или расшифрование) по состоянию контекста.

```
EVP_CIPHER_CTX_set_padding(EVP_CIPHER_CTX *x, int padding)
```

устанавливает режим дополнения до целого блока. По умолчанию режим дополнения включен. Если он выключен (для чего в параметре `padding`) следует указать 0, то общий объем

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

зашифрованных данных должен быть кратен размеру блока. Иначе при функция финализации возвратит ошибку.

```
int EVP_DecryptFinal(EVP_CIPHER_CTX *ctx, unsigned char
      *out, int *outl)

int EVP_EncryptFinal(EVP_CIPHER_CTX *ctx, unsigned char out,
      int *outl)

int EVP_CipherFinal(EVP_CIPHER_CTX *ctx, unsigned char out,
      int *outl)

int EVP_DecryptFinal_ex(EVP_CIPHER_CTX *ctx, unsigned char
      *out, int *outl)

int EVP_EncryptFinal_ex(EVP_CIPHER_CTX *ctx, unsigned char out,
      int *outl)

int EVP_CipherFinal_ex(EVP_CIPHER_CTX *ctx, unsigned char out,
      int *outl)
```

Завершают операцию шифрования, возможно (в случае блоковых шифров и включенного padding) возвращая результат обработки последнего (неполного) блока.

Функции без суффикса `_ex` делают то же самое, что и аналогичные функции с этим суффиксом.

```
int EVP_CIPHER_CTX_cleanup(EVP_CIPHER_CTX *a)
```

Очищает контекст, затирая ключ, вектор инициализации и прочие сенситивные данные.

Используется для контекстов, размещенных в статической или автоматической памяти. Для контекстов, созданных с помощью функции **EVP_CIPHER_CTX_new** следует использовать функцию

```
void EVP_CIPHER_CTX_free(EVP_CIPHER_CTX *a)
```

.

17.5 Формирование ключа из пароля

В некоторых случаях (например, при создании закрытых ключей подписи) необходимо защитить данные с помощью шифрования с использованием ключа, выведенного из вводимого пользователем пароля (пассфразы).

```
int EVP_BytesToKey(const EVP_CIPHER *type, EVP_MD *md,
      const unsigned char *salt, const unsigned char *data, int
      datal, int
      count, unsigned char *key, unsigned char *iv)
```

Функция выполняет формирование ключа и вектора инициализации для алгоритма шифрования `type` на основе данных `data` длиной `datal` с помощью алгоритма хэширования `md`. Если параметр `salt` не `NULL`, он должен указывать на 8-байтовый буфер с дополнительным материалом для вывода ключа.

Созданные ключ и вектор инициализации помещаются в буфера, указатели на которые передаются в параметрах `key` и `iv` соответственно.

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

Размеры этих буферов должны быть достаточны для размещения ключа и вектора инициализации для передаваемого алгоритма шифрования.

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

18 Служебные типы и функции

18.1 BIO

Тип BIO — абстракция ввода-вывода в OpenSSL, позволяющая основным функциям не обрабатывать отдельно каждый вид источника или получателя данных. Объекты BIO в OpenSSL бывают двух подтипов — оконечные (файлы, сокеты, память и т.п.) и фильтры. Описание фильтров выходит за пределы данного документа.

Приложение может воспользоваться существующими типами BIO, как это сделано в примерах из данного документа, а может создать свой. Над каждым объектом BIO определены следующие операции:

```
int BIO_read(BIO *b, void *buf, int len);
```

Пытается прочесть из объекта ***b*** ***len*** байт данных в буфер ***buf***. Возвращает количество прочитанных байт.

```
int BIO_gets(BIO *b, char *buf, int size);
```

Для большинства типов BIO выполняет операцию, аналогичную **fgets** — читает из ***b*** в ***buf*** до ближайшего конца строки, но не более ***size*** байт. Однако, для фильтров, представляющих операцию хэширования, эта операция означает «выдать текущее значение хэш-функции», и для ряда типов BIO она не определена. Возвращает количество прочитанных байт.

```
int BIO_write(BIO *b, const void *buf, int len);
```

Пытается записать в ***b*** ***len*** байт из буфера ***buf***. Возвращает количество записанных байт.

```
int BIO_puts(BIO *b, const char *buf);
```

Пытается записать ASCIIZ-строку из ***buf*** в ***b***. Возвращает количество записанных байт.

```
int BIO_printf(BIO *b, const char *format, ...);
```

Форматирует выводимые данные согласно указанному формату и выводит их в ***b*** вызовом **BIO_write**.

