УТВЕРЖДЕН

РБ.ЮСКИ.08013-01 90 01-ЛУ

БИБЛИОТЕКА ФУНКЦИЙ AVCRYPTMAIL

Описание интерфейса

РБ.ЮСКИ.08013-01 90 01

Листов 152

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

2008

Интерфейс библиотеки AvCryptMail 4.0

Редакция 4.0.0

История создания:

Редакция 4.0.0

Реализована поддерка удостоверяющих подписей (CounterSignature)

В функции **AvCmSign**, **AvCmSignAndEncrypt**, **AvCmMsgSign** добавлен флаг **AVCMF_ADD_ONLY_CERT .**

В функцию **AvCmOpenMsg** добавлены флаги **AVCMF_OPEN_FOR_CALC_BELTHASH**, **AVCMF_OPEN_FOR_CALC_BHFHASH**.

В функцию **AvCmLogin** добавлен флаг **AVCMF_RELOGIN**.

В функцию **AvCmMsgSetAttribute** добавлены атрибуты: **AVCM_PKCS7_CERTS**, **AVCM_PKCS7_CRLS**.

В функцию **AvCmOpenCertEnum** добавлены атрибуты: **AVCM_PKCS7_CERTS**, **AVCM_PKCS7_CRLS**.

Добавлены функции обмена данными по протоколу TLS: **AvCmTLSCreateConnect**, **AvCmTlsGet**, **AvCmTlsPost**.

Добавлена функция вывода информации в отладочный лог: **AvCmDebugLog**.

Редакция 3.4.4

В функцию **AvCmGetCertAttr** добавлены параметры **AVCM_POLICYINFO_COUNT, AVCM_POLICYINFO_OID, AVCM_LDAP_PATH, AVCM_BASE_SERIAL_AS_STRING, AVCM_BASE_ISSUER_AS_STRING**.

Редакция 3.3.7

Простой протокол регистрации сертификата SCEP (**AvCmCreateScep**, **AvCmScepGet**, **AvCmScepSet**, **AvCmScepExequte**).

Редакция 3.3.4

В функции **AvCmGetCrlAttr** – параметр **AVCM_PUB_KEY_ALG_OID** заменен на **AVCM_SIGN_ALG_OID**.

В функцию **AvCmGetCertAttr** добавлен параметр **AVCM_AUTHORITY_KEY_IDENTIFIER**.

Редакция 3.3.3

В функции **AvCmGetCertAttr**, **AvCmOpenCertEnum**, **AvCmEnumGet** добавлена поддержка атрибутных сертификатов.

Для функции **AvCmLogin** добавлен параметр **AVCM_SLOTID**.

Добавлена функция **AvCmMsgSetAttribute** добавления дополнительных атрибутов в подписываемое сообщение

18 декабря 2009 г. – версия 3.0.0
Добавлена функция **CmCheckCrlDP** контроля точек распространения СОС
12 июня 2009 г. – версия 2.36.0
Добавлена обработка ответов **OCSP сервера**.
Добавлена функция **AvCmMsgOCSPGetResponse** получение дескриптора информации о статусе сертификата.
Добавлена функция **AvCmMsgOCSPGetResponseAttr** извлечение атрибутов информации о статусе сертификата.
Добавлена функция **AvCmMsgOCSPAddResponse** добавление информации о статусе сертификата в подписанное сообщение.
В функцию **AvCmGetSignAttr** добавлены атрибуты **AVCM_OCSP_SIGN_RESPONSE_COUNT**, **AVCM_OCSP_SIGN_RESPONSE_BYNUM.**
В функцию **AvCmGetCertAttr** добавлен параметр **AVCM_CERT_SHA1**.
В функцию **AvCmOpenCertEnum**, добавлен параметр **AVCM_TYPE.**

3 января 2008 г. – версия 2.32.3
Добавлена функция **AvCmMsgVerifySignAtIndexForDate** проверка подписи в сообщении по номеру на заданную дату.

3 января 2008 г. – версия 2.32.3
Добавлена функция **AvCmMsgUpdate** поблочной обработки данных сообщения.
В функцию **AvCmOpenMsg** добавлены флаги **AVCMF_OPEN_FOR_SIGN, AVCMF_OPEN_FOR_VERIFYSIGN, AVCMF_DETACHED**.

27 июня 2007 г. – версия 2.29.2
В функцию **AvCmImport** добавлен атрибут **AVCM_OPERATION_REPORT**.
В функцию **AvCmGenerateRequest** добавлен атрибут **AVCM_OPERATION_REPORT**.
В функцию **AvCmImport** добавлен флаг **AVCMF_SELECT_MY_CERT**.

26 июня 2007 г. – версия 2.29.1
Изменены функции **AvCmMsgVerifySign**, **AvCmMsgVerifySignAtIndex**, **AvCmVerifyRawDataSign** – добавлен флаг **AVCMF_NO_CRL_VERIFY**.
В функцию **AvCmGenerateRequest** добавлен атрибут **AVCM_EXPORT_PATH**.
В функцию **AvCmImport** добавлен атрибут **AVCM_IMPORT_PATH**.

11 июня 2007 г. – версия 2.28.6
Изменены функции **AvCmSignRawData** и **AvCmVerifyRawDataSign**, добавлен флаг **AVCMF_RAW_SIGN**.
Функция **AvCmMsgGetSTBSign** объявлена как устаревшая.

29 мая 2007 г. – версия 2.28.5
В функцию **AvCmGenerateRequest** добавлен атрибут **AVCM_CERT_PROLONGATION,** добавлен флаг **AVCMF_ALLOW_TO_SELECT_FILE**.
В функцию **AvCmImport** добавлен атрибут **AVCM_ANY_FILE**.

18 мая 2007 г. – версия 2.28.4
В функцию **AvCmGetCertAttr** добавлен параметр **AVCM_CRL_DISTRIBUTION_POINTS**.

14 октября 2005 г. – версия 2.23.3
Доработана функция **AvCmOpenCertEnum**, добавлен параметр **AVCM_P_NON_REPUDIABLE**.
Доработана функция **AvCmVerifyCertStatus**.
Добавлена функция **AvCmVerifyRawDataSign**.

29 сентября 2005 г. – версия 2.23.2
Добавлена функция **AvCmVerifyCertStatus**.
Добавлена функция **AvCmGetCertStatusAttr**.
Добавлен код ошибки **AVCMR_CERT_NOT_REVOKED.**
Изменена функция **AvCmMsgVerifySignAtIndex**, добавлен флаг **AVCMF_NO_CERT_VERIFY.**

14 сентября 2005 г. – версия 2.23.1
Добавлена функция **AvCmSignRawData**.
Для функции **AvCmLogin** добавлен параметр **AVCM_DBT_E_MEMORY**.
Для функции **AvCmImport** добавлен параметр **AVCM_STORES**.

8 сентября 2005 г. – версия 2.22.12
Добавлена функция **AvCmFlush**.

18 июля 2005 г. – версия 2.22.9
В функцию **AvCmGetRequestAttr** добавлен параметр **AVCM_MSCA_COMPATIBLE**.
В функцию **AvCmGetCertAttr** добавлены параметры **AVCM_SUBJECT_ATTR_COUNT, AVCM_ISSUER_ATTR_COUNT, AVCM_SUBJECT_ATTR_OID, AVCM_ISSUER_ATTR_OID, AVCM_EXT_BLOB.**
В функцию **AvCmGetSignAttr** добавлены параметры: **AVCM_AUTH_OID, AVCM_UNAUTH_OID, AVCM_AUTH_BLOB, AVCM_UNAUTH_BLOB.**

25 апреля 2005 г. – версия 2.22.1
Добавлена функция **AvCmMsgAddCert**.

6 апреля 2005 г. – версия 2.21.0

Для функций **AvCmMsgVerifySignAtIndex**, **AvCmVerifySign**, **AvCmDecryptAndVerifySign**, **AvCmMsgVerifySign** добавлен флаг **AVCMF_VERIFY_ON_SIGN_DATE**.

28 марта 2005 г. – версия 2.20.7

Для функции **AvCmImport** добавлены параметры **AVCM_PKCS10_REQUEST, AVCM_PKCS7_REQUEST** и **AVCM_RESULT_HANDLE**.
Изменена функция **AvCmOpenCertEnum**, добавлена ошибка **AVCMR_REQUEST_DENIED**, добавлен флаг **AVCMF_REQUEST_RESULT**.

24 марта 2005 г. – версия 2.20.6

Для функции **AvCmLogin** добавлен флаг **AVCMF_IGNORE_CRL_ABSENCE**.
Для функций **AvCmMsgImportCerts**, **AvCmVerifySign**, **AvCmDecryptAndVerifySign**, **AvCmMsgVerifySign** добавлен флаг **AVCMF_IMPORT_CRL**.

3 марта 2005 г. – версия 2.20.1

Изменена функция **AvCmLogin**, добавлен атрибут **AVCM_COMMON_NAME**.

25 февраля 2005 г. – версия 2.20.0

Добавлен тип данных **AvCmGenReqParam**.
Добавлен тип данных **AvCmHreq**.
Добавлен тип данных **AvCmFindReqParam**..
Добавлена функция **AvCmGenerateRequest**.
Добавлена функция **AvCmGetRequestAttr**.
Добавлена функция **AvCmFindRequest**.
Добавлена ошибка **AVCMR_TOKEN_NOT_FOUND.**
Добавлена ошибка **AVCMR_ NO_CONTENT** .

24 января 2005 г. – версия 2.18.9

Добавлен флаг в функции **AvCmSign**, **AvCmSignAndEncrypt**, **AvCmMsgSign**.

4 января 2005 г. – версия 2.18.6

Добавлен тип данных **AvCmHcrl**.
Добавлен тип данных **AvCmFindCrlParam**.
Добавлена функция **AvCmFindCrl**.
Добавлена функция **AvCmGetCrlAttr**.

21 декабря 2004 г. – версия 2.18.4

Изменена функция **AvCmVerifySign**, добавлен флаг **AVCMF_NO_CERT_VERIFY.**
Изменена функция **AvCmDecryptAndVerifySign**, добавлен флаг **AVCMF_NO_CERT_VERIFY.**
Изменена функция **AvCmMsgVerifySign**, добавлен флаг **AVCMF_NO_CERT_VERIFY.**

8 декабря 2004 г. – версия 2.18.2

Изменена функция **AvCmOpenCertEnum**, добавлен параметр поиска **AVCM_CERT_ISSUERS_CHAIN.**

Добавлена структура данных **AvCmImportParam.**

Добавлена функция **AvCmImport.**

23 ноября 2004 г. – версия 0.14.0

Изменена функция **AvCmLogin**.

2 июня 2004 г. – версия 0.13.0

Добавлена функция **AvCmEnumDlg.**

Добавлена функция **AvCmMsgGetSTBSign.**

Добавлена функция **AvCmShowObjectInfo.**

В функции **AvCmOpenCertEnum** и **AvCmEnumAddCerts** добавлены параметры **AVCM_KEY_NOT_BEFORE, AVCM_KEY_NOT_AFTER.**

В функцию **AvCmGetCertAttr** добавлен параметр **AVCM_BLOB.**

19 мая 2004 г. – версия 0.12.1

Изменён принцип работы с дескрипторами сертификатов.

Добавлена функция **AvCmMsgImportCerts.**

Добавлена функция **AvCmMsgVerifySignAtIndex.**

Из функции **AvCmGetCertAttr** удалён атрибут **AVCM_PROPERTY_AS_STRING**.

07 мая 2004 г. – версия 0.12.0

Удалены флаги **AVCMF_IN_SMIME**, **AVCMF_OUT_SMIME**.

Добавлена функция **AvCmEnumAddCerts.**

Добавлена функция **AvCmInitEx.**

В функцию **AvCmSetMsgContent** добавлена возможность добавить данные в содержимое сообщения при помощи флага **AVCMF_APPEND**. Изменено поведение данной функции.

Добавлены новые типы хранилища в функцию **AvCmLogin**: **AVCM_DBT_FILE, AVCM_DBT_ARCHIVE_FILE, AVCM_DBT_ORACLE, AVCM_DBT_ARCHIVE_MEMORY;** добавлены параметры **AVCM_DB_CONNECTSTR, AVCM_DB_FILE_PATH, AVCM_DB_ARCHIVE_FILE_PATH, AVCM_DB_ARCHIVE_PTR, AVCM_DB_ARCHIVE_SIZE.**

Удалён тип хранилища **AVCM_DBT_ODBC.**

Из функций **AvCmSign**, **AvCmSignAndEncrypt**, **AvCmMsgSign**, удалён флаг **AVCMF_AS_NEW_MESSAGE.**

Из функции **AvCmGetMsgParam** удалена возможность получения формата **MF_DATA.**

27 апреля 2004 г. – версия 0.11.2

Добавлена функция **AvCmOpenInnerMsg.**

Изменено функционирование функции **AvCmMsgDecrypt**.

В функцию **AvCmGetMsgParam** добавлена возможность получения формата вложенного сообщения при помощи атрибута **AVCM_INNER_FORMAT**.

8 декабря 2003 г. – версия 0.8.2

Добавлены параметры **AVCM_PUB_KEY_ID** и **AVCM_PASSWORD** в функцию **AvCmLogin**.

6 декабря 2003 г. – версия 0.8.1

Добавлен флаг **AVCMF_IMPORT** в функции **AvCmVerifySign** и **AvCmDecryptAndVerifySign**.

Добавлен флаг **AVCMF_MESSAGE** в функцию **AvCmPutCert**.

30 октября 2003 г. – версия 0.8.0

Добавлен флаг **AVCMF_ALL_CERT** в функцию **AvCmOpenCertEnum**.

Добавлен параметр **AVCM_VALID** в функцию **AvCmGetCertAttr**.

Добавлена функция **AvCmPutCert**.

15 октября 2003 г. – версия 0.7.0

19 сентября 2003 г. – версия 0.6.0

Добавлена функция **AvCmSetMsgContent**.

Добавлен флаг **AVCMF_DETACHED**.

Удален флаг **AVCMF_EXTRACT**.

Изменена функция **AvCmGetCertAttr**:
добавлены флаги **AVCM_NOT_CERT_BEFORE, AVCM_NOT_CERT_AFTER, AVCM_NOT_KEY_BEFORE, AVCM_NOT_KEY_AFTER**.

23 июля 2003 г. – начало работ над версией 0.5.7

Удален флаг **AVCMF_EXTRACT**.

Добавлена ошибка **AVCMR_OTHER_RECIPIENT.**

2 июля 2003 г. – начало работ над версией 0.5.6

Добавлены флаги **AVCMF_IN_RAW_DATA**, **AVCMF_IN_PKCS7**, **AVCMF_IN_SMIME**, **AVCMF_OUT_PKCS7**, **AVCMF_OUT_SMIME**.

Изменена функция **AvCmGetCertAttr**:
добавлен атрибут **AVCM_EXT_OID_NAME**;
добавлены атрибуты **AVCM_EXT_KEY_USAGE_COUNT, AVCM_EXT_KEY_USAGE_OID, AVCM_EXT_KEY_USAGE_NAME.**

Изменена функция **AvCmOpenCertEnum**:
добавлен атрибут **AVCM_EXT_KEY_USAGE_OID**.

Удален флаг **AVCMF_ADD_SIGN.**

30 июня 2003 г. – версия 0.5.5

Добавлены ошибки:
AVCMR_AVCSP_INIT_FAILED
AVCMR_REGISTRY_ERROR

Для функции **AvCmGetCertAttr** добавлен атрибут **AVCM_PROPERTY_AS_STRING**.

Изменена функция **AvCmGetSignAttr**.

Удалены атрибуты **AVCM_SIGNER_ID_CHOICE, AVCM_ID_ISS_SER_NUM, AVCM_ID_KEY_ID, AVCM_ID_CER_HASH.**

Изменена функция **AvCmFindCertBySign**.

Изменена функция **AvCmGetCertAttr**.

Изменена функция **AvCmOpenCertEnum**.
Изменены наименования атрибутов функций.
Удалена функция **AvCmCheckDate.**
Добавлена ошибка **AVCMR_BAD_FORMED_SIGN.**
Добавлен флаг **AVCMF_AS_NEW_MESSAGE.**

12 июня 2003 г. – версия 0.5.4
Добавлена функция **AvCmGetObjectInfo**.
Изменена функция **AvCmGetCertAttr**.

11 июня 2003 г. – версия 0.5.3
Удален флаг **AVCMF_FORCE_CLOSING.**
Удален флаг **AVCMF_GEN_NO_SIGN.**
Удален флаг **AVCMF_ADD_NO_CERT.**
Удалены параметры функции авторизации: **AVCMU_CONTAINER, AVCMU_ISSUER AVCMU_SERIAL_NUM**
Изменены параметры функций: **AvCmDuplicateHandle**, **AvCmCloseHandle**, **AvCmGetMsgParam**, **AvCmMsgSign**, **AvCmMsgVerifySign**, **AvCmMsgEncrypt**, **AvCmMsgDecrypt**, **AvCmGetMsg**, **AvCmGetMsgContent**, **AvCmGetMsgSign**, **AvCmGetSignAttr**, **AvCmFindCertBySign**, **AvCmGetCertAttr**, **AvCmEnumGet** – удален параметр дескриптор сессии.
Изменена функция **AvCmGetErrorInfo**.

28 мая 2003 г. – версия интерфейса 0.5.1
Изменены функции: **AvCmGetSignAttr**, **AvCmGetCertAttr** .
Добавлены функции: **AvCmCheckDate.**
Добавлен пример проверки ЭЦП.

13 мая 2003 г. – завершение первого варианта интерфейса версии 0.5.

6 мая 2003 г. – начало работ над версией 0.5. Предполагается переработка обработки ошибок и добавления низкоуровневой работы с сообщениями. Предполагается значительная переработка интерфейса.
Изменены функции: **AvCmLogin** и **AvCmGetErrorInfo**.
Добавлены функции:
AvCmInit, **AvCmDuplicateHandle**, **AvCmGetMsg**, **AvCmMsgSign**, **AvCmMsgDecrypt**, **AvCmGetMsgParam**, **AvCmGetMsgContent**, **AvCmMsgEncrypt**, **AvCmSignAndEncrypt**, **AvCmDecryptAndVerifySign**.
Удалена функция **AvCmGetSignCount.**

10 апреля 2003 г. – описан формат подписанного сообщения совместимый с *PKCS#7*. редакция 0.4.2.

3 апреля 2003 г. – подготовлена редакция 0.4. Внесены изменения и уточнения.
Добавлена поддержка *S/MIME*.

25 марта 2003 г. – подготовлена редакция 0.3. Добавлены функции
AvCmOpenCertEnum, AvCmEnumGet, AvCmCloseHandle, AvCmEncrypt, AvCmDecrypt, AvCmGetErrorMessage.

21 марта 2003 г. – подготовлена редакция 0.2. Добавлены функции **AvCmGetMsgSign, AvCmCloseSign, AvCmGetSignAttr, AvCmCloseCert, AvCmGetCertAttr.**

19 марта 2003 г. – подготовлена редакция 0.1

17 марта 2003 г. – начало работ по интерфейсу библиотеки.

Аннотация

Библиотека AvCryptMail обеспечивает возможность клиентам библиотеки обмениваться подписанными и/или зашифрованными сообщениями.

Для работы библиотеки необходимо хранилище сертификатов и СОС (списков отозванных сертификатов), заполненное необходимыми сертификатами и СОС.

В данной версии предполагается работа с хранилищем в базе данных Sybase Anywhere, Oracle, файловым хранилищем (в формате PKCS#7), либо системным хранилищем. Также возможна работа с архивным хранилищем – без личной части.

Для выработки ЭЦП и расшифрования сообщений необходим личный ключ, располагающийся на съемном носителе и соответствующий сертификат.

Библиотека не содержит функций Центра сертификации, выдающего сертификаты абонентам системы.

Библиотека в процессе работы использует криптопровайдер AvCSP™, обращаясь к нему через Microsoft Crypto API 1.

Библиотека поддерживает многопоточные обращения.

Оглавление

Аннотация 10

Оглавление 11

Архитектура библиотеки 14
Порядок работы с библиотекой 14
Поддерживаемые форматы сообщений 15
Поддерживаемые криптографические алгоритмы 16

Использование библиотеки 17
Инициализация библиотеки 17
Аутентификация пользователя и завершение сеанса 17
Использование дескрипторов объектов библиотеки 17
Обработка ошибок 18
Распределение памяти 18
Создание подписанного сообщения 18
Проверка ЭЦП с помощью высокоуровневых функций 19
Проверка ЭЦП с помощью низкоуровневых функций 19
Создание подписанного сообщения без текста сообщения 20
Проверка ЭЦП сообщения без текста сообщения 20
Отбор сертификатов для отправки зашифрованных сообщений 20
Зашифрование сообщения 21
Расшифрование сообщения 21
Применение нескольких криптографических операций к одному сообщению 21
Создание подписанного и зашифрованного сообщения 22
Обработка подписанных и зашифрованных сообщений 22
Генерация запроса на сертификат 22
Импорт сертификатов и СОС 22
Проверка ЭЦП подписанного сообщения с использованием статуса сертификата полученного от OCSP сервера 23
Проверка ЭЦП подписанного сообщения с использованием удостоверяющей подписи (CounterSignature) 23
Простой протокол регистрации сертификата SCEP 24
Дополнительные функции 24

Список функций библиотеки 25

Описание типов 28
Типы, определяемые библиотекой AvCryptMail 28
Используемые типы Microsoft 30

Описание функций 31
Инициализация 31
Инициализация библиотеки 31
Расширенная инициализация библиотеки 32
Авторизация 33
Авторизация пользователя (открытие сессии) 33
Закрытие сессии 36
Сброс данных сессии 37
Высокоуровневые функции сообщений 38

Генерация подписанного сообщение 38
Проверка подписи в сообщении и извлечение исходного сообщения 40
Зашифрование сообщения....42
Расшифрование сообщения....44
Подпись и зашифрование сообщения 46
Расшифрование, проверка ЭЦП и извлечение исходного сообщения....48
Работа с дескрипторами объектов....50
Получение параметров объекта....50
Дублирование дескриптора открытого объекта....52
Закрытие дескриптора открытого объекта 53
Низкоуровневые функции обработки сообщений 54
Открытие/создание сообщения....54
Открытие внутреннего сообщения....56
Получение параметров открытого сообщения....57
Установка дополнительных атрибутов подписанного сообщения 59
Подпись открытого сообщения 60
Генерация СТБ-подписи открытого сообщения 62
Проверка подписи в сообщении 64
Проверка подписи в сообщении по номеру 66
Проверка подписи в сообщении по номеру на заданную дату....68
Зашифрование сообщения....69
Расшифрование сообщения....70
Получение/экспорт подготовленного сообщения....71
Извлечение содержимого сообщения 72
Установка содержимого сообщения 73
Поблочная обработка данных сообщения 74
Извлечение информации о подписях из открытого сообщения....75
Получение атрибутов подписи по данным подписи 76
Импортирование сертификатов и СОС из сообщения в хранилище 80
Импортирование сертификатов в открытое сообщение....81
Функции работы с сертификатами....82
Поиск сертификата по данным подписи....82
Получение дескриптора сертификата из двоичных(der) данных 83
Получение атрибутов сертификата 84
Получение статуса сертификата....91
Получение атрибутов статуса сертификата....93
Создание контекста поиска и перебора сертификатов....94
Добавление сертификатов в контекст поиска и перебора сертификатов 99
Диалог создания контекста поиска и перебора сертификатов 103
Перебор сертификатов и информации о подписях....104
Помещение сертификата из сообщения в справочник сертификатов 105
Помещение сертификата/СОС в справочник сертификатов....106
Функции работы со списками отозванных сертификатов 108
Поиск списка отозванных сертификатов....108
Получение атрибутов открытого списка отозванных сертификатов....110
Функции работы с запросами на сертификат....112
Генерация запроса на сертификат 112
Получение атрибутов запроса на сертификат 114
Поиск запроса на сертификат 116
Низкоуровневые криптографические функции 117
Выработка ЭЦП для блока данных 117
Проверка ЭЦП для блока данных....119

Сервисные функции....121
Показ окна информации об объекте библиотеки....121
Функция получения информации об ошибке....122
Вывод информации в отладочный лог....124
Функции обработки ответов OCSPRespondera....125
Получение информации о статусе сертификата....125
Извлечение атрибутов информации о статусе сертификата....126
Добавление информации о статусе сертификата в подписанное сообщение....128
Контроль точек распространения COC....129
Контроль точек распространения COC....129
Взаимодействие с сервером SCEP....130
Создание соединения с сервером SCEP....130
Выполнение операции соединения с сервером SCEP....131
Получение данных соединения с сервером SCEP....132
Установка данных соединения с сервером SCEP....134
Обмен данными по протоколу TLS....135
Создание соединения с сервером TLS....135
Получение данных по протоколу TLS....136
Отправка данных по протоколу TLS....137

Возвращаемые коды ошибок....138

Примеры....140
Генерация подписанного сообщения при помощи низкоуровневых функций....140
Проверка ЭЦП под сообщением....141

Приложение 1. Формат файла с описанием операции....142

Текущие вопросы реализации....142

Описание форматов сообщений....143
Общий синтаксис....143
Сообщение типа «Данные»....143
Сообщение типа «Подписанные данные»....144
Тип данных SignedData....144
Тип данных SignerInfo....146
Процесс генерации цифрового дайджеста сообщения....148
Процесс зашифрования цифрового дайджеста....148
Выдержки из PKCS #9: Типы атрибутов, используемые в PKCS #7....149

Архитектура библиотеки

Порядок работы с библиотекой

Для начала работы с библиотекой требуется выполнить инициализацию библиотеки. Затем необходимо произвести авторизацию пользователя и указать параметры подключения к хранилищу сертификатов. В случае успешной авторизации функция вернет дескриптор сессии, которой необходимо использовать при всех последующих обращениях к библиотеке. Полученный дескриптор можно использовать только в потоке (thread), в котором он был создан, если необходима работа с библиотекой из нескольких потоков, в каждом из них необходимо создать свою сессию (произвести авторизацию).

Важным элементом в работе с библиотекой является работа с дескрипторами объектов библиотеки. Функции библиотеки при необходимости возвращают дескрипторы объектов, созданных/открытых в библиотеке, для дальнейшего использования в других вызовах функций библиотеки. После того, как необходимость в дальнейшем использования объекта, управляемого дескриптором, его необходимо освободить функцией закрытия дескриптора.

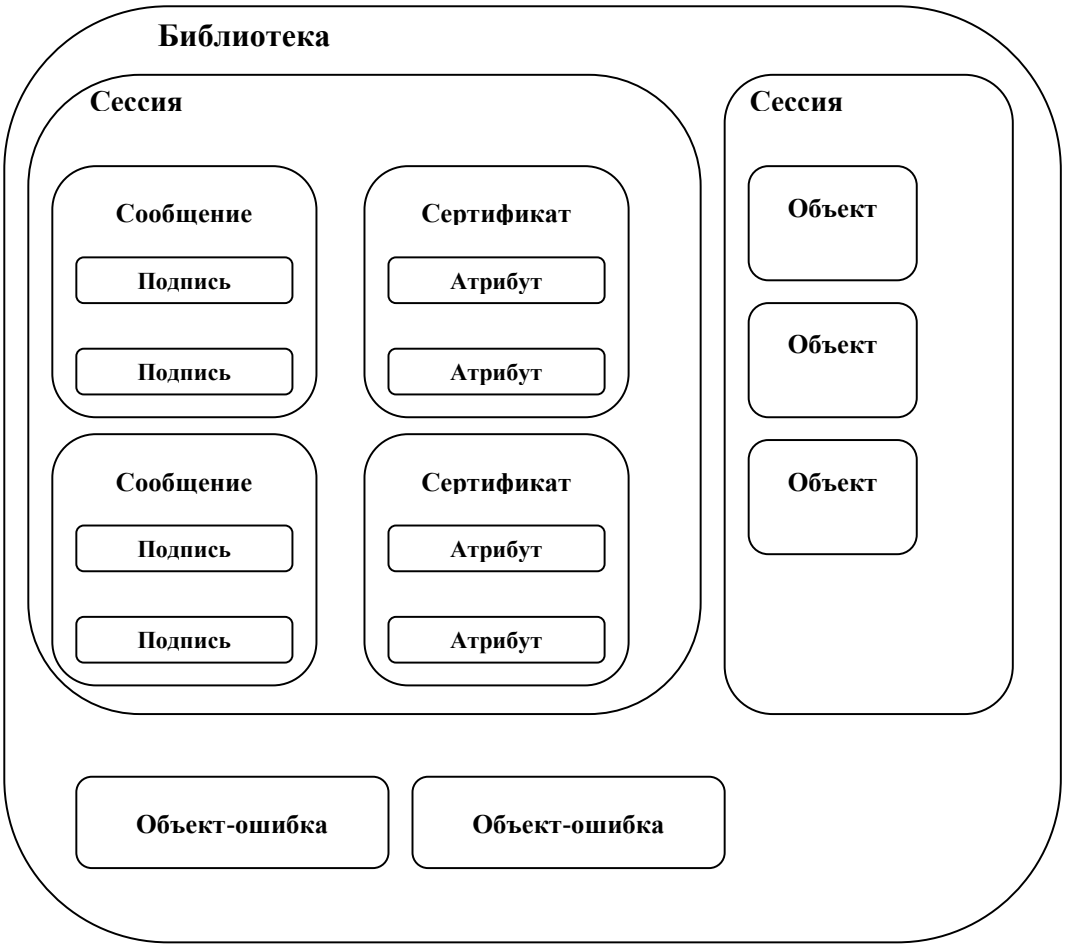

Рис. 1. Архитектура библиотеки

Библиотека содержит как низкоуровневые, так и высокоуровневые функции работы с криптографическими сообщениями. Высокоуровневые функции обеспечивают более простой интерфейс в работе с сообщениями, но не позволяют использовать все возможности библиотеки, поэтому в случае, если требуется большая гибкость, необходимо использовать низкоуровневые функции.

Помимо функций работы с криптографическими сообщениями библиотека содержит функции работы с сертификатами, необходимыми для работы с криптографическими сообщениями.

Все функции библиотеки в случае возникновения ошибки возвращают информацию об ошибке, при использовании библиотеки необходимо в обязательном порядке проверять код возврата функций библиотеки.

Поддерживаемые форматы сообщений

Библиотека поддерживает следующие форматы сообщений:

- *PKCS#7 Data* – данный формат предназначен для включения содержания сообщения в сообщения других форматов.
- *PKCS#7 SignedData* – сообщение в данном формате содержит поле данных сообщения и список подписей. Также такое сообщение может содержать список сертификатов и списки отозванных сертификатов.
- *PKCS#7 EnvelopedData* – сообщение в данном формате содержит зашифрованное разовым ключом поле данных сообщения и список получателей сообщения, при этом для каждого получателя в сообщение включен разовый ключ, на котором было зашифровано поле данных(содержание) сообщения, зашифрованный на открытом ключе получателя.

Сообщения могут быть вложены друг в друга. Формат *Data* всегда вкладывается в форматы *SignedData* и *EnvelopedData*. Другим стандартным вложением является вложение подписанного сообщения в зашифрованное. В результате подписанное и зашифрованное сообщение состоит из трех вложенных сообщений. Высокоуровневые функции библиотеки предназначены для работы именно с такими вложенными форматами. Низкоуровневые функции могут быть использованы при больших вложениях, например подписанное в подписанном и т. д.

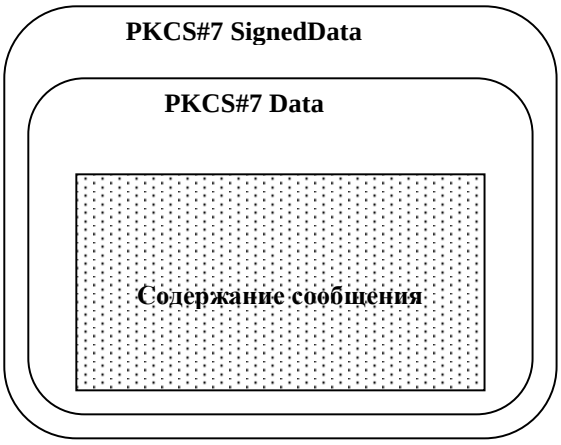

Рис. 2.Подписанное сообщение

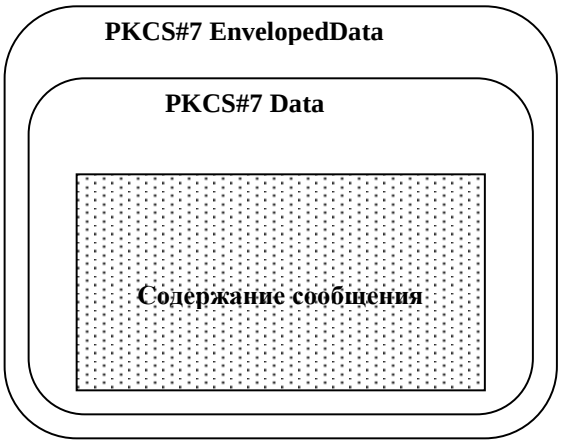

Рис. 3.Зашифрованное сообщение

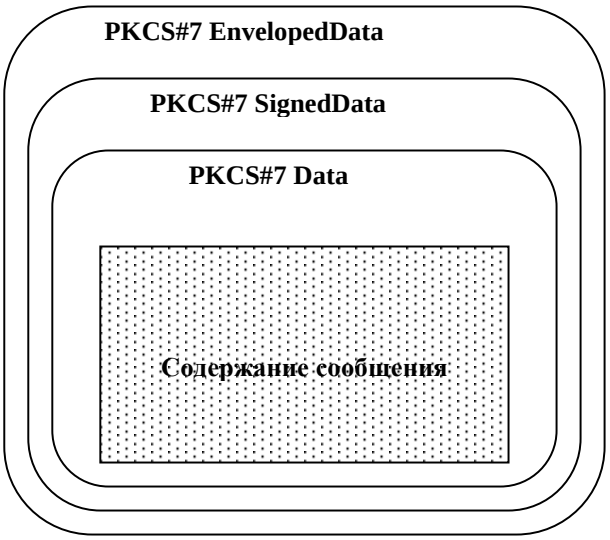

Рис. 4. Подписанное и зашифрованное сообщение

Поддерживаемые криптографические алгоритмы

Библиотека реализует интерфейс к криптографическим алгоритмам, реализованным в программных компонентах продукта РБ.ЮСКИ.13001-03 «Программный комплекс «Комплект Абонента АВЕСТ» AvUCK. Криптографические алгоритмы указаны в следующих документах:

– программный комплекс «Персональный менеджер сертификатов АВЕСТ» AvPCM. Руководство оператора (РБ.ЮСКИ.08003-02 34 01);

– программное средство криптографической защиты информации «Криптопровайдер Avest CSP» AvCSP. Руководство оператора (РБ.ЮСКИ.08000-02 34 01);

– программное средство криптографической защиты информации «Криптопровайдер Avest CSP BEL» AvCSPBEL. Руководство оператора (РБ.ЮСКИ.12004-01 34 01);

– программное средство криптографической защиты информации «Криптопровайдер Avest CSP BIGN» AvCSPBIGN. Руководство оператора (РБ.ЮСКИ.12005-01 34 01).

Использование библиотеки

Инициализация библиотеки

После загрузки библиотеки в адресное пространство процесса необходимо выполнить инициализацию библиотеки функцией **AvCmInit** или функцией **AvCmInitEx**. Ни один вызов библиотечных функций до выполнения инициализации невозможен – любой вызов библиотечной функции вернет ошибку. По окончанию работы с библиотекой необходимо выполнить очистку ресурсов библиотеки этой же функцией.

Аутентификация пользователя и завершение сеанса

После выполнения инициализации можно создать сессию работы с библиотекой с авторизацией пользователя. Для этого необходимо вызвать функцию **AvCmLogin**, передав ей параметры подключающегося пользователя и параметры подключения к справочникам сертификатов. При этом необходимо наличие в считывающем устройстве носителя с контейнером личных ключей пользователя. Библиотека самостоятельно выдаст диалог с просьбой ввести пароль, на котором зашифрован контейнер.

В процессе работы библиотеке требуются справочники сертификатов и СОС, поэтому в библиотеку также необходимо передать информацию о том, где располагаются справочники. При успешной авторизации библиотека возвращает *дескриптор сессии(соединения),* который будет использоваться при дальнейших обращениях к библиотеке.

Все объекты библиотеки, такие как сообщение, сертификат и т. д., создаются и в дальнейшем принадлежат сессии с дескриптором которой они были созданы, при вызове функции **AvCmLogout** все объекты данной сессии будут закрыты.

Для каждого из потоков(*Thread*) процесса, использующих библиотеку, приложению необходимо создать отдельную сессию. Вызов функции из одного потока с передачей функции дескриптора сессии, полученного в другом потоке недопустим, вызванная функция вернет ошибку **AVCMR_BAD_THREAD**. Для передачи объектов библиотеки между сессиями необходимо использовать функцию **AvCmDuplicateHandle**.

Помимо работы многопоточных приложений, несколько сессий приложению потребуется в случае необходимости параллельного использования нескольких личных ключей.

По окончанию работы с библиотекой необходимо закрыть соединение функцией **AvCmLogout**.

Для получения информации о пользователе и его сертификате необходимо использовать функцию **AvCmGetObjectInfo**.

Использование дескрипторов объектов библиотеки

Некоторые функции библиотеки возвращают дескрипторы объектов библиотеки, например дескриптор сертификата. После использования объекта, указанного возвращенным дескриптором, обязательно необходимо освободить используемый объект, вызвав функцию **AvCmCloseHandle**. Все объекты (за исключением ошибок) принадлежат сессии, в которой они были созданы. После закрытия сессии все объекты сессии закрываются и дальнейшее использование дескрипторов объектов данной сессии недопустимо.

Необходимо обратить внимание на то что, некоторые объекты являются дочерними по отношению к другим, и закрытие родительского объекта может привести к тому, что использование дескриптора дочернего объекта будет далее невозможно. В настоящий момент в библиотеке объект подпись является дочерним объектом объекта подписанное сообщение. Сертификаты являются дочерними объектами сессии. При необходимости объект может быть продублирован для использования в другой сессии функцией **AvCmDuplicateHandle**.

Параметры объекта библиотеки, например количество дочерних объектов, можно получить используя функцию **AvCmGetObjectInfo**.

Обработка ошибок

Большинство функций библиотеки имеют одинаковый механизм сигнализирования о происшедших в процессе работы ошибках. Функции возвращают значения типа **AvCmResult**.

Вызывающая программа должна проверять код возврата типа **AvCmResult**, и в случае если он не равен **AVCMR_SUCCESS**(0), программа может получить дополнительную информацию об ошибке, вызвав функцию получения дополнительной информации об ошибке **AvCmGetErrorInfo**, например, получить текстовое сообщение с описанием ошибки.

При этом каждая сессия хранит подробную информацию о последней возникшей в данной сессии ошибке. Приложение может получить расширенную информацию, передав при вызове дескриптор сессии библиотеки, в которой ошибка произошла. Если дескриптор сессии недоступен в функции обработки ошибки, можно, указав специальный флаг, выполнить поиск ошибки по идентификатору потока (*threadId*), вызвавшему функцию **AvCmGetErrorInfo**. Иначе будет возвращено только общее описание ошибки.

Распределение памяти

Некоторые функции библиотеки, например функция, создающая подписанное сообщение **AvCmSign**, возвращают сообщения, длина которых не может быть заранее определена вызывающей функцию программой. Поэтому необходим механизм определения необходимого размера памяти для выходного сообщения. В настоящий момент реализованы два метода для определения необходимого объема памяти:

1. Двойной вызов требуемой функции. При первом вызове в функцию в качестве указателя выходного сообщения передается 0. Функция заполняет по переданному указателю размера выходного буфера необходимый размер и возвращает значение **AVCMR_SUCCESS(0).** После этого вызывающая программа должна распределить необходимый объем памяти для выходного сообщения и вызвать функцию еще раз, указав правильные параметры выходного буфера и его размера. При первом вызове будет возвращен не точный размер выходного сообщения, а размер, достаточный для размещения выходного сообщения. При втором вызове функция вернет действительный размер получившего выходного сообщения.

При этом необходимо отметить что использование данной методики может приводить к перерасходу процессорного времени в криптографических функциях высокого уровня, поскольку разбор сообщения будет производится дважды.

2. Во флагах, передаваемых функции, указывается флаг **AVCMF_ALLOC**. В качестве параметра указателя выходного буфера должен быть указан указатель на незаполненный указатель выходного буфера. В этом случае функция самостоятельно распределит память в куче процесса функцией *Win32 API* **HeapAlloc** и вернет указатель созданного буфера через указатель на указатель буфера, переданный функции на входе. После использования выходного сообщения, вызвавшая программа должна самостоятельно очистить выделенную память функцией *Win32 API* **HeapFree.**

Создание подписанного сообщения

Создание подписанного сообщения требует наличия корректного личного сертификата, предназначенного для выработки ЭЦП, в справочнике сертификатов и наличия контейнера соответствующего личного ключа на носителе. Возможно как создание нового сообщения в

формате *PKCS#7 SignedData*, так и добавление подписи к уже существующему сообщению в этом формате, подписанному другими абонентами.

Существуют два способа создания подписанного сообщения: с использованием высокоуровневой функции **AvCmSign** и с использованием низкоуровневых функций обработки сообщений:

1. Использование высокоуровневой функции **AvCmSign**.

Функции необходимо передать входные данные, которые могут быть как данными для подписи и включения в подписываемое сообщение, так и сообщением, подписанным другими абонентами, для включения в сообщение новой подписи. Созданное сообщение содержит все необходимые данные для идентификации подписавшего, а также опционально может содержать все необходимые сертификаты и СОС для последующей проверки ЭЦП.

2. Использование низкоуровневых функций.
   - Необходимо создать или открыть сообщение функцией **AvCmOpenMsg**.
   - Затем выполнить выработку ЭЦП функцией **AvCmMsgSign**.
   - Экспортировать полученное подписанное сообщение **AvCmGetMsg**.
   - Закрыть созданное сообщение функцией **AvCmCloseHandle**.

Проверка ЭЦП с помощью высокоуровневых функций

Проверка подписей подписанного сообщения может быть выполнена функцией **AvCmVerifySign**. Функции необходимо передать в качестве входных данных подписанное сообщение, в случае успешной проверки ЭЦП функция вернет содержание сообщения, подписи которого были проверены.

Если в подписанном сообщении несколько подписей, то будет выполнена проверка всех подписей сообщения, при этом корректным сообщением считается только сообщение, все подписи которого верны. Если необходимо проверять только некоторые подписи сообщения, то необходимо использовать низкоуровневые функции проверки ЭЦП.

Для уточнения атрибутов подписей и сертификатов, соответствующих этим подписям функция возвращает дескриптор перечисления подписей и сертификатов типа **AvCmHsgnCertEnum**. данный дескриптор необходимо использовать в цикле перебора перечисления функцией **AvCmEnumGet**. При каждом вызове возвращается дескриптор объекта *Подпись и сертификат*, который можно передавать функциям получения атрибутов подписи и сертификата **AvCmGetSignAttr** и **AvCmGetCertAttr**. После сбора информации о подписях и соответствующих сертификатах необходимо закрыть все полученные дескрипторы (функция **AvCmCloseHandle**).

Проверка ЭЦП с помощью низкоуровневых функций

Проверка подписей подписанного сообщения представляет более сложный процесс, который может быть описан следующим образом:

1. Открытие подписанного сообщения (функция **AvCmOpenMsg**).
2. Проверка типа сообщения (функция **AvCmGetMsgParam**).
3. Разбор подписей, имеющихся в данном сообщении (функции **AvCmGetMsgParam** и **AvCmGetMsgSign**).
4. Принятие решения какие из подписей требуют проверки. Для получения атрибутов подписи, таких как, например серийный номер подписавшего сертификата ключа и сертификата его издателя используется функция **AvCmGetSignAttr**.
5. Поиск сертификатов, соответствующих этим подписям выполняет функция **AvCmFindCertBySign**.
6. Проверка атрибутов найденных сертификатов для того, чтобы решить, можно ли проверять подпись владельца данного сертификата (функция **AvCmGetCertAttr**).

7. Проверка подписей в сообщении, используя найденные сертификаты (функция **AvCmMsgVerifySign**).
8. Получить проверенное содержание сообщения функцией **AvCmGetMsgContent**.
9. Закрытие всех ранее открытых дескрипторов – дескрипторов найденных сертификатов, дескрипторов подписей в сообщении и дескриптора сообщения (функция **AvCmCloseHandle**).

Создание подписанного сообщения без текста сообщения

Библиотека позволяет создать подписанное сообщение, содержимое которого не включено в само сообщение. То есть, подписанное сообщение содержит только данные о подписях, но не подписанный текст. Сообщения такого вида могут быть использованы в случае необходимости хранения подписанного текста сообщения отдельно от информации о подписях сообщения.

Процедура создания такого сообщения аналогична процедуре создания обычного подписанного сообщения. При этом могут использоваться как высокоуровневая функция **AvCmSign**, так и низкоуровневая функция **AvCmMsgSign**. Этим функциям необходимо указать флаг **AVCMF_DETACHED**. Результатом работы при указании этого флага будет сгенерированное сообщение без подписанного текста сообщения.

Проверка ЭЦП сообщения без текста сообщения

Проверка подписей сообщения, подписанный текст которого не включен в само сообщение, может быть выполнен только при помощи низкоуровневых функций обработки сообщений. При этом в процедуру проверки добавляется еще один шаг: установка подписанного текста сообщения. Этот шаг должен быть выполнен после проверки типа сообщения.

Для установки подписанного текста необходимо использовать функцию **AvCmSetMsgContent**.

Отбор сертификатов для отправки зашифрованных сообщений

Для зашифрования сообщения, отправляемого абоненту или абонентам необходимо наличие сертификата/сертификатов абонентов, предназначенных для обмена зашифрованными сообщениями. Поэтому перед генерацией зашифрованного сообщения нужно отобрать сертификаты абонентов для которых будет генерироваться зашифрованное сообщение. Для отбора сертификатов используется функции **AvCmOpenCertEnum**, **AvCmEnumGet** и **AvCmCloseHandle**. Процесс перебора сертификатов осуществляется следующим образом:

1. Функцией **AvCmOpenCertEnum** создается контекст перебора сертификатов, при этом необходимо указать набор критериев отбора необходимых сертификатов. В качестве критериев могут быть указаны, например, издатель сертификата, идентификатор открытого ключа и другие атрибуты сертификатов.
2. В цикле вызывается функция **AvCmEnumGet** с указанием контекста перебора, до тех пор, пока функция не вернет ошибку **AVCMR_NOT_FOUND,** означающую что больше не найдено сертификатов, отвечающих заданным критериям отбора. Если вызов функции завершился успешно, будет возвращен дескриптор найденного сертификата, который может быть использован в дальнейшем, например для зашифрования сообщения.
3. По окончанию перебора функцией **AvCmCloseHandle** закрывается контекст перебора.
4. Полученные на этапе 2 дескрипторы найденных сертификатов после использования необходимо освободить функцией **AvCmCloseHandle**.

Зашифрование сообщения

После того как сертификаты получателей зашифрованного сообщения отобраны можно выполнить зашифрование сообщения для этих абонентов. Зашифрование может выполняться как высокоуровневой функцией **AvCmEncrypt,** так и низкоуровневой **AvCmMsgEncrypt**. В обоих случаях необходимо передать массив дескрипторов сертификатов получателей зашифрованного сообщения. Обратим внимание на то, что передаваемые сертификаты должны быть предназначены для обмена зашифрованными сообщениями. В случае успешного выполнения будет сгенерировано зашифрованное сообщение в формате *PKCS#7 EnvelopedData*. Зашифрованное сообщение включает в себя всю необходимую информацию для того чтобы владельцы сертификатов для которых было выполнено зашифрование, при наличии соответствующих личных ключей могли расшифровать сообщение.

1. Использование высокоуровневой функции **AvCmEncrypt**.
   Этой функции необходимо передать шифруемое сообщение и массив сертификатов получателей зашифрованного сообщения.
2. Использование низкоуровневых функций. Предварительно необходимо отобрать сертификаты, для владельцев которых будет зашифровано сообщение.
   - Необходимо создать или открыть сообщение функцией **AvCmOpenMsg**.
   - Зашифровать сообщение для отобранных абонентов функцией **AvCmMsgEncrypt**.
   - Экспортировать полученное подписанное сообщение **AvCmGetMsg**.
   - Закрыть созданное сообщение функцией **AvCmCloseHandle**.

Расшифрование сообщения

Полученное зашифрованное сообщение в формате *PKCS#7 EnvelopedData* можно расшифровать если в списке получателей зашифрованного сообщения был указан сертификат, владельцем которого является аутентифицированный в данный момент пользователь.

Расшифрование может быть выполнено как высокоуровневой функцией **AvCmDecrypt**, так и с помощью низкоуровневых функций.

1. Использование высокоуровневой функции **AvCmDecrypt**.

Этой функции необходимо передать зашифрованное сообщение. Функция попробует обнаружить среди получателей сообщения сертификат аутентифицированного в данный момент пользователя и затем расшифровать сообщение.

2. Использование низкоуровневых функций.

- Необходимо открыть зашифрованное сообщение функцией **AvCmOpenMsg**.
- Расшифровать сообщение функцией **AvCmMsgDecrypt**.
- Получить расшифрованное содержание сообщения функцией **AvCmGetMsgContent**.
- Закрыть сообщение функцией **AvCmCloseHandle**.

Применение нескольких криптографических операций к одному сообщению

Стандартной операцией при генерации сообщений является подписывание сообщения с последующим его зашифрованием. Данная операция, и обратная операция расшифрование и проверка подписей могут быть выполнены как высокоуровневыми функциями, так и последовательностью вызовов низкоуровневых функций.

В случае использования других комбинаций криптографических преобразований необходимо использовать соответствующие комбинации вызовов низкоуровневых функций библиотеки.

Создание подписанного и зашифрованного сообщения

Данная операция может быть выполнена вызовом высокоуровневой функции **AvCmSignAndEncrypt**. Использование этой функции аналогично использованию функций **AvCmSign** и **AvCmEncrypt**.

Подпись и зашифрование сообщений также может быть выполнено последовательностью вызовов низкоуровневых функций:

1. Необходимо создать сообщение функцией **AvCmOpenMsg**.
2. Выполнить выработку ЭЦП функцией **AvCmMsgSign**. При этом автоматически будет создано сообщение в формате *подписанное сообщение*, в которое будет вложено в качестве содержимого исходное сообщение.
3. Зашифровать сообщение для отобранных абонентов функцией **AvCmMsgEncrypt**. При этом автоматически будет создано сообщение в формате *зашифрованное сообщение*, в которое будет вложено в качестве содержимого подписанное сообщение.
4. Экспортировать полученное подписанное сообщение **AvCmGetMsg**.
5. Закрыть созданное сообщение функцией **AvCmCloseHandle**.

Обработка подписанных и зашифрованных сообщений

Данная операция может быть выполнена вызовом высокоуровневой функции **AvCmDecryptAndVerifySign**. Использование этой функции аналогично использованию функций **AvCmVerifySign** и **AvCmDecrypt**.

Подпись и зашифрование сообщений также может быть выполнено последовательностью вызовов низкоуровневых функций:

1. Необходимо открыть сообщение функцией **AvCmOpenMsg**.
2. Расшифровать сообщение функцией **AvCmMsgDecrypt**.
3. Получить расшифрованное подписанное сообщение в формате *подписанное сообщение* при помощи функции **AvCmOpenInnerMsg**.
4. Проверить подписи функцией **AvCmMsgVerifySign**.
5. Получить исходное содержание сообщения функцией **AvCmGetMsgContent**.
6. Закрыть созданное и внутренне сообщения функцией **AvCmCloseHandle**.
7. Закрыть внутренне сообщения функцией **AvCmCloseHandle**.

Генерация запроса на сертификат

Библиотека позволяет создать запрос на сертификат, являющийся продолжением действующего в настоящий момент личного сертификата, для передачи запроса в Центр сертификации. При генерации запроса будут сгенерированы необходимые пары ключей. Запрос может быть подписан действующим в настоящий момент личным сертификатом. Центр сертификации после проверки подписи под запросом может выдать сертификат и затем этот сертификат может быть импортирован в справочник сертификатов используемый библиотекой, и использоваться в дальнейшем для выработки ЭЦП или для расшифрования сообщений.

Для генерации запроса на сертификат необходимо использовать функцию **AvCmGenerateRequest**.

Импорт сертификатов и СОС

Библиотека может импортировать сертификаты и СОС. Импорт сертификатов и СОС возможен как в автоматическом, так и в ручном режиме. В ручном режиме при вызове функции импорта библиотека отобразит мастер импорта сертификатов и СОС и позволит пользователю

выбрать объекты для импорта, после импорта предложит выбрать контейнер с личным ключом и подпишет список доверенных удостоверяющих центров.

Для импорта с выводом мастера импорта необходимо открыть файл PKCS #7 SignedData, в котором находятся сертификаты и/или СОС для импорта, функцией **AvCmOpenMsg**. Затем необходимо вызвать функцию для запуска мастера импорта **AvCmMsgImportCerts** с указанием в качестве флагов 0.

Для импорта сертификата или СОС в автоматическом режиме необходимо использовать функцию **AvCmImport**.

Проверка ЭЦП подписанного сообщения с использованием статуса сертификата полученного от OCSP сервера

Проверка ЭЦП подписанного сообщения с использованием статуса сертификата полученного от OCSP сервера может быть описана следующим образом:

1. Открытие подписанного сообщения (функция **AvCmOpenMsg**).
2. Принятие решения о целесообразности проверки ЭЦП сообщения используя функции: **AvCmGetMsgParam**, **AvCmGetMsgSign**, **AvCmGetSignAttr**, **AvCmFindCertBySign**, **AvCmGetCertAttr**.
3. Проверка подписей в сообщении (функции **AvCmMsgVerifySign** либо **AvCmMsgVerifySignAtIndex**) без проверки валидности сертификата используя флаг **AVCMF_NO_CERT_VERIFY**.
4. Получить информацию о статусе сертификата (фикция **AvCmMsgOCSPGetResponse** - для проверки на текущее время, функция **AvCmGetSignAttr** – для проверки на требуемую дату, для сообщений с добавленной информацией о статусе сертификата).
5. На основании анализа атрибутов информации о статусе сертификата (функция **AvCmMsgOCSPGetResponseAttr**): **AVCM_OCSP_STATUS, AVCM_OCSP_THIS_UPDATE, AVCM_OCSP_NEXT_UPDATE** – принять решение о валидности сертификата и ЭЦП сообщения на требуемую дату.
6. При необходимости сохранить информацию о статусе сертификата в проверяемом сообщении (функции **AvCmMsgOCSPAddResponse**, **AvCmGetMsg**).

Закрытие всех ранее открытых дескрипторов – дескрипторов найденных сертификатов, дескрипторов подписей в сообщении и дескриптора сообщения (функция **AvCmCloseHandle**).

Проверка ЭЦП подписанного сообщения с использованием удостоверяющей подписи (CounterSignature)

Проверка ЭЦП подписанного сообщения включает в себя проверку действительности сертификата на момент проверки ЭЦП. В некоторых системах к времени окончательного приема документа, срок действия отдельного сертификата может закончится, в данном случае возникает вопрос о действительности всего документа. В целях исключения такой ситуации предлагается принимающей стороне (службе) производить удостоверение проверяемой ЭЦП сертификатом службы в момент первой проверки ЭЦП и в дальнейшем судить о действительности сообщения в целом по действительности удостоверяющей подписи. Алгоритм реализации данного сценария сервера может быть описан следующим образом:

1. Открытие подписанного сообщения (функция **AvCmOpenMsg**).
2. Получение количества ЭЦП в сообщении (функция **AvCmGetMsgParam**). Далее для каждой ЭЦП:
3. Проверка ЭЦП в сообщении (функция **AvCmMsgVerifySignAtIndex**) без проверки валидности сертификата используя флаг **AVCMF_NO_CERT_VERIFY**. Если ЭЦП не верна документ отвергаем.

4. Получение количества удостоверяющих подписей для данной ЭЦП (**AvCmGetSignAttr**). Если удостоверяющих подписей нет – 6.
5. Проверка целостности самой удостоверяющей ЭЦП (**AvCmGetSignAttr** с параметром AVCM_MSGSIGN_VALID). Если ЭЦП не верна документ отвергаем.Так как удостоверяющая подпись тоже может быть удостоверена проверяем ее начиная с 4.
6. Проверяем действительность сертификата открытого ключа, на основании которого выполнена ЭЦП (**AvCmFindCertBySign**, **AvCmGetCertAttr** с параметром AVCM_VALID). Если сертификат не действителен документ отвергаем.
7. Получаем атрибуты сертификата для определения какой службой выполнена ЭЦП (**AvCmGetCertAttr**). Если это «первая» ЭЦП или удостоверяющую подпись требуется переудостоверить, добавляем удостоверяющую подпись для данной ЭЦП (AvCmMsgSign с флагом AVCMF_COUNTERSIGNATURE). Если проверяем последнюю ЭЦП документа, при необходимости получаем новый блоб сообщения (**AvCmGetMsg**), документ принимается иначе проверяем следующую ЭЦП документа начиная с 3.

Закрытие всех ранее открытых дескрипторов – дескрипторов найденных сертификатов, дескрипторов подписей в сообщении и дескриптора сообщения (функция **AvCmCloseHandle**). См. пример CounterSignature.dpr.

Простой протокол регистрации сертификата SCEP

Библиотека поддерживает простой протокол регистрации сертификата SCEP. Алгоритм реализации протокола должен быть реализован следующим образом:

1. Создание соединения с сервером SCEP (функция **AvCmCreateScep**).
2. Установка обрабатываемого запроса на сертификат, либо текущей транзакции (функция **AvCmScepSet**).
3. Установление связи контейнера личных ключей с открытым ключом запроса на сертификат (функции **AvCmScepGet**, **AvCmScepSet**, **AvCmScepExequte**).
4. Выполнение запроса к серверу SCEP (функция **AvCmScepExequte**).
5. Получение текущей транзакции либо выпущенного сертификата (функция **AvCmScepGet**).

Дополнительные функции

Для удобства использования библиотека содержит дополнительные функции.

Список функций библиотеки

Инициализация библиотеки

1. **AvCmInit** – первоначальная инициализация библиотеки и очистка ресурсов библиотеки по окончанию работы с ней
2. **AvCmInitEx** – первоначальная инициализация библиотеки и очистка ресурсов библиотеки по окончанию работы с ней с установкой рабочего каталога библиотеки AvCryptMail.

Авторизация пользователя – создание и уничтожение сессий

3. **AvCmLogin** – авторизация пользователя и создание сессии
4. **AvCmLogout** – отключение пользователя и закрытие сессии
5. **AvCmFlush** – сброс изменений в справочниках

Высокоуровневые функции обработки сообщений

6. **AvCmSign** – создание подписанного сообщения
7. **AvCmVerifySign** – проверка подписи в сообщении и извлечение исходного сообщения
8. **AvCmEncrypt** – генерация зашифрованного сообщения
9. **AvCmDecrypt** – расшифрование полученного зашифрованного сообщения
10. **AvCmSignAndEncrypt** – генерация подписанного сообщения с последующим его зашифрованием
11. **AvCmDecryptAndVerifySign** – расшифрование сообщения с последующей проверкой подписи, и получения исходного сообщения

Функции работы с объектами при помощи дескрипторов объектов

12. **AvCmGetObjectInfo** – получение параметров открытого объекта библиотеки
13. **AvCmDuplicateHandle** – создание копии объекта
14. **AvCmCloseHandle** – закрытие объекта библиотеки доступного с использованием дескриптора

Низкоуровневые функции обработки сообщений

15. **AvCmOpenMsg** – открытие или создание сообщения
16. **AvCmOpenInnerMsg** – открытие и разбор вложенного сообщения
17. **AvCmGetMsg** – получение сгенерированного сообщения
18. **AvCmGetMsgContent** – получение содержимого сообщения
19. **AvCmSetMsgContent** – установка содержимого сообщения
20. **AvCmMsgSign** – подпись открытого сообщения
21. **AvCmMsgGetSTBSign** – получение СТБ-подписи открытого сообщения
22. **AvCmMsgVerifySign** – проверка подписей в открытом подписанном сообщении
23. **AvCmMsgVerifySignAtIndex** – проверка подписей в открытом подписанном сообщении по номеру
24. **AvCmMsgVerifySignAtIndexForDate** – проверка подписей в открытом подписанном сообщении по номеру на заданную дату
25. **AvCmMsgEncrypt** – зашифрование открытого сообщения
26. **AvCmMsgDecrypt** – расшифрование открытого зашифрованного сообщения
27. **AvCmGetMsgParam** – получение параметров открытого сообщения
28. **AvCmGetMsgSign** – извлечение подписи одного из подписавших сообщение
29. **AvCmGetSignAttr** – извлечение атрибутов подписи
30. **AvCmMsgImportCerts** – импортирование сертификатов и СОС из сообщения

31. **AvCmMsgAddCert** – импорт массива открытых сертификатов и СОС в подписанное сообщение
32. **AvCmMsgSetAttribute** – добавления дополнительных атрибутов в подписываемое сообщение

Функции работы с сертификатами

33. **AvCmFindCertBySign** – поиск сертификата по его подписи
34. **AvCmGetCertAttr** – извлечение атрибутов сертификата
35. **AvCmVerifyCertStatus** – проверка статуса сертификата
36. **AvCmGetCertStatusAttr** – извлечение атрибутов статуса сертификата
37. **AvCmOpenCertEnum** – создание контекста поиска сертификата путем перебора подмножества сертификатов, удовлетворяющих определенным условиям
38. **AvCmEnumAddCerts** – добавление сертификатов в контекст поиска сертификата путем перебора подмножества сертификатов, удовлетворяющих определенным условиям
39. **AvCmEnumDlg** – диалог создания контекста поиска сертификата.
40. **AvCmEnumGet** – перебор сертификатов и подписей.
41. **AvCmPutCert** – помещение сертификата в справочник сертификатов
42. **AvCmImport** – импорт сертификата или списка отозванных сертификатов в соответствующие справочники сертификатов/СОС

Функции работы со списками отозванных сертификатов

43. **AvCmFindCrl** – поиск списка отозванных сертификатов в справочниках сертификатов/СОС
44. **AvCmGetCrlAttr** – извлечение атрибутов открытого списка отозванных сертификатов

Функции работы с запросами на сертификат

45. **AvCmGenerateRequest** – генерация запроса на сертификат
46. **AvCmGetRequestAttr** – извлечение атрибутов запроса на сертификат.
47. **AvCmFindRequest** – поиск запроса на сертификат

Низкоуровневые криптографические функции

48. **AvCmSignRawData** – выработка ЭЦП для блока данных.
49. **AvCmVerifyRawDataSign** – проверки ЭЦП заданным алгоритмом для переданных данных.

Функции обработки ответов OCSPRespondera

50. **AvCmMsgOCSPGetResponse** – получение дескриптора информации о статусе сертификата.
51. **AvCmMsgOCSPGetResponseAttr** – извлечение атрибутов информации о статусе сертификата.
52. **AvCmMsgOCSPAddResponse** – добавление информации о статусе сертификата в подписанное сообщение.

Простой протокол регистрации сертификата SCEP

53. **AvCmCreateScep** – создание соединения с сервером SCEP.
54. **AvCmScepExequte** – выполнение операции соединения с сервером SCEP.
55. **AvCmScepGet** – получение данных соединения с сервером SCEP.
56. **AvCmScepSet** – установка данных соединения с сервером SCEP.

Сервисные функции

57. **AvCmShowObjectInfo** – показ окна с информацией по объекту библиотеки.
58. **AvCmGetErrorInfo** – получение описания ошибки библиотеки по дескриптору ошибки, полученному при вызове одной из функций библиотеки

Описание типов

Типы, определяемые библиотекой AvCryptMail

AvCmLong – целое четырехбайтовое (восьмибайтовое для win64) число без знака.

AvCmHandle(AvCmLong) – дескриптор объекта библиотеки, данный тип используется для обращения к распределенным объектам библиотеки, например к сертификатам, сообщениям и т. д.

AvCmResult(DWORD) – тип возвращаемого библиотекой кода результата. Функции библиотеки возвращают в качестве результата код завершения операции. В случае успешного выполнения – **AVCMR_SUCCESS (0)**, в случае ошибки – код ошибки. См. Обработка ошибок

AvCmHc(AvCmHandle) – дескриптор подключившегося пользователя.

AvCmHmsg(AvCmHandle) – дескриптор открытого сообщения в формате *PKCS#7*.

AvCmHsign(AvCmHandle) – дескриптор одной из подписей сообщения.

AvCmHcert(AvCmHandle) – дескриптор открытого сертификата.

AvCmHcertstat(AvCmHandle) – дескриптор статуса сертификата.

AvCmHsignCert(AvCmHandle) – дескриптор подписи и соответствующего сертификата, полученные в результате вызова функций проверки ЭЦП.

AvCmHenum(AvCmHandle) – дескриптор открытого контекста перебора объектов.

AvCmHcertEnum(AvCmHenum) – дескриптор открытого контекста перебора сертификатов.

AvCmHsgnCertEnum(AvCmHenum) – дескриптор открытого контекста перебора информации о подписях и соответствующих сертификатах подписанного сообщения.

AvCmHcrl (AvCmHandle) – дескриптор открытого списка отозванных сертификатов.

AvCmHreq (AvCmHandle) – дескриптор открытого запроса на сертификат.

AvCmConnectionParam – данные необходимые при подключении для идентификации пользователя и подключения к справочникам сертификатов/СОС. В настоящее время планируется хранение справочников сертификатов в базе данных.

```
struct AvCmConnectionParam {
   DWORD param_id;  // идентификатор параметра
   void * param;    // значение параметра
};
```

AvCmEnumGetParam – атрибуты поиска сертификатов в справочнике сертификатов.

```
struct AvCmEnumGetParam {
   DWORD param_id;     // идентификатор параметра
   void * param_spec;  // уточнение идентификатора
   void * param;       // значение параметра
```

```
};
```

AvCmImportParam – данные используемые при импорте сертификатов и списков отозванных сертификатов.

```
struct AvCmImportParam {
   DWORD param_id;  // идентификатор параметра
   void * param;    // значение параметра
};
```

AvCmFindCrlParam – данные используемые при поиске списков отозванных сертификатов.

```
struct AvCmFindCrlParam {
   DWORD param_id;  // идентификатор параметра
   void * param;    // значение параметра
};
```

AvCmGenReqParam – данные используемые при генерации запроса на сертификат.

```
struct AvCmGenReqParam {
   DWORD param_id;  // идентификатор параметра
   void * param;    // значение параметра
};
```

AvCmFindReqParam – данные используемые при поиске запроса на сертификат.

```
struct AvCmFindReqParam {
   DWORD param_id;  // идентификатор параметра
   void * param;    // значение параметра
};
```

AvCmCertStatParam – данные используемые при запросе статуса сертификата.

```
struct AvCmCertStatParam {
   DWORD param_id;  // идентификатор параметра
   void * param;    // значение параметра
};
```

Используемые типы Microsoft

DWORD – целое четырехбайтовое число без знака

FILETIME – дата и время

SYSTEMTIME – дата и время. При этом дата и время всегда возвращаются для текущего часового пояса (согласно настроек операционной системы).

Для получения дополнительной информации о типах Microsoft обратитесь к MSDN.

Описание функций

Инициализация

Инициализация библиотеки

Функция **AvCmInit** предназначена для выполнения первоначальной инициализации библиотеки и очистки ресурсов библиотеки по окончанию работы с ней. Данная функция должна быть вызвана с параметром **AVCMF_STARTUP** перед вызовом других функций библиотеки и с параметром **AVCMF_SHUTDOWN** по окончанию работы с ней.

```
AvCmResult AvCmInit(
    DWORD flags);
```

Параметры

[in] DWORD flags – режим работы функции

- **AVCMF_STARTUP** – данный флаг необходимо использовать для первоначальной инициализации библиотеки.
- **AVCMF_SHUTDOWN** – данный флаг необходимо использовать для освобождения ресурсов библиотеки перед ее выгрузкой из адресного пространства процесса.

Результат

Код возврата типа **AvCmResult**

Возможные значения:

- **AVCMR_SUCCESS (0)** – успешное выполнение функции.
- **AVCMR_BUSY** – невозможно освободить ресурсы – имеются открытые объекты библиотеки.
- **AVCMR_ALREADY_INITIALIZED** – повторная попытка инициализации библиотеки.
- **AVCMR_NOT_INITIALIZED** – библиотека не была инициализирована.
- **AVCMR_INTERNAL_ERROR** – внутренняя ошибка библиотеки.
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.
- **AVCMR_AVCSP_INIT_FAILED** – инициализация криптопровайдера Авест завершилась неудачно.
- **AVCMR_REGISTRY_ERROR** – ошибка доступа к реестру Windows.

- Ошибка *Win32 API*.

Расширенная инициализация библиотеки

Функция **AvCmInitEx** предназначена для выполнения первоначальной инициализации библиотеки и очистки ресурсов библиотеки по окончанию работы с ней. Данная функция должна быть вызвана с параметром **AVCMF_STARTUP** перед вызовом других функций библиотеки и с параметром **AVCMF_SHUTDOWN** по окончанию работы с ней. От **AvCmInit** отличается наличием параметра, указывающего рабочий каталог библиотеки AvCryptMail.

```
AvCmResult AvCmInitEx(
    const char* cpszWorkDir,
    DWORD flags);
```

Параметры

[in] const char* cpszWorkDir – путь к рабочему каталогу библиотеки. Если равен NULL, то рабочим будет считаться текущий каталог.

[in] DWORD flags – режим работы функции

- **AVCMF_STARTUP** – данный флаг необходимо использовать для первоначальной инициализации библиотеки.
- **AVCMF_SHUTDOWN** – данный флаг необходимо использовать для освобождения ресурсов библиотеки перед ее выгрузкой из адресного пространства процесса.

Результат

Код возврата типа **AvCmResult**

Возможные значения:

- **AVCMR_SUCCESS (0)** – успешное выполнение функции.
- **AVCMR_BUSY** – невозможно освободить ресурсы – имеются открытые объекты библиотеки.
- **AVCMR_ALREADY_INITIALIZED** – повторная попытка инициализации библиотеки.
- **AVCMR_NOT_INITIALIZED** – библиотека не была инициализирована.
- **AVCMR_INTERNAL_ERROR** – внутренняя ошибка библиотеки.
- **AVCMR_BAD_PARAM** – функции переданы неверные параметры.
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.
- **AVCMR_AVCSP_INIT_FAILED** – инициализация криптопровайдера Авест завершилась неудачно.
- **AVCMR_REGISTRY_ERROR** – ошибка доступа к реестру Windows.

- Ошибка *Win32 API*.

Авторизация

Авторизация пользователя (открытие сессии)

Функция **AvCmLogin** предназначена для авторизации пользователя для дальнейшей работы с библиотекой. При работе с базой данных функции должны быть переданы или параметры подключения или дескриптор открытого соединения с базой. По окончанию работы должна быть вызвана функция **AvCmLogout**.

```
AvCmResult AvCmLogin(
    size_t conn_param_count,
    const AvCmConnectionParam * conn_params,
    AvCmHc * hc,
    DWORD flags);
```

Параметры

[in] size_t conn_param_count – количество параметров идентификации пользователя

[in] const AvCmConnectionParam * conn_params – массив параметров идентификации подключающегося пользователя и параметры подключения:

В настоящее время поддерживается следующие параметры идентификации пользователя при подключении:

Параметры подключения к базе данных сертификатов:

- **AVCM_DB_TYPE** – тип базы данных сертификатов, в которой находятся справочники сертификатов/СОС:
  - **AVCM_DBT_MS_REGISTRY** – хранилище стандарта Microsoft в системном реестре Windows.
  - **AVCM_DBT_FILE** – хранилище в файлах стандарта PKCS7.
  - **AVCM_DBT_ARCHIVE_FILE** – архивное хранилище в файле стандарта PKCS7.
  - **AVCM_DBT_ARCHIVE_MEMORY** – архивное хранилище в памяти в стандарте PKCS7.
  - **AVCM_DBT_E_MEMORY** –пустое хранилище в памяти с личным сертификатом, прочитанным с носителя ключей. После выполнения такой авторизации необходимо найти и загрузить справочник сертификатов корневых УЦ при помощи функции **AvCmImport**.

  Если параметр **AVCM_DB_TYPE** не передан, то будет произведено подключение к базе данных с использованием ADO. Если также не передан параметр **AVCM_DB_CONNECTSTR**, то библиотека для подключения к базе данных использует файл конфигурации AvCmMsg.ini,

- **AVCM_DB_CONNECTSTR** – строка ADO ConnectString для соединения с базой..

В случае использования в качестве хранилища сертификатов архивного хранилища в буфере памяти в стандарте PKCS7, должен быть указан путь к файлам хранилища:

- **AVCM_DB_ARCHIVE_PTR** – Указатель на начало буфера с архивным хранилищем.

- **AVCM_DB_ARCHIVE_SIZE** – Указатель на переменную типа DWORD, хранящую размер буфера с архивным хранилищем.

При работе с архивным хранилищем сертификатов особым образом работает проверка подписей. Проверяется доверие сертификатам на то время, когда было подписано сообщение; самоподписанные сертификаты, если их сроки действия подходят под дату подписи сообщения, считаются доверенными.

Параметры аутентификации пользователя:

- **AVCM_PUB_KEY_ID** – идентификатор открытого ключа сертификата пользователя, чьим личным ключом будет произведена авторизация. Если этот параметр не указан, то библиотека выдаст окно со списком сертификатов пользователя для выбора того сертификата, при помощи которого будет произведена авторизация.
- **AVCM_COMMON_NAME** – атрибут CommonName субъекта сертификата пользователя, чьим личным ключом будет произведена авторизация. Если этот параметр не указан, то библиотека выдаст окно со списком сертификатов пользователя для выбора того сертификата, при помощи которого будет произведена авторизация.

Одновременное указание параметров **AVCM_PUB_KEY_ID** и **AVCM_COMMON_NAME** считается ошибкой.

- **AVCM_PASSWORD** – пароль доступа к контейнеру личных ключей. Если в персональном справочнике только один сертификат, то библиотека произведет попытку авторизации с этим сертификатом, не отображая окно выбора личного сертификата.
- **AVCM_SLOTID** – (указатель на AvCmLong) логический идентификатор слота устройства, доступ к которому осуществляется через PKCS#11. Слоты нумеруются с 1.

`[out] AvCmHc * hc` – возвращаемый в случае успешного подключения дескриптор сессии.

`[in] DWORD flags` – режимы подключения

- **AVCMF_NO_AUTH** – подключение без аутентификации пользователя. Использование сессии без открытия контейнера личных ключей и поиска личного сертификата. Использование такой сессии для операций, требующих наличия личного сертификата и контейнера личных ключей, невозможно
- **AVCMF_FORCE_TOKEN_CONTROL** – контроль наличия вставленного носителя с личным ключом
- **AVCMF_DENY_TOKEN_CONTROL** – запрет контроля наличия вставленного носителя с личным ключом
- **AVCMF_IGNORE_CRL_ABSENCE** – игнорировать отсутствие СОС.
- **AVCMF_RELOGIN** – выбор личного сертификата для текущей сесии.

Результат

Код возврата типа **`AvCmResult`**

Возможные значения:

- **AVCMR_SUCCESS** – успешное подключение пользователя
- **AVCMR_CONTAINER_NOT_FOUND** – не найден контейнер с личными ключами на носителе
- **AVCMR_DEVICE_NOT_FOUND** – не найден носитель
- **AVCMR_BAD_PASSWORD** – пароль неверен
- **AVCMR_NO_DB_PARAMS** – не указаны параметры подключения к базе данных
- **AVCMR_DB_NOT_FOUND** – невозможно подключится к базе данных

- **AVCMR_CERT_STORE_NOT_FOUND** – не найдено или пусто хранилище сертификатов
- **AVCMR_CERT_ NOT_FOUND** – не найден личный сертификат
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.
- **AVCMR_BAD_PARAM** – параметр функции неверен
- **AVCMR_TOKEN_NOT_FOUND** – носитель с личным ключом не установлен

Закрытие сессии

Функция **AvCmLogout** предназначена для отключения пользователя и закрытие сессии. После успешного выполнения функции использовать дескриптор сессии больше нельзя. Также могут освобождены все созданные в данном подключении объекты, например сертификаты, поэтому все полученные дескрипторы объектов библиотеки становятся недействительными, и использовать их в дальнейшем нельзя.

```
AvCmResult AvCmLogout(
    AvCmHc hc,
    DWORD flags);
```

Параметры

[in] AvCmHc hc – дескриптор открытой сессии

[in] DWORD flags – режимы отключения
В настоящий момент нет поддерживаемых флагов.

Результат

Код возврата типа **AvCmResult**
Возможные значения:

- **AVCMR_SUCCESS** – успешное отключение пользователя
- **AVCMR_BAD_HC** – дескриптор соединения неверен
- **AVCMR_BUSY** – невозможно закрыть соединение, соединение занято
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.

Сброс данных сессии

Функция **AvCmFlush** предназначена для сброса данных сессии – сохранения изменений в справочниках сертификатов. Рекомендуется вызывать данную функцию после использования механизмов импорта сертификатов и СОС.

```
AvCmResult AvCmFlush(
    AvCmHc hc,
    DWORD flags);
```

Параметры

[in] AvCmHc hc – дескриптор открытой сессии

[in] DWORD flags – режимы работы функции.
В настоящий момент нет поддерживаемых флагов.

Результат

Код возврата типа **AvCmResult**
Возможные значения:

- **AVCMR_SUCCESS** – успешное отключение пользователя
- **AVCMR_BAD_HC** – дескриптор соединения неверен
- **AVCMR_BUSY** – невозможно закрыть соединение, соединение занято
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.

Высокоуровневые функции сообщений

Генерация подписанного сообщение

Функция **AvCmSign** предназначена для генерации подписанного сообщения. На вход функции подается сообщение, на выходе – сообщение, подписанное активным сертификатом пользователя в формате. *PKCS#7 SignedData*, либо просто сообщение отформатированное как *PKCS#7 SignedData* – которое можно в дальнейшем подписать (в зависимости от переданных флагов). Также возможна генерация дополнительной подписи в подписанном сообщении. Размещение памяти для подписанного сообщения может быть произведено самой функцией при помощи функции *Win32 API* **HeapAlloc**.

```
AvCmResult AvCmSign(
    AvCmHc hc,
    const void * input_message,
    size_t input_size,
    void * output_buffer,
    size_t * output_size,
    DWORD flags);
```

Параметры

[in] AvCmHc hc – дескриптор открытой сессии

[in] const void * input_message – указатель на входное сообщение

[in] size_t input_size – размер входного сообщения

[in,out] void * output_buffer –указатель на буфер выходного сообщения Если передан 0 в качестве указателя, функция заполнит требуемый размер.

[in,out] size_t * output_size – указатель на размер буфера выходного сообщения. Если размер меньше необходимого, функция заполнит требуемый размер. В случае успешного выполнения будет заполнен размер получившегося сообщения.

[in] DWORD flags – режимы генерации подписанного сообщения

- **AVCMF_IN_RAW_DATA** – входное сообщение будет вставлено в новое сообщение *PKCS#7 SignedData* в качестве содержания сообщения.
- **AVCMF_IN_PKCS7** – входное сообщение в формате *PKCS#7 SignedData.*
- **AVCMF_DETACHED** – подписанное содержимое не будет включено в сообщение.
- **AVCMF_ADD_ALL_CERT** – в выходное сообщение будут включены все необходимые сертификаты и СОС
- **AVCMF_ADD_SIGN_CERT** – в выходное сообщение будет включен только сертификат подписавшего.
- **AVCMF_ALLOC** – функция должна самостоятельно распределить память для выходного сообщения функцией *Win32 API* **HeapAlloc**, после использования выходного сообщения, вызвавшая программа должна самостоятельно очистить выделенную память функцией **HeapFree**. См. также Распределение памяти.

Если указан данный флаг, то **output_buffer** должен быть указателем на указатель на буфер для извлекаемого атрибута подписи.

- **AVCMF_REPEAT_AUTHENTICATION** – перед выработкой ЭЦП библиотека потребует повторного ввода пароля к контейнеру личных ключей, проверит наличие контейнера на вставленном носителе и убедится в правильности введенного пароля.

Результат

Код возврата типа **`AvCmResult`**

Возможные значения:

- **AVCMR_SUCCESS** – успешное выполнение операции
- **AVCMR_BAD_HC** – дескриптор соединения неверен
- **AVCMR_BUFFER_TOO_SMALL** – выходной буфер слишком мал
- **AVCMR_NO_INPUT** – входное сообщение имеет нулевой размер
- **AVCMR_BAD_FORMAT –** неверный формат сообщения
- **AVCMR_BAD_PASSWORD** – пароль неверен
- **AVCMR_CONTAINER_NOT_FOUND** – не найден контейнер с личными ключами на носителе
- **AVCMR_DEVICE_NOT_FOUND** – не найден носитель
- **AVCMR_ALLOC_ERROR –** ошибка при распределении памяти
- **AVCMR_CERT_NOT_FOUND** – сертификат не найден
- **AVCMR_CERT_SIGN_INVALID** – подпись под сертификатом неверна
- **AVCMR_CERT_EXPIRED** – срок действия личного ключа сертификата неверен
- **AVCMR_CERT_CA_NOT_FOUND** – сертификат издателя не найден
- **AVCMR_CERT_CA_INVALID** – подпись под сертификатом издателя неверна
- **AVCMR_CERT_NOT_FOR_SIGN** – сертификат не предназначен для подписи
- **AVCMR_CRL_NOT_FOUND** – СОС издателя не найден
- **AVCMR_CRL_INVALID** – подпись под СОС издателя неверна или СОС устарел
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.
- **AVCMR_USER_NO_AUTH** – сессия создана без авторизации пользователя.
- **AVCMR_TOKEN_NOT_FOUND** – носитель с личным ключом не установлен
- **AVCMR_REPEAT_AUTHENTICATION_ERROR** – Вставлен неверный носитель или неверно задан личный пароль.

Проверка подписи в сообщении и извлечение исходного сообщения

Функция **AvCmVerifySign** предназначена для проверки подписи и получения исходного сообщения. На вход функции подается подписанное сообщение, на выходе – проверенное содержание сообщения (в случае необходимости), и информация о результате проверки ЭЦП. В случае, если в сообщении несколько подписей, будут проверены **все** подписи. Сообщение считается корректным только в случае, если все подписи верны. Также функция возвращает дескриптор списка сертификатов, чьи подписи проверялись.

Размещение памяти для извлеченного сообщения может быть произведено самой функцией при помощи функции *Win32 API* **HeapAlloc**.

```
AvCmResult AvCmVerifySign(
    AvCmHc hc,
    const void * input_message,
    size_t input_size,
    AvCmHsgnCertEnum * hsgn_cert_enum,
    void * output_buffer,
    size_t * output_size,
    DWORD flags);
```

Параметры

[in] AvCmHc hc – дескриптор открытой сессии

[in] const void * input_message – указатель на входное сообщение

[in] size_t input_size – размер входного сообщения

[out] AvCmHsgnCertEnum * hsgn_cert_enum – дескриптор списка данных о подписи и соответствующих сертификатах, подписи которых были проверены. Для получения самих данных подписей и сертификатов необходимо использовать функцию **AvCmEnumGet**.
После получения дескрипторов их можно использовать для получения информации о проверенных подписях (например, даты и времени подписи) и данных о сертификатах подписавших (функции **AvCmGetSignAttr** и **AvCmGetCertAttr**).
После завершения работы со списком данных подписей и сертификатов список необходимо закрыть вызовом **AvCmCloseHandle**.

[in,out] void * output_buffer – указатель на буфер сообщения без подписи

[in,out] size_t * output_size – указатель на размер буфера выходного сообщения. На выходе функция заполнит требуемый размер.

[in] DWORD flags – режимы генерации подписанного сообщения

- **AVCMF_NO_OUTPUT** – не заполнять выходное сообщение
- **AVCMF_IMPORT** – импортирует сертификаты и списки отозванных сертификатов из сообщения в хранилище сертификатов
- **AVCMF_ALLOC** – функция должна самостоятельно распределить память для выходного сообщения функцией *Win32 API* **HeapAlloc**, после использования выходного сообщения, вызвавшая программа должна самостоятельно очистить выделенную память функцией **HeapFree**. См. также Распределение памяти.

Если указан данный флаг, то **output_buffer** должен быть указателем на указатель на буфер для извлекаемого атрибута подписи.

- **AVCMF_NO_CERT_VERIFY** – не проверять доверие к сертификату подписавшего. При указании этого флага не будет производится проверка цепочки сертификатов до сертификата корневого ЦС. Также не будет производится поиск сертификата в СОС центра сертификации.

Внимание!
Целостность справочников сертификатов при проверке ЭЦП с указанным флагом **AVCMF_NO_CERT_VERIFY** должна контролироваться иными средствами.

- **AVCMF_IMPORT_CRL** – импортировать все необходимые СОС из сообщения.

- **AVCMF_VERIFY_ON_SIGN_DATE** – проверять ЭЦП на дату выработки электронной подписи, а не на текущую дату.

Внимание!
Контроль подлинности даты выработки ЭЦП в электронном сообщении должен обеспечиваться иными средствами.

Результат

Код возврата типа **`AvCmResult`**
Возможные значения:

- **AVCMR_SUCCESS** – успешная проверка подписи
- **AVCMR_ALLOC_ERROR –** ошибка при распределении памяти
- **AVCMR_BAD_FORMAT –** неверный формат сообщения
- **AVCMR_BAD_HC** – дескриптор соединения неверен
- **AVCMR_BAD_HMSG** – дескриптор открытого сообщения неверен
- **AVCMR_BAD_HCERT** – дескриптор сертификата неверен
- **AVCMR_BUFFER_TOO_SMALL** – выходной буфер слишком мал
- **AVCMR_CERT_NOT_FOR_SIGN** – сертификат не предназначен для подписи
- **AVCMR_NO_SIGN –** Сообщение не имеет подписи, выработанной на ключе данного сертификата.
- **AVCMR_SIGN_INVALID –** подпись неверна
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.
- **AVCMR_NO_CONTENT** – Сообщение не имеет содержимого.

Зашифрование сообщения

Функция **AvCmEncrypt** предназначена для зашифрования сообщения. На вход функции подается сообщение, на выходе – сообщение в формате *PKCS#7 EnvelopedData,* зашифрованное для определенных абонентов. Размещение памяти для зашифрованного сообщения может быть произведено самой функцией при помощи функции *Win32 API* **HeapAlloc.**

```
AvCmResult AvCmEncrypt(
    AvCmHc hc,
    const void * input_message,
    size_t input_size,
    size_t cert_count,
    const AvCmHcert * certificates,
    void * output_buffer,
    size_t * output_size,
    DWORD flags);
```

Параметры

[in] AvCmHc hc – дескриптор открытой сессии

[in] const void * input_message – указатель на входное сообщение

[in] size_t input_size – размер входного сообщения

[in] size_t cert_count – количество абонентов, для которых создается зашифрованное сообщение

[in] const AvCmHcert * certificates – массив дескрипторов сертификатов получателей сообщения

[in,out] void * output_buffer – указатель на буфер выходного сообщения Если передан 0 в качестве указателя, функция заполнит требуемый размер.

[in,out] size_t * output_size – указатель на размер буфера выходного сообщения. Если меньше необходимого, функция заполнит требуемый размер. В случае успешного выполнения будет заполнен размер получившегося сообщения.

[in] DWORD flags – режимы генерации подписанного сообщения

2. **AVCMF_ALLOC** – функция должна самостоятельно распределить память для выходного сообщения функцией *Win32 API* **HeapAlloc**, после использования выходного сообщения, вызвавшая программа должна самостоятельно очистить выделенную память функцией **HeapFree**. См. также Распределение памяти.

Если указан данный флаг, то **output_buffer** должен быть указателем на указатель на буфер для извлекаемого атрибута подписи.

Результат

Код возврата типа **AvCmResult**

Возможные значения:

- **AVCMR_SUCCESS** – успешное выполнение операции

- **AVCMR_BAD_HC** – дескриптор соединения неверен
- **AVCMR_BAD_HCERT** – дескриптор сертификата неверен
- **AVCMR_BUFFER_TOO_SMALL** – выходной буфер слишком мал
- **AVCMR_NO_INPUT** – входное сообщение имеет нулевой размер
- **AVCMR_ALLOC_ERROR –** ошибка при распределении памяти
- **AVCMR_CERT_NOT_FOUND** – сертификат не найден
- **AVCMR_CERT_SIGN_INVALID** – подпись под сертификатом неверна
- **AVCMR_CERT_EXPIRED** – срок действия личного ключа сертификата неверен
- **AVCMR_CERT_CA_NOT_FOUND** – сертификат издателя не найден
- **AVCMR_CERT_CA_INVALID** – подпись под сертификатом издателя неверна
- **AVCMR_CERT_NOT_FOR_CRYPT** – сертификат не предназначен для зашифрования
- **AVCMR_CRL_NOT_FOUND** – СОС издателя не найден
- **AVCMR_CRL_INVALID** – подпись под СОС издателя неверна или СОС устарел
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.
- **AVCMR_NO_RECIPIENTS** – отсутствуют получатели зашифрованного сообщения.

Расшифрование сообщения

Функция **AvCmDecrypt** предназначена для расшифрования полученного зашифрованного сообщения. На вход функции подается сообщение в формате *PKCS#7 EnvelopedData***,** в списке получателей которого имеется и авторизованный в настоящий момент пользователь. Размещение памяти для расшифрованного сообщения может быть произведено самой функцией при помощи функции *Win32 API* **HeapAlloc**.

```
AvCmResult AvCmDecrypt(
    AvCmHc hc,
    const void * input_message,
    size_t input_size,
    void * output_buffer,
    size_t * output_size,
    DWORD flags);
```

Параметры

[in] AvCmHc hc – дескриптор открытой сессии

[in] const void * input_message – указатель на входное зашифрованное сообщение

[in] size_t input_size – размер входного сообщения

[in,out] void * output_buffer – указатель на буфер выходного сообщения. Если передан 0 в качестве указателя, функция заполнит требуемый размер.

[in,out] size_t * output_size – указатель на размер буфера выходного сообщения. Если меньше необходимого, функция заполнит требуемый размер. В случае успешного выполнения будет заполнен размер получившегося сообщения.

[in] DWORD flags – режимы генерации подписанного сообщения

- **AVCMF_ALLOC** – функция должна самостоятельно распределить память для выходного сообщения функцией *Win32 API* **HeapAlloc**, после использования выходного сообщения, вызвавшая программа должна самостоятельно очистить выделенную память функцией **HeapFree**. См. также Распределение памяти**.**

Если указан данный флаг, то **output_buffer** должен быть указателем на указатель на буфер для извлекаемого атрибута подписи.

Результат

Код возврата типа **AvCmResult**

Возможные значения:

- **AVCMR_SUCCESS** – успешное выполнение операции
- **AVCMR_BAD_HC** – дескриптор соединения неверен
- **AVCMR_BAD_FORMAT –** неверный формат сообщения
- **AVCMR_BUFFER_TOO_SMALL** – выходной буфер слишком мал
- **AVCMR_NO_INPUT** – входное сообщение имеет нулевой размер
- **AVCMR_ALLOC_ERROR –** ошибка при распределении памяти

- **AVCMR_CONTAINER_NOT_FOUND** – не найден контейнер с личными ключами на носителе
- **AVCMR_DEVICE_NOT_FOUND** – не найден носитель
- **AVCMR_CERT_NOT_FOUND** – сертификат не найден
- **AVCMR_CERT_SIGN_INVALID** – подпись под сертификатом неверна
- **AVCMR_CERT_EXPIRED** – срок действия личного ключа сертификата неверен
- **AVCMR_CERT_CA_NOT_FOUND** – сертификат издателя не найден
- **AVCMR_CERT_CA_INVALID** – подпись под сертификатом издателя неверна
- **AVCMR_CRL_NOT_FOUND** – СОС издателя не найден
- **AVCMR_CRL_INVALID** – подпись под СОС издателя неверна или СОС устарел
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.
- **AVCMR_USER_NO_AUTH** – сессия создана без авторизации пользователя.
- **AVCMR_OTHER_RECIPIENT** – невозможно расшифровать сообщение: среди получателей сообщения отсутствует владелец текущей сессии.
- **AVCMR_TOKEN_NOT_FOUND** – носитель с личным ключом не установлен

Подпись и зашифрование сообщения

Функция **AvCmSignAndEncrypt** предназначена для генерации подписанного сообщения с последующим его зашифрованием. На вход функции подается сообщение, на выходе – зашифрованное сообщение в формате *PKCS#7 EnvelopedData* с вложенным зашифрованным подписанным сообщением? подписанным активным сертификатом пользователя в формате *PKCS#7 SignedData*.

Также возможна генерация дополнительной подписи в подписанном сообщении. Размещение памяти для подписанного сообщения может быть произведено самой функцией при помощи функции *Win32 API* **HeapAlloc**.

```
AvCmResult AvCmSignAndEncrypt(
    AvCmHc hc,
    const void * input_message,
    size_t input_size,
    size_t cert_count,
    const AvCmHcert * certificates,
    void * output_buffer,
    size_t * output_size,
    DWORD flags);
```

Параметры

[in] AvCmHc hc – дескриптор открытой сессии

[in] const void * input_message – указатель на входное сообщение

[in] size_t input_size – размер входного сообщения

[in] size_t cert_count – количество абонентов, для которых создается зашифрованное сообщение

[in] const AvCmHcert * certificates – массив дескрипторов сертификатов получателей зашифрованного сообщения

[in,out] void * output_buffer –указатель на буфер выходного сообщения. Если передан 0 в качестве указателя, функция заполнит требуемый размер.

[in,out] size_t * output_size – указатель на размер буфера выходного сообщения. Если меньше необходимого, функция заполнит требуемый размер. В случае успешного выполнения будет заполнен размер получившегося сообщения.

[in] DWORD flags – режимы генерации подписанного сообщения

- **AVCMF_IN_RAW_DATA** – входное сообщение будет вставлено в новое сообщение *PKCS#7 SignedData* в качестве содержания сообщения.
- **AVCMF_IN_PKCS7** – входное сообщение в формате *PKCS#7 SignedData.*
- **AVCMF_ADD_ALL_CERT** – в выходное сообщение будут включены все необходимые сертификаты и СОС
- **AVCMF_ADD_SIGN_CERT** – в выходное сообщение будет включен только сертификат подписавшего.

- **AVCMF_ALLOC** – функция должна самостоятельно распределить память для выходного сообщения функцией *Win32 API* **HeapAlloc**, после использования выходного сообщения, вызвавшая программа должна самостоятельно очистить выделенную память функцией **HeapFree**. См. также Распределение памяти.

Если указан данный флаг, то **output_buffer** должен быть указателем на указатель на буфер для извлекаемого атрибута подписи.

- **AVCMF_REPEAT_AUTHENTICATION** – перед выработкой ЭЦП библиотека потребует повторного ввода пароля к контейнеру личных ключей, проверит наличие контейнера на вставленном носителе и убедится в правильности введенного пароля.

Результат

Код возврата типа **`AvCmResult`**

Возможные значения:

- **AVCMR_SUCCESS** – успешное выполнение операции
- **AVCMR_BAD_HC** – дескриптор соединения неверен
- **AVCMR_BAD_HCERT** – дескриптор сертификата неверен
- **AVCMR_BUFFER_TOO_SMALL** – выходной буфер слишком мал
- **AVCMR_NO_INPUT** – входное сообщение имеет нулевой размер
- **AVCMR_BAD_FORMAT –** неверный формат сообщения
- **AVCMR_BAD_PASSWORD** – пароль неверен
- **AVCMR_CONTAINER_NOT_FOUND** – не найден контейнер с личными ключами на носителе
- **AVCMR_DEVICE_NOT_FOUND** – не найден носитель
- **AVCMR_ALLOC_ERROR –** ошибка при распределении памяти
- **AVCMR_CERT_NOT_FOUND** – сертификат не найден
- **AVCMR_CERT_SIGN_INVALID** – подпись под сертификатом неверна
- **AVCMR_CERT_EXPIRED** – срок действия личного ключа сертификата неверен
- **AVCMR_CERT_CA_NOT_FOUND** – сертификат издателя не найден
- **AVCMR_CERT_CA_INVALID** – подпись под сертификатом издателя неверна
- **AVCMR_CERT_NOT_FOR_SIGN** – сертификат не предназначен для подписи
- **AVCMR_CERT_NOT_FOR_CRYPT** – сертификат не предназначен для зашифрования
- **AVCMR_CRL_NOT_FOUND** – СОС издателя не найден
- **AVCMR_CRL_INVALID** – подпись под СОС издателя неверна или СОС устарел
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.
- **AVCMR_USER_NO_AUTH** – сессия создана без авторизации пользователя.
- **AVCMR_TOKEN_NOT_FOUND** – носитель с личным ключом не установлен
- **AVCMR_REPEAT_AUTHENTICATION_ERROR** – Вставлен неверный носитель или неверно задан личный пароль.
- **AVCMR_NO_RECIPIENTS** – отсутствуют получатели зашифрованного сообщения.

Расшифрование, проверка ЭЦП и извлечение исходного сообщения

Функция **AvCmDecryptAndVerifySign** предназначена для расшифрования сообщения с последующей проверкой подписи, и получения исходного сообщения. На вход функции подается обрабатываемое сообщение и дескриптор подписавшего сертификата, на выходе – исходное сообщение в случае необходимости и информация о результате проверки ЭЦП. Размещение памяти для извлеченного сообщения может быть произведено самой функцией при помощи функции *Win32 API* **HeapAlloc**.

```
AvCmResult AvCmDecryptAndVerifySign(
    AvCmHc hc,
    const void * input_message,
    size_t input_size,
    AvCmHsgnCertEnum * hsgn_cert_enum,
    void * output_buffer,
    size_t * output_size,
    DWORD flags);
```

Параметры

[in] AvCmHc hc – дескриптор открытой сессии

[in] const void * input_message – указатель на входное сообщение

[in] size_t input_size – размер входного сообщения

[out] AvCmHsgnCertEnum * hsgn_cert_enum – дескриптор списка данных о подписи и соответствующих сертификатах, подписи которых были проверены. Для получения самих данных подписей и сертификатов необходимо использовать функцию **AvCmEnumGet**.
После получения дескрипторов их можно использовать для получения информации о проверенных подписях (например, даты и времени подписи) и данных о сертификатах подписавших (функции **AvCmGetSignAttr** и **AvCmGetCertAttr**).
После завершения работы со списком данных подписей и сертификатов список необходимо закрыть вызовом **AvCmCloseHandle**.

[in,out] void * output_buffer – указатель на буфер сообщения без подписи

[in,out] size_t * output_size – указатель на размер буфера выходного сообщения. На выходе функция заполнит требуемый размер.

[in] DWORD flags – режимы генерации подписанного сообщения

- **AVCMF_NO_OUTPUT** – не заполнять выходное сообщение
- **AVCMF_IMPORT** – импортирует сертификаты и списки отозванных сертификатов из сообщения в хранилище сертификатов
- **AVCMF_ALLOC** – функция должна самостоятельно распределить память для выходного сообщения функцией *Win32 API* **HeapAlloc**, после использования выходного сообщения, вызвавшая программа должна самостоятельно очистить выделенную память функцией **HeapFree**. См. также Распределение памяти.

Если указан данный флаг, то **output_buffer** должен быть указателем на указатель на буфер для извлекаемого атрибута подписи.

- **AVCMF_NO_CERT_VERIFY** – не проверять доверие к сертификату подписавшего. При указании этого флага не будет производится проверка цепочки сертификатов до сертификата корневого ЦС. Также не будет производится поиск сертификата в СОС центра сертификации.

Внимание!
Целостность справочников сертификатов при проверке ЭЦП с указанным флагом **AVCMF_NO_CERT_VERIFY** должна контролироваться иными средствами.

- **AVCMF_IMPORT_CRL** – импортировать все необходимые СОС из сообщения.

- **AVCMF_VERIFY_ON_SIGN_DATE** – проверять ЭЦП на дату выработки электронной подписи, а не на текущую дату.

Внимание!
Контроль подлинности даты выработки ЭЦП в электронном сообщении должен обеспечиваться иными средствами.

Результат

Код возврата типа **`AvCmResult`**

Возможные значения:

- **AVCMR_SUCCESS** – успешная проверка подписи
- **AVCMR_ALLOC_ERROR –** ошибка при распределении памяти
- **AVCMR_BAD_FORMAT –** неверный формат сообщения
- **AVCMR_BAD_HC** – дескриптор соединения неверен
- **AVCMR_BAD_HCERT** – дескриптор сертификата неверен
- **AVCMR_BUFFER_TOO_SMALL** – выходной буфер слишком мал
- **AVCMR_CERT_NOT_FOR_SIGN** – сертификат не предназначен для подписи
- **AVCMR_NO_SIGN –** Сообщение не имеет подписи, выработанной на ключе данного сертификата.
- **AVCMR_NO_INPUT** – входное сообщение имеет нулевой размер
- **AVCMR_CONTAINER_NOT_FOUND** – не найден контейнер с личными ключами на носителе
- **AVCMR_DEVICE_NOT_FOUND** – не найден носитель
- **AVCMR_CERT_NOT_FOUND** – сертификат не найден
- **AVCMR_CERT_SIGN_INVALID** – подпись под сертификатом неверна
- **AVCMR_CERT_EXPIRED** – срок действия личного ключа сертификата неверен
- **AVCMR_CERT_CA_NOT_FOUND** – сертификат издателя не найден
- **AVCMR_CERT_CA_INVALID** – подпись под сертификатом издателя неверна
- **AVCMR_CERT_NOT_FOR_SIGN** – сертификат не предназначен для подписи
- **AVCMR_CERT_NOT_FOR_CRYPT** – сертификат не предназначен для зашифрования
- **AVCMR_CRL_NOT_FOUND** – СОС издателя не найден
- **AVCMR_CRL_INVALID** – подпись под СОС издателя неверна или СОС устарел
- **AVCMR_SIGN_INVALID –** подпись неверна
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.
- **AVCMR_USER_NO_AUTH** – сессия создана без авторизации пользователя.
- **AVCMR_OTHER_RECIPIENT** – невозможно расшифровать сообщение: среди получателей сообщения отсутствует владелец текущей сессии.
- **AVCMR_NO_CONTENT** – Сообщение не имеет содержимого.

Работа с дескрипторами объектов

Получение параметров объекта

Функция **AvCmGetObjectInfo** предназначена для получения параметров открытого объекта библиотеки.

```
AvCmResult AvCmGetObjectInfo(
    AvCmHandle handle,
    DWORD param_id,
    void * output_buffer,
    size_t * output_size,
    DWORD flags);
```

Параметры

[in] AvCmHandle handle – дескриптор объекта, параметр которого необходимо получить

[in] DWORD param_id – идентификатор параметра, который требуется получить. В настоящий момент поддерживаются следующие параметры:

- **AVCM_MY_CERT** – получение дескриптора личного сертификата. В этом случае в качестве указателя на выходной буфер должен быть передан указатель на **AvCmHcert**. После использования личного сертификата, его дескриптор должен быть освобожден функцией **AvCmCloseHandle**. Данный параметр может быть использован только при передачи в качестве параметра **handle** дескриптора сессии.
- **AVCM_CHILDREN_COUNT** – получение количество дочерних объектов. В этом случае в качестве указателя на выходной буфер должен быть передан указатель на **DWORD. В** качестве параметра **handle** может быть передан дескриптор любого объекта.

[in,out] void * output_buffer – указатель на буфер выходного сообщения. Если передан 0 в качестве указателя, функция заполнит требуемый размер.

[in,out] size_t * output_size – указатель на размер буфера выходного сообщения. Если размер меньше необходимого, функция заполнит требуемый размер. В случае успешного выполнения будет заполнен размер получившегося сообщения.

[in] DWORD flags – режим работы функции.
В настоящий момент нет поддерживаемых флагов.

Результат

Код возврата типа **DWORD**
Возможные значения:

- **AVCMR_SUCCESS** (0) – успешное выполнение операции
- **AVCMR_BUFFER_TOO_SMALL** – выходной буфер слишком мал
- **AVCMR_ALLOC_ERROR –** ошибка при распределении памяти
- **AVCMR_INTERNAL_ERROR** – внутренняя ошибка библиотеки.
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.

- **AVCMR_BAD_PARAM** – параметр функции неверен

Дублирование дескриптора открытого объекта

Функция **AvCmDuplicateHandle** предназначена для создания копии объекта, управляемого переданным дескриптором объекта библиотеки, полученного в результате одного из вызовов библиотеки. Данная операция, например, необходима для передачи объектов библиотеки между сессиями библиотеки.

Дублировать можно только объекты, не являющиеся вложенными в другие объекты библиотеки, к вложенным объектам относится, например, подпись, поскольку объект подпись вложен в объект сообщение и не может существовать отдельно от сообщения. К объектам библиотеки, доступным для копирования, относятся: сообщение, сертификат, контекст поиска сертификатов.

```
AvCmResult AvCmDuplicateHandle(
    AvCmHandle source_handle,
    AvCmHc dest_hc,
    AvCmHandle * copy_of_handle,
    DWORD flags);
```

Параметры

[in] AvCmHandle source_handle – дескриптор исходного объекта.

[in] AvCmHc dest_hc – дескриптор сессии создаваемой копии объекта

[out] AvCmHandle * copy_of_handle – указатель на область памяти, куда будет помещен дескриптор созданной копии объекта.

[in] DWORD flags – параметры копирования объекта
В настоящий момент нет поддерживаемых флагов.

Результат

Код возврата типа **AvCmResult**
Возможные значения:

- **AVCMR_SUCCESS** – успешное выполнение операции
- **AVCMR_BAD_HC** – дескриптор соединения неверен
- **AVCMR_BAD_HANDLE** – дескриптор объекта неверен
- **AVCMR_BUSY** – невозможно дублировать объект, объект в данный момент используется
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.

Закрытие дескриптора открытого объекта

Функция **AvCmCloseHandle** предназначена для закрытия дескриптора объекта библиотеки, полученного в результате одного из вызовов библиотеки. К объектам библиотеки, доступных через получаемый дескриптор, относятся: подписанное сообщение, подпись из подписанного сообщения, сертификат, контекст поиска сертификатов.

```
AvCmResult AvCmCloseHandle(
    AvCmHandle handle,
    DWORD flags);
```

Параметры

[in] AvCmHandle handle – дескриптор открытого ранее объекта.

[in] DWORD flags – параметры закрытия объекта

Результат

Код возврата типа **AvCmResult**

Возможные значения:

- **AVCMR_SUCCESS** – успешное выполнение операции
- **AVCMR_BAD_HANDLE** – дескриптор объекта неверен
- **AVCMR_BUSY** – невозможно закрыть объект, объект в данный момент используется
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.

Низкоуровневые функции обработки сообщений

Открытие/создание сообщения

Функция **AvCmOpenMsg** предназначена для создания или открытия и разбора подписанного или зашифрованного сообщения. На вход функции подается сообщение в формате *PKCS#7* или данные будущего криптографического сообщения. В результате разбора или создания (если не произошло ошибки) функция возвращает через параметр **hmsg** дескриптор открытого сообщения для дальнейшей работы с ним. По окончанию использования открытого сообщения его необходимо закрыть при помощи **AvCmCloseHandle**.

```
AvCmResult AvCmOpenMsg(
    AvCmHc hc,
    const void * message_data,
    size_t message_size,
    AvCmHmsg * hmsg,
    DWORD flags);
```

Параметры

[in] AvCmHc hc – дескриптор открытой сессии

[in] const void * message_data – указатель на блок памяти, содержащий открываемое сообщение

[in] size_t message_size – размер открываемого сообщения

[out] AvCmHmsg * hmsg – указатель на дескриптор открытого сообщения, который будет получен в случае успешного открытия сообщения.

[in] DWORD flags – параметры(флаги) работы.

- **AVCMF_IN_RAW_DATA** – входные данные будут вставлены в выходное сообщение как *PKCS#7 Data.*
- **AVCMF_IN_PKCS7** – входное сообщение в формате *PKCS #7.*
- **AVCMF_OPEN_FOR_SIGN** – открыть сообщение для подписи с поблочной обработкой данных.
- **AVCMF_OPEN_FOR_VERIFYSIGN** – открыть сообщение для проверки подписи с поблочной обработкой данных.
- **AVCMF_OPEN_FOR_CALC_BELTHASH** – открыть сообщение для вычисления хэш с поблочной обработкой данных.
- **AVCMF_OPEN_FOR_CALC_BHFHASH** – открыть сообщение для вычисления хэш с поблочной обработкой данных.
- **AVCMF_DETACHED** – исходные данные не включены в сообщение.

Результат

Код возврата типа **AvCmResult**
Возможные значения:

- **AVCMR_SUCCESS** – успешное выполнение операции
- **AVCMR_BAD_HC** – дескриптор соединения неверен
- **AVCMR_NO_INPUT** – входное сообщение имеет нулевой размер
- **AVCMR_BAD_FORMAT –** неверный формат сообщения

- **AVCMR_ALLOC_ERROR –** ошибка при распределении памяти
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.

Открытие внутреннего сообщения

Функция **AvCmOpenInnerMsg** предназначена для открытия и разбора вложенного сообщения, после обработки сообщения верхнего уровня подписанного или зашифрованного сообщения. На вход функции подается дескриптор сообщения, которое содержит вложенное сообщение в формате *PKCS#7*. В результате разбора или создания (если не произошло ошибки) функция возвращает через параметр **hmsg_inner** дескриптор открытого вложенного сообщения для дальнейшей работы с ним. По окончанию использования открытого сообщения его необходимо закрыть при помощи **AvCmCloseHandle**.

Данная функция может применяться для работы с сообщениями, содержащими вложенные сообщения, например для подписанных и затем зашифрованных сообщений.

```
AvCmResult AvCmOpenInnerMsg(
    AvCmHmsg hmsg,
    AvCmHmsg * hmsg_inner,
    DWORD flags);
```

Параметры

[in] AvCmHmsg hmsg – указатель на дескриптор открытого сообщения, содержащее вложенное сообщение.

[out] AvCmHmsg * hmsg_inner – указатель на дескриптор открытого сообщения, который будет получен в случае успешного открытия сообщения.

[in] DWORD flags – параметры(флаги) работы.
В настоящий момент нет поддерживаемых флагов.

Результат

Код возврата типа **AvCmResult**
Возможные значения:

- **AVCMR_SUCCESS** – успешное выполнение операции
- **AVCMR_BAD_HMSG** – дескриптор сообщения неверен
- **AVCMR_BAD_FORMAT –** неверный формат сообщения
- **AVCMR_ALLOC_ERROR –** ошибка при распределении памяти
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.

Получение параметров открытого сообщения

Функция **AvCmGetMsgParam** предназначена для получения параметров открытого сообщения, например количества подписей под сообщением, открытым функцией **AvCmOpenMsg** в том случае, если открытое сообщение является подписанным сообщением. На вход функции подается дескриптор открытого сообщения, идентификатор параметра, который необходимо получить, и указатель на буфер и его размер, которые будет заполнены в случае успешного выполнения функции. Размещение памяти для извлекаемого атрибута может быть произведено самой функцией при помощи функции *Win32 API* **HeapAlloc**. См. Распределение памяти.

```
AvCmResult AvCmGetMsgParam(
    AvCmHmsg hmsg,
    DWORD attr_id,
    void * output_buffer,
    size_t * output_size,
    DWORD flags);
```

Параметры

[in] AvCmHmsg hmsg – дескриптор открытого ранее сообщения.

[in] DWORD attr_id – идентификатор извлекаемого атрибута:

- **AVCM_FORMAT** – вид открытого сообщения:
  Результат: **DWORD:**
  - **AVCM_MF_RAW_DATA** – данные не имеют структуры *PKCS#7.*
  - **AVCM_MF_SIGNED_DATA** – подписанное сообщение в формате PKCS#7 SignedData.
  - **AVCM_MF_ENVELOPED_DATA** – зашифрованное сообщение в формате *PKCS#7 EnvelopedData.*

- **AVCM_INNER_FORMAT** – формат вложенного сообщения:
  Результат: **DWORD:**
  - **AVCM_MF_NONE** – сообщение не содержит вложенное сообщение.
  - **AVCM_MF_RAW_DATA** – данные не имеют структуры *PKCS#7.*
  - **AVCM_MF_SIGNED_DATA** – подписанное сообщение в формате *PKCS#7 SignedData.*
  - **AVCM_MF_ENVELOPED_DATA** – зашифрованное сообщение в формате *PKCS#7 EnvelopedData.*

- **AVCM_SIGN_COUNT** – количество подписей в подписанном сообщении.
  Результат: **size_t (DWORD)**

[in,out] void * output_buffer – указатель на буфер для извлекаемого атрибута сообщения

[in,out] size_t * output_size – указатель на размер буфера извлекаемого атрибута подписи. Если размер меньше необходимого и не указан флаг **AVCMF_ALLOC**, функция заполнит требуемый размер. . В случае успешного выполнения будет заполнен размер полученного значения.

[in] DWORD flags – параметры

- **AVCMF_ALLOC** – функция должна самостоятельно распределить память для выходного сообщения функцией *Win32 API* **HeapAlloc**, после использования выходного сообщения, вызвавшая программа должна самостоятельно очистить выделенную память функцией **HeapFree**. См. также Распределение памяти**.**

Если указан данный флаг, то **output_buffer** должен быть указателем на указатель на буфер для извлекаемого атрибута подписи.

Результат

Код возврата типа **`AvCmResult`**

Возможные значения:

- **AVCMR_SUCCESS** – успешное выполнение операции
- **AVCMR_BAD_HMSG** – дескриптор открытого сообщения неверен
- **AVCMR_ALLOC_ERROR –** ошибка при распределении памяти
- **AVCMR_BUFFER_TOO_SMALL –** выходной буфер слишком мал
- **AVCMR_BAD_FLAGS –** функции переданы неверные флаги.

Установка дополнительных атрибутов подписанного сообщения

Функция **AvCmMsgSetAttribute** предназначена для установки дополнительных атрибутов открытого сообщения для подписи. На вход функции подается дескриптор открытого сообщения, идентификатор параметра, идентификатор атрибута, который необходимо установить, и указатель на буфер и его размер, который содержит данные устанавливаемого атрибута.

```
AvCmResult AvCmMsgSetAttribute(
    AvCmHmsg hmsg,
    DWORD attr_id,
    DWORD attr_param,
    void * input_buffer,
    size_t input_size,
    DWORD flags);
```

Параметры

[in] AvCmHmsg hmsg – дескриптор открытого сообщения.

[in] DWORD attr_id – идентификатор параметра:

- **AVCM_ATTRIBUTE_CERTS** – атрибутный сертификат
- **AVCM_PKCS7_CERTS** – сертификат открытых ключей
- **AVCM_PKCS7_CRLS** – СОС
- **AVCM_AUTH_AS_STRING** – авторизованный атрибут
- **AVCM_UNAUTH_AS_STRING** – неавторизованный атрибут:

[in] void * attr_param – указатель на идентификатор (OID) устанавливаемого атрибута подписи, для AVCM_ATTRIBUTE_CERTS, AVCM_PKCS7_CERTS, AVCM_PKCS7_CRLS указатель на дескриптор добавляемого объекта в сообщение.

[in] void * input_buffer – указатель на буфер содержимого добавляемого атрибута.

[in] size_t input_size – указатель на размер буфера.

[in] DWORD flags – параметры

Результат

Код возврата типа **AvCmResult**

Возможные значения:

- **AVCMR_SUCCESS** – успешное выполнение операции
- **AVCMR_BAD_ATTR –** идентификатор атрибута неверен
- **AVCMR_BAD_HMSG** – дескриптор открытого сообщения неверен
- **AVCMR_BAD_FLAGS –** функции переданы неверные флаги.

Подпись открытого сообщения

Функция **AvCmMsgSign** предназначена для выработки подписи под сообщением, открытым или созданным функцией **AvCmOpenMsg**. При этом, в зависимости от текущего формата сообщения, подпись может быть вставлена в текущее подписанное сообщение или может быть сгенерировано новое подписанное сообщение в формате *PKCS#7 SignedData*, а текущее сообщение будет включено в новое как поле данных сообщения. В первом случае будет подписано только поле данных исходного сообщения, а во втором – все исходное сообщение. Включение новой подписи возможно, только если формат исходного сообщения *PKCS#7 SignedData*. Подпись вырабатывается на личном ключе, открытого в текущей сессии. При этом в сообщение могут быть добавлены необходимые для проверки подписи сообщения сертификаты и СОС. В случае успешного выполнения, независимо от предыдущего формата сообщения, новым форматом сообщения будет *PKCS#7 SignedData*.

```
AvCmResult AvCmMsgSign(
    AvCmHmsg hmsg,
    DWORD flags);
```

Параметры

[in] AvCmHmsg hmsg – дескриптор открытого сообщения (дескриптор подписи, при выработке контрподписи).

[in] DWORD flags – режимы генерации подписанного сообщения

- **AVCMF_DETACHED –** подписанное содержимое не будет включено в сообщение.
- **AVCMF_ADD_ALL_CERT** – в выходное сообщение будут включены все необходимые сертификаты и СОС.
- **AVCMF_ADD_ONLY_CERT** – в выходное сообщение будут включены все необходимые сертификаты.
- **AVCMF_ADD_SIGN_CERT** – в выходное сообщение будет включен только сертификат подписавшего.
- **AVCMF_REPEAT_AUTHENTICATION** – перед выработкой ЭЦП библиотека потребует повторного ввода пароля к контейнеру личных ключей, проверит наличие контейнера на вставленном носителе и убедится в правильности введенного пароля.
- **AVCMF_COUNTER_SIGNATURE** – выработка контрподписи.

Результат

Код возврата типа **AvCmResult**

Возможные значения:

- **AVCMR_SUCCESS** – успешное выполнение операции
- **AVCMR_BAD_HMSG** – дескриптор открытого сообщения неверен
- **AVCMR_BAD_FORMAT –** неверный формат сообщения
- **AVCMR_BAD_PASSWORD** – пароль неверен
- **AVCMR_CONTAINER_NOT_FOUND** – не найден контейнер с личными ключами на носителе
- **AVCMR_DEVICE_NOT_FOUND** – не найден носитель
- **AVCMR_ALLOC_ERROR –** ошибка при распределении памяти
- **AVCMR_CERT_NOT_FOUND** – сертификат не найден
- **AVCMR_CERT_SIGN_INVALID** – подпись под сертификатом неверна

- **AVCMR_CERT_EXPIRED** – срок действия личного ключа сертификата неверен
- **AVCMR_CERT_CA_NOT_FOUND** – сертификат издателя не найден
- **AVCMR_CERT_CA_INVALID** – подпись под сертификатом издателя неверна
- **AVCMR_CERT_NOT_FOR_SIGN** – сертификат не предназначен для подписи
- **AVCMR_CRL_NOT_FOUND** – СОС издателя не найден
- **AVCMR_CRL_INVALID** – подпись под СОС издателя неверна или СОС устарел
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.
- **AVCMR_USER_NO_AUTH** – сессия создана без авторизации пользователя.
- **AVCMR_TOKEN_NOT_FOUND** – носитель с личным ключом не установлен
- **AVCMR_REPEAT_AUTHENTICATION_ERROR** – Вставлен неверный носитель или неверно задан личный пароль.

Генерация СТБ-подписи открытого сообщения

Устаревшая, используйте **AvCmSignRawData** *вместо неё.*

Функция **AvCmMsgGetSTBSign** предназначена для выработки подписи под сообщением (или содержимым сообщения), открытым или созданным функцией **AvCmOpenMsg**. При этом никаких изменений над открытым сообщением проведено не будет, а выработанная по стандарту СТБ подпись будет выдана в буфер.

```
AvCmResult AvCmMsgGetSTBSign(
    AvCmHmsg hmsg,
    void * output_buffer,
    size_t * output_size,
    DWORD flags);
```

Параметры

[in] AvCmHmsg hmsg – дескриптор открытого сообщения.

[in,out] void * output_buffer – указатель на буфер для подписи. Если передан 0 в качестве указателя, функция заполнит требуемый размер.

[in,out] size_t * output_size – указатель на размер буфера для подписи. Если размер меньше необходимого, функция заполнит требуемый размер. В случае успешного выполнения будет заполнен размер получившегося сообщения.

[in] DWORD flags – режимы генерации подписанного сообщения.

- **AVCMF_ALLOC** – функция должна самостоятельно распределить память для выходного сообщения функцией *Win32 API* **HeapAlloc**, после использования выходного сообщения, вызвавшая программа должна самостоятельно очистить выделенную память функцией **HeapFree**. См. также Распределение памяти.

Если указан данный флаг, то **output_buffer** должен быть указателем на указатель на буфер для извлекаемого атрибута подписи.

Результат

Код возврата типа **AvCmResult**

Возможные значения:

- **AVCMR_SUCCESS** – успешное выполнение операции
- **AVCMR_BAD_HMSG** – дескриптор открытого сообщения неверен
- **AVCMR_BAD_FORMAT** – неверный формат сообщения
- **AVCMR_CONTAINER_NOT_FOUND** – не найден контейнер с личными ключами на носителе
- **AVCMR_DEVICE_NOT_FOUND** – не найден носитель
- **AVCMR_CERT_NOT_FOUND** – сертификат не найден
- **AVCMR_CERT_SIGN_INVALID** – подпись под сертификатом неверна
- **AVCMR_CERT_EXPIRED** – срок действия личного ключа сертификата неверен
- **AVCMR_CERT_CA_NOT_FOUND** – сертификат издателя не найден
- **AVCMR_CERT_CA_INVALID** – подпись под сертификатом издателя неверна
- **AVCMR_CERT_NOT_FOR_SIGN** – сертификат не предназначен для подписи
- **AVCMR_CRL_NOT_FOUND** – СОС издателя не найден
- **AVCMR_CRL_INVALID** – подпись под СОС издателя неверна или СОС устарел

- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.
- **AVCMR_ALLOC_ERROR –** ошибка при распределении памяти
- **AVCMR_BUFFER_TOO_SMALL** – выходной буфер слишком мал
- **AVCMR_USER_NO_AUTH** – сессия создана без авторизации пользователя.

Проверка подписи в сообщении

Функция **AvCmMsgVerifySign** предназначена для проверки подписи в открытом подписанном сообщении. Функции необходимо передать дескриптор сертификата, подпись которого необходимо проверить. Параметры проверки ЭЦП передаются при помощи флагов функции.

```
AvCmResult AvCmMsgVerifySign(
    AvCmHmsg hmsg,
    AvCmHcert hcert,
    DWORD flags);
```

Параметры

[in] AvCmHmsg hmsg – дескриптор открытого сообщения.

[in] AvCmHcert hcert – дескриптор сертификата, подпись которого будет проверена

[in] DWORD flags – режимы проверки подписи сообщения

- **AVCMF_NO_CERT_VERIFY** – не проверять доверие к сертификату подписавшего. При указании этого флага не будет производится проверка цепочки сертификатов до сертификата корневого ЦС. Также не будет производится поиск сертификата в СОС центра сертификации.

Внимание!
Целостность справочников сертификатов при проверке ЭЦП с указанным флагом должна контролироваться иными средствами.

- **AVCMF_NO_CRL_VERIFY** – не проверять отсутствие сертификата в СОС удостоверяющего центра.

Внимание!
Целостность справочников сертификатов при проверке ЭЦП с указанным флагом должна контролироваться иными средствами.

- **AVCMF_IMPORT_CRL** – импортировать все необходимые СОС из сообщения.
- **AVCMF_VERIFY_ON_SIGN_DATE** – проверять ЭЦП на дату выработки электронной подписи, а не на текущую дату.

Внимание!
Контроль подлинности даты выработки ЭЦП в электронном сообщении должен обеспечиваться иными средствами.

Результат

Код возврата типа **AvCmResult**

Возможные значения:

- **AVCMR_SUCCESS** – успешная проверка подписи
- **AVCMR_ALLOC_ERROR –** ошибка при распределении памяти
- **AVCMR_BAD_FORMAT –** неверный формат сообщения
- **AVCMR_BAD_HMSG** – дескриптор открытого сообщения неверен

- **AVCMR_BAD_HCERT** – дескриптор сертификата неверен
- **AVCMR_BUFFER_TOO_SMALL** – выходной буфер слишком мал
- **AVCMR_CERT_NOT_FOR_SIGN** – сертификат не предназначен для подписи
- **AVCMR_NO_SIGN –** Сообщение не имеет подписи, выработанной на ключе данного сертификата.
- **AVCMR_SIGN_INVALID –** подпись неверна
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.
- **AVCMR_NO_CONTENT** – сообщение не имеет содержимого.

Проверка подписи в сообщении по номеру

Функция **AvCmMsgVerifySignAtIndex** предназначена для проверки подписи в открытом подписанном сообщении. Функции необходимо передать номер подписи, которую необходимо проверить. Параметры проверки ЭЦП передаются при помощи флагов функции.

```
AvCmResult AvCmMsgVerifySignAtIndex(
    AvCmHmsg hmsg,
    size_t sign_index,
    DWORD flags);
```

Параметры

[in] AvCmHmsg hmsg – дескриптор открытого сообщения.

[in] size_t sign_index – номер требуемой подписи.

[in] DWORD flags – режимы проверки подписи сообщения

- **AVCMF_VERIFY_ON_SIGN_DATE** – проверять ЭЦП на дату выработки электронной подписи, а не на текущую дату.

Внимание!
Контроль подлинности даты выработки ЭЦП в электронном сообщении должен обеспечиваться иными средствами.

- **AVCMF_NO_CERT_VERIFY** – не проверять доверие к сертификату подписавшего. При указании этого флага не будет производится проверка цепочки сертификатов до сертификата корневого ЦС. Также не будет производится поиск сертификата в СОС центра сертификации.

Внимание!
Целостность справочников сертификатов при проверке ЭЦП с указанным флагом должна контролироваться иными средствами.

- **AVCMF_NO_CRL_VERIFY** – не проверять отсутствие сертификата в СОС удостоверяющего центра.

Внимание!
Целостность справочников сертификатов при проверке ЭЦП с указанным флагом должна контролироваться иными средствами.

Результат

Код возврата типа **AvCmResult**

Возможные значения:

- **AVCMR_SUCCESS** – успешная проверка подписи
- **AVCMR_ALLOC_ERROR –** ошибка при распределении памяти
- **AVCMR_BAD_FORMAT –** неверный формат сообщения
- **AVCMR_BAD_HMSG** – дескриптор открытого сообщения неверен
- **AVCMR_BUFFER_TOO_SMALL** – выходной буфер слишком мал
- **AVCMR_SIGN_INVALID –** подпись неверна
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.

- **AVCMR_NO_CONTENT** – сообщение не имеет содержимого.

Проверка подписи в сообщении по номеру на заданную дату

Функция **AvCmMsgVerifySignAtIndexForDate** предназначена для проверки подписи в открытом подписанном сообщении. Функции необходимо передать номер подписи, которую необходимо проверить и дату на которую будет проверятся валидность подписавшего сертификата. Параметры проверки ЭЦП передаются при помощи флагов функции.

```
AvCmResult AvCmMsgVerifySignAtIndex(
    AvCmHmsg hmsg,
    SystemTime * verifydate,
    size_t sign_index,
    DWORD flags);
```

Параметры

[in] AvCmHmsg hmsg – дескриптор открытого сообщения.

[in] size_t sign_index – номер требуемой подписи.

[in] SystemTime * verifydate - дату на которую будет проверятся валидность подписавшего сертификата.

[in] DWORD flags – режимы проверки подписи сообщения

Результат

Код возврата типа **AvCmResult**

Возможные значения:

- **AVCMR_SUCCESS** – успешная проверка подписи
- **AVCMR_ALLOC_ERROR –** ошибка при распределении памяти
- **AVCMR_BAD_FORMAT –** неверный формат сообщения
- **AVCMR_BAD_HMSG** – дескриптор открытого сообщения неверен
- **AVCMR_BUFFER_TOO_SMALL** – выходной буфер слишком мал
- **AVCMR_SIGN_INVALID –** подпись неверна
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.
- **AVCMR_NO_CONTENT** – сообщение не имеет содержимого.

Зашифрование сообщения

Функция **AvCmMsgEncrypt** предназначена для зашифрования открытого сообщения. На вход функции подается дескриптор открытого сообщения и список сертификатов получателей сообщения. Список сертификатов получателей необходимо получить при помощи функций перебора/поиска сертификатов. См. Отбор сертификатов для отправки зашифрованных сообщений. В случае успешного выполнения операции, исходное сообщение будет заменено на зашифрованное для выбранных абонентов.

```
AvCmResult AvCmMsgEncrypt(
    AvCmHmsg hmsg,
    size_t cert_count,
    const AvCmHcert * certificates,
    DWORD flags);
```

Параметры

[in] AvCmHmsg hmsg – дескриптор открытого сообщения, которое необходимо зашифровать.

[in] size_t cert_count – количество абонентов, для которых создается зашифрованное сообщение

[in] const AvCmHcert * certificates – массив дескрипторов сертификатов получателей сообщения

[in] DWORD flags – режимы генерации подписанного сообщения
В настоящий момент нет поддерживаемых флагов.

Результат

Код возврата типа **AvCmResult**
Возможные значения:

- **AVCMR_SUCCESS** – успешное выполнение операции
- **AVCMR_BAD_HCERT** – дескриптор сертификата неверен
- **AVCMR_NO_INPUT** – входное сообщение имеет нулевой размер
- **AVCMR_ALLOC_ERROR –** ошибка при распределении памяти
- **AVCMR_CERT_NOT_FOUND** – сертификат не найден
- **AVCMR_CERT_SIGN_INVALID** – подпись под сертификатом неверна
- **AVCMR_CERT_EXPIRED** – срок действия личного ключа сертификата неверен
- **AVCMR_CERT_CA_NOT_FOUND** – сертификат издателя не найден
- **AVCMR_CERT_CA_INVALID** – подпись под сертификатом издателя неверна
- **AVCMR_CERT_NOT_FOR_CRYPT** – сертификат не предназначен для зашифрования
- **AVCMR_CRL_NOT_FOUND** – СОС издателя не найден
- **AVCMR_CRL_INVALID** – подпись под СОС издателя неверна или СОС устарел
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.
- **AVCMR_NO_RECIPIENTS** – отсутствуют получатели зашифрованного сообщения.

Расшифрование сообщения

Функция **AvCmMsgDecrypt** предназначена для расшифрования открытого зашифрованного сообщения. Зашифрованное сообщение должно быть предварительно открыто функцией **AvCmMsgOpen**. В списке получателей сообщения должен быть и авторизованный в настоящий момент пользователь.

В случае удачного расшифрования следует проверить тип вложенного сообщения, если оно является сообщением в формате *PKCS#7*, то его можно открыть функцией **AvCmOpenInnerMsg** и подвергнуть дальнейшей обработки функциями библиотеки. Если дополнительная обработка не требуется, то расшифрованное сообщение следует извлечь функцией **AvCmGetMsgContent**.

```
AvCmResult AvCmMsgDecrypt(
    AvCmHmsg hmsg,
    DWORD flags);
```

Параметры

[in] AvCmHmsg hmsg – дескриптор открытого сообщения.

[in] DWORD flags – режимы расшифрования сообщения.
В настоящий момент нет поддерживаемых флагов.

Результат

Код возврата типа **AvCmResult**
Возможные значения:

- **AVCMR_SUCCESS** – успешное выполнение операции
- **AVCMR_BAD_HMSG** – дескриптор открытого сообщения неверен
- **AVCMR_BAD_FORMAT –** неверный формат сообщения
- **AVCMR_ALLOC_ERROR –** ошибка при распределении памяти
- **AVCMR_CONTAINER_NOT_FOUND** – не найден контейнер с личными ключами на носителе
- **AVCMR_DEVICE_NOT_FOUND** – не найден носитель
- **AVCMR_CERT_NOT_FOUND** – сертификат не найден
- **AVCMR_CERT_SIGN_INVALID** – подпись под сертификатом неверна
- **AVCMR_CERT_EXPIRED** – срок действия личного ключа сертификата неверен
- **AVCMR_CERT_CA_NOT_FOUND** – сертификат издателя не найден
- **AVCMR_CERT_CA_INVALID** – подпись под сертификатом издателя неверна
- **AVCMR_CRL_NOT_FOUND** – СОС издателя не найден
- **AVCMR_CRL_INVALID** – подпись под СОС издателя неверна или СОС устарел
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги
- **AVCMR_USER_NO_AUTH** – сессия создана без авторизации пользователя.
- **AVCMR_OTHER_RECIPIENT** – невозможно расшифровать сообщение: среди получателей сообщения отсутствует владелец текущей сессии.
- **AVCMR_TOKEN_NOT_FOUND** – носитель с личным ключом не установлен

Получение/экспорт подготовленного сообщения

Функция **AvCmGetMsg** предназначена для получения сгенерированного сообщения в одном из экспортируемых форматов. Данная функция должна применяться после выполнения необходимых операций над сообщением, например выработки подписи и зашифрования. В настоящий момент поддерживается формат экспорта *PKCS#7*.

На вход функции подается дескриптор открытого и подготовленного сообщения и указание требуемого формата в виде соответствующего флага, и указатель на буфер и его размер, которые будет заполнены в случае успешного выполнения функции. Размещение памяти для извлекаемого атрибута может быть произведено самой функцией при помощи функции *Win32 API* **HeapAlloc**. См. Распределение памяти.

```
AvCmResult AvCmGetMsg(
    AvCmHmsg hmsg,
    void * output_buffer,
    size_t * output_size,
    DWORD flags);
```

Параметры

[in] AvCmHmsg hmsg – дескриптор сообщения.

[in,out] void * output_buffer – указатель на буфер для экспортируемого сообщения.

[in,out] size_t * output_size – указатель на размер буфера. Если размер меньше необходимого и не указан флаг **AVCMF_ALLOC**, функция заполнит требуемый размер. В случае успешного выполнения будет заполнен размер полученного значения.

[in] DWORD flags – параметры

- **AVCMF_OUT_PKCS7** – выходное сообщение в формате *PKCS #7*.
- **AVCMF_ALLOC** – функция должна самостоятельно распределить память для выходного сообщения функцией *Win32 API* **HeapAlloc**, после использования выходного сообщения, вызвавшая программа должна самостоятельно очистить выделенную память функцией **HeapFree**. См. также Распределение памяти.

Если указан данный флаг, то **output_buffer** должен быть указателем на указатель на буфер для извлекаемого атрибута подписи.

Результат

Код возврата типа **AvCmResult**

Возможные значения:

- **AVCMR_SUCCESS** – успешное выполнение операции
- **AVCMR_BAD_HMSG** – дескриптор открытого сообщения неверен
3. **AVCMR_ALLOC_ERROR –** ошибка при распределении памяти
4. **AVCMR_BUFFER_TOO_SMALL** – выходной буфер слишком мал
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.

Извлечение содержимого сообщения

Функция **AvCmGetMsgContent** предназначена для извлечения содержимого сообщения. Данная функция может применяться после выполнения необходимых операций над полученным сообщением, например расшифрования и проверки подписей.

В первом режиме работы на вход функции подается дескриптор открытого сообщения и указатель на буфер и его размер, которые будет заполнены в случае успешного выполнения функции. Размещение памяти для извлекаемого атрибута может быть произведено самой функцией при помощи функции *Win32 API* **HeapAlloc**. См. Распределение памяти.

```
AvCmResult AvCmGetMsgContent(
    AvCmHmsg hmsg,
    void * output_buffer,
    size_t * output_size,
    DWORD flags);
```

Параметры

[in] AvCmHmsg hmsg – дескриптор сообщения.

[in,out] void * output_buffer – указатель на буфер для содержимого сообщения.

[in,out] size_t * output_size – указатель на размер буфера. Если размер меньше необходимого и не указан флаг **AVCMF_ALLOC**, функция заполнит требуемый размер. Если работа функции завершится успешно, функция укажет действительный размер заполненных данных.

[in] DWORD flags – параметры

- **AVCMF_ALLOC** – функция должна самостоятельно распределить память для выходного сообщения функцией *Win32 API* **HeapAlloc**, после использования выходного сообщения, вызвавшая программа должна самостоятельно очистить выделенную память функцией **HeapFree**. См. также Распределение памяти.

Если указан данный флаг, то **output_buffer** должен быть указателем на указатель на буфер для извлекаемого атрибута подписи.

Результат

Код возврата типа **AvCmResult**

Возможные значения:

- **AVCMR_SUCCESS** – успешное выполнение операции
- **AVCMR_BAD_HMSG** – дескриптор открытого сообщения неверен
5. **AVCMR_ALLOC_ERROR –** ошибка при распределении памяти
6. **AVCMR_BUFFER_TOO_SMALL** – выходной буфер слишком мал
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.
- **AVCMR_NO_CONTENT** – сообщение не имеет содержимого.

Установка содержимого сообщения

Функция **AvCmSetMsgContent** предназначена для установки содержимого сообщения либо добавления новых данных к нему (флаг **AVCMF_APPEND**).

На вход функции подается дескриптор открытого сообщения, указатель на буфер, в котором находится содержимое сообщения или новые данные для добавления в содержимое и размер буфера.

```
AvCmResult AvCmSetMsgContent(
    AvCmHmsg hmsg,
    void * input_buffer,
    size_t input_size,
    DWORD flags);
```

Параметры

[in] AvCmHmsg hmsg – дескриптор сообщения.

[in] void * input_buffer – указатель на буфер содержимого сообщения или новых данных для содержимого сообщения..

[in] size_t input_size – указатель на размер буфера.

[in] DWORD flags – параметры

- **AVCMF_APPEND** – функция добавляет переданный буфер к уже установленному содержимому сообщения, вместо того, чтобы заменить его.

Результат

Код возврата типа **AvCmResult**

Возможные значения:

- **AVCMR_SUCCESS** – успешное выполнение операции
- **AVCMR_BAD_HMSG** – дескриптор открытого сообщения неверен
- **AVCMR_ALLOC_ERROR –** ошибка при распределении памяти
- **AVCMR_BAD_FLAGS –** функции переданы неверные флаги.

Поблочная обработка данных сообщения

Функция **AvCmMsgUpdate** предназначена для поблочной обработки данных сообщения.

На вход функции подается дескриптор открытого сообщения, указатель на буфер, в котором находится содержимое сообщения или новые данные для добавления в содержимое и размер буфера, выходные данные содержат указатель на буфер содержимого нового сообщения или исходных данных обрабатываемого сообщения и их размер. Выходные данные должны быть обработаны до следующего вызова функции. В данной реализации функция может применяться только для сообщений открытых с флагами **(AVCMF_OPEN_FOR_SIGN, AVCMF_OPEN_FOR_VERIFYSIGN, AVCMF_OPEN_FOR_ENCRYPT, AVCMF_OPEN_FOR_DECRYPT).**

```
AvCmResult AvCmMsgUpdate(
    AvCmHmsg hmsg,
    void * in_data,
    size_t in_size,
    void * out_data,
    size_t out_size,
    DWORD flags);
```

Параметры

[in] AvCmHmsg hmsg – дескриптор сообщения.

[in] void * in_data – указатель на буфер содержимого сообщения или новых данных для содержимого сообщения.

[in] size_t in_size –размер буфера.

[out] void * out_data – указатель на буфер содержимого нового сообщения или исходных данных обрабатываемого сообщения.

[out] size_t out_size – указатель на размер выходного буфера.

[in] DWORD flags – параметры

- **AVCMF_UPDATE_FINAL** – указывает что переданы все данные сообщения.

Результат

Код возврата типа **AvCmResult**

Возможные значения:

- **AVCMR_SUCCESS** – успешное выполнение операции
- **AVCMR_BAD_HMSG** – дескриптор открытого сообщения неверен
- **AVCMR_ALLOC_ERROR –** ошибка при распределении памяти
- **AVCMR_BAD_FLAGS –** функции переданы неверные флаги.

Извлечение информации о подписях из открытого сообщения

Функция **AvCmGetMsgSign** предназначена для извлечения одной из подписей данного подписанного сообщения. На вход функции подается дескриптор ранее открытого сообщения и номер требуемой подписи (от 0 до количества подписей минус 1).

> Количество подписей необходимо получить функцией **AvCmGetMsgParam**.

```
AvCmResult AvCmGetMsgSign(
    AvCmHmsg hmsg,
    size_t sign_number,
    AvCmHsign * hsign,
    DWORD flags);
```

Параметры

[in] AvCmHmsg hmsg – дескриптор открытого ранее сообщения.

[in] size_t sign_number – номер требуемой подписи.

[out] AvCmHsign * hsign – дескриптор подписи, см. Описание типов.

[in] DWORD flags – параметры (флаги) работы.
В настоящий момент нет поддерживаемых флагов.

Результат

Код возврата типа **AvCmResult**
Возможные значения:

- **AVCMR_SUCCESS** – успешное выполнение операции
- **AVCMR_BAD_HMSG** – дескриптор открытого сообщения неверен
- **AVCMR_BAD_NUMBER** – в сообщении нет подписи с указанным номером
- **AVCMR_ALLOC_ERROR –** ошибка при распределении памяти
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.

Получение атрибутов подписи по данным подписи

Функция **AvCmGetSignAttr** предназначена для извлечения атрибутов подписи. Дескриптор подписи должен предварительно быть получен из открытого сообщения функцией **AvCmGetMsgSign**. Также функция может получать информацию о подписи типа **AvCmHsignCert**, полученную от функций **AvCmVerifySign** и **AvCmDecryptAndVerifySign**.

Размещение памяти для извлекаемого атрибута может быть произведено самой функцией при помощи функции *Win32 API* **HeapAlloc**.

```
AvCmResult AvCmGetSignAttr(
    AvCmHandle handle,
    DWORD attr_id,
    const void * attr_param,
    void * output_buffer,
    size_t * output_size,
    DWORD flags);
```

Параметры

[in] AvCmHandle handle – дескриптор объекта подписи, атрибут которой необходимо извлечь, в зависимости от способа получения:

- **AvCmHsign hsign –** дескриптор подписи, полученной функцией **AvCmGetMsgSign**.
- **AvCmHsignCert * hsign_cert** – дескриптор информации о подписи и соответствующем сертификате, полученную от функций **AvCmVerifySign** и **AvCmDecryptAndVerifySign**.

[in] DWORD attr_id – идентификатор извлекаемого атрибута:

- **AVCM_VERSION** – версия подписи
  Результат:
  output_size: sizeof(DWORD)
  output_buffer: DWORD – версия подписи

- **AVCM_ISSUER_AS_STRING** – имя (*X.509 Name*) издателя подписавшего сертификата в виде строки, в том случае, если это поле сертификата можно представить в виде строки. Если атрибут невозможно представить в виде строки *ASCIIZ*, будет возвращена ошибка **AVCMR_BAD_FORMAT**.
  Результат:
  output_size: размер строки
  output_buffer: строка *ASCIIZ*

- **AVCM_SERIAL_AS_STRING** – серийный номер подписавшего сертификата. Если атрибут невозможно представить в виде строки *ASCIIZ*, будет возвращена ошибка **AVCMR_BAD_FORMAT**.
  Результат:
  output_size: размер серийного номера
  output_buffer: серийный номер в формате строки *ASCIIZ*

- **AVCM_HASH_ALG_OID** – идентификатор алгоритма хэширования

Результат:
output_size: размер строки идентификатора алгоритма хэширования
output_buffer: строка *ASCIIZ* идентификатора алгоритма хэширования

- **AVCM_SIGN_ALG_OID** – идентификатор алгоритма подписи
  Результат:
  output_size: размер строки идентификатора алгоритма подписи
  output_buffer: строка *ASCIIZ* идентификатора алгоритма подписи

- **AVCM_SIGN** – подпись
  Результат:
  output_size: размер подписи
  output_buffer: подпись в виде последовательности байт

- **AVCM_SIGN_DATE_TIME** – дата и время выработки подписи, которые находятся в списке подписанных атрибутов сообщения.
  output_size: sizeof(SYSTEMTIME)
  output_buffer: дата и время в формате **SYSTEMTIME.** При этом дата и время всегда возвращаются для текущего часового пояса. См. *MSDN*

- **AVCM_AUTH_COUNT** – количество подписанных атрибутов
  Результат:
  output_size: sizeof(DWORD)
  output_buffer: DWORD – количество подписанных атрибутов

- **AVCM_UNAUTH_COUNT** – количество неподписанных атрибутов
  Результат:
  output_size: sizeof(DWORD)
  output_buffer: DWORD – количество неподписанных атрибутов

- **AVCM_AUTH_AS_STRING** – подписанный атрибут в виде строки *ASCIIZ*, в том случае, если данный атрибут можно представить в виде строки. При этом необходима передача номера либо идентификатора атрибута (параметр **attr_param**). Если атрибут невозможно представить в виде строки *ASCIIZ*, будет возвращена ошибка **AVCMR_BAD_FORMAT.**
  Результат:
  output_size: размер строки атрибута
  output_buffer: строка *ASCIIZ* атрибута

- **AVCM_AUTH_OID** – идентификатор подписанного атрибута в виде строки *ASCIIZ.* При этом необходима передача номера атрибута (параметр **attr_param**).
  Результат:
  output_size: размер строки идентификатора подписанного атрибута
  output_buffer: строка *ASCIIZ* идентификатора подписанного атрибута

- **AVCM_AUTH_BLOB** – значение подписанного атрибута в виде *BLOB.* При этом необходима передача номера либо идентификатора атрибута (параметр **attr_param**).
  Результат:
  output_size: размер буфера
  output_buffer: буфер со значением подписанного атрибута

- **AVCM_UNAUTH_AS_STRING** – неподписанный атрибут в виде строки *ASCIIZ*, в том случае, если данный атрибут можно представить в виде строки. При этом необходима передача номера либо идентификатора атрибута (параметр **attr_param**). Если атрибут невозможно представить в виде строки *ASCIIZ*, будет возвращена ошибка **AVCMR_BAD_FORMAT**.
  Результат:
  output_size: размер строки атрибута
  output_buffer: строка *ASCIIZ* атрибута

- **AVCM_UNAUTH_OID** – идентификатор неподписанного атрибута в виде строки *ASCIIZ*. При этом необходима передача номера атрибута (параметр **attr_param**).
  Результат:
  output_size: размер строки идентификатора атрибута
  output_buffer: строка *ASCIIZ* идентификатора атрибута

- **AVCM_UNAUTH_BLOB** – значение неподписанного атрибута в виде *BLOB*. При этом необходима передача номера либо идентификатора атрибута (параметр **attr_param**).
  Результат:
  output_size: размер буфера
  output_buffer: буфер со значением атрибута
- **AVCM_OCSP_SIGN_RESPONSE_COUNT** – количество дескрипторов информации о статусе сертификата.
  Результат:
  output_size: sizeof(DWORD)
  output_buffer: количество дескрипторов информации о статусе сертификата
- **AVCM_OCSP_SIGN_RESPONSE_BYNUM** – дескриптор информации о статусе сертификата. При этом необходима передача номера дескриптора (параметр **attr_param**).
  Результат:
  output_size: sizeof(AvCmHandle)
  output_buffer: дескриптор информации о статусе сертификата

[in] const void * attr_param – дополнительный параметр извлекаемого атрибута, применяется при извлечении подписанных и неподписанных атрибутов. При необходимости получения атрибутов по номеру необходимо передать указатель на **size_t** с номером извлекаемого дополнения. При получении дополнения по идентификатору необходимо передать указатель на *ASCIIZ* строку, содержащую *OID* атрибута.

[in,out] void * output_buffer – указатель на буфер для извлекаемого атрибута подписи

[in,out] size_t * output_size – указатель на размер буфера извлекаемого атрибута подписи. Если размер меньше необходимого, функция заполнит требуемый размер. В случае успешного выполнения будет заполнен размер полученного значения.

[in] DWORD flags – параметры
- **AVCMF_ATTR_BY_NUM** – атрибут подписи по номеру
- **AVCMF_ATTR_BY_OID** – атрибут подписи по его идентификатору объекта (*OID*)

- **AVCMF_ALLOC** – функция должна самостоятельно распределить память для выходного сообщения функцией *Win32 API* **HeapAlloc**, после использования выходного сообщения, вызвавшая программа должна самостоятельно очистить выделенную память функцией **HeapFree**. См. также Распределение памяти.

Если указан данный флаг, то **output_buffer** должен быть указателем на указатель на буфер для извлекаемого атрибута подписи.

Результат

Код возврата типа **`AvCmResult`**

Возможные значения:

- **AVCMR_SUCCESS –** успешное выполнение операции
- **AVCMR_BAD_HSIGN –** дескриптор подписи неверен
- **AVCMR_BAD_ATTR –** идентификатор атрибута неверен
- **AVCMR_ALLOC_ERROR –** ошибка при распределении памяти
- **AVCMR_BUFFER_TOO_SMALL –** выходной буфер слишком мал
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.
- **AVCMR_NOT_FOUND** – объект(атрибут объекта) не найден.

Импортирование сертификатов и СОС из сообщения в хранилище

Функция **AvCmMsgImportCerts** предназначена для импортирования сертификатов и СОСов из сообщения в хранилище. В зависимости от переданного флага функция может добавлять все сертификаты и СОС либо выводить диалог мастера импорта сертификатов.

```
AvCmResult AvCmMsgImportCerts(
    AvCmHmsg hmsg,
    DWORD flags);
```

Параметры

[in] AvCmHmsg hmsg – дескриптор объекта сообщения, из которого нужно импортировать сертификаты и СОС.

[in] DWORD flags – параметры

- **AVCMF_IMPORT_ALL_CERTS** – импортировать все сертификаты и СОС без вывода диалогового окна.
- **AVCMF_IMPORT_CRL** – импортировать все необходимые СОС без вывода диалогового окна.

Результат

Код возврата типа **AvCmResult**

Возможные значения:

- **AVCMR_SUCCESS –** успешное выполнение операции
- **AVCMR_BAD_HMSG –** дескриптор сообщения неверен
- **AVCMR_BAD_FLAGS –** функции переданы неверные флаги.

Импортирование сертификатов в открытое сообщение

Функция **AvCmMsgAddCert** предназначена для импортирования открытых сертификатов и СОС в подписанное сообщение. Данная функция может использоваться для создания сообщения формата PKCS#7 Signed Data без включенного текста сообщения и его подписей, с включенным набором сертификатов и СОС для транспортных нужд.

```
AvCmResult AvCmMsgAddCert(
    AvCmHmsg hmsg,
    size_t cert_count,
    const AvCmHandle * certificates,
    DWORD flags);
```

Параметры

[in] AvCmHmsg hmsg – дескриптор объекта сообщения, в который нужно импортировать сертификаты и/или СОС.

[in] size_t cert_count – количество импортируемых сертификатов и/или СОС.

[in] const AvCmHandle * certificates – массив импортируемых объектов (**AvCmHcert** или **AvCmHcrl**).

[in] DWORD flags – параметры
В настоящий момент нет поддерживаемых флагов.

Результат

Код возврата типа **AvCmResult**
Возможные значения:

- **AVCMR_SUCCESS –** успешное выполнение операции
- **AVCMR_BAD_HMSG –** дескриптор сообщения неверен
- **AVCMR_BAD_FLAGS –** функции переданы неверные флаги.

Функции работы с сертификатами

Поиск сертификата по данным подписи

Функция **AvCmFindCertBySign** предназначена для поиска сертификата по его подписи. В случае успешного поиска функция вернет дескриптор сертификата, сгенерировавшего эту подпись. Поиск может выполнятся как в локальном справочнике сертификатов, так и в самом сообщении. Зарезервирована возможность поиска сертификата в сетевой инфраструктуре.

По окончанию использования найденного сертификата его необходимо закрыть при помощи **AvCmCloseHandle**.

```
AvCmResult AvCmFindCertBySign(
    AvCmHsign hsign,
    AvCmHcert * hcert,
    DWORD flags);
```

Параметры

[in] AvCmHsign hsign – дескриптор подписи, по которой необходимо найти соответствующий сертификат. См. Описание типов

[out] AvCmHcert * hcert – дескриптор найденного сертификата

[in] DWORD flags – параметры
В настоящий момент нет поддерживаемых флагов.

Результат

Код возврата типа **AvCmResult**
Возможные значения:

- **AVCMR_SUCCESS** – успешное выполнение операции
- **AVCMR_BAD_HSIGN** – дескриптор подписи неверен
- **AVCMR_CERT_NOT_FOUND** – сертификат не найден
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.

Получение дескриптора сертификата из двоичных(der) данных

Функция **AvCmOpenCert** предназначена для получения дескриптора сертификата из двоичных данных (der). По окончанию использования найденного сертификата его необходимо закрыть при помощи **AvCmCloseHandle**.

```
AvCmResult AvCmOpenCert(
    AvCmHc hc,
    void* input_data,
    size_t input_size,
    AvCmHcert * hcert,
    DWORD flags);
```

Параметры

[in] AvCmHc hc – дескриптор открытой сессии

[in] void* input_data – указатель на буфер содержимого сертификата

[in] size_t input_size – размер буфера

[out] AvCmHcert * hcert – дескриптор полученного сертификата

[in] DWORD flags – параметры
В настоящий момент нет поддерживаемых флагов.

Результат

Код возврата типа **AvCmResult**

Получение атрибутов сертификата

Функция **AvCmGetCertAttr** предназначена для извлечения атрибутов сертификата (атрибутного сертификата). Дескриптор сертификата должен предварительно быть получен из открытого сообщения функцией **AvCmFindCertBySign** или перечислением сертификатов в хранилище.

Также функция может получать атрибуты сертификата типа **AvCmHsignCert**, полученного от функций **AvCmVerifySign** и **AvCmDecryptAndVerifySign**.

Размещение памяти для подписанного сообщения может быть произведено самой функцией при помощи функции *Win32 API* **HeapAlloc**.

```
AvCmResult AvCmGetCertAttr(
    AvCmHandle handle,
    DWORD attr_id,
    const void * attr_param,
    void * output_buffer,
    size_t * output_size,
    DWORD flags);
```

Параметры

[in] AvCmHandle handle – дескриптор сертификата, атрибут которого необходимо извлечь, в зависимости от способа получения:

- **AvCmHcert hcert** – дескриптор сертификата, полученного функциями **AvCmFindCertBySign** и **AvCmOpenCertEnum**.
- **AvCmHsignCert hsign_cert** – дескриптор информации о подписи и соответствующем сертификате, полученную от функций **AvCmVerifySign** и **AvCmDecryptAndVerifySign**.

[in] DWORD attr_id – идентификатор извлекаемого атрибута:

- **AVCM_VERSION** – версия сертификата
  Результат:
  output_size: sizeof(DWORD)
  output_buffer: DWORD – версия сертификата

- **AVCM_VALID** – результат полной проверки корректности сертификата
  Результат:
  output_size: sizeof(AvCmResult)
  output_buffer: AvCmResult
  0 – сертификат корректен
  значение отличное от 0 - сертификат не корректен

Для уточнения причины некорректности сертификата необходимо использовать функцию **AvCmGetErrorInfo**. Другим способом проверки действительности сертификата является вызов функции **AvCmVerifyCertStatus**.

- **AVCM_SERIAL_AS_STRING** – серийный номер сертификата. Если атрибут невозможно представить в виде строки *ASCIIZ*, будет возвращена ошибка **AVCMR_BAD_FORMAT**.
  Результат:
  output_size: размер серийного номера
  output_buffer: серийный номер в формате строки *ASCIIZ*

- **AVCM_SERIAL_AS_INTEGER** – серийный номер сертификата в виде *BLOB*. Серийный номер будет возвращен таким образом, что первый значащий байт будет последним.
  Результат:
  output_size: размер серийного номера
  output_buffer: серийный номер в виде большого целого числа (последовательности байт)

- **AVCM_SIGN_ALG_OID** – идентификатор алгоритма подписи
  Результат:
  output_size: размер строки идентификатора алгоритма подписи
  output_buffer: строка *ASCIIZ* идентификатора алгоритма подписи

- **AVCM_ISSUER_AS_STRING** – имя (*X.509 Name*) издателя сертификата в виде строки, в том случае, если это поле сертификата можно представить в виде строки. Если атрибут невозможно представить в виде строки *ASCIIZ*, будет возвращена ошибка **AVCMR_BAD_FORMAT**.
  Результат:
  output_size: размер строки
  output_buffer: строка *ASCIIZ*

- **AVCM_ISSUER_ATTR** – атрибут имени (*X.509 Name*) издателя сертификата в виде строки, в том случае, если это поле сертификата можно представить в виде строки. При этом необходима передача номера либо идентификатора атрибута (параметр **attr_param**).
  Результат:
  output_size: размер строки
  output_buffer: строка *ASCIIZ*

- **AVCM_ISSUER_ATTR_OID** – идентификатор объекта (OID) атрибута имени владельца сертификата в виде строки. При этом необходима передача номера атрибута (параметр **attr_param**).
  Результат:
  output_size: размер строки
  output_buffer: строка *ASCIIZ*

- **AVCM_ISSUER_ATTR_COUNT** – количество атрибутов имени (*X.509 Name*) издателя сертификата.
  Результат:
  output_size: sizeof(DWORD)
  output_buffer: DWORD – количество атрибутов имени

- **AVCM_NOT_BEFORE** – дата/время начала действия сертификата
  output_size: sizeof(SYSTEMTIME)
  output_buffer: дата и время в формате **SYSTEMTIME.** При этом дата и время всегда возвращаются для текущего часового пояса. См. *MSDN*

- **AVCM_NOT_AFTER** – дата/время окончания действия сертификата
  output_size: sizeof(SYSTEMTIME)
  output_buffer: дата и время в формате **SYSTEMTIME.** При этом дата и время всегда возвращаются для текущего часового пояса. См. *MSDN*

- **AVCM_NOT_KEY_BEFORE** – дата/время начала действия личного ключа
  output_size: sizeof(SYSTEMTIME)
  output_buffer: дата и время в формате **SYSTEMTIME.** При этом дата и время всегда возвращаются для текущего часового пояса. См. *MSDN*

- **AVCM_NOT_KEY_AFTER** – дата/время окончания действия личного ключа
  output_size: sizeof(SYSTEMTIME)
  output_buffer: дата и время в формате **SYSTEMTIME.** При этом дата и время всегда возвращаются для текущего часового пояса. См. *MSDN*

- **AVCM_SUBJECT_ATTR** – атрибут имени (*X.509 Name*) владельца сертификата в виде строки, в том случае, если это поле сертификата можно представить в виде строки. При этом необходима передача номера либо идентификатора атрибута (параметр **attr_param**).
  Результат:
  output_size: размер строки
  output_buffer: строка *ASCIIZ*

- **AVCM_SUBJECT_ATTR_OID** – идентификатор объекта (OID) атрибута имени владельца сертификата в виде строки. При этом необходима передача номера атрибута (параметр **attr_param**).
  Результат:
  output_size: размер строки
  output_buffer: строка *ASCIIZ*

- **AVCM_SUBJECT_ATTR_COUNT** – количество атрибутов имени (*X.509 Name*) владельца сертификата.
  Результат:
  output_size: sizeof(DWORD)
  output_buffer: DWORD – количество атрибутов имени

- **AVCM_PUB_KEY_ID** – идентификатор открытого ключа сертификата (значение атрибута subjectKeyIdentifier (2.5.29.14))
  Результат:
  output_size: размер идентификатора открытого ключа
  output_buffer: строка *ASCIIZ* идентификатора
- **AVCM_AUTHORITY_KEY_IDENTIFIER** – идентификатор ключа центра сертификатов (значение атрибута authorityKeyIdentifier (2.5.29.35))
  Результат:
  output_size: размер идентификатора ключа
  output_buffer: строка *ASCIIZ* идентификатора

- **AVCM_PUB_KEY_ALG_OID** – идентификатор алгоритма открытого ключа сертификата
  Результат:
  output_size: размер строки идентификатора
  output_buffer: строка *ASCIIZ* идентификатора

- **AVCM_PUB_KEY** – открытый ключ сертификата
  Результат:
  output_size: размер открытого ключа
  output_buffer: открытый ключ сертификата в виде последовательности байт

- **AVCM_SUBJ_ALT_NAME_ATTR** – атрибут альтернативного имени (*X.509 AltName*) владельца сертификата в виде строки, в том случае, если это поле сертификата можно представить в виде строки. При этом необходима передача номера либо идентификатора атрибута (параметр **attr_param**).
  Результат:
  output_size: размер строки
  output_buffer: строка *ASCIIZ*

- **AVCM_EXT_COUNT** – количество дополнений в сертификате
  Результат:
  output_size: sizeof(DWORD)
  output_buffer: DWORD – количество дополнений

- **AVCM_EXT_OID** – идентификатор объекта (OID) дополнения. При этом необходима передача номера дополнения (параметр **attr_param**).
  Результат:
  output_size: размер строки
  output_buffer: строка *ASCIIZ*

- **AVCM_EXT_OID_NAME** – наименование идентификатора объекта (OID) дополнения, в том случае, если данный OID зарегистрирован в операционной системе, или сам OID в противном случае. При этом необходима передача номера либо идентификатора дополнения (параметр **attr_param**).
  Результат:
  output_size: размер строки наименования идентификатора
  output_buffer: строка *ASCIIZ* наименования идентификатора или значение OID.

- **AVCM_EXT_CRITICAL** – признак критичности дополнения в виде числа формата **DWORD**. При этом 0 означает отсутствие критичности данного дополнения, а любое отличное от 0 значение – критичность дополнения. При этом необходима передача номера либо идентификатора дополнения (параметр **attr_param**).
  Результат:
  output_size: sizeof(DWORD)
  output_buffer: DWORD – булевое значение критичности

- **AVCM_EXT_AS_STRING** – дополнение сертификата в виде строки *ASCIIZ* , в том случае, если данное дополнение можно представить в виде строки. При этом необходима передача номера либо идентификатора дополнения (параметр **attr_param**). Если дополнение невозможно представить в виде строки *ASCIIZ*, будет возвращена ошибка **AVCMR_BAD_FORMAT**.
  Результат:
  output_size: размер строки
  output_buffer: строка *ASCIIZ*

- **AVCM_EXT_BLOB** – значение дополнения сертификата в виде *BLOB*. При этом необходима передача номера либо идентификатора дополнения (параметр **attr_param**).
  Результат:
  output_size: размер буфера
  output_buffer: буфер со значением дополнения

- **AVCM_EXT_KEY_USAGE_COUNT** – количество элементов в списке ограничений применения ключа сертификата
  Результат:
  output_size: sizeof(DWORD)
  output_buffer: DWORD – количество элементов в списке

- **AVCM_EXT_KEY_USAGE_OID** – идентификатор объекта (OID) из списка ограничений применения ключа сертификата. При этом необходима передача номера в списке (параметр **attr_param**).
  Результат:
  output_size: размер строки
  output_buffer: строка *ASCIIZ*

- **AVCM_EXT_KEY_USAGE_NAME** – наименование идентификатора объекта (OID) из списка ограничений применения ключа сертификата, в том случае, если данный OID зарегистрирован в операционной системе. При этом необходима передача номера либо идентификатора в списке (параметр **attr_param**).
  Результат:
  output_size: размер строки наименования идентификатора
  output_buffer: строка *ASCIIZ* наименования идентификатора

- **AVCM_BLOB** – DER-представление сертификата.
  Результат:
  output_size: размер буфера
  output_buffer: буфер с сертификатом

- **AVCM_CRL_DISTRIBUTION_POINTS** – список точек распространения СОС в виде строки *ASCIIZ* **"точка 1\nточка 2\nточка 3"**, где **\n** – символ перевода строки. В случае отсутствия точек распространения СОС в сертификате возвращается ошибка **AVCMR_NOT_FOUND**.
  Результат:
  output_size: размер строки
  output_buffer: строка *ASCIIZ*

- **AVCM_CERT_SHA1** – значение хэш-функции SHA1 от блоба сертификата в шестнадцатеричном представлении.
  Результат:
  output_size: размер значения хэш функции SHA1
  output_buffer: значение хэш функции SHA1

- **AVCM_POLICYINFO_COUNT** – количество пунктов политик сертификата
  Результат:
  output_size: sizeof(DWORD)
  output_buffer: DWORD – количество пунктов политик сертификата

- **AVCM_POLICYINFO_OID** – описание политики сертификата. При этом необходима передача номера пункта политики сертификата (параметр **attr_param**).
  Результат:
  output_size: размер строки
  output_buffer: строка *ASCIIZ*

- **AVCM_LDAP_PATH** – путь доступа к сертификату по протоколу LDAP, если в качестве хранилища сертификатов используется LDAP.
  Результат:
  output_size: размер строки
  output_buffer: строка *ASCIIZ*

Только для атрибутных сертификатов:

- **AVCM_ATTR_COUNT** – количество дополнительных атрибутов атрибутного сертификата
  Результат:
  output_size: sizeof(DWORD)
  output_buffer: DWORD – количество дополнительных атрибутов

- **AVCM_ATTR_OID** – идентификатор объекта (OID) атрибута. При этом необходима передача номера атрибута (параметр **attr_param**).
  Результат:
  output_size: размер строки
  output_buffer: строка *ASCIIZ*

- **AVCM_ATTR_OID_NAME** – наименование идентификатора объекта (OID) атрибута, в том случае, если данный OID зарегистрирован в операционной системе, или сам OID в противном случае. При этом необходима передача номера либо идентификатора атрибута (параметр **attr_param**).
  Результат:
  output_size: размер строки наименования идентификатора
  output_buffer: строка *ASCIIZ* наименования идентификатора или значение OID.

- **AVCM_ATTR_AS_STRING** – атрибут сертификата в виде строки *ASCIIZ*. При этом необходима передача номера либо идентификатора дополнения (параметр **attr_param**).
  Результат:
  output_size: размер строки
  output_buffer: строка *ASCIIZ*

- **AVCM_ATTR_BLOB** – значение атрибута сертификата в виде *BLOB*. При этом необходима передача номера либо идентификатора дополнения (параметр **attr_param**).
  Результат:
  output_size: размер буфера
  output_buffer: буфер со значением дополнения

- **AVCM_BASE_SERIAL_AS_STRING** – серийный номер исходного сертификата в виде строки *ASCIIZ*, на который выпущен атрибутный сертификат.
  Результат:
  output_size: размер строки

output_buffer: строка *ASCIIZ*

- **AVCM_BASE_ISSUER_AS_STRING** – имя издателя исходного сертификата в виде строки *ASCIIZ*, на который выпущен атрибутный сертификат.
  Результат:
  output_size: размер строки
  output_buffer: строка *ASCIIZ*

[in] const void * attr_param – параметр извлекаемого атрибута, применяется при извлечении дополнений сертификата. При необходимости получения дополнения по номеру необходимо передать указатель на **size_t** с номером извлекаемого дополнения. При получении дополнения по идентификатору необходимо передать указатель на *ASCIIZ* строку, содержащую *OID* дополнения. Что именно передается данным параметром указывается при помощи флагов функции.

[in,out] void * output_buffer – указатель на буфер для извлекаемого атрибута сертификата

[in,out] size_t * output_size – указатель на размер буфера извлекаемого атрибута сертификата. Если размер меньше необходимого, функция заполнит требуемый размер. В случае успешного выполнения будет заполнен размер полученного значения.

[in] DWORD flags – параметры

- **AVCMF_ATTR_BY_NUM** – поиск атрибута сертификата по номеру
- **AVCMF_ATTR_BY_OID** – поиск атрибута сертификата по его идентификатору объекта (*OID*)
- Данные флаги применяются при поиске атрибутов *Имени субъекта*, *Имени издателя* и дополнительных атрибутов сертификата. При этом либо номер, либо идентификатор должен быть передан параметром **attr_param**.
- **AVCMF_ALLOC** – функция должна самостоятельно распределить память для выходного сообщения функцией *Win32 API* **HeapAlloc**, после использования выходного сообщения, вызвавшая программа должна самостоятельно очистить выделенную память функцией **HeapFree**. См. также Распределение памяти.

Если указан данный флаг, то **output_buffer** должен быть указателем на указатель на буфер для извлекаемого атрибута подписи.

Результат

Код возврата типа **AvCmResult**
Возможные значения:

- **AVCMR_SUCCESS** – успешное выполнение операции
- **AVCMR_BAD_HANDLE** – дескриптор объекта неверен
- **AVCMR_BAD_ATTR** – идентификатор атрибута неверен
- **AVCMR_ALLOC_ERROR** – ошибка при распределении памяти
- **AVCMR_BUFFER_TOO_SMALL** – выходной буфер слишком мал
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.
- **AVCMR_NOT_FOUND** – объект(атрибут объекта) не найден.
- **AVCMR_BAD_FORMAT** – неверный формат.

Получение статуса сертификата

Функция **AvCmVerifyCertStatus** предназначена для получения объекта статуса сертификата, либо информации о доверии сертификату.

Объект статуса сертификата является дочерним объектом объекта сессии (AvCmHc).

```
AvCmResult AvCmVerifyCertStatus(
    AvCmHandle handle,
    size_t param_count,
    const AvCmCertStatParam * params,
    AvCmLong * status_ok,
    AvCmHcertstat * hstatus,
    AvCmLong flags);
```

Параметры

[in] AvCmHandle handle – дескриптор сертификата, статус которого необходимо проверить, в зависимости от способа получения:

- **AvCmHcert hcert** – дескриптор сертификата, полученного функциями **AvCmFindCertBySign** и **AvCmOpenCertEnum**.
- **AvCmHsignCert hsign_cert** – дескриптор информации о подписи и соответствующем сертификате, полученную от функций **AvCmVerifySign** и **AvCmDecryptAndVerifySign**.

[in] size_t param_count – количество параметров

[in] const AvCmCertStatParam * params – параметры. Для указания параметров необходим массив структур **AvCmCertStatParam** (см. Типы, определяемые библиотекой AvCryptMail) в котором указаны параметры проверки статуса сертификата. В элементе структуры **param_id**, указывается тип параметра, а в поле **param** указатель на значение параметра. В настоящий момент поддерживаются следующие критерии поиска:

Значения **param_id:**

- **AVCM_CHECK_MODE** – указывает режим проверки статуса.
  Значение указываемое **param указатель** на **AvCmLong:**
  - **AVCM_CM_OFFLINE** – проверка статуса с использованием локального СОС. Данный режим определён по умолчанию
  - **AVCM_CM_ONLINE** – проверка статуса сертификата с обращением OCSP серверу.

[in] AvCmLong * status_ok – статус сертификата: 0 – сертификат недействителен, подробности в возвращаемом объекте статус сертификата, любое ненулевое значение - сертификат действителен.

[in] AvCmHcertstat * hstatus – дескриптор объекта параметров статуса сертификата. Данный дескриптор возвращается только в случае, если сертификат недействителен. После использования объект статуса необходимо закрыть функцией **AvCmCloseHandle**.

[in] AvCmLong flags – параметры
В настоящий момент нет поддерживаемых флагов.

Результат

Код возврата типа **`AvCmResult`**
Возможные значения:

- **AVCMR_SUCCESS** – успешное выполнение операции
- **AVCMR_BAD_HANDLE** – дескриптор объекта неверен
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.

Получение атрибутов статуса сертификата

Функция **AvCmGetCertStatusAttr** предназначена для извлечения атрибутов статуса сертификата. Дескриптор статуса сертификата должен предварительно быть получен функцией **AvCmVerifyCertStatus**.

```
AvCmResult AvCmGetCertStatusAttr(
    AvCmHcertstat hstatus,
    DWORD attr_id,
    void * output_buffer,
    size_t * output_size,
    DWORD flags);
```

Параметры

[in] AvCmHcertstatus hstatus – дескриптор статуса сертификата, атрибут которого необходимо извлечь.

[in] DWORD attr_id – идентификатор извлекаемого атрибута:

- **AVCM_CS_UNTRUST_REASON** – причина недоверия сертификату
  Результат:
  output_size: sizeof(DWORD)
  output_buffer: DWORD
  Результат может принимать одно из следующих значений:
  AVCM_CSR_REVOKED – сертификат отозван
  AVCM_CSR_UNKNOWN – причина недоверия не может быть установлена

- **AVCM_CS_REVOCATION_TIME** – дата/время отзыва сертификата. В случае, если сертификат не отозван, возвращается ошибка **AVCMR_CERT_NOT_REVOKED**.
  output_size: sizeof(SYSTEMTIME)
  output_buffer: дата и время в формате **SYSTEMTIME.** При этом дата и время всегда возвращаются для текущего часового пояса. См. *MSDN*

- **AVCM_CS_REVOCATION_REASON** – причина отзыва сертификата. В случае, если сертификат не отозван, возвращается ошибка **AVCMR_CERT_NOT_REVOKED**. В случае, если причина отзыва сертификата не указана, в качестве причины возвращается 0 (unspecified).
  Результат:
  output_size: sizeof(DWORD)
  output_buffer: DWORD – причина отзыва сертификата.

Результат

Код возврата типа **AvCmResult**
Возможные значения:

- **AVCMR_SUCCESS** – успешное выполнение операции
- **AVCMR_BAD_HANDLE** – дескриптор объекта неверен
- **AVCMR_BAD_ATTR** – идентификатор атрибута неверен
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.
- **AVCMR_CERT_NOT_REVOKED** – сертификат не отозван

Создание контекста поиска и перебора сертификатов

Функция **AvCmOpenCertEnum** предназначена для создания контекста поиска сертификата путем перебора подмножества сертификатов, удовлетворяющих определенным условиям. В случае успешного выполнения функция вернет дескриптор перебора сертификатов **hcert_enum,** который необходимо передавать в дальнейшем функции **AvCmEnumGet** для получения по очереди найденных сертификатов. Для поиска можно задать столько атрибутов, сколько необходимо.

При этом следует отметить, что по умолчанию перебор осуществляется только среди корректных сертификатов, то есть для каждого найденного сертификата производится полная проверка его корректности. При необходимости можно получить и сертификат без проверки его корректности, для этого нужно указать флаг **AVCMF_ALL_CERT**.

По окончанию перебора сертификатов дескриптор контекста перебора необходимо закрыть функцией **AvCmCloseHandle**. В том случае если функция **AvCmOpenCertEnum** вернула ошибку, закрывать дескриптор не надо, поскольку он не был создан.

```
AvCmResult AvCmOpenCertEnum(
    AvCmHc hc,
    size_t param_count,
    const AvCmEnumGetParam * params,
    AvCmHcertEnum * hcert_enum,
    DWORD flags);
```

Параметры

[in] AvCmHc hc – дескриптор открытой сессии

[in] size_t param_count – количество критериев поиска

[in] const AvCmEnumGetParam * params – критерии поиска. Для указания критериев необходим массив структур **AvCmEnumGetParam** (см. Типы, определяемые библиотекой AvCryptMail) в котором указаны критерии поиска. В элементе структуры **param_id**, указывается тип критерия, а в поле **param** соответствующее значение критерия. В настоящий момент поддерживаются следующие критерии поиска:
Значения **param_id:**

- **AVCM_PURPOSE** – указывает предназначение искомых сертификатов
  Значение **param_spec: 0**
  Значение указываемое **param** типа **DWORD** может быть комбинацией следующих флагов**:**
  - **AVCM_P_SIGN** – поиск сертификатов, предназначенных для подписи.
  - **AVCM_P_CRYPT** – поиск сертификатов, предназначенных для зашифрования.
  - **AVCM_P_NON_REPUDIABLE –** перебор только неотзываемых сертификатов.

- **AVCM_TYPE**– указывает тип сертификатов
  Значение **param_spec: 0**
  Значение указываемое **param** типа **DWORD** может быть следующим**:**
  - **AVCM_TYPE_MY** – поиск из личных сертификатов.
  - **AVCM_TYPE_ROOT** – поиск из сертификатов доверенных УЦ.

- **AVCM_EXT_KEY_USAGE_OID** – отбор сертификатов по наличию в его списке ограничений применения ключа сертификата заданного идентификатора объекта (*OID*) в виде строки *ASCIIZ*.

Значение **param:** указатель на строку значения требуемого ограничения применения ключа сертификата в виде строки *ASCIIZ*

- **AVCM_SERIAL_AS_STRING** – поиск по серийному номеру сертификата
  Значение **param:** указатель на строку серийного номера сертификата вида *ASCIIZ*

- **AVCM_SERIAL_AS_INTEGER** – поиск по серийному номеру сертификата в виде *BLOB*. Последовательность байт серийного номера должна быть такой же, как ее возвращает функция **AvCmGetCertAttr**.
  Значение **param_spec:** размер серийного номера в байтах
  Значение **param:** указатель на серийный номер сертификата в виде большого целого числа (последовательности байт)

- **AVCM_ISSUER_AS_STRING** – поиск по полному имени (*X.509 Name*) издателя сертификата в виде строки
  Значение **param_spec: 0**
  Значение **param:** указатель на строку имени издателя вида *ASCIIZ*

- **AVCM_ISSUER_ATTR** – поиск по атрибуту имени (*X.509 Name*) издателя сертификата в виде строки. При этом необходима передача идентификатора искомого объекта (поле **param_spec**).
  Значение **param_spec:** указатель на строку идентификатора искомого объекта (*OID*)
  Значение **param:** указатель на строку атрибута имени издателя вида *ASCIIZ*

- **AVCM_NOT_BEFORE** – поиск по дате/времени начала действия сертификата
  Значение **param_spec** указывает на условие, применяемое при поиске. Возможные значения:
  - **AVCM_D_GREATER** – дата и время начала действия искомого сертификата должна быть больше или равна указанной дате и времени.
  - **AVCM_D_LESS** – дата и время начала действия искомого сертификата должна быть меньше или равна указанной дате и времени.

  Значение **param:** указатель на дату и время в формате **SYSTEMTIME**. При этом дата и время должны быть указаны для текущего часового пояса. См. *MSDN*

- **AVCM_NOT_AFTER** – поиск по дате/времени окончания действия сертификата
  Значение **param_spec** указывает на условие, применяемое при поиске. Возможные значения:
  - **AVCM_D_GREATER** – дата и время окончания действия искомого сертификата должна быть больше или равна указанной дате и времени.
  - **AVCM_D_LESS** – дата и время окончания окончания искомого сертификата должна быть меньше или равна указанной дате и времени.

  Значение **param:** указатель на дату и время в формате **SYSTEMTIME**. При этом дата и время должны быть указаны для текущего часового пояса. См. *MSDN*

- **AVCM_KEY_NOT_BEFORE** – поиск по дате/времени начала действия личного ключа
  Значение **param_spec** указывает на условие, применяемое при поиске. Возможные значения:

- **AVCM_D_GREATER** – дата и время начала действия искомого личного ключа должна быть больше или равна указанной дате и времени.
- **AVCM_D_LESS** – дата и время начала действия искомого личного ключа должна быть меньше или равна указанной дате и времени.

Значение **param:** указатель на дату и время в формате **SYSTEMTIME**. При этом дата и время должны быть указаны для текущего часового пояса. См. *MSDN*

- **AVCM_KEY_NOT_AFTER** – поиск по дате/времени окончания действия личного ключа
  Значение **param_spec** указывает на условие, применяемое при поиске. Возможные значения:
  - **AVCM_D_GREATER** – дата и время окончания действия искомого личного ключа должна быть больше или равна указанной дате и времени.
  - **AVCM_D_LESS** – дата и время окончания окончания искомого личного ключа должна быть меньше или равна указанной дате и времени.

  Значение **param:** указатель на дату и время в формате **SYSTEMTIME**. При этом дата и время должны быть указаны для текущего часового пояса. См. *MSDN*

- **AVCM_SUBJECT_AS_STRING** – поиск по полному имени (*X.509 Name*) владельца сертификата в виде строки
  Результат:
  Значение **param_spec: 0**
  Значение **param:** указатель на строку имени издателя вида *ASCIIZ*

- **AVCM_SUBJECT_ATTR** – поиск по атрибуту имени (*X.509 Name*) владельца сертификата в виде строки. При этом необходима передача идентификатора искомого объекта (поле **param_spec**).
  Значение **param_spec:** указатель на строку идентификатора искомого объекта (*OID*)
  Значение **param:** указатель на строку атрибута владельца сертификата вида *ASCIIZ*

- **AVCM_PUB_KEY_ID** – поиск по идентификатору открытого ключа сертификата в виде строки шестнадцатеричных цифр
  Значение **param_spec: 0**
  Значение **param:** указатель на строку идентификатора открытого ключа вида *ASCIIZ*

- **AVCM_PUB_KEY** – поиск по открытому ключу сертификата
  Значение **param_spec:** размер открытого ключа в байтах
  Значение **param:** указатель на открытый ключ (последовательности байт)

- **AVCM_SUBJ_ALT_NAME_ATTR** – поиск по атрибуту альтернативного имени (*X.509 AltName*) владельца сертификата в виде строки. При этом необходима передача идентификатора искомого объекта (поле **param_spec**).
  Значение **param_spec:** указатель на строку идентификатора искомого атрибута (*OID*)
  Значение **param:** указатель на строку атрибута альтернативного имени владельца сертификата вида *ASCIIZ*

- **AVCM_EXT_AS_STRING** – поиск по дополнению сертификата в виде строки *ASCIIZ*. При этом необходима передача идентификатора искомого объекта (поле **param_spec**).
  Значение **param_spec:** указатель на строку идентификатора искомого дополнения (*OID*)
  Значение **param:** указатель на строку значения искомого дополнения вида *ASCIIZ*

- **AVCM_CERT_ISSUERS_CHAIN** – поиск цепочки сертификатов издателей, начиная от сертификата, чей дескриптор передан через параметр **param**. Если в качестве **param** передан **0**, то будет возвращена цепочка сертификатов издателей личного сертификата (сертификата авторизованного пользователя). Данный параметр несовместим с другими параметрами поиска.
  Значение **param_spec: 0**
  Значение **param:** указатель на дескриптор сертификата, цепочка сертификатов издателей которого ищется
- **AVCM_ATTRIBUTE_CERTS** – поиск атрибутных сертификатов выпущенных на сертификат указанный в **param**. Если в качестве **param** передан **0**, то будут возвращены атрибутные сертификаты, выпущенные на личный сертификат (сертификата авторизованного пользователя). Данный параметр несовместим с другими параметрами поиска.
  Значение **param_spec: 0**
  Значение **param:** указатель на дескриптор сертификата, если **param** указывает на дескриптор подписанного сообщения – будут возвращены атрибутные сертификаты, помещенные в данное сообщение, если **param** указывает на дескриптор информации о подписи и соответствующем сертификате – будут возвращены атрибутные сертификаты, помещенные в данное сообщение и выпущенные на соответствующий сертификат.
- **AVCM_PKCS7_CERTS** – поиск сертификатов добавленных в сообщение. Данный параметр несовместим с другими параметрами поиска.
  Значение **param_spec: 0**
  Значение **param:** указатель на дескриптор подписанного сообщения.
- **AVCM_PKCS7_CRLS** – поиск СОС добавленных в сообщение. Данный параметр несовместим с другими параметрами поиска.
  Значение **param_spec: 0**
  Значение **param:** указатель на дескриптор подписанного сообщения.

[out] AvCmHcertEnum * hcert_enum – дескриптор перебора сертификатов, возвращаемый функцией в случае успешной инициализации перебора. По окончании перебора его необходимо освободить функцией **AvCmCloseHandle**.

[in] DWORD flags – параметры (флаги) работы.

- **AVCMF_ALL_CERT** – данный флаг необходимо использовать при необходимости перебора не только корректных сертификатов, а всех сертификатов, имеющихся в хранилище сертификатов.
- **AVCMF_REQUEST_RESULT** – поиск не только среди сертификатов, но и проверка среди отказов на сертификат. Данный флаг используется при поиске сертификата, выданного по имеющемуся запросу на сертификат. Совместно с этим флагом можно передавать только один атрибут поиска: **AVCM_PUB_KEY_ID**. В случае нахождения отказа на сертификат, будет возвращена ошибка **AVCMR_REQUEST_DENIED.**

Результат

Код возврата типа **AvCmResult**
Возможные значения:

- **AVCMR_SUCCESS** – успешное выполнение операции
- **AVCMR_BAD_HC** – дескриптор соединения неверен

- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.
- **AVCMR_BAD_PARAM** – параметр функции неверен
- **AVCMR_REQUEST_DENIED** – сертификат не найден, но найден отказ УЦ в выдаче сертификата по данному запросу

Добавление сертификатов в контекст поиска и перебора сертификатов

Функция **AvCmEnumAddCerts** предназначена для добавления сертификатов в открытый функцией **AvCmOpenCertEnum** контекст поиска сертификата, путем перебора подмножества сертификатов, удовлетворяющих определенным условиям. При успешном вызове сбрасывается указатель текущего сертификата функции **AvCmEnumGet**, и её нужно будет вызвать с флагом **AVCMF_START** для начала получения по очереди найденных сертификатов. Для поиска можно задать столько атрибутов, сколько необходимо.

При этом следует отметить, что по умолчанию перебор осуществляется только среди корректных сертификатов, то есть для каждого найденного сертификата производится полная проверка его корректности. При необходимости можно получить и сертификат без проверки его корректности, для этого нужно указать флаг **AVCMF_ALL_CERT**.

В случае, если количество сертификатов, подпадающих под условия (и сертификатов, уже отобранных в контекст) очень велико, работа данной функции может сильно замедлиться.

```
AvCmResult AvCmEnumAddCerts(
    AvCmHcertEnum hcert_enum,
    size_t param_count,
    const AvCmEnumGetParam * params,
    DWORD flags);
```

Параметры

[in] AvCmHcertEnum hcert_enum – дескриптор открытого перебора сертификатов.

[in] size_t param_count – количество критериев поиска

[in] const AvCmEnumGetParam * params – критерии поиска. Для указания критериев необходим массив структур **AvCmEnumGetParam** (см. Типы, определяемые библиотекой AvCryptMail) в котором указаны критерии поиска. В элементе структуры **param_id**, указывается тип критерия, а в поле **param** соответствующее значение критерия. В настоящий момент поддерживаются следующие критерии поиска:
Значения **param_id:**

- **AVCM_PURPOSE** – указывает предназначение искомых сертификатов
  Значение **param_spec: 0**
  Значение указываемое **param** типа **DWORD:**
  - **AVCM_P_SIGN** – поиск сертификатов, предназначенных для подписи.
  - **AVCM_P_CRYPT** – поиск сертификатов, предназначенных для зашифрования.

- **AVCM_EXT_KEY_USAGE_OID** – отбор сертификатов по наличию в его списке ограничений применения ключа сертификата заданного идентификатора объекта (*OID*) в виде строки *ASCIIZ*.
  Значение **param:** указатель на строку значения требуемого ограничения применения ключа сертификата в виде строки *ASCIIZ*

- **AVCM_SERIAL_AS_STRING** – поиск по серийному номеру сертификата
  Значение **param:** указатель на строку серийного номера сертификата вида *ASCIIZ*

- **AVCM_SERIAL_AS_INTEGER** – поиск по серийному номеру сертификата

Значение **param_spec:** размер серийного номера в байтах
Значение **param:** указатель на серийный номер сертификата в виде большого целого числа (последовательности байт)

- **AVCM_ISSUER_AS_STRING** – поиск по полному имени (*X.509 Name*) издателя сертификата в виде строки
  Значение **param_spec: 0**
  Значение **param:** указатель на строку имени издателя вида *ASCIIZ*

- **AVCM_ISSUER_ATTR** – поиск по атрибуту имени (*X.509 Name*) издателя сертификата в виде строки. При этом необходима передача идентификатора искомого объекта (поле **param_spec**).
  Значение **param_spec:** указатель на строку идентификатора искомого объекта (*OID*)
  Значение **param:** указатель на строку атрибута имени издателя вида *ASCIIZ*

- **AVCM_NOT_BEFORE** – поиск по дате/времени начала действия сертификата
  Значение **param_spec** указывает на условие, применяемое при поиске. Возможные значения:
  - **AVCM_D_GREATER** – дата и время начала действия искомого сертификата должна быть больше или равна указанной дате и времени.
  - **AVCM_D_LESS** – дата и время начала действия искомого сертификата должна быть меньше или равна указанной дате и времени.

  Значение **param:** указатель на дату и время в формате **SYSTEMTIME**. При этом дата и время должны быть указаны для текущего часового пояса. См. *MSDN*

- **AVCM_NOT_AFTER** – поиск по дате/времени окончания действия сертификата
  Значение **param_spec** указывает на условие, применяемое при поиске. Возможные значения:
  - **AVCM_D_GREATER** – дата и время окончания действия искомого сертификата должна быть больше или равна указанной дате и времени.
  - **AVCM_D_LESS** – дата и время окончания окончания искомого сертификата должна быть меньше или равна указанной дате и времени.

  Значение **param:** указатель на дату и время в формате **SYSTEMTIME**. При этом дата и время должны быть указаны для текущего часового пояса. См. *MSDN*

- **AVCM_KEY_NOT_BEFORE** – поиск по дате/времени начала действия личного ключа
  Значение **param_spec** указывает на условие, применяемое при поиске. Возможные значения:
  - **AVCM_D_GREATER** – дата и время начала действия искомого личного ключа должна быть больше или равна указанной дате и времени.
  - **AVCM_D_LESS** – дата и время начала действия искомого личного ключа должна быть меньше или равна указанной дате и времени.

  Значение **param:** указатель на дату и время в формате **SYSTEMTIME**. При этом дата и время должны быть указаны для текущего часового пояса. См. *MSDN*

- **AVCM_KEY_NOT_AFTER** – поиск по дате/времени окончания действия личного ключа
  Значение **param_spec** указывает на условие, применяемое при поиске. Возможные значения:
  - **AVCM_D_GREATER** – дата и время окончания действия искомого личного ключа должна быть больше или равна указанной дате и времени.

- **AVCM_D_LESS** – дата и время окончания окончания искомого личного ключа должна быть меньше или равна указанной дате и времени.

Значение **param:** указатель на дату и время в формате **SYSTEMTIME**. При этом дата и время должны быть указаны для текущего часового пояса. См. *MSDN*

- **AVCM_SUBJECT_AS_STRING** – поиск по полному имени (*X.509 Name*) владельца сертификата в виде строки
  Результат:
  Значение **param_spec: 0**
  Значение **param:** указатель на строку имени издателя вида *ASCIIZ*

- **AVCM_SUBJECT_ATTR** – поиск по атрибуту имени (*X.509 Name*) владельца сертификата в виде строки. При этом необходима передача идентификатора искомого объекта (поле **param_spec**).
  Значение **param_spec:** указатель на строку идентификатора искомого объекта (*OID*)
  Значение **param:** указатель на строку атрибута владельца сертификата вида *ASCIIZ*

- **AVCM_PUB_KEY_ID** – поиск по идентификатору открытого ключа сертификата в виде строки шестнадцатеричных цифр
  Значение **param_spec: 0**
  Значение **param:** указатель на строку идентификатора открытого ключа вида *ASCIIZ*

- **AVCM_PUB_KEY** – поиск по открытому ключу сертификата
  Значение **param_spec:** размер открытого ключа в байтах
  Значение **param:** указатель на открытый ключ (последовательности байт)

- **AVCM_SUBJ_ALT_NAME_ATTR** – поиск по атрибуту альтернативного имени (*X.509 AltName*) владельца сертификата в виде строки. При этом необходима передача идентификатора искомого объекта (поле **param_spec**).
  Значение **param_spec:** указатель на строку идентификатора искомого атрибута (*OID*)
  Значение **param:** указатель на строку атрибута альтернативного имени владельца сертификата вида *ASCIIZ*

- **AVCM_EXT_AS_STRING** – поиск по дополнению сертификата в виде строки *ASCIIZ*. При этом необходима передача идентификатора искомого объекта (поле **param_spec**).
  Значение **param_spec:** указатель на строку идентификатора искомого дополнения (*OID*)
  Значение **param:** указатель на строку значения искомого дополнения вида *ASCIIZ*

[in] DWORD flags – параметры (флаги) работы.

- **AVCMF_ALL_CERT** – данный флаг необходимо использовать при необходимости перебора не только корректных сертификатов, а всех сертификатов, имеющихся в хранилище сертификатов.

Результат

Код возврата типа **AvCmResult**
Возможные значения:

- **AVCMR_SUCCESS** – успешное выполнение операции

- **AVCMR_BAD_HENUM** – дескриптор перебора неверен
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.
- **AVCMR_BAD_PARAM** – параметр функции неверен

Диалог создания контекста поиска и перебора сертификатов

Функция **AvCmEnumDlg** предназначена для создания контекста поиска сертификатов путём предложения пользователю выбрать нужные ему сертификаты из списка.

```
AvCmResult AvCmEnumDlg(
    AvCmHc hc,
    const char* cpszDlgCaption,
    const char* cpszLabel,
    const char* cpszOkButtonCaption,
    AvCmHcertEnum * hcert_enum,
    DWORD flags);
```

Параметры

[in] AvCmHc hc – дескриптор открытой сессии.

[in] const char* cpszDlgCaption – заголовок окна выбора сертификатов.

[in] const char* cpszLabel – текст над списком сертификатов.

[in] const char* cpszOkButtonCaption – надпись на кнопке завершения диалога.

[out] AvCmHcertEnum * hcert_enum – дескриптор перебора сертификатов, возвращаемый функцией в случае успешной инициализации перебора. По окончании перебора его необходимо освободить функцией **AvCmCloseHandle**.

[in] DWORD flags – параметры (флаги) работы.

- **AVCMF_ONLY_ENCR_CERTS** – данный флаг необходимо использовать при необходимости показа только сертификатов, предназначенных для шифрования.
- **AVCMF_ALL_CERT** – данный флаг необходимо использовать при необходимости показа не только корректных сертификатов, а всех сертификатов, имеющихся в хранилище сертификатов.

Результат

Код возврата типа **AvCmResult**

Возможные значения:

- **AVCMR_SUCCESS** – успешное выполнение операции
- **AVCMR_BAD_HC** – дескриптор соединения неверен
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.
- **AVCMR_BAD_PARAM** – параметр функции неверен

Перебор сертификатов и информации о подписях

Функция **AvCmEnumGet** предназначена для перебора объектов в открытом перечислении.

Функцию необходимо вызывать пока не будут найдены требуемые объекты или не будет возвращена ошибка **AVCMR_NOT_FOUND**, означающая что больше нет объектов в данном перечислении.

Данная функция может быть использована как для поиска сертификатов в хранилище сертификатов, так и для перебора информации о подписях и сертификатах, полученных от функций **AvCmVerifySign** и **AvCmDecryptAndVerifySign**.

```
AvCmResult AvCmEnumGet(
    AvCmHenum henum,
    AvCmHandle * handle,
    DWORD flags);
```

Параметры

[in] AvCmHenum henum – либо дескриптор перебора сертификатов **AvCmHcertEnum**, полученный от функции **AvCmOpenCertEnum**, либо дескриптор перебора информации о подписях и сертификатах **AvCmHsgnEnum**.

[out] AvCmHandle * handle – дескриптор найденного объекта, в зависимости от переданного дескриптора перечисления:

- **AvCmHcert * hcert** – дескриптор найденного сертификата (атрибутного сертификата).
- **AvCmHsignCert * hsign_cert** – дескриптор информации о подписи и соответствующем сертификате.

После использования объект необходимо закрыть функцией **AvCmCloseHandle**.

[in] DWORD flags – параметры (флаги) работы.

- **AVCMF_NEXT** – получить следующий по порядку элемент перечисления
- **AVCMF_START** – начать перебор элементов с начала

Результат

Код возврата типа **AvCmResult**

Возможные значения:

- **AVCMR_SUCCESS** – успешное выполнение операции
- **AVCMR_BAD_HENUM** – дескриптор перебора неверен
- **AVCMR_NOT_FOUND** – объект не найден
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.

Помещение сертификата из сообщения в справочник сертификатов

Функция **AvCmPutCert** предназначена для помещения сертификата, извлеченного из сообщения в справочник сертификатов.

```
AvCmResult AvCmPutCert(
    AvCmHandle handle,
    DWORD flags);
```

Параметры

[in] AvCmHandle handle – дескриптор сертификата (**AvCmHcert**).

[in] AvCmLong flags – параметры
В настоящий момент нет поддерживаемых флагов.

Результат

Код возврата типа **AvCmResult**
Возможные значения:

- **AVCMR_SUCCESS** – успешное выполнение операции
- **AVCMR_BAD_HCERT** – дескриптор сертификата неверен
- **AVCMR_ALREADY_EXISTS** – сертификат уже внесен в справочник сертификатов
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.

Помещение сертификата/СОС в справочник сертификатов

Функция **AvCmImport** предназначена для импорта поддерживаемых объектов, таких как: сертификат, список отозванных сертификатов или запрос на сертификат, в соответствующие справочники сертификатов/СОС.

```
AvCmResult AvCmImport(
    AvCmHc hc,
    DWORD obj_type,
    const void * input_data,
    size_t input_size,
    size_t param_count,
    const AvCmImportParam * params,
    DWORD flags);
```

Параметры

[in] AvCmHc hc – дескриптор открытой сессии

[in] DWORD obj_type – тип импортируемого объекта;

- **AVCM_CERTIFICATE** – *DER*-представление сертификата;
- **AVCM_CRL** – *DER*-представление списка отозванных сертификатов;
- **AVCM_PKCS10_REQUEST** – *DER*-представление запроса на сертификат в формате *PKCS#10*;
- **AVCM_PKCS7_REQUEST** – *DER*-представление подписанной заявки на сертификат в формате *PKCS#7 SignedData*;
- **AVCM_ANY_FILE** – импорт сертификатов и/или СОС из файла, при этом будет отображён стандартный мастер импорта сертификатов/СОС из файла с выбором файла. Параметры **input_data** и **input_size** игнорируются.

[in] const void * input_data – указатель на область данных, содержащую импортируемые данные;

[in] size_t input_size – размер импортируемых данных в байтах;

[in] size_t param_count – количество параметров импорта;

[in] const AvCmImportParam * params – массив структур, описывающих параметры импорта.

Возможные значения:

- Значение **param_id: AVCM_RESULT_HANDLE** – открыть импортированный объект (сертификат, СОС, запрос на сертификат) и вернуть дескриптор импортированного и открытого объекта. После использования дескриптор должен быть освобожден функцией **AvCmCloseHandle**.
  Значение **param:** указатель на **AvCmHandle** дескриптор импортированного объекта.
- Значение **param_id: AVCM_CRL_ISSUER_SUBJECT** – при импорте списка отозванных сертификатов этим параметром можно передать поле **Subject** издателя СОС для того чтобы убедиться что мы импортируем СОС именно необходимого нам издателя.

Значение **`param:`** указатель на строку **Subject** издателя СОС вида *ASCIIZ*

- **AVCM_IMPORT_PATH** – применяется совместно с типом объекта **AVCM_ANY_FILE,** для того чтобы указать каталог по умолчанию для выбора импортируемого файла.

Значение **`param:`** указатель на строку пути к каталогу импорта по умолчанию формата *ASCIIZ*.

- **AVCM_OPERATION_REPORT** – указывает на необходимость создания информационного файла с описанием выполненных операций. Описание формата приведено в приложении Приложение 1.

Значение **`param:`** указатель на строку имени файла для сохранения отчета об операции формата *ASCIIZ*.

`[in] DWORD flags` – режимы импорта:

- **AVCMF_RETURN_HANDLE_IF_EXISTS** – при указании данного флага функция вернет дескриптор открытого объекта (через атрибут **AVCM_RESULT_HANDLE**) не только в случае успешного импорта, но и в случае если данный объект уже имеется в справочниках, то есть при ошибке **AVCMR_ALREADY_EXISTS.**
- **AVCMF_SELECT_MY_CERT** – после импорта сертификатов обязательно вывести диалог выбора контейнера с личным ключом. Используется только при импорте сертификатов **AVCM_ANY_FILE**.

Результат

Код возврата типа **`AvCmResult`**

Возможные значения:

- **AVCMR_SUCCESS** – успешное выполнение операции
- **AVCMR_BAD_HC** – дескриптор соединения неверен
- **AVCMR_NO_INPUT** – входное сообщение имеет нулевой размер
- **AVCMR_BAD_FORMAT –** неверный формат входных данных
- **AVCMR_ALLOC_ERROR –** ошибка при распределении памяти
- **AVCMR_CERT_NOT_FOUND** – сертификат издателя СОС не найден
- **AVCMR_CERT_SIGN_INVALID** – подпись под сертификатом неверна
- **AVCMR_CERT_EXPIRED** – срок действия личного ключа сертификата неверен
- **AVCMR_CRL_INVALID** – подпись под СОС издателя неверна или СОС устарел
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.
- **AVCMR_USER_NO_AUTH** – сессия создана без авторизации пользователя.
- **AVCMR_BAD_CRL_ISSUER** – издатель СОС отличается от ожидаемого издателя, переданного параметром **AVCM_CRL_ISSUER_SUBJECT**.
- **AVCMR_OLD_CRL** – в справочнике списков отозванных сертификатов имеется более новый СОС данного издателя.
- **AVCMR_ALREADY_EXISTS** – сертификат уже внесен в справочник сертификатов
- **AVCMR_UNSUPPORTED_CONNECTION_TYPE** – импорт в данный тип хранилища не поддерживается.

Функции работы со списками отозванных сертификатов

Поиск списка отозванных сертификатов

Функция **AvCmFindCrl** предназначена для поиска и открытия списка отозванных сертификатов в справочниках сертификатов/СОС. Открытый список отозванных сертификатов должен быть закрыт функцией **AvCmCloseHandle** по окончанию использования.

```
AvCmResult AvCmFindCrl(
    AvCmHc hc,
    size_t param_count,
    const AvCmFindCrlParam * params,
    AvCmHcrl * hcrl,
    DWORD flags);
```

Параметры

[in] AvCmHc hc – дескриптор открытой сессии

[in] size_t param_count – количество критериев отбора сертификатов. В настоящий момент поддерживается отбор только по одному критерию.

[in] const AvCmFindCrlParam * params – критерии поиска. Для указания критериев необходим массив структур **AvCmFindCrlParam** (см. Типы, определяемые библиотекой AvCryptMail) в котором указаны критерии поиска СОС. В элементе структуры **param_id**, указывается тип критерия, а в поле **param** соответствующее значение критерия. В настоящий момент поддерживаются следующие критерии поиска:
Значения **param_id:**

- **AVCM_CRL_ISSUER_SUBJECT** – поиск по полному имени (*X.509 Name*) издателя списка отозванных сертификатов в виде строки
  Значение **param:** указатель на строку имени издателя вида *ASCIIZ*

- **AVCM_CRL_ISSUER_CERT** – поиск по открытому сертификату издателя списка отозванных сертификатов в виде строки
  Значение **param:** указатель на дескриптор сертификата типа **AvCmHcert**

[out] AvCmHcrl * hcrl – дескриптор найденного списка отозванных сертификатов.

[in] DWORD flags – режимы поиска.
В настоящий момент нет поддерживаемых флагов.

Результат

Код возврата типа **AvCmResult**
Возможные значения:

- **AVCMR_SUCCESS** – успешное выполнение операции
- **AVCMR_BAD_HC** – дескриптор соединения неверен
- **AVCMR_ALLOC_ERROR –** ошибка при распределении памяти
- **AVCMR_CRL_NOT_FOUND** – СОС не найден

- **AVCMR_CRL_INVALID** – подпись под СОС издателя неверна или СОС устарел
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.

Получение атрибутов открытого списка отозванных сертификатов

Функция **AvCmGetCrlAttr** предназначена для извлечения атрибутов списка отозванных сертификатов.

```
AvCmResult AvCmGetCrlAttr(
    const AvCmHcrl hcrl,
    DWORD attr_id,
    void * output_buffer,
    size_t * output_size,
    DWORD flags);
```

Параметры

[in] AvCmHcrl hcrl – дескриптор открытого списка отозванных сертификатов, атрибут которого необходимо извлечь.

[in] DWORD attr_id – идентификатор извлекаемого атрибута:

- **AVCM_VERSION** – версия списка отозванных сертификатов
  Результат:
  output_size: sizeof(DWORD)
  output_buffer: DWORD – версия сертификата

- **AVCM_ISSUER_AS_STRING** – имя (*X.509 Name*) издателя списка отозванных сертификатов в виде строки, в том случае, если это поле COC можно представить в виде строки. Если атрибут невозможно представить в виде строки *ASCIIZ*, будет возвращена ошибка **AVCMR_BAD_FORMAT**.
  Результат:
  output_size: размер строки
  output_buffer: строка *ASCIIZ*

- **AVCM_THIS_UPDATE** – дата/время издания списка отозванных сертификатов
  output_size: sizeof(SYSTEMTIME)
  output_buffer: дата и время в формате **SYSTEMTIME.** При этом дата и время всегда возвращаются для текущего часового пояса. См. *MSDN*

- **AVCM_NEXT_UPDATE** – дата/время окончания действия списка отозванных сертификатов
  output_size: sizeof(SYSTEMTIME)
  output_buffer: дата и время в формате **SYSTEMTIME.** При этом дата и время всегда возвращаются для текущего часового пояса. См. *MSDN*

- **AVCM_PUB_KEY_ID** – идентификатор открытого ключа сертификата издателя COC
  Результат:
  output_size: размер идентификатора открытого ключа
  output_buffer: строка *ASCIIZ* идентификатора

- **AVCM_SIGN_ALG_OID** – идентификатор алгоритма ЭЦП
  Результат:
  output_size: размер строки идентификатора

output_buffer: строка *ASCIIZ* идентификатора

- **AVCM_BLOB** – DER-представление списка отозванных сертификатов.
  Результат:
  output_size: размер буфера
  output_buffer: буфер с COC

- **AVCM_SHA1_HASH** – SHA-1 хэш от DER-представление списка отозванных сертификатов.
  Результат:
  output_size: размер буфера, не меньше 20 байт
  output_buffer: буфер с значением вычисленной хэш функции SHA-1. Всегда занимает 20 байт.

[in,out] void * output_buffer – указатель на буфер для извлекаемого атрибута сертификата

[in,out] size_t * output_size – указатель на размер буфера извлекаемого атрибута сертификата. Если размер меньше необходимого, функция заполнит требуемый размер. В случае успешного выполнения будет заполнен размер полученного значения.

[in] DWORD flags – параметры

- **AVCMF_ALLOC** – функция должна самостоятельно распределить память для выходного сообщения функцией *Win32 API* **HeapAlloc**, после использования выходного сообщения, вызвавшая программа должна самостоятельно очистить выделенную память функцией **HeapFree**. См. также Распределение памяти**.**

Если указан данный флаг, то **output_buffer** должен быть указателем на указатель на буфер для извлекаемого атрибута подписи.

Результат

Код возврата типа **AvCmResult**
Возможные значения:

- **AVCMR_SUCCESS** – успешное выполнение операции
- **AVCMR_BAD_HCRL** – дескриптор COC неверен
- **AVCMR_ALLOC_ERROR –** ошибка при распределении памяти
- **AVCMR_BAD_ATTR** – идентификатор атрибута неверен
- **AVCMR_BAD_FORMAT** – неверный формат.
- **AVCMR_BUFFER_TOO_SMALL** – выходной буфер слишком мал
- **AVCMR_NOT_FOUND** – объект(атрибут объекта) не найден.
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.

Функции работы с запросами на сертификат

Генерация запроса на сертификат

Функция **AvCmGenerateRequest** предназначена для генерации запроса на сертификат. При вызове данной функции библиотека отобразит мастер создания запроса на сертификат. Для генерации запроса по умолчанию будут использованы шаблоны сертификатов, находящиеся в каталоге, в котором установлена библиотека. В случае успешного выполнения функция вернет дескриптор созданного и помещенного в справочник запросов, запроса на сертификат. Открытый запрос должен быть закрыт функцией **AvCmCloseHandle** по окончанию использования.

```
AvCmResult AvCmGenerateRequest(
    AvCmHc hc,
    size_t param_count,
    const AvCmGenReqParam * params,
    AvCmHreq * hreq,
    DWORD flags);
```

Параметры

[in] AvCmHc hc – дескриптор открытой сессии

[in] size_t param_count – количество передаваемых параметров генерации запроса.

[in] const AvCmGenReqParam * params – параметров генерации запроса. Для указания параметров необходим массив структур **AvCmGenReqParam** (см. Типы, определяемые библиотекой AvCryptMail) в котором указаны дополнительные параметры генерации запроса на сертификат.

Значения **param_id:**

- **AVCM_TEMPLATE** – путь к файлу шаблона на сертификат. Если указан путь к шаблону, то мастер не будет предлагать пользователю выбор шаблона на сертификат.
  Значение **param:** указатель на строку пути к файлу шаблона на сертификат открытого ключа формата *ASCIIZ*.
- **AVCM_CERT_PROLONGATION** – сгенерировать запрос в продолжение переданного сертификата.
  Значение **param:** дескриптор сертификата типа **AvCmHcert**.
- **AVCM_EXPORT_PATH** – применяется совместно с флагом **AVCMF_ALLOW_TO_SELECT_FILE,** для того чтобы указать каталог по умолчанию для сохранения файла созданного запроса.
  Значение **param:** указатель на строку пути к каталогу экспорта по умолчанию формата *ASCIIZ*.
- **AVCM_OPERATION_REPORT** – указывает на необходимость создания информационного файла с описанием выполненных операций. Описание формата приведено в приложении Приложение 1.
  Значение **param:** указатель на строку имени файла для сохранения отчета об операции формата *ASCIIZ*.

[out] AvCmHreq * hreq – дескриптор созданного запроса на сертификат.

`[in] DWORD flags` – режимы генерации запроса на сертификат:

AVCMF_ALLOW_TO_SELECT_FILE – после генерации запроса предложить пользователю сохранить запрос в файл.

Результат

Код возврата типа **`AvCmResult`**

Возможные значения:

- **AVCMR_SUCCESS** – успешное выполнение операции
- **AVCMR_BAD_HC** – дескриптор соединения неверен
- **AVCMR_USER_CANCEL** – создание запроса отменено пользователем
- **AVCMR_ALLOC_ERROR –** ошибка при распределении памяти
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.

Получение атрибутов запроса на сертификат

Функция **AvCmGetRequestAttr** предназначена для извлечения атрибутов запроса на сертификат.

```
AvCmResult AvCmGetRequestAttr(
    AvCmHreq hreq,
    DWORD attr_id,
    void * output_buffer,
    size_t * output_size,
    DWORD flags);
```

Параметры

[in] const AvCmHreq * hreq – дескриптор открытого запроса на сертификат

[in] DWORD attr_id – идентификатор извлекаемого атрибута:

- **AVCM_PUB_KEY_ID** – идентификатор открытого ключа запроса на сертификат
  Результат:
  output_size: размер идентификатора открытого ключа
  output_buffer: строка *ASCIIZ* идентификатора

- **AVCM_BLOB** – DER-представление запроса на сертификат.
  Результат:
  output_size: размер буфера
  output_buffer: буфер с запросом на сертификат

- **AVCM_MSCA_COMPATIBLE** – получение режима генерации запроса. В этом случае в качестве указателя на выходной буфер должен быть передан указатель на DWORD. Результатом будет ненулевое значение в том случае, если библиотека сгенерировала запрос для Microsoft Certificate Authority.
  output_size: размер DWORD
  output_buffer: указатель на DWORD

[in,out] void * output_buffer – указатель на буфер для извлекаемого атрибута запроса на сертификат

[in,out] size_t * output_size – указатель на размер буфера извлекаемого атрибута запроса на сертификат. Если размер меньше необходимого, функция заполнит требуемый размер. В случае успешного выполнения будет заполнен размер полученного значения.

[in] DWORD flags – параметры

- **AVCMF_ALLOC** – функция должна самостоятельно распределить память для выходного сообщения функцией *Win32 API* **HeapAlloc**, после использования выходного сообщения, вызвавшая программа должна самостоятельно очистить выделенную память функцией **HeapFree**. См. также Распределение памяти.

Если указан данный флаг, то **output_buffer** должен быть указателем на указатель на буфер для извлекаемого атрибута подписи.

Результат

Код возврата типа **`AvCmResult`**

Возможные значения:

- **AVCMR_SUCCESS** – успешное выполнение операции
- **AVCMR_BAD_HC** – дескриптор соединения неверен
- **AVCMR_ALLOC_ERROR –** ошибка при распределении памяти
- **AVCMR_BAD_ATTR** – идентификатор атрибута неверен
- **AVCMR_BAD_FORMAT** – неверный формат.
- **AVCMR_BUFFER_TOO_SMALL** – выходной буфер слишком мал
- **AVCMR_NOT_FOUND** – объект(атрибут объекта) не найден.
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.

Поиск запроса на сертификат

Функция **AvCmFindRequest** предназначена для поиска и открытия запроса на сертификат.

```
AvCmResult AvCmFindRequest(
    AvCmHc hc,
    size_t param_count,
    const AvCmFindReqParam * params,
    const AvCmHreq * hreq,
    DWORD flags);
```

Параметры

[in] AvCmHc hc – дескриптор открытой сессии

[in] size_t param_count – количество критериев поиска на сертификат. В настоящий момент поддерживается отбор только по одному критерию.

[in] const AvCmFindReqParam * params – критерии поиска. Для указания критериев необходим массив структур **AvCmFindReqParam** (см. Типы, определяемые библиотекой AvCryptMail) в котором указаны критерии поиска СОС. В элементе структуры **param_id**, указывается тип критерия, а в поле **param** соответствующее значение критерия. В настоящий момент поддерживаются следующие критерии поиска:
Значения **param_id:**

- **AVCM_PUB_KEY_ID** – поиск по идентификатору открытого ключа запроса на сертификат
  Значение **param:** указатель на строку идентификатора открытого ключа вида *ASCIIZ*

[out] AvCmHreq * hreq – дескриптор найденного списка отозванных сертификатов.

[in] DWORD flags – режимы поиска.
В настоящий момент нет поддерживаемых флагов.

Результат

Код возврата типа **AvCmResult**
Возможные значения:

- **AVCMR_SUCCESS** – успешное выполнение операции
- **AVCMR_BAD_HC** – дескриптор соединения неверен
- **AVCMR_ALLOC_ERROR –** ошибка при распределении памяти
- **AVCMR_BAD_ATTR** – идентификатор атрибута неверен
- **AVCMR_BAD_FORMAT** – неверный формат.
- **AVCMR_NOT_FOUND** – объект(атрибут объекта) не найден.
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.

Низкоуровневые криптографические функции

Выработка ЭЦП для блока данных

Функция **AvCmSignRawData** предназначена для выработки ЭЦП заданным алгоритмом для переданного блока данных на ключе текущего пользователя (сессии). Функция вернёт выработанную ЭЦП в виде закодированного *ASN.1 BitString*, пригодном для включения в другие структуры *ASN.1*, такие как сертификат *X509* и т. д.

```
AvCmResult AvCmSignRawData(
    AvCmHc hc,
    const char* hash_and_sign_oid,
    const void* data,
    size_t data_size,
    void * output_buffer,
    size_t * output_size,
    DWORD flags);
```

Параметры

[in] AvCmHc hc – дескриптор открытой сессии

[in] const char* hash_and_sign_oid – OID алгоритмов хэширования и выработки ЭЦП при помощи которых необходимо подписать переданный блок данных. Если текущий криптопровайдер не поддерживает данные алгоритмы, функция вернёт ошибку **AVCMR_ALG_NOT_SUPPORTED**. Если передан нулевой указатель, то OID алгоритмов хэширования и выработки ЭЦП будет считан из конфигурационного файла библиотеки.

[in] const void* message_data – указатель на блок подписываемых данных

[in] size_t message_size – размер подписываемых данных

[in,out] void * output_buffer –указатель на буфер для помещения выработанной подписи. Если передан 0 в качестве указателя, функция заполнит требуемый размер.

[in,out] size_t * output_size – указатель на размер буфера подписи. Если размер меньше необходимого, функция заполнит требуемый размер. В случае успешного выполнения будет заполнен размер полученной подписи.

[in] DWORD flags – параметры (флаги) работы.

- **AVCMF_RAW_SIGN** – вернуть подпись без *ASN.1* кодирования;
- **AVCMF_ALLOC** – функция должна самостоятельно распределить память для выходного сообщения функцией *Win32 API* **HeapAlloc**, после использования выходного сообщения, вызвавшая программа должна самостоятельно очистить выделенную память функцией **HeapFree**. См. также Распределение памяти.

Если указан данный флаг, то **output_buffer** должен быть указателем на указатель на буфер для извлекаемого атрибута подписи.

Результат

Код возврата типа **AvCmResult**

Возможные значения:

- **AVCMR_SUCCESS** – успешное выполнение операции
- **AVCMR_BAD_HC** – дескриптор соединения неверен
- **AVCMR_ALLOC_ERROR –** ошибка при распределении памяти
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.
- **AVCMR_ALG_NOT_SUPPORTED** – алгоритм не поддерживается
- **AVCMR_BUFFER_TOO_SMALL** – выходной буфер слишком мал

Проверка ЭЦП для блока данных

Функция **AvCmVerifyRawDataSign** предназначена для проверки ЭЦП заданным алгоритмом для переданных данных на открытом ключе переданного дескриптором сертификата. Значение ЭЦП передаётся в виде закодированного *ASN.1 BitString.*

```
AvCmResult AvCmVerifyRawDataSign(
    AvCmHcert hcert,
    const char* hash_and_sign_oid,
    const void* data_to_be_verified,
    size_t data_size,
    const void * sign_value,
    size_t sign_size,
    DWORD flags);
```

Параметры

[in] AvCmHcert hcert – дескриптор сертификата подписавшего

[in] const char* hash_and_sign_oid – OID алгоритмов хэширования и выработки ЭЦП при помощи которых необходимо проверить подпись переданных данных. Если текущий криптопровайдер не поддерживает данные алгоритмы, функция вернёт ошибку **AVCMR_ALG_NOT_SUPPORTED**. Если передан нулевой указатель, то OID алгоритмов хэширования и выработки ЭЦП будет считан из конфигурационного файла библиотеки.

[in] const void* data_to_be_verified – указатель на проверяемые данные

[in] size_t message_size – размер проверяемых данных

[in] const void * sign_value –указатель на значение подписи. Значение ЭЦП передаётся в виде закодированного *ASN.1 BitString*.

[out] size_t sign_size – размер подписи.

[in] DWORD flags – режимы проверки подписи сообщения

- **AVCMF_RAW_SIGN** – подпись без *ASN.1* кодирования;
- **AVCMF_NO_CERT_VERIFY** – не проверять доверие к сертификату подписавшего. При указании этого флага не будет производится проверка цепочки сертификатов до сертификата корневого ЦС. Также не будет производится поиск сертификата в СОС центра сертификации.

Внимание!
Целостность справочников сертификатов при проверке ЭЦП с указанным флагом должна контролироваться иными средствами.

- **AVCMF_NO_CRL_VERIFY** – не проверять отсутствие сертификата в СОС удостоверяющего центра.

Внимание!
Целостность справочников сертификатов при проверке ЭЦП с указанным флагом должна контролироваться иными средствами.

Результат

Код возврата типа **`AvCmResult`**

Возможные значения:

- **AVCMR_SUCCESS** – успешное выполнение операции
- **AVCMR_BAD_HCERT** – дескриптор сертификата неверен
- **AVCMR_ALLOC_ERROR –** ошибка при распределении памяти
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.
- **AVCMR_ALG_NOT_SUPPORTED** – алгоритм не поддерживается
- **AVCMR_BUFFER_TOO_SMALL** – выходной буфер слишком мал
- **AVCMR_CERT_NOT_FOR_SIGN** – сертификат не предназначен для подписи
- **AVCMR_SIGN_INVALID –** подпись неверна

Сервисные функции

Показ окна информации об объекте библиотеки

Функция **AvCmShowObjectInfo** предназначена для показа окна информации об объекте библиотеки.

```
AvCmResult AvCmShowObjectInfo(
    AvCmHandle handle,
    HWnd hwnd,
    const char* cpszDlgCaption,
    const char* cpszLabel,
    const char* cpszOkButtonCaption,
    DWORD flags);
```

Параметры

[in] AvCmHc handle – дескриптор объекта, который нужно показать. В настоящий момент поддерживаются следующие объекты: сертификат (AvCmHcert), подпись (AvCmHsign) и набор сертификатов (AvCmHcertEnum).

[in] HWnd hwnd – дескриптор окна, дочерним к которому должен быть диалог (или 0).

[in] const char* cpszDlgCaption – заголовок окна информации (если 0 – используется стандартный заголовок).

[in] const char* cpszLabel – текст над списком сертификатов (если для данного объекта нужен).

[in] const char* cpszOkButtonCaption – надпись на кнопке завершения диалога (если 0 – используется стандартная надпись).

[in] DWORD flags – параметры (флаги) работы.

Результат

Код возврата типа **AvCmResult**
Возможные значения:

- **AVCMR_SUCCESS** – успешное выполнение операции
- **AVCMR_BAD_HANDLE** – дескриптор неверен
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.
- **AVCMR_BAD_PARAM** – параметр функции неверен

Функция получения информации об ошибке

Функция **AvCmGetErrorInfo** возвращает различные параметры возникшей в процессе работы одной из функций библиотеки ошибки, например, текстовое описание ошибки. Приложение обязано проверять возникновение ошибки по коду возврата каждой из функций библиотеки. При этом любой код возврата, отличный от 0, является признаком ошибки. Код ошибки может быть использован для получения дополнительной информации о возникшей ошибке. При этом каждая сессия хранит подробную информацию о последней возникшей ошибке. Приложение может получить расширенную информацию, передав при вызове дескриптор сессии библиотеки, в которой ошибка произошла. Если дескриптор сессии недоступен в функции обработки ошибки, можно, указав флаг **AVCMF_THREAD_ERROR**, выполнить поиск ошибки по идентификатору потока (*threadId*), вызвавшему функцию **AvCmGetErrorInfo**. Если дескриптор сессии не передан, или детальная информация об этой ошибке уже недоступна (ошибка не была последней возникшей ошибкой сессии или сессия недоступна), будет возвращено только общее описание ошибки.

Размещение памяти для сообщения может быть произведено самой функцией при помощи функции *Win32 API* **HeapAlloc.**

```
AvCmResult AvCmGetErrorInfo(
    AvCmHc hc,
    AvCmResult error_code,
    DWORD param_id,
    void * output_buffer,
    size_t * output_size,
    DWORD flags);
```

Параметры

[in] AvCmHc hc – дескриптор сессии, в которой произошла ошибка или 0, если сессия недоступна

[in] AvCmResult error_code – код ошибки, описание которой необходимо получить

[in] DWORD param_id – идентификатор параметра, который требуется получить. В настоящий момент поддерживаются следующие параметры:

- **AVCM_SHORT_STRING** – краткое описание ошибки
- **AVCM_DESCRIPTION** – подробное описание ошибки
- **AVCM_ERROR_CODE** – код ошибки библиотеки. В этом случае в качестве указателя на выходной буфер должен быть передан указатель на **DWORD.**

[in,out] void * output_buffer – указатель на буфер выходного сообщения. Если передан 0 в качестве указателя, функция заполнит требуемый размер.

[in,out] size_t * output_size – указатель на размер буфера выходного сообщения. Если размер меньше необходимого, функция заполнит требуемый размер. В случае успешного выполнения будет заполнен размер получившегося сообщения.

[in] DWORD flags – режимы генерации подписанного сообщения:

- **AVCMF_THREAD_ERROR** – получить ошибку сессии по идентификатору вызывающего потока. В этом случае, параметр **hc** игнорируется.
- **AVCMF_ALLOC** – функция должна самостоятельно распределить память для выходного сообщения функцией *Win32 API* **HeapAlloc**, после использования выходного

сообщения, вызвавшая программа должна самостоятельно очистить выделенную память функцией **HeapFree**. См. также Распределение памяти.

Если указан данный флаг, то **output_buffer** должен быть указателем на указатель на буфер для извлекаемого атрибута подписи.

Результат

Код возврата типа **`AvCmResult`**

Возможные значения:

- **AVCMR_SUCCESS** (0) – успешное выполнение операции
- **AVCMR_BUFFER_TOO_SMALL** – выходной буфер слишком мал
- **AVCMR_ALLOC_ERROR –** ошибка при распределении памяти
- **AVCMR_INTERNAL_ERROR** – внутренняя ошибка библиотеки.
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.
- **AVCMR_BAD_PARAM** – параметр функции неверен

- Ошибка *Win32 API.*

Вывод информации в отладочный лог

Функция **AvCmDebugLog** предназначена для показа окна информации об объекте библиотеки.

```
AvCmResult AvCmDebugLog(
    const ansichar* cpszDlgCaption,
    const ansichar* cpszLabel,
    const ansichar* cpszOkButtonCaption,
    DWORD flags);
```

Параметры

[in] AvCmHc handle – дескриптор объекта, который нужно показать. В настоящий момент поддерживаются следующие объекты: сертификат (AvCmHcert), подпись (AvCmHsign) и набор сертификатов (AvCmHcertEnum).

[in] HWnd hwnd – дескриптор окна, дочерним к которому должен быть диалог (или 0).

[in] const char* cpszDlgCaption – заголовок окна информации (если 0 – используется стандартный заголовок).

[in] const char* cpszLabel – текст над списком сертификатов (если для данного объекта нужен).

[in] const char* cpszOkButtonCaption – надпись на кнопке завершения диалога (если 0 – используется стандартная надпись).

[in] DWORD flags – параметры (флаги) работы.

Результат

Код возврата типа **AvCmResult**
Возможные значения:

- **AVCMR_SUCCESS** – успешное выполнение операции
- **AVCMR_BAD_HANDLE** – дескриптор неверен
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.
- **AVCMR_BAD_PARAM** – параметр функции неверен

Функции обработки ответов OCSPRespondera

Получение информации о статусе сертификата

Функция **AvCmMsgOCSPGetResponse** предназначена для получения дескриптора информации о статусе сертификата от сервера OCSP.

```
AvCmResult AvCmMsgOCSPGetResponse (
    AvCmHandle handle,
    DWORD sign_index,
    const AvCmHOCSP * hr,
    DWORD flags);
```

Параметры

[in] AvCmHandle handle – дескриптор объекта, для которого необходимо получить информацию о статусе сертификата. Поддерживаются следующие объекты: сертификат (AvCmHcert), подпись (AvCmHsign) и подписанное сообщение (AvCmHmsg).

[in] DWORD sign_index – номер подписи в сообщении, если в качестве дескриптора объекта используется подписанное сообщение.

[out] AvCmHOCSP * hr – дескриптор информации о статусе сертификата.

[in] DWORD flags – параметры (флаги) работы.

Результат

Код возврата типа **AvCmResult**

Возможные значения:

- **AVCMR_SUCCESS** – успешное выполнение операции
- **AVCMR_BAD_HANDLE** – дескриптор неверен
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.

Извлечение атрибутов информации о статусе сертификата

Функция **AvCmMsgOCSPGetResponseAttr** предназначена извлечения атрибутов информации о статусе сертификата.

```
AvCmResult AvCmMsgOCSPGetResponseAttr (
    AvCmHOCSP hr,
    DWORD attr_id,
    const void * attr_param,
    void * output_buffer,
    size_t * output_size,
    DWORD flags);
```

Параметры

[in] AvCmHOCSP hr – дескриптор информации о статусе сертификата, атрибуты которого необходимо извлечь.

[in] DWORD attr_id – идентификатор извлекаемого атрибута:

- **AVCM_OCSP_STATUS** – статус сертификата, на время когда была получена информация о статусе сертификата
  Результат:
  output_size: sizeof(DWORD)
  output_buffer: DWORD – статус сертификата;
  Возможные значения статуса сертификата**:**
  - **AVCM_OCSP_STATUS_GOOD** – сертификат действителен;
  - **AVCM_OCSP_STATUS_REVOKED** – сертификат отозван;
  - **AVCM_OCSP_STATUS_UNKNOWN** – OCSP серверу не удалось определить статус сертификата;
  - **AVCM_OCSP_STATUS_BAD** – не удалось проверить целостность информации о статусе сертификата.

- **AVCM_OCSP_THIS_UPDATE** – дата текущего обновления ответа OCSP сервера.
  Результат:
  output_size: sizeof(SystemTime)
  output_buffer: дата текущего обновления ответа OCSP сервера.

- **AVCM_OCSP_NEXT_UPDATE** – дата следующего обновления ответа OCSP сервера (не обязательный атрибут).
  Результат:
  output_size: sizeof(SystemTime)
  output_buffer: дата следующего обновления ответа OCSP сервера.

- **AVCM_OCSP_REV_TIME** – дата отзыва сертификата (если сертификат отозван).
  Результат:
  output_size: sizeof(SystemTime)
  output_buffer: дата отзыва сертификата.

- **AVCM_OCSP_REV_REASON_AS_DWORD** – код причины отзыва сертификата (если сертификат отозван).
  Результат:
  output_size: sizeof(DWORD)

output_buffer: код причины отзыва сертификата.

- **AVCM_OCSP_RESPONSE_CERT** – дескриптор сертификата OCSP сервера.
  Результат:
  output_size: sizeof(AvCmHandle)
  output_buffer: дескриптор сертификата.

[in] const void * attr_param – параметр извлекаемого атрибута (в данной версии не используется).

[in,out] void * output_buffer – указатель на буфер для извлекаемого атрибута.

[in,out] size_t * output_size – указатель на размер буфера извлекаемого атрибута. Если размер меньше необходимого, функция заполнит требуемый размер. В случае успешного выполнения будет заполнен размер полученного значения.

[in] DWORD flags – параметры

- **AVCMF_ALLOC** – функция должна самостоятельно распределить память для выходного сообщения функцией *Win32 API* **HeapAlloc**, после использования выходного сообщения, вызвавшая программа должна самостоятельно очистить выделенную память функцией **HeapFree**. См. также Распределение памяти.

Если указан данный флаг, то **output_buffer** должен быть указателем на указатель на буфер для извлекаемого атрибута подписи.

Результат

Код возврата типа **AvCmResult**
Возможные значения:

- **AVCMR_SUCCESS** – успешное выполнение операции
- **AVCMR_BAD_HANDLE** – дескриптор объекта неверен
- **AVCMR_BAD_ATTR** – идентификатор атрибута неверен
- **AVCMR_ALLOC_ERROR** – ошибка при распределении памяти
- **AVCMR_BUFFER_TOO_SMALL** – выходной буфер слишком мал
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.
- **AVCMR_NOT_FOUND** – объект(атрибут объекта) не найден.
- **AVCMR_BAD_FORMAT** – неверный формат.

Добавление информации о статусе сертификата в подписанное сообщение

Функция **AvCmMsgOCSPAddResponse** предназначена для добавления информации о статусе сертификата в подписанное сообщение.

Размещение памяти для сообщения может быть произведено самой функцией при помощи функции *Win32 API* **HeapAlloc**.

```
AvCmResult AvCmMsgOCSPAddResponse(
    AvCmHandle handle,
    DWORD sign_index,
    AvCmHOCSP hr,
    DWORD flags);
```

Параметры

[in] AvCmHandle handle – дескриптор подписанного сообщения или дескриптор информации о подписи

[in] DWORD sign_index – номер подписи, для подписанного сообщения

[in] AvCmHOCSP hr – дескриптор информации о статусе сертификата

[in] DWORD flags – в данной реализации не используется и должно быть установлено в 0.

Результат

Код возврата типа **AvCmResult**

Возможные значения:

- **AVCMR_SUCCESS** (0) – успешное выполнение операции
- **AVCMR_INTERNAL_ERROR** – внутренняя ошибка библиотеки.
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.
- **AVCMR_BAD_PARAM** – параметр функции неверен

Контроль точек распространения СОС

Контроль точек распространения СОС

Функция **AvCmCheckCrlDP** предназначена для получения текучего СОС с указанных точек распространения СОС, проверки его целостности и помещение в справочник сертификатов и СОС. Если имеется несколько альтернативных точек распространения СОС, они должны быть разделены символами перевода строки (#13#10) , если хотя бы в одной точке распределения СОС имеется более новый СОС – результат функции будет **AVCMR_SUCCESS.** Путь доступа к точкам распространения СОС может быть получен из сертификата вызовом функции **AvCmGetCertAttr** с параметром **AVCM_CRL_DISTRIBUTION_POINTS**, либо напрямую получен из удостоверяющего центра. Поддерживаются точки распространения СОС по протоколам HTTP и LDAP.

```
AvCmResult AvCmCheckCrlDP(
    AvCmHc hc,
    PChar CrlDPPath,
    DWORD flags);
```

Параметры

[in] AvCmHc hc – дескриптор открытой сессии.

[in] PChar CrlDPPath – путь доступа к точкам распространения СОС.

[in] DWORD flags – параметры (флаги) работы.
В настоящий момент нет поддерживаемых флагов.

Результат

Код возврата типа **AvCmResult**
Возможные значения:

- **AVCMR_SUCCESS** – успешное выполнение операции.
- **AVCMR_BASE** – неверно задан путь к точкам распределения СОС.
- **AVCMR_BAD_HANDLE** – дескриптор неверен.
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.
- **AVCMR_INTERNAL_ERROR** – внутренняя ошибка библиотеки.
- **AVCMR_CRL_NOT_FOUND** – СОС не найден.
- **AVCMR_CERT_CA_CRL_NOT_FOUND** – Сертификат издателя СОС не найден.
- **AVCMR_CRL_INVALID** – Подпись под СОС издателя неверна.
- **AVCMR_OLD_CRL** – В справочнике списков отозванных сертификатов имеется более новый (такой же) СОС данного издателя.

Взаимодействие с сервером SCEP

Создание соединения с сервером SCEP

Функция **AvCmCreateScep** предназначена для получения дескриптора соединения с сервером SCEP.

```
AvCmResult AvCmCreateScep(
    AvCmHc hc,
    PChar URL,
    const AvCmHandle * hs,
    DWORD flags);
```

Параметры

[in] AvCmHc hc – дескриптор открытой сессии.

[in] PChar URL – путь доступа к серверу SCEP.

[out] AvCmHandle * hs – дескриптор соединения с сервером SCEP.

[in] DWORD flags – параметры (флаги) работы.
В настоящий момент нет поддерживаемых флагов.

Результат

Код возврата типа **AvCmResult**
Возможные значения:

- **AVCMR_SUCCESS** – успешное выполнение операции.
- **AVCMR_SCEP_ERROR** – ошибка при обращении к серверу SCEP.
- **AVCMR_BAD_HANDLE** – дескриптор неверен.
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.
- **AVCMR_INTERNAL_ERROR** – внутренняя ошибка библиотеки.

Выполнение операции соединения с сервером SCEP

Функция **AvCmScepExecute** предназначена для выполнения опреции соединения с сервером SCEP.

```
AvCmResult AvCmScepExecute(
    AvCmHandle hs,
    AvCmLong oper_id,
    DWORD flags);
```

Параметры

[in] AvCmHandle hs – дескриптор соединения с сервером SCEP.

[in] AvCmLong oper_id – идентификатор операции:

- **AVCM_SCEP_PKCS_REQ** – запрос к серверу SCEP на обработку запроса на сертификат.
- **AVCM_SCEP_LOGIN** – установление связи контейнера личных ключей с открытым ключом запроса на сертификат.

[in] DWORD flags – параметры

Результат

Код возврата типа **AvCmResult**
Возможные значения:

- **AVCMR_SUCCESS** – успешное выполнение операции.
- **AVCMR_SCEP_PENDING** – запрос на сертификат ждет ручного утверждения.
- **AVCMR_SCEP_ERROR** – ошибка при обращении к серверу SCEP.
- **AVCMR_BAD_HANDLE** – дескриптор объекта неверен.
- **AVCMR_BAD_ATTR** – идентификатор атрибута неверен.

Получение данных соединения с сервером SCEP

Функция **AvCmScepGet** предназначена для получения данных соединения с сервером SCEP.

```
AvCmResult AvCmScepGet(
    AvCmHandle hs,
    AvCmLong param_id,
    const void * param_spec,
    void * out_data,
    size_t * output_size,
    DWORD flags);
```

Параметры

[in] AvCmHandle hs – дескриптор соединения с сервером SCEP.

[in] AvCmLong param_id – идентификатор извлекаемых данных:

- **AVCM_CONTAINERCOUNT** – количество контейнеров на текущем носителе
  Результат:
  output_size: sizeof(AvCmLong)
  output_buffer: AvCmLong – количество контейнеров;

- **AVCM_CONTAINERNAME** – имя контейнера, номер контейнера, начиная с 0 указывается как указатель на **AvCmLong** в **param_spec**.
  Результат:
  output_size: размер строки
  output_buffer: имя контейнера в формате строки *ASCIIZ*

- **AVCM_SCEP_PKCS_REQ** – результат обработки запроса на сертификат сервером SCEP.
  Результат:
  output_size: размер сообщения
  output_buffer: обрезанное сообщение PKCS#7 содержащее выпученный сертификат (первый в сообщении) и сертификаты УЦ.

- **AVCM_SCEP_TRANSACTIONID** – ID транзакция текущего запроса к серверу SCEP.
  Результат:
  output_size: размер строки
  output_buffer: ID транзакции в формате строки *ASCIIZ*.

[in] const void * param_spec – параметр извлекаемых данных.

[in,out] void * output_buffer – указатель на буфер для извлекаемых данных.

[in,out] size_t * output_size – указатель на размер буфера извлекаемых данных. Если размер меньше необходимого, функция заполнит требуемый размер. В случае успешного выполнения будет заполнен размер полученного значения.

[in] DWORD flags – параметры

- **AVCMF_ALLOC** – функция должна самостоятельно распределить память для выходного сообщения функцией *Win32 API* **HeapAlloc**, после использования выходного сообщения,

вызвавшая программа должна самостоятельно очистить выделенную память функцией **HeapFree**. См. также Распределение памяти.

Если указан данный флаг, то **output_buffer** должен быть указателем на указатель на буфер для извлекаемого атрибута подписи.

Результат

Код возврата типа **`AvCmResult`**

Возможные значения:

- **AVCMR_SUCCESS** – успешное выполнение операции.
- **AVCMR_SCEP_PENDING** – запрос на сертификат ждет ручного утверждения.
- **AVCMR_SCEP_ERROR** – ошибка при обращении к серверу SCEP.
- **AVCMR_BAD_HANDLE** – дескриптор объекта неверен.
- **AVCMR_BAD_ATTR** – идентификатор атрибута неверен.
- **AVCMR_ALLOC_ERROR** – ошибка при распределении памяти.
- **AVCMR_BUFFER_TOO_SMALL** – выходной буфер слишком мал.
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.
- **AVCMR_NOT_FOUND** – объект(атрибут объекта) не найден.
- **AVCMR_BAD_FORMAT** – неверный формат.

Установка данных соединения с сервером SCEP

Функция **AvCmScepSet** предназначена для установки данных соединения с сервером SCEP.

```
AvCmResult AvCmScepSet(
    AvCmHandle hs,
    AvCmLong param_id,
    void * in_data,
    size_t in_size,
    DWORD flags);
```

Параметры

[in] AvCmHandle hs – дескриптор соединения с сервером SCEP.

[in] AvCmLong param_id – идентификатор устанавливаемых данных:

- **AVCM_PASSWORD** – пароль доступа к контейнеру личных ключей;
- **AVCM_CONTAINERNAME** – имя контейнера;
- **AVCM_SCEP_REQUEST** – блоб запроса на сертификат;
- **AVCM_SCEP_TRANSACTIONID** – ID транзакции запроса к серверу SCEP;

[in]void * in_data – указатель на блок памяти, содержащий устанавливаемые данные.

[in] size_t * in_size – размер устанавливаемых данных.

[in] DWORD flags – в данной реализации не используется и должно быть установлено в 0.

Результат

Код возврата типа **AvCmResult**

Возможные значения:

- **AVCMR_SUCCESS** (0) – успешное выполнение операции
- **AVCMR_INTERNAL_ERROR** – внутренняя ошибка библиотеки.
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.
- **AVCMR_BAD_PARAM** – параметр функции неверен.
- **AVCMR_NOT_FOUND** – объект не найден.

Обмен данными по протоколу TLS

Создание соединения с сервером TLS

Функция **AvCmTLSCreateConnect** предназначена для получения дескриптора соединения с сервером TLS.

```
AvCmResult AvCmTLSCreateConnect(
    AvCmHc hc,
    size_t conn_param_count,
    const AvCmConnectionParam * conn_params,
    AvCmHandle * htls,
    DWORD flags);
```

Параметры

[in] AvCmHc hc – дескриптор открытой сессии.

[in] size_t conn_param_count – количество параметров идентификации соединения.

[in] const AvCmConnectionParam * conn_params – массив параметров идентификации соединения:

В настоящее время поддерживается следующие параметры идентификации соединения:

- **AVCM_HTTPS_URL** – URL TLS сервера;
- **AVCM_HTTPS_PROXY_VERSION** – версия прокси сервера (1, 4(Socks4), 5(Socks5));
- **AVCM_HTTPS_PROXY** – URL прокси сервера;
- **AVCM_HTTPS_USERAGENT** – USERAGENT;
- **AVCM_HTTPS_CONTENTTYPE** – CONTENTTYPE;
- **AVCM_HTTPS_READTIMEOUT** – READTIMEOUT;

[out] AvCmHandle * htls – дескриптор соединения с сервером TLS.

[in] DWORD flags – параметры (флаги) работы.
В настоящий момент нет поддерживаемых флагов.

Результат

Код возврата типа **AvCmResult**
Возможные значения:

- **AVCMR_SUCCESS** – успешное выполнение операции.
- **AVCMR_SERVER_ERROR** – ошибка при обращении к серверу TLS.
- **AVCMR_TLS_INCOMPLETE_CREDENTIALS** – сервер запросил аутентификацию клиента, но переданный мандат не содержит требуемый сертификат клиента.
- **AVCMR_BAD_HANDLE** – дескриптор неверен.
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.
- **AVCMR_INTERNAL_ERROR** – внутренняя ошибка библиотеки.

Получение данных по протоколу TLS

Функция **AvCmTlsGet** предназначена для получения данных по протоколу TLS.

```
AvCmResult AvCmTlsGet(
    AvCmHandle htls,
    PChar URL,
    const void ** res_data,
    size_t * res_size,
    DWORD flags);
```

Параметры

[in] AvCmHandle htls – дескриптор соединения с сервером TLS.
[in] PChar URL – путь доступа к данным.
[in,out] void ** res_data – указатель на указатель для извлекаемых данных.
[in,out] void * res_size – указатель на размер извлекаемых данных.
[in] DWORD flags – параметры (флаги) работы.
В настоящий момент нет поддерживаемых флагов.

Результат

Код возврата типа **AvCmResult**
Возможные значения:

- **AVCMR_SUCCESS** – успешное выполнение операции.
- **AVCMR_SERVER_ERROR** – ошибка при обращении к серверу TLS.
- **AVCMR_TLS_INCOMPLETE_CREDENTIALS** – сервер запросил аутентификацию клиента, но переданный мандат не содержит требуемый сертификат клиента.
- **AVCMR_BAD_HANDLE** – дескриптор неверен.
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.
- **AVCMR_INTERNAL_ERROR** – внутренняя ошибка библиотеки.

Отправка данных по протоколу TLS

Функция **AvCmTlsPost** предназначена для получения данных по протоколу TLS.

```
AvCmResult AvCmTlsPos(
    AvCmHandle htls,
    PChar URL,
    const void data,
    size_t data_size,
    const void ** res_data,
    size_t * res_size,
    DWORD flags);
```

Параметры

[in] AvCmHandle htls – дескриптор соединения с сервером TLS.
[in] PChar URL – путь доступа к данным.
[in] void data – указатель на отправляемые данные.
[in] size_t data_size –размер отправляемых данных.
[in,out] void ** res_data – указатель на указатель для извлекаемых данных.
[in,out] size_t * res_size – указатель на размер извлекаемых данных.
[in] DWORD flags – параметры (флаги) работы.
В настоящий момент нет поддерживаемых флагов.

Результат

Код возврата типа **AvCmResult**
Возможные значения:

- **AVCMR_SUCCESS** – успешное выполнение операции.
- **AVCMR_SERVER_ERROR** – ошибка при обращении к серверу TLS.
- **AVCMR_TLS_INCOMPLETE_CREDENTIALS** – сервер запросил аутентификацию клиента, но переданный мандат не содержит требуемый сертификат клиента.
- **AVCMR_BAD_HANDLE** – дескриптор неверен.
- **AVCMR_BAD_FLAGS** – функции переданы неверные флаги.
- **AVCMR_INTERNAL_ERROR** – внутренняя ошибка библиотеки.

Возвращаемые коды ошибок

Ошибка	Описание
AVCMR_SUCCESS	Успешное выполнение функции (0)
AVCMR_ALG_NOT_SUPPORTED	алгоритм не поддерживается
AVCMR_ALLOC_ERROR	Ошибка при распределении памяти
AVCMR_ALREADY_EXISTS	Объект уже существует
AVCMR_ALREADY_INITIALIZED	Повторная попытка инициализации библиотеки
AVCMR_AVCSP_INIT_FAILED	Инициализация криптопровайдера Авест завершилась неудачно
AVCMR_BAD_ATTR	Идентификатор атрибута неверен
AVCMR_BAD_DATE	Формат даты/времени некорректен
AVCMR_BAD_FLAGS	Функции переданы неверные флаги
AVCMR_BAD_FORMAT	Неверный формат
AVCMR_BAD_FORMED_SIGN	Неверный формат подписи СТБ
AVCMR_BAD_HANDLE	Дескриптор объекта неверен
AVCMR_BAD_HC	Дескриптор соединения неверен
AVCMR_BAD_HCERT	Дескриптор сертификата неверен
AVCMR_BAD_HENUM	Дескриптор перебора неверен
AVCMR_BAD_HMSG	Дескриптор открытого сообщения неверен
AVCMR_BAD_HSIGN	Дескриптор подписи неверен
AVCMR_BAD_NUMBER	В сообщении нет подписи с указанным номером
AVCMR_BAD_PARAM	Параметр функции неверен
AVCMR_BAD_PASSWORD	Пароль неверен
AVCMR_BAD_THREAD	Попытка обратится к сессии, созданной в одном потоке, из другого потока
AVCMR_BUFFER_TOO_SMALL	Выходной буфер слишком мал
AVCMR_BUSY	Невозможно закрыть объект, объект занят
AVCMR_CERT_ NOT_FOUND	Не найден сертификат
AVCMR_CERT_CA_INVALID	Подпись под сертификатом издателя неверна
AVCMR_CERT_CA_NOT_FOUND	Сертификат издателя не найден
AVCMR_CERT_EXPIRED	Срок действия личного ключа сертификата неверен
AVCMR_CERT_NOT_FOR_CRYPT	Сертификат не предназначен для зашифрования
AVCMR_CERT_NOT_FOR_SIGN	Сертификат не предназначен для подписи
AVCMR_CERT_NOT_FOUND	Сертификат не найден
AVCMR_CERT_SIGN_INVALID	Подпись под сертификатом неверна
AVCMR_CERT_STORE_NOT_FOUND	Не найдено или пусто хранилище сертификатов
AVCMR_CONTAINER_NOT_FOUND	Не найден контейнер с личными ключами на носителе
AVCMR_CRL_INVALID	Подпись под СОС издателя неверна или СОС устарел
AVCMR_CRL_NOT_FOUND	СОС издателя не найден
AVCMR_DATE_NOT_VALID	Дата не подходящая
AVCMR_DB_NOT_FOUND	Невозможно подключится к базе данных
AVCMR_DEVICE_NOT_FOUND	Не найден носитель
AVCMR_INTERNAL_ERROR	Внутренняя ошибка библиотеки
AVCMR_NO_DB_PARAMS	Не указаны параметры подключения к базе данных

AVCMR_NO_INPUT	Входное сообщение имеет нулевой размер
AVCMR_NO_SIGN	Сообщение не имеет подписи, выработанной на ключе данного сертификата.
AVCMR_NOT_FOUND	Объект (атрибут объекта) не найден
AVCMR_NOT_IMPLEMENTED	Функция не реализована
AVCMR_NOT_INITIALIZED	Библиотека не была инициализирована
AVCMR_REGISTRY_ERROR	Ошибка доступа к реестру Windows
AVCMR_SIGN_INVALID	Подпись неверна
AVCMR_USER_NO_AUTH	Сессия создана без авторизации пользователя
AVCMR_WIN32_ERROR	Произошла ошибка при вызове функций Win32
AVCMR_OTHER_RECIPIENT	Невозможно расшифровать сообщение: среди сертификатов получателей сообщения отсутствует личный сертификат аутентифицированного пользователя
AVCMR_CTL_NOT_FOUND	Не найден СДЦС (список доверяемых центров сертификации)
AVCMR_CERT_REVOKED	Сертификат отозван
AVCMR_CERT_NOT_TRUSTED	Нет доверия сертификату
AVCMR_CRL_EXPIRED	Срок действия СОС истек
AVCMR_CRL_ISSUER_NOT_FOUND	Сертификат издателя СОС не найден
AVCMR_CRL_ISSUER_EXPIRED	Срок действия сертификата издателя СОС истек
AVCMR_CERT_STORE_BAD_VERSION	Версия библиотеки не соответствует версии хранилища сертификатов.
AVCMR_MY_STORE_EMPTY	В личном хранилище отсутствуют действующие сертификаты
AVCMR_USER_CANCEL	Действие отменено пользователем
AVCMR_RA_EXT_KEY_USAGE_NOT_ALLOWED	Сертификат содержит расширенное использование ключа не поддерживаемое центром регистрации
AVCMR_RA_EXT_NOT_ALLOWED	Сертификат содержит дополнение, не поддерживаемое центром регистрации
AVCMR_UNSUPPORTED_CONNECTION_TYPE	Импорт в данный тип хранилища не поддерживается

Примеры

Генерация подписанного сообщения при помощи низкоуровневых функций

```
#include <string>
#include "AvCryptMail.h"

class AvCmException
{
public:
 std::string mErrMessage;
 AvCmResult mErrCode;

public:
 AvCmException(
       AvCmResult aErrCode,
       const char * aErrMessage)
       : mErrCode(aErrCode),
         mErrMessage(aErrMessage)
 {}
};

void CheckAvCmResult(AvCmResult aRc)
{
 AvCmResult rc;
 char err_buff[4096];
 size_t output_size = sizeof(err_buff);

 if( aRc != AVCMR_SUCCESS )
 {
       rc = AvCmGetErrorInfo(
                  0,
                  aRc,
                  AVCM_DESCRIPTION,
                  err_buff,
                  &output_size,
                  AVCMF_THREAD_ERROR);
       if( rc != AVCMR_SUCCESS )
       {
             sprintf(err_buff, "AvCmGetErrorInfo error 0x%x", rc);
       }
       throw AvCmException(aRc, err_buff);
 }
}


int Sign(
       const void * aInput, size_t aInputLen,
       void ** aOutput, size_t * aOutputLen)
{
 AvCmResult rc;

 // Initialization...
 rc = AvCmInit( AVCMF_STARTUP );
 CheckAvCmResult( rc );

 // Login...
 AvCmHc hc;
 rc = AvCmLogin(
            0,
```

```
            0,
            &hc,
            0);
 CheckAvCmResult( rc );

 AvCmHmsg hmsg;

 // Creating signed message...
 rc = AvCmOpenMsg(
            hc,
            aInput,
            aInputLen,
            &hmsg,
            AVCMF_IN_RAW_DATA);
 CheckAvCmResult( rc );

 // Signing...
 rc = AvCmMsgSign(
            hmsg,
            0);
 CheckAvCmResult( rc );

 // Getting signed message...
 char * output_msg;
 size_t output_size;
 rc = AvCmGetMsg(
            hmsg,
            0,
            aOutputLen,
            AVCMF_OUT_PKCS7);
 CheckAvCmResult( rc );

 *aOutput = new char[*aOutputLen];
 rc = AvCmGetMsg(
            hmsg,
            *aOutput,
            aOutputLen,
            AVCMF_OUT_PKCS7);
 CheckAvCmResult( rc );


 // Closing signed message...
 rc = AvCmCloseHandle( hmsg, 0 );
 CheckAvCmResult( rc );


 // Logout...
 rc = AvCmLogout(hc, 0);
 CheckAvCmResult( rc );

 // Deinitialization...
 rc = AvCmInit( AVCMF_SHUTDOWN );
 CheckAvCmResult( rc );


 return rc;
}
```

Проверка ЭЦП под сообщением

Приложение 1. Формат файла с описанием операции

Файл имеет формат Windows INI файла в кодировке Windows CP-1251.

Для каждой операции заполняется секция INI файла. Имя секции должно иметь вид OPERxxxx, где xxxx – произвольный номер операции.

Генерация запроса на сертификат

```
[OPER1]
Operation=gen_request
OutputFileName=<полный путь к файлу с сгенерированным запросом>
PublicKeyID=<шестнадцатеричное представление идентификатора сгенерированного открытого ключа>
TemplateFilename=<полный путь к файлу с использованным шаблоном сертификата>
ProlongationFor=<шестнадцатеричное представление идентификатора сертификата, для которого подготовлен запрос на продолжение>
```

Импорт сертификата

```
[OPER1]
Operation=import_cert
InputFileName=<полный путь к файлу с импортируемым сертификатом>
PublicKeyID=<шестнадцатеричное представление идентификатора импортированного сертификата>
Imported=MY,ROOT,CA
```

Импорт СОС

```
[OPER1]
Operation=import_crl
InputFileName=<полный путь к файлу с импортируемым СОС>
PublicKeyID=<шестнадцатеричное представление идентификатора ключа издателя импортированного СОС>
```

Текущие вопросы реализации

Описание форматов сообщений

Библиотека использует для обрабатываемых сообщений формат, определенный в документе *RSA Laboratories PKCS #7: Cryptographic Message Syntax Standard*. Настоящая глава описывает используемые библиотекой варианты форматов. Описание представлено в соответствии со стандартом *ASN.1*. Стандарт PKCS #7 описан исходя из применения алгоритма выработки ЭЦП RSA и вследствие этого, требует доработок и уточнений для работы с другими алгоритмами выработки ЭЦП.

Общий синтаксис

Общий синтаксис сообщения для обмена состоит из типа сообщения и связанного с ним содержимого сообщения. Синтаксис имеет *ASN.1* тип `ContentInfo`:

```
ContentInfo ::= SEQUENCE {
  contentType ContentType,
  content
    [0] EXPLICIT ANY DEFINED BY contentType OPTIONAL }

ContentType ::= OBJECT IDENTIFIER
```

Поля типа `ContentInfo` имеют следующее назначение:

`contentType` описывает тип сообщения. Это идентификатор объекта. Стандарт *PKCS #7* определяет шесть типов:

- *данные* (`data`)
- *подписанные данные* (`signedData`)
- *электронный конверт* (`envelopedData`)
- *электронный конверт подписанных данных* (`signedAndEnvelopedData`)
- *хэшированные данные* (`digestedData`)
- *зашифрованные данные* (`encryptedData`)

В библиотеке используются только три из описанных типов сообщений: *данные, подписанные данные* и *электронный конверт*. В дальнейшем описании будут описываться только эти типы сообщений.

Поле `content` является содержимым сообщения. Содержимое сообщения должно соответствовать его типу. Стандарт описывает данное поле как опциональное, библиотека не поддерживает данную возможность.

Примечание

В случае если содержимое представляет собой подписанное сообщение, то алгоритм хэширования применяется к октетам содержимого в кодировке *DER*. Если же содержимым является зашифрованное сообщение, то алгоритм шифрования сообщения применяется к октетам содержимого в кодировке *BER* с определенной длиной.

Сообщение типа «Данные»

Тип сообщения *данные* – это просто строка октетов. Она должна иметь тип *ASN.1* `Data`:

```
Data ::= OCTET STRING
```

Тип данных *данные* описывает произвольные строки октетов, например ASCII текстовые файлы, интерпретация таких данных производится приложениями.

Сообщение типа «Подписанные данные»

Тип сообщения *подписанные данные* состоит из содержимого любого типа и зашифрованных электронных дайджестов (цифровых подписей) содержимого любым числом подписавших, включая 0. Любое содержимое может быть подписано любым количеством лиц. Синтаксис имеет специальный вариант, когда в сообщении нет ни одной подписи. Данный вариант сообщения предназначен для распространения сертификатов и СОС.

Процесс создания подписанного сообщения включает в себя следующие шаги:

1. Для каждого подписавшего лица вычисляется дайджест сообщения путем вычисление хэш-функции от содержимого сообщения при помощи используемого подписывающим лицом алгоритма хэширования. В случае, если производится авторизация любой другой информации, кроме самого сообщения, дайджест сообщения и иная авторизуемая информация хэшируются алгоритмом хэширования подписывающего лица, и результат данного хэширования в дальнейшем является дайджестом сообщения.

2. Для каждого подписывающего лица, дайджест сообщения и ассоциированная информация[1] зашифровывается секретным ключом подписывающего лица.

3. Для каждого подписывающего лица, зашифрованный дайджест сообщения и иная информация, специфичная для подписывающего, собирается в структуру `SignerInfo`, описанную ниже. Сертификаты и списки отозванных сертификатов для каждого подписывающего лица, а также не относящиеся ни к кому из подписывающих также собираются на данном шаге процесса создания подписанного сообщения.

4. Алгоритмы хэширования для всех подписывающих и структуры `SignerInfo` для всех подписывающих лиц собираются вместе с содержимым сообщения типа `SignedData`, как описано далее.

Получатель проверяет ЭЦП расшифровав зашифрованный дайджест сообщения для каждого подписавшего лица с использованием открытого ключа подписавшего и сравнивает полученный дайджест сообщения с подсчитанным функцией хэширования, то есть выполняет проверку ЭЦП. Открытый ключ подписавшего содержится или в сертификате, включенном в сообщение или может быть однозначно найден в хранилище сертификатов по уникальному имени издателя и серийному номеру сертификата, хранящимся в структуре `SignerInfo`.

Тип данных SignedData

Тип данных *подписанные данные* (`signedData`) имеет тип *ASN.1* `SignedData`:

```
SignedData ::= SEQUENCE {
  version Version,
  digestAlgorithms DigestAlgorithmIdentifiers,
  contentInfo ContentInfo,
  certificates
      [0] IMPLICIT ExtendedCertificatesAndCertificates
```

[1] Это перевод PKCS#7: For each signer, the message digest and **associated information** are encrypted with the signer's private key. В текущей реализации наличие такой информации не предусматривается.

```
      OPTIONAL,
  crls
    [1] IMPLICIT CertificateRevocationLists OPTIONAL,
  signerInfos SignerInfos }

DigestAlgorithmIdentifiers ::=
  SET OF DigestAlgorithmIdentifier

SignerInfos ::= SET OF SignerInfo
```

Поля структуры `SignedData` имеют следующее назначение:

- `version` – это номер версии синтаксиса структуры. Должен иметь значение 1.

- `digestAlgorithms` – набор идентификаторов алгоритмов хэширования. Набор может включать любое количество элементов, включая 0. Каждый идентификатор определяет алгоритм хэширования и любое ассоциированные параметры, при помощи которого был выработан цифровой дайