

УТВЕРЖДЕН
СЕИУ.00009-01 33 01 - ЛУ

Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

СРЕДСТВО КРИПТОГРАФИЧЕСКОЙ ЗАЩИТЫ ИНФОРМАЦИИ
MagPro КриптоПакет вер. 1.0

**Библиотека libcrypto.
Руководство программиста**

СЕИУ.00009-01 33 01
Листов 55

Литера О

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).

Авторские права на СКЗИ «МагПро КриптоПакет» принадлежат ООО «Криптоком». В продукте использован исходный код OpenSSL, ©The OpenSSL Project, 1998-2004. «МагПро» является зарегистрированным товарным знаком ООО «Криптоком».

Содержание

1	Создание и обработка заявок, сертификатов и списков отзыва	5
1.1	Формирование заявки	5
1.2	Анализ заявок, сертификатов и CRL	7
1.2.1	Серийный номер сертификата	7
1.2.2	Сроки действия сертификата и CRL	7
1.2.3	Наименование владельца сертификата/заявки и УЦ выпустившего сертификат/CRL	8
1.2.4	Расширения X509v3	8
1.2.5	Низкоуровневые функции проверки подписи	9
1.3	Проверка соответствия сертификата и секретного ключа	9
1.4	Манипулирование X509 Distinguished name.	9
1.4.1	Создание и освобождение	9
1.4.2	Добавление элементов	10
1.4.3	Просмотр и поиск элементов	10
1.4.4	Анализ отдельного поля имени	11
1.4.5	Манипуляции с именем в целом	11
1.5	Манипуляции с расширениями X509v3	12
2	Создание и обработка сообщений S/MIME	13
2.1	Создание зашифрованного сообщения S/MIME	17
2.2	Расшифрование зашифрованного сообщения S/MIME	17
2.3	Создание подписанного сообщения S/MIME	17
2.4	Проверка подписи под сообщением S/MIME	18
3	Функции, предназначенные для создания и обработки сообщений S/MIME	19
3.1	Флаги в параметрах функций	19
3.2	Зашифрование данных: PKCS7_encrypt	19
3.3	Расшифрование сообщения: PKCS7_decrypt	19
3.4	Подпись данных: PKCS7_sign	20
3.5	Добавление второй подписи: PKCS7_add_sign	20
3.6	Проверка подписей: PKCS7_verify	21
3.7	Запись сообщения S/MIME: SMIME_write_PKCS7	21
3.8	Чтение сообщения S/MIME: SMIME_read_PKCS7	22
3.9	Другие способы чтения и записи структур PKCS7	22
4	Подсчет хэш-сумм	24
4.1	Пример хэширования данных	25
5	Симметричное шифрование	27
5.1	Формирование ключа из пароля	31
6	Работа с ключевыми парами	32
6.1	Загрузка закрытого ключа из файла.	32
6.2	Загрузка закрытого ключа из аппаратного хранилища.	32
6.2.1	Формат идентификаторов ключа на аппаратных носителях	32
6.2.2	Создание методов взаимодействия с пользователем	33

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

7	Работа с цепочками сертификатов и хранилищем доверенных сертификатов УЦ	35
7.1	Создание хранилища сертификатов	35
7.2	Параметры проверки сертификатов	36
7.3	Процедура проверки сертификата	37
7.4	Процедура поиска нужного сертификата	38
7.5	Функция обратного вызова проверки сертификатов	39
7.6	Метод поиска сертификатов в файле	39
7.7	Метод поиска сертификатов в директории	40
7.8	Создание собственных методов поиска сертификатов	40
8	Работа с протоколом онлайн проверки статус сертификатов	43
8.1	Формирование OCSP-запроса	43
8.2	Анализ OCSP-ответа	44
8.3	API для реализации сервера OCSP	47
9	Служебные типы и функции	50
9.1	BIO	50
9.1.1	Файловый BIO	51
9.1.2	Сокетный BIO	51
9.1.3	BIO в памяти	51
9.2	API стеков OpenSSL	52
9.3	Аргументы функций чтения и записи формата PEM	53
9.4	Вывод значений строк из ASN.1 структур	53

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

1 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 абстракции ввода вывода BIO.

Интерфейс этих функций единообразен для всех типов данных и описан в разделе 9.3

1.1 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 1.4);
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 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` с помощью `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* представляет собой структуру `X509_NAME`, способы создания которой описаны в разделе 1.4.

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

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

Устанавливает публичный ключ заявки из указанной структуры ключевой пары. См раздел 6.

```
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 соответствующей 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 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 1.5.

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

Подписывает заявку закрытый 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 алгоритмами ГОСТ Р 34.10 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 4).

Следует иметь в виду, что после 31 декабря 2007 года алгоритм ГОСТ Р 34.10-94 должен использоваться только для проверки ранее выработанных подписей; ключи для этого алгоритма создаваться не должны.

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

1.2 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 загрузки PEM или DER-формата, описанные в разделе 9.3

1.2.1 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. Освободить 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 структуре 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 представлять собой 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**.

1.2.2 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).

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

Значение ASN1_TIME может быть выведено с помощью функции

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

1.2.3 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 выпустившего УЦ (issuer) сертификата и CRL. Функции анализа структуры X509_NAME описаны в разделе 1.4.

1.2.4 Расширения X509v3

```
int X509_get_ext_count(X509 *cert)
```

```
int X509_CRL_get_ext_count(X509_CRL *x);
```

Возвращает количество расширений, содержащихся в сертификате /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 1.5

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

1.2.5 Низкоуровневые 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.

1.3 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 несоответствия 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.

1.4 Манипулирование X509 Distinguished name.

1.4.1 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.

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

1.4.2 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. Многочисленные 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.

1.4.3 Просмотр и поиск элементов

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

Возвращает число 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
```

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

```
*name, ASN1_OBJEКТ *obj, int lastpos);
```

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

1.4.4 Анализ отдельного поля имени

```
ASN1_OBJEКТ * X509_NAME_ENTRY_get_object(X509_NAME_ENTRY
    *ne);
```

Возвращает OID имени в виде структуры ASN1_OBJEКТ.

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 помощью **ASN1_STRING_print_ex**, описанной в разделе 9.4.

1.4.5 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. Задаёт 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* задаёт отступ от начала строки (используется в основном, если используемые флаги задают многострочный вывод).

Определены следующие 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.

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.

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

XN_FLAG_SPC_EQ — Выводить 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 9.4.

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.

1.5 Манипуляции с расширениями X509v3

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

Получает OID указанного расширения.

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

Получает данные указанного расширения.

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

Получает флаг критичности.

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

2 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/sectkey.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/sectey.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

2.1 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**.

2.2 Расшифрование зашифрованного сообщения S/MIME

Типичные действия при расшифровании зашифрованного сообщения S/MIME:

1. Загрузить свой 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.

2.3 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 свой 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 (ВВО) 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**.

2.4 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 (ВВО) для входного сообщения и выходных данных. В примере оба объекта построены на файлах.
3. Подготовить указатель на ВВО 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 ВВО 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.

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

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

3.1 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_CRLFEOF`, `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).

3.2 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.

3.3 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

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

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.

3.4 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_NOSMIMESCAP — не формировать атрибут `SMIMESCapabilities`.

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**.

3.5 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**.

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

3.6 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 7;
- 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**).

3.7 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 отдельной 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 отдельная. 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 должен быть указан при вызове **PKCS7_sign**, сформировавшем структуру *p7*.

3.8 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 отдельной 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.

3.9 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 9.3.

```
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 9.3.

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

4 Подсчет хэш-сумм

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 предоставляется подгружаемым модулем, то можно указывать `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`.

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)
```

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

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_MD_CTX_cleanup(EVP_MD_CTX *ctx)
```

Очищает использованную структуру `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`, после получения значения хэш-суммы выполняет очистку структуры.

```
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.

```
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 `EVP_MD_CTX`.

4.1 Пример хэширования данных

```
void hashfd(int fd)
{
    EVP_MD_CTX *ctx;
    ssize_t blocksize;
```

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

```

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

5 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.

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 функций, возвращающих информацию 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 ??).

```
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_CIPHER_STREAM_CIPHER` — потоковый шифр

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

EVP_CIPHER_ECB_MODE — режим простой замены (ECB)
EVP_CIPHER_CBC_MODE — режим сцепления блоков шифртекста (CBC)
EVP_CIPHER_CFB_MODE — режим гаммирования с обратной связью (CFB)
EVP_CIPHER_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 ор-комбинацию следующих констант:

EVP_CIPHER_VARIABLE_LENGTH — шифр переменной длины
EVP_CIPHER_CUSTOM_IV — шифр требует пользовательского вектора инициализации
EVP_CIPHER_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_CIPHER_CTRL_INIT — Требуется инициализация параметров посредством **EVP_CIPHER_CTX_ctrl**
EVP_CIPHER_CUSTOM_KEY_LENGTH — Используется нестандартный способ определения длины ключа.
EVP_CIPHER_NO_PADDING — не используется стандартный способ дополнения до полной длины блока.
EVP_CIPHER_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 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, unsigned char *iv)
```

```
int EVP_CipherInit_ex(EVP_CIPHER_CTX *ctx, const EVP_CIPHER
    *type, ENGINE *impl, unsigned char *key, unsigned char *iv, int
    env)
```

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

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 расшифровании.

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.

Кроме того если параметр `cipher` не равен `NULL`, они производят очистку контекста вызовом **EVP_CIPHER_CTX_init**.

Поэтому при использовании этих функций 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 результата зашифрования/расшифрования в буфер `out` и длину полученного шифртекста/открытого текста в `outl`.

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

В ряде режимов шифров объем результата может отличаться от объема исходных данных, например в случае блочных шифров последние байты данных, не составляющие целого блока не будут обработаны 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 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 блочных 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 чувствительные 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)
```

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

5.1 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.

Созданный ключ и вектор инициализации помещаются в буфера, указатели на которые передаются в параметрах `key` и `iv` соответственно.

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

6 Работа с ключевыми парами

Ключевые пары асимметричных 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.

Поскольку генерация долговременных ключевых пар производится с помощью программы `makekey`, а эфемерные ключевые пары используемые в S/MIME и TLS создаются внутри соответствующих функций, в данном разделе рассматривается только API загрузки закрытых ключей.

6.1 Загрузка закрытого ключа из файла.

Для загрузки закрытого ключа из файла используется функция `PEM_read_bio_PrivateKey`, аналогичная по интерфейсу остальным функциям чтения, описанным в разделе 9.3.

6.2 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`.

key_id идентификатор ключа на аппаратном токене. См раздел 6.2.1

ui_method указатель на набор функций, используемых для взаимодействия с пользователем. CRYPTOcom engine использует его только для запроса пароля, на котором зашифрован ключ.

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.

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

Возвращает указатель на engine по её идентификатору. Например:

```
ENGINE *e=ENGINE_by_id("cryptocom");
```

6.2.1 Формат идентификаторов ключа на аппаратных носителях

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

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

Где ТИП-УСТРОЙСТВА может быть ACCORD, 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.

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

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

6.2.2 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 вызывается, 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.

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

Все функции, входящие в состав метода возвращают 1 в случае успешного завершения, 0 в случае ошибки, и -1 в случае возникновения специальной ситуации, 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. Получить доступ к содержащейся в ней информации можно с помощью следующих функций:

```
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, предполагающий 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

7 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`.

7.1 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 7.6) и

```
X509_LOOKUP_METHOD *X509_LOOKUP_hash_dir(void);
```

(см. раздел 7.7).

Кроме того, приложение может реализовать собственный метод поиска сертификатов, позволяющий хранить доверенные корневые сертификаты и списки отзыва, например, в базе данных (см. раздел 7.8).

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 7.5.

Также можно установить ряд параметров поиска и верификации:

- Флаги, указывающие на набор выполняемых проверок (функция **X509_STORE_set_flags**).

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

- Область применения сертификата, которая требуется в данном случае (функция **X509_STORE_set_purpose**).
- Признак доверенности (функция **X509_STORE_set_trust**).

7.2 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. Область 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 внутренней 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 идентификатор, 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 корневому 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.

7.3 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. структура 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);
```

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

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 7.4).

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`.

7.4 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*.

```
struct x509_object_st
{
    int type;
```

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

```

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.

7.5 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 СА, 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 незначительна, и вернуть 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`

7.6 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)
    
```

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

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 использовать метод поиска сертификатов в хэшированной директории.

7.7 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`, сгенерированные 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`.

7.8 Создание собственных методов поиска сертификатов

Для создания собственных методов поиска сертификатов следует определить функции, 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)
```

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

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_shutdown`.

```
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_OBJECT *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.

В стандартных методах не реализована.

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

```
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.

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

8 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 протокола 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.

8.1 Формирование OCSP-запроса

`OCSP_REQUEST* OCSP_REQUEST_new()`

Создает пустую структуру `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 структуры 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 удостоверяющего центра и серийный номер сертификата.

`OCSP_CERTID *OCSP_cert_id_new(const EVP_MD *dgst, X509_NAME
*issuerName,`

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

```
ASN1_BIT_STRING* issuerKey, ASN1_INTEGER *serialNumber);
```

Позволяет сформировать OSCP_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 1.2.

Если какая-нибудь из структур, передаваемых 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 OSCP_request_add1_nonce(OSCP_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 попсе. Если в качестве длины передано `-1`, в качестве длины будет использована константа `OSCP_DEFAULT_NONCE_LENGTH`. Проверка попсе впоследствии осуществляется путем сравнения структуры запроса со структурой полученного ответа с помощью функции **OSCP_check_nonce** (см раздел 8.2).

Необязательная подпись под запросом может быть выработана с помощью функции:

```
int OSCP_request_sign(OSCP_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 OSCP-сервером) и поле флагов.

В настоящий момент поддерживается единственный флаг `OSCP_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_OSCP_REQUEST` или `i2d_OSCP_REQUEST_bio`, аналогичный прочим `i2d`-функциям.

8.2 Анализ OSCP-ответа

Разбор полученного от сервера OSCP-ответа производится с помощью функций `d2i_OSCP_RESPONSE`, `d2i_OSCP_RESPONSE_bio`, `d2i_OSCP_RESPONSE_fp`. Полученная структура `OSCP_RESPONSE` должна быть впоследствии освобождена вызовом **OSCP_RESPONSE_free**. При этом будут также освобождены все данные, на которые указывают указатели типов `OSCP_BASICRES`, `OSCP_SINGLERESP` и `ASN1_GENERALIZEDTIME` получаемые с помощью функций, описанных в данном разделе.

```
int OSCP_response_status(OSCP_RESPONSE *resp)
```

Возвращает статус — одну из констант:

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

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 следующей 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,

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

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 — Организация 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

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

```
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_SINGLERESP` 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.

8.3 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 идентификатор сертификата, содержащийся в запросе.

```
int OCSP_id_get0_info(ASN1_OCTET_STRING **piNameHash,
    ASN1_OBJECT **pmd,
    ASN1_OCTET_STRING **pikeyHash,
```

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

```
ASN1_INTEGER **pserial, OCSPP_CERTID *cid);
```

Возвращает информацию о идентификаторе сертификата, в частности OID использованного алгоритма хеширования и серийный номер сертификата.

Для определения соответствующего сертификата удостоверяющего центра, OCSPP-сервер должен подготовить структуры OCSPP_CERTID с использованием того же алгоритма хеширования с помощью функции **OCSPP_cert_to_id** используя NULL в качестве идентифицируемого сертификата, и сравнить их с содержащимся в запросе запроса с помощью функции

```
int OCSPP_id_issuer_cmp(OCSPP_CERTID *a, OCSPP_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_OCSPP_CERTSTATUS_UNKNOWN).

Информация о статусе сертификата добавляется в структуру OCSPP_BASIC_RESP с помощью функции

```
OCSPP_SINGLERESP *OCSPP_basic_add1_status(OCSPP_BASICRESP *rsp,
    OCSPP_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 **OCSPP_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 помощью функции

```
int OCSPP_copy_nonce(OCSPP_BASICRESP *resp, OCSPP_REQUEST *req);
```

После этого ответ подписывается с помощью функции:

```
int OCSPP_basic_sign(OCSPP_BASICRESP *brsp,
    X509 *signer, EVP_PKEY *key, const EVP_MD *dgst,
    STACK_OF(X509) *certs, unsigned long flags);
```

аналогичной **OCSPP_request_sign**. Эта функция поддерживает следующие флаги:

OCSPP_NOCERTS — Не включать в ответ сертификат ключа, которым выработана подпись и сертификаты, указанные в параметре **certs**.

OCSPP_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 хеширования, и использование именно этого алгоритма обязательно согласно RFC 2560.

и преобразуется в OCSPP_RESPONSE с помощью функции

```
OCSPP_RESPONSE *OCSPP_RESPONSE_create(int status, OCSPP_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.

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

9 Служебные типы и функции

9.1 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, сокет, 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 понадобится работать автору приложения, использующего МагПро КриптоПакет.

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

9.1.1 Файловый ВІО

Файловый ВІО построен поверх стандартного типа `FILE`.

Создать файловый ВІО можно одной из следующих функций:

```
ВІО *ВІО_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.

```
ВІО *ВІО_new_fp(FILE *stream, int flags);
```

Созданный объект привязывается к потоку *stream*. Возможные флаги:

`ВІО_CLOSE` — закрыть поток при выполнении `ВІО_free`.

`ВІО_NOCLOSE` — не закрывать поток при выполнении `ВІО_free`. Этот флаг следует указывать, если объект привязывается к одному из стандартных потоков `stdin`, `stdout` или `stderr`.

`ВІО_FP_TEXT` — установить поток в текстовый режим (влияет на работу только под Win32).

Файловый ВІО поддерживает функции `ВІО_gets` и `ВІО_puts`, а также `ВІО_flush` (вызывает `fflush(stream)`), `ВІО_reset` (вызывает `fseek(stream, 0, 0)`), `ВІО_seek` (вызывает `fseek(stream, offset, 0)`), `ВІО_tell` (вызывает `ftell(stream)`), `ВІО_eof` (вызывает `feof(stream)`).

9.1.2 Сокетный ВІО

Сокетный ВІО строится поверх сокетов.

```
ВІО *ВІО_new_socket(int sock, int close_flag);
```

Если параметр *close_flag* истинен, сокет будет закрыт при вызове `ВІО_free`. Сокетный ВІО поддерживает `ВІО_gets`, `ВІО_puts`, `ВІО_flush` и `ВІО_eof`, но по природе сокетов не поддерживает операцию `seek` и связанные.

9.1.3 ВІО в памяти

ВІО в памяти — это ВІО, 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

```
ВІО *ВІО_new_mem_buf(void *buf, int len)
```

Если *len* равно `-1`, *buf* предполагается ASCIIZ-строкой, и его длина определяется вызовом `strlen`. Вызов `ВІО_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 `ВІО_new(ВІО_s_mem())`. При записи в такой объект необходимая память будет выделяться автоматически. При чтении из такого объекта прочитанные данные удаляются из него. Вызов `ВІО_reset` для такого объекта очищает его данные.

Оба подтипа поддерживают операции `ВІО_gets` и `ВІО_puts`. Вызов `ВІО_eof` возвращает истину, если в объекте нет данных. Вызов `ВІО_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 данных 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`.

9.2 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

9.3 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 вернуть 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 команднорочных, 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 **fp* — 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.

9.4 Вывод значений строк из ASN.1 структур

```
int ASN1_STRING_print_ex(BIO *out, ASN1_STRING *str, unsigned long flags);
```

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

```
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_STRFLGS_ESC_CTRL` — Выводить символы с кодами 0-32 в виде шестнадцатиричных 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 преобразования 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 шестнадцатиричный 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`.

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