Все функции работы с BIO возвращают -2, если соответствующая операция для данного объекта не определена. Функции чтения/записи возвращают 0 или -1 в случае ошибки или если данных нет. Интерфейс не специфицирует, является ли 0 или -1 ошибкой. Так, вызов **BIO_read** на неблокирующем сокете может вернуть 0 и если соединение закрыто, и если на данный момент в сокете отсутствуют данные. В такой ситуации следует вызвать

```
int BIO_should_retry(BIO *b)
```

Если этот вызов вернул 0, то произошла ошибка (соединение закрыто), а если 1 — отсутствуют данные, вызов следует повторить через некоторое время.

```
void BIO_free(BIO *b);
```

В то время как функции создания объектов BIO для каждого типа свои, функция удаления их — общая. Эта функция удаляет объект, предварительно проделав все необходимые действия по корректному завершению операций (закрывает файл или сокет, очищает память, и т.п.).

Ниже описаны основные типы BIO, с которыми может понадобиться работать автору приложения, использущего МагПро КриптоПакет.

18.1.1 Файловый BIO

Файловый BIO построен поверх стандартного типа FILE.

Создать файловый BIO можно одной из следующих функций:

```
BIO *BIO_new_file(const char *filename, const char *mode);
```

Параметры ***filename*** и ***mode*** имеют то же значение, что и для стандартной функции **fopen**. Объект будет привязан к результату вызова **fopen** с этими аргументами.

```
BIO *BIO_new_fp(FILE *stream, int flags);
```

Созданный объект привязывается к потоку ***stream***. Возможные флаги:

`BIO_CLOSE` — закрыть поток при выполнении **BIO_free**.
`BIO_NOCLOSE` — не закрывать поток при выполнении **BIO_free**. Этот флаг следует указывать, если объект привязывается к одному из стандартных потоков `stdin`, `stdout` или `stderr`.
`BIO_FP_TEXT` — установить поток в текстовый режим (влияет на работу только под Win32).

Файловый BIO поддерживает функции **BIO_gets** и **BIO_puts**, а также **BIO_flush** (вызывает `fflush(stream)`), **BIO_reset** (вызывает `fseek(stream,0,0)`), **BIO_seek** (вызывает `fseek(stream,offset,0)`), **BIO_tell** (вызывает `ftell(stream)`), **BIO_eof** (вызывает `feof(stream)`).

18.1.2 Сокетный BIO

Сокетный BIO строится поверх сокетов.

```
BIO *BIO_new_socket(int sock, int close_flag);
```

Если параметр ***close_flag*** истинен, сокет будет закрыт при вызове **BIO_free**. Сокетный BIO поддерживает **BIO_gets**, **BIO_puts**, **BIO_flush** и **BIO_eof**, но по природе сокетов не поддерживает операцию seek и связанные.

18.1.3 BIO в памяти

BIO в памяти — это BIO, данные которого хранятся в памяти. Он бывает двух подтипов — только для чтения и для чтения-записи.

Объект только для чтения создается на основании буфера в памяти вызовом

```
BIO *BIO_new_mem_buf(void *buf, int len)
```

Если ***len*** равно -1, ***buf*** предполагается ASCIIZ-строкой, и его длина определяется вызовом **strlen**. Вызов **BIO_reset** для такого объекта приводит его в исходное состояние, т.е. позволяет начать читать сначала. Освобождение такого объекта не приводит к освобождению памяти, на которой он построен.

Объект для чтения-записи создается вызовом `BIO_new(BIO_s_mem())`. При записи в такой объект необходимая память будет выделяться автоматически. При чтении из такого объекта прочитанные данные удаляются из него. Вызов **BIO_reset** для такого объекта очищает его данные.

Оба подтипа поддерживают операции **BIO_gets** и **BIO_puts**. Вызов **BIO_eof** возвращает истину, если в объекте нет данных. Вызов **BIO_ctrl_pending** возвращает количество байт данных, имеющихся в объекте на данный момент.

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

Вызов `BIO_set_mem_eof_return(BIO *b, int v)` позволяет настраивать поведение функций **BIO_read** и **BIO_gets** в ситуации, когда даных в объекте нет. Если ***v*** равно 0, эти функции будут возвращать 0, а вызов `BIO_read_retry(b)` будет возвращать ложь. Если ***v*** не равно 0, то эти функции будут возвращать ***v*** (во избежание двусмысленностей следует устанавливать ***v*** в значение -1), а `BIO_read_retry(b)` будет возвращать истину.

Вызов

```
long BIO_get_mem_data(BIO *b, char **pp)
```

возвращает количество байт данных в объекте, и через ***pp*** — указатель на эти данные в памяти. Это позволяет получить доступ к данным без лишнего копирования.

Операция чтения из объекта для чтения-записи на практике приводит к копированию остатка в начало буфера, т.е. является сравнительно дорогой. Поэтому в OpenSSL нередко используется идиома создания временного объекта только для чтения на том же буфере:

```
BIO *temp_ro_b; char *p; long len;
...
len = BIO_get_mem_data(rw_b, &p);
temp_ro_b = BIO_new_mem_buf(p, len);
```

и дальнейшее чтение производится из `temp_ro_b`.

18.2 API стеков OpenSSL

В случаях, когда в какую-либо функцию OpenSSL необходимо передать или вернуть из нее набор однотипных объектов (таких, как сертификаты получателей зашифрованного письма), как правило, используется стек. Стеки в OpenSSL реализованы макросами. Они всегда хранят не сами объекты, а указатели на них, подразумевая, что объекты выделены динамически. Данные, необходимые для реализации структуры стека, тоже выделяются динамически, и работа с ними ведется через указатели.

Стек объектов типа `TYPE` объявляется следующим образом:

STACK_OF(TYPE) *stack_var;

Он создается вызовом sk_TYPE_new_null() и пополняется вызовами sk_TYPE_push(stack_var, object_ptr). После использования стек следует освободить вызовом sk_TYPE_pop_free(stack_var,TYPE_free_func). При этом будут удалены и содержащиеся в нем объекты, каждый отдельным вызовом **TYPE_free_func(obj_ptr)**. Для передачи наборов в функции этих вызовов, как правило, достаточно.

Для обработки стеков, возвращаемых функциями OpenSSL (например, набора сертификатов, использованных для подписи, который возвращает функция **PKCS7_get0_signers**) требуются также способы доступа к содержимому стека. Наиболее часто используется идиома

```
for (i=0; i<sk_TYPE_num(stack_var); i++)
        handle(sk_TYPE_value(stack_var, i));
```

Здесь **sk_TYPE_num** возвращает количество элементов в стеке, а **sk_TYPE_value** возвращает указатель на элемент с указанным индексом.

Функция **PKCS7_get0_signers**, как следует из «0» в ее имени, не копирует возвращаемые ею сертификаты, поэтому удалять возвращенный ею стек следует не вызовом sk_X509_pop_free(stack_var, X509_free), а вызовом sk_X509_free(stack_var). Этот вызов удаляет только структуру стека, но не содержащиеся в нем объекты.

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

18.3 Аргументы функций чтения и записи формата PEM

Все функции чтения из формата PEM (**PEM_read_bio_<*type*>** и **PEM_read_<*type*>**) получают следующие параметры:

`BIO *bp` — указатель на объект `BIO`, из которого функция читает данные (**PEM_read_bio_<*type*>**).

`FILE *fp` — указатель на объект `FILE`, из которого функция читает данные (**PEM_read_<*type*>**).

`<TYPE> **x` — если этот параметр равен `NULL`, он игнорируется. Если он ненулевой, но `*x` равно `NULL`, указатель на созданную при чтении структуру типа `<TYPE>` будет возвращен через этот параметр. Если же `*x` не `NULL`, то функция попытается повторно использовать структуру, на которую указывает это значение. В любом случае возвращаемым значением функции будет указатель на структуру, в которую прочитаны данные.

`pem_password_cb *cb` — указатель на функцию обратного вызова для запроса пароля в случае зашифрованной PEM-структуры (в норме это только закрытые ключи). Эта функция имеет прототип

```
int cb(char *buf, int size, int rwflag, void *u);
```

Здесь ***buf*** — буфер, в который эта функция должна записать пароль, ***size*** — размер этого буфера, ***rwflag*** устанавливается в 0 при запросе пароля для операции чтения и в 1 — для операции записи (типичная функция обратного вызова должна запросить подтверждение пароля, если этот флаг равен 1), параметр ***u*** передается без изменений из функции чтения PEM. Функция обратного вызова должна возвратить количество символов в пароле, или 0 в случае ошибки.

`void *u` — пользовательские данные для функции запроса пароля, если она определена. Если она не определена (т.е. `cb==NULL`), в этом параметре ожидается пароль в виде ASCIIZ-строки. Если он также равен `NULL`, при необходимости будет использована встроенная функция запроса пароля, подходящая для командностроочных, но не графических приложений.

Функции чтения возвращают указатель на заполненную структуру в случае успеха или `NULL` в случае неудачи.

Функции записи в PEM получают параметры:

`BIO *bp` — указатель на объект `BIO`, в который функция пишет данные (**PEM_write_bio_<*type*>**).

`FILE *bp` — указатель на объект `FILE`, в который функция пишет данные (**PEM_write_<*type*>**).

`TYPE *x` — указатель на структуру, которую функция преобразует в формат PEM.

Функции записи возвращают 1 в случае успеха, и 0 в случае неудачи.

18.4 Вывод значений строк из ASN.1 структур

```
int ASN1_STRING_print_ex(BIO *out, ASN1_STRING *str, unsigned long
    flags);
```

```
int ASN1_STRING_print_ex_fp(FILE *fp, ASN1_STRING *str, unsigned long
      flags);
```

Выводит содержимое ASN.1 строки ***str*** в `BIO` ***out***. Формат вывода управляется с помощью флагов.

Наиболее распространенная комбинация флагов обозначается константой `ASN1_STRFLGS_RFC2253`. В случае, если строка может содержать русские буквы, рекомендуется использовать `ASN1_STRFLGS_RFC2253 & ~ASN1_STRFLGS_ESC_MSB`. В этом случае вывод должен интерпретироваться как имеющий кодировку UTF-8.

Поддерживаются следующие флаги

`ASN1_STRFLGS_ESC_2253` — Выводить в виде шестнадцатиричных кодов непечатаемые символы, определенные как таковые в RFC2253

`ASN1_STRFLFS_ESC_CTRL` — Выводить символы с кодами 0-32 в виде шестнацатиричных кодов

`ASN1_STRFLGS_ESC_MSB` — Выводить в виде шестнадцатиричных кодов все символы, кроме латинских букв и знаков препинания

`ASN1_STRFLGS_ESC_QUOTE` — Взять выводимую строку в кавычки

`ASN1_STRFLGS_UTF8_CONVERT` — Преобразовать символы не входящие в ASCII в представление UTF-8 из любого допустимого в ASN.1 представление

`ASN1_STRFLGS_IGNORE_TYPE` — Игнорировать ASN.1 тип строки и рассматривать строку как содержащую один байт на символ

`ASN1_SHOW_TYPE` — вывести перед строкой её ASN.1 тип.

`ASN1_STRFLGS_DUMP_ALL` — Вывести шестнадцатиричный дамп содержимого строки вместо преобразоваиня в печатную форму.

`ASN1_STRFLGS_DUMP_UNKNOWN` — Выводить как дамп только строки, тип которых не подразумевает печатного представления.

`ASN1_STRFLGS_DUMP_DER` — При выводе в виде шестнацатиричный дампа выводить полное ASN.1 представление строки, включая тэг типа и поле длины.

`ASN1_STRFLGS_RFC2253` — Комбинация флагов `ASN1_STRFLGS_ESC_2253`, `ASN1_STRFLGS_ESC_CTRL`, `ASN1_STRFLGS_ESC_MSB`, `ASN1_STRFLGS_UTF8_CONVERT`, `ASN1_STRFLGS_DUMP_UNKNOWN`,`ASN1_STRFLGS_DUMP_DER`.

18.5 База данных идентификаторов объектов

OpenSSL поддерживает базу данных ASN.1 идентификаторов объектов. Каждому объекту кроме собственно ASN.1 OID-а ставится в соответствие целочисленный код (NID), используемый в большинстве внутренних функций OpenSSL, короткое имя (SN) используемое в конфигурационном файле и опциях командной строки и длинное имя (LN) которое может быть использовано для визуализации.

Для преобразования между этими представлениями предназначены следующие функции:

```
ASN1_OBJECT *OBJ_nid2obj(int n)
```

Возвращает структуру `ASN1_OBJECT`, соответствующую указанному nid. Возвращается указатель на константную структуру из базы данных объектов. Освобождать выделенную под эту струкутру память не нужно.

```
const char *OBJ_nid2ln(int n)
```

Возвращает указатель на длинное описание объекта. Освобождать память не нужно.

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

```
const char *OBJ_nid2sn(int n)
```

Возвращает указатель на короткое имя.

```
int OBJ_obj2nid(const ASN1_OBJECT *o)
```

Возвращает nid, соответствующий указанному OID или `NID_undef`, если объект не зарегистрирован в базе.

```
int OBJ_sn2nid(const char *sn)
```

Возвращает nid по короткому имени или `NID_undef`, если переданная строка не является именем объекта.

```
int OBJ_ln2nid(const char *ln)
```

Возвращает nid по длинному имени

```
int OBJ_txt2nid(const char *s)
```

Возвращает nid по текстовому представлению объекта, которое может быть длинным именем, коротким именем или OID в десятично-точечной нотации.

```
ASN1_OBJECT *OBJ_txt2obj(const char *s, int no_name)
```

Преобразует текстовое представление в структуру `ASN1_OBJECT`. Если параметр `no_name` не равен нулю, воспринимает только десятично-точечную натацию OID-ов. Иначе воспринимает также и длинные и короткие имена. Эта функция может обрабатывать OID-ы, не зарегистрированные в базе объектов.

```
int OBJ_obj2txt(char *buf, int buf_len, const ASN1_OBJECT *a,
       int no_name)
```

Преобразует структуру `ASN1_OBJECT`, в том числе и содержащую не зарегистрированный в базе OID, в тексттовое представление.

Результат записывается в буфер `buf`, если его длина не превосходит `buf_len`. Если требуемая длина больше, то будет записана та часть текстового предстаавления, которая уместится, и возвращен необходимый размер буфера.

Если `no_name` равно 0, то если у данного OID имеется long name, будет возвращено оно, иначе, если есть, short name, и только при отсутствии в базе того и другого будет использована точечно-цифровая нотация.

18.6 Система диагностики ошибок libcrypto

В libcrypto существует стэк ошибок, в который может быть помещено несколько ошибок, начиная от самого нижнего уровня, и кончая самой высокоуровневой операцией. В многонитевом приложении каждая нить имеет свой собственный стэк ошибок.

В стэке ошибок вместо сообщений хранятся коды ошибок. Для того, чтобы можно было получить описание ошибки в формате, понятном человеку, следует загрузить таблицу строковых сообщений для данной библиотеки.

```
void ERR_load_error_strings(void);
```

Загружает таблицу сообщений библиотеки libcrypto.

Для получения человеко-читаемых сообщений об ошибке используются функции

```
void ERR_print_errors_fp(FILE *fp);
```

выводит текущее содержимое стэка ошибок в указанный файл и очищает стэк.

```
void ERR_print_errors(BIO *bp);
```

выводит текущее содержимое стэка ошибок в указанный объект ввода вывода BIO.

Поскольку libcrypto позволяет создавать объекты BIO работающие с буфером памяти, данную функцию можно использовать для формирования сообщения об ошибке в строке.

Получение машинно-читаемой информации об ошибке возможно с помощью следующих функций:

```
unsigned long ERR_get_error(void);
```

Возвращает первый код ошибки и удаляет его из очереди

```
unsigned long ERR_peek_error(void);
```

Возвращает первый код ошибки и не удаляет его из очереди

```
 unsigned long ERR_peek_last_error(void);
```

Возвращает последний код ошибки, не удаляя его.

Система обработки ошибок связывает с ошибкой информацию о месте её возникновения — файл исходного текста и номер строки в нём.

Получить эту информацию можно с помощью функций

```
 unsigned long ERR_get_error_line(const char **file, int
       *line)

unsigned long ERR_peek_error_line(const char **file, int *line);

 unsigned long ERR_peek_last_error_line(const char **file,
       int *line);
```

Эти функции возвращают код ошибки и помещают в переданные переменные указатель на строку имени файла и номер строки.

С ошибкой может быть также связана дополнительная информация, специфичная для данной библиотечной функции. Получить эту информацию можно с помощью функций

```
unsigned long ERR_get_error_line_data(const char **file, int *line,
       const char **data, int *flags);

unsigned long ERR_peek_error_line_data(const char **file, int *line,
       const char **data, int *flags);

unsigned long ERR_peek_last_error_line_data(const char **file,
       int *line , const char **data, int *flags);
```

Эти функции помещают в переменную *data указатель на дополнительные данные, а в переменной *flags устанавливают биты ERR_TXT_STRING, если данные размещены в статической памяти, и ERR_TXT_MALLOCED, если данные размещены с помощью функции **OPENSSL_malloc** и их после использования требуется освободить с помощью **OPENSSL_free**.

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

19 Требования к встраиванию библиотеки в приложения

19.1 Требования по организации передачи данных по каналам связи

При реализации передачи данных (сообщений, ключевой или аутентифицирующей информации в зашифрованном виде) между двумя абонентами на основе криптоядра СКЗИ (без использования протокола TLS) необходимо использовать протокол обмена информацией обеспечивающий:

- аутентификацию обоих абонентов связи;
- целостность передаваемого блока данных;
- защиту от повторного использования ключей шифрования (перекрытия ключей);
- защиту от повторов, а также навязывания ложных данных.

Кроме того, при выработке общего ключа абонент отправитель должен убедиться в том, что абонент получатель выработал ключ, совпадающий с ключом, который выработал абонент отправитель.

Примечание: Целостность передаваемых данных при большой скорости передачи рекомендуется осуществлять посредством выработки ЭЦП.

Приложения, реализующие собственные протоколы защиты данных, передаваемых по каналам передачи данных, необходимо сертифицировать на соответствие «Требованиям к шифровальным (криптографическим) средствам, предназначенным для защиты информации, не содержащей сведений, составляющих государственную тайну».

19.2 Требование по использованию криптоалгоритмов. Алгоритм ЭЦП

Цифровая подпись предназначена для аутентификации лица, подписавшего электронное сообщение. Кроме того, использование ЭЦП предоставляет возможность обеспечить следующие свойства при передаче в системе подписанного сообщения:

- осуществить контроль целостности передаваемого подписанного сообщения,
- доказательно подтвердить авторство лица, подписавшего сообщение,
- защитить сообщение от возможной подделки.

Для обеспечения заявленных свойств ЭЦП при развертывании сети связи, защищенной с использованием СКЗИ «МагПро Криптопакет», требуется обеспечить:

1. Невозможность отказа абонента, подписавшего сообщение, от своей подписи. Для этого должна быть выполнена процедура закрепления пары закрытый ключ, открытый ключ ЭЦП за администратором безопасности и каждым абонентом. Например, каждый абонент должен лично являться в УЦ и расписаться за владение сертификатом своего открытого ключа и сертификатом открытого ключа УЦ.
2. Надежное хранение в тайне своего закрытого ключа ЭЦП.
3. Невозможность подделки справочника сертификатов и стоп-листа абонентов. Выполняется с помощью проверки подписи УЦ под сертификатами и стоп-листом.
4. Открытые ключи ЭЦП не могут быть переданы по сети связи, к которой имеет доступ противник, без использования механизмов обеспечения целостности и достоверности.

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

19.3 Требование по использованию криптоалгоритмов. Шифрование

Использования алгоритма ГОСТ 28147-89 в режиме простой замены возможно только для шифрования неструктурированных данных (ключей, синхропосылок, полученных с ДСЧ).

Для всех режимов алгоритма шифрования следует обеспечить отсутствие перекрытий ключей для режимов гаммирования и гаммирования с обратной связью.

Примечание: Данное свойство может быть обеспечено передачей сообщений в канал связи зашифрованными на ключе, выработанном с использованием ДСЧ. При этом ключ шифрования сообщения может передаваться зашифрованным на ключе парной связи.

Кроме того, рекомендуется:

1. на открытые данные вычислять имитовставку в соответствии с ГОСТ 28147-89;
2. контролировать объем данных зашифрованных на одном ключе (не более 4 Мбайт);
3. формирование синхропосылки следует производить с использованием ДСЧ;
4. предусмотреть в формате данных подлежащих шифрованию информацию, защищающую от повторов ранее переданных сообщений;
5. если для защиты электронных сообщений используются как электронная цифровая подпись, так и шифрование, то сначала вырабатывать подпись, а затем шифровать подписанное сообщение.

20 Конфигурирование СКЗИ «МагПро КриптоПакет»

Для корректного функционирования СКЗИ «МагПро Криптопакет» требуется задание некоторых параметров конфигурации в конфигурационном файле СКЗИ «МагПро Криптопакет».

Поддерживается несколько вариантов поиска конфигурационного файла:

1. Общесистемный конфигурационный файл. Его расположение зависит от операционной системы:

ОС	Расположение файла
Win32	`C:\openssl\openssl.cnf`
Debian Linux	`/etc/ssl-r/openssl.cnf`
Другие Linux	`/opt/cryptopack/openssl.cnf`
FreeBSD	`/usr/local/cryptopack/openssl.cnf`
Solaris	`/opt/cryptopack/openssl.cnf`

2. Конфигурационный файл текущей сессии. Имя этого файла задается переменной окружения `OPENSSL_CONF`
3. Конфигурационный файл текущей операции. Задается опцией `-config` тех команд утилиты `openssl`, которые поддерживают конфигурационные файлы для текущей операции, или соответствующими опциями других приложений.

При любой операции используется только один конфигурационный файл. Поэтому, если используется файл текущей сессии или текущей операции, в него должны быть перенесены из общесистемного файла настройки СКЗИ «МагПро Криптопакет».

20.1 Конфигурирование поддеркжи алгоритмов ГОСТ

По умолчанию библиотеки libcrypto и libssl, входящие в состав СКЗИ «МагПро Криптопакет», предоставляют только ту же самую функциональность, что и соответствующие библиотеки из состава OpenSSL 0.9.8.

Чтобы включить поддержку алгоритмов ГОСТ, необходимо описать в конфигурационном файле подключение модуля engine libcryptocom.

При установке соответствующего пакета автоматически выполняется подключение этого модуля в системном конфигурационном файле.

Для подключения необходимо добавить в конфигурационный файл следующую информацию:

1. До названия первой секции (первая строка [в квадратных скобках]) следует поместить команду `openssl_conf`, указывающую на секцию с глобальными параметрами конфигурации (по умолчанию этой секции не существует в файле, ее необходимо добавить):

   ```
   openssl_conf = openssl_def
   ```

2. Добавить в конфигурационный файл (например, в конец файла) секцию, указанную выше, и вставить в нее команду `engines`, указывающую на секцию со списком модулей, которые необходимо подгрузить:

   ```
   [openssl_def]
   engines = engine_section
   ```

3. Добавить в конфигурационный файл секцию `engines`, содержащую строку с ID МагПро Engine и название секции, описывающей его конфигурацию:

```
[engine_section]
cryptocom = cryptocom_section %(для Cryptocom Engine)
```

4. Добавить в конфигурационный файл секцию, описывающую конфигурацию библиотеки. Эта секция должна содержать по меньшей мере две строки — в одной указывается полный путь к модулю, во второй указывается его ID.

```
[cryptocom_section]
engine_id = cryptocom
default_algorithms = ALL
```

Кроме этого, в секцию конфигурации библиотеки libcryptocom может быть включена конфигурационная информация самой библиотеки.

Эта информация включает в себя три параметра:

`RNG` — Тип датчика случайных чисел. Допустимые значения: `PROGRAM`, `YARROW`, `ACCORD`, `SOBOL`.

`RNG_PARAMS` — Дополнительные параметры датчика случайных чисел. Для программного датчика этот параметр указывает на расположения файла начального заполнения программного ДСЧ, если его местоположение не совпадает с умолчательным.

Для датчика YARROW этот параметр указывает TCP-порт на сетевом loopback интерфейсе, через который осуществляется доступ к yarrowd. По умолчанию 7670.

При установке пакета значение параметра `RNG` в системном конфигурационом файле устанавливается в `YARROW`. Если вы не планируете использовать на данной машине датчик YARROW и не устанавливаете соответствующий пакет, необходимо отредактировать конфигурационный файл вручную.

`CRYPT_PARAMS` — Параметры алгоритма шифрования ГОСТ 28147-89. Значением опции является OID параметров алгоритма шифрования (см табл. 2), который будет использоваться для зашифрования документов. На работу TLS этот параметр не влияет, так как параметры шифрования жестко фиксированы в спецификации шифрсьютов TLS.

Эти параметры конфигурации могут быть также заданы в environment с помощью переменных с теми же именами и значениями. Значения, заданные в environment, имеют приоритет перед значениями в конфигурационном файле.

Некоторые приложения (например, apache/mod_ssl, stunnel, openvpn) не считывают конфигурационный файл libcrypto, а предоставляют собственные средства конфигурации, позволяющие загружать модули engine. При использовании этих приложений необходимо в их конфигурационном файле указать использование engine с идентификатором cryptocom, а параметры библиотеки libcryptocom передавать через environment.

20.2 Конфигурирование информации о владельце сертификатов

Сертификаты формата X.509 содержат информацию о владельце сертификата, в том числе название организации, подразделения, местонахождение и так далее.

Эта информация может задаваться явно при создании заявки на получение сертификата, но в системном конфигурационном файле могут быть указаны умолчательные значения, которые позволят при создании заявки на сертификат указывать только собственно имя владельца.

В начале файла `openssl.cnf` определяются следующие переменные

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения

Имя	значение
C	страна, двухбуквенный код по ISO-630
L	местоположение (название населенного пункта)
O	название организации
OU	название подразделения

Если в значениях этих переменных используются русские буквы, они должны быть в кодировке UTF-8.

Значения переменных CN (имя владельца сертификата) и E (адрес электронной почты), заданные в конфигурационном файле, при создании заявок с помощью скрипта `mkreq` не используются никогда. Они могут быть использованы только при непосредственном вызове команды `openssl req` из командной строки.

Значение адреса электронной почты по умолчанию в Unix-системах формируется из имени текущего пользователя и доменного имени, содержащегося в файле `/etc/mailname`, а в Windows берется из настроек Microsoft Internet Account Manager (т.е. будет использован адрес первого найденного аккаунта Outlook или Outlook Express).

В случае отсутствия файла `/etc/mailname` или отсутствия настроенного аккаунта Outlook Express скрипт `mkreq` завершается с ошибкой, если адрес электронной почты не был указан явно.

20.3 Формат файла конфигурации библиотеки libcrypto

В библиотеке libcrypto приложения могут автоматически конфигурировать определенные аспекты библиотеки с использованием основного конфигурационного файла СКЗИ «МагПро Криптопакет» или опционально - с использованием альтернативного конфигурационного файла. Утилита openssl включает эту функциональность: любая команда утилиты использует основной конфигурационный файл, если в этой команде не указана опция, использующая альтернативный конфигурационный файл.

Чтобы включить конфигурирование библиотеки, умолчательный раздел конфигурационного файла должен содержать соответствующую строку, указывающую на главный конфигурационный раздел. Имя, используемое утилитой openssl по умолчанию - openssl_conf. Другие приложения могут использовать альтернативные имена, например myapplication_conf.

Конфигурационный раздел должен состоять из набора пар "имя-значение которые содержат информацию о конфигурации конкретных модулей. Имя представляет имя конфигурационного модуля, значение же зависит от характера модуля; например, оно может представлять собой дальнейший конфигурационный раздел, содержащий информацию, специфичную для этого конфигурационного модуля. Например:

```
openssl_conf = openssl_init

[openssl_init]

oid_section = new_oids
engines = engine_section

[new_oids]
```

```
... Здесь новые OID ...

[engine_section]

... Здесь информация об engine ...
```

В этом примере указаны два конфигурационных модуля. Один для объектов ASN.1, другой для конфигурации ENGINE.

20.3.1 Конфигурационный модуль для объектов ASN.1

Имя этого модуля - oid_section. Значение этой величины указывает на раздел, содержащий пары «имя - значение» для OID: имя - короткое и длинное имена OID, значение - численная форма OID. Хотя некоторые из команд утилиты openssl уже имеют собственную функциональность раздела для объектов ASN.1, но не все. С использованием данного конфигурационного модуля все команды утилиты openssl и все приложения, вызывающие конфигурирование, могут видеть новые объекты. Например:

```
[new_oids]

some_new_oid = 1.2.3.4
some_other_oid = 1.2.3.5
```

Также возможно указать в качестве значения длинное имя, за которым следуют запятая и численная форма OID. Например:

```
shortName = some object long name, 1.2.3.4
```

20.3.2 Конфигурационный модуль ENGINE

Этот конфигурационный модуль ENGINE называется engines. Значение этой переменной указывает на раздел, содержащий дальнейшую конфигурационную информацию ENGINE.

Раздел, на который указывает переменная engines, содержит таблицу имен различных engine (хотя см. ниже engine_id) и дальнейших разделов, содержащих конфигурационную информацию, относящуюся к конкретным ENGINE.

Каждый раздел, относящийся к конкретному ENGINE, используется для установки умолчательных алгоритмов, загрузки динамической информации, выполнения инициализации и отдачи управляющих команд. Какая именно операция выполняется, зависит от имени команды, которое указано в качестве имени в паре «имя - значение». Команды, поддерживаемые в настоящее время, перечислены ниже.

Например:

```
[engine_section]

# Конфигурация ENGINE с именем "foo"
```

```
foo = foo_section
# Конфигурация ENGINE с именем "bar"
bar = bar_section

[foo_section]
... Команды, относящиеся к foo ENGINE ...

[bar_section]
... Команды, относящиеся к "bar" ENGINE ...
```

Команда engine_id используется для явного указания имени ENGINE. Если эта команда используется, она должна быть первой. Например:

```
[engine_section]
# здесь должен описываться ENGINE "foo"
foo = foo_section

[foo_section]
# Для этого ENGINE используется имя "myfoo" вместо "foo"
engine_id = myfoo
```

Команда dynamic_path загружает и добавляет ENGINE, находящийся по указанному пути. Она эквивалентна отдаче ENGINE dynamic (ENGINE, отвечающему за загрузку других ENGINE) управляющих команд: SO_PATH с аргументом path, затем LIST_ADD со значением 2 и LOAD. Если требуется другая модель загрузки, ее можно реализовать, передавая нужный набор управляющих команд непосредственно ENGINE dynamic (с помощью соответствующих управляющих команд).

Команда init определяет, инициализировать ENGINE или нет. Если ее значение 0, ENGINE не будет инициализирован, если ее значение равно 1, делается попытка немедленно инициализировать ENGINE. Если команда init не присутствует, то будет попытка инициализировать ENGINE после того, как будут выполнены все остальные команды из данного раздела.

Команда default_algorithms устанавливает умолчательные алгоритмы, которые ENGINE будет предоставлять при использовании функций **ENGINE_set_default_string()**.

Если имя команды не соответствует ни одной из указанных выше команд, считается, что это управляющая команда, которая посылается ENGINE. Значение этой команды передается в качестве аргумента команды. Если значением является строка EMPTY, никаких аргументов не передается.

Например:

```
[engine_section]

# Конфигурация ENGINE "foo"
foo = foo_section
```

```
[foo_section]
# Загрузка engine из DSO
dynamic_path = /some/path/fooengine.so
# управляющая команда, специфичная для foo.
some_ctrl = some_value
# еще одна управляющая команда, не имеющая аргументов.
other_ctrl = EMPTY
# Предоставить все умолчательные алгоритмы
default_algorithms = ALL
```

20.3.3 Примечания

Если конфигурационный файл пытается подставить значение несуществующей переменной, возвращается ошибка и файл не загружается. Это может произойти при попытке подставить значение несуществующей переменной среды.

Это можно обойти, включив умолчательное значение в умолчательный раздел: тогда, если поиск среди переменных среды не удастся, будет использована умолчательная переменная. Чтобы это работало корректно, умолчательная величина должна быть в конфигурационном файле определена до подстановки. В разделе 20.3.4 указан пример того, как это делается.

Если одна и та же переменная в одном и том же разделе упоминается несколько раз, то все значения, кроме последнего, игнорируются. В определенных обстоятельствах, например в distinguished name сертификата, одно и то же поле может встречаться несколько раз. Это обычно обходится с помощью игнорирования всех символов до первой точки. Например:

```
1.OU="My first OU"
2.OU="My Second OU"
```

20.3.4 Примеры

Здесь приводится пример конфигурационного файла, использующего некоторые возможности, указанные выше.

```
# Это умолчательный раздел.

HOME=/temp
RANDFILE= ${ENV::HOME}/.rnd
configdir=$ENV::HOME/config

[ section_one ]

# Теперь мы в первом разделе.

# Кавычки позволяют включать пробелы в конце и начале значений.
any = " any variable name "

other = A string that can \
```

```
cover several lines \
by including \\ characters

message = Hello World\n

[ section_two ]

greeting = $section_one::message
```

Следующий пример показывает, как выполняется безопасная подстановка переменных среды.

Предположим, что вы хотите, чтобы переменная tmpfile указывала на временный файл. Каталог, в котором она находится, может быть определен переменной среды TEMP или TMP, но этим переменным может вообще не быть присвоено никакого значения. Если вы просто укажете имена этих переменных среды, а соответствующие переменные окажутся несуществующими, это приведет к ошибке при попытке загрузить файл конфигурации. С использованием умолчательного раздела можно учесть значения обеих перменных, причем в данном случае TEMP имеет приоритет, а если ни одна не указана, используется /tmp.

```
TMP=/tmp
# Вышеуказанная величина используется, если переменная среды TMP
не определена
TEMP=$ENV::TMP
# Вышеуказанная величина используется, если переменная TEMP не
определена
tmpfile=${ENV::TEMP}/tmp.filename
```

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

Порядковый № изменения	Подпись лица, ответственного за изменение	Дата внесения изменения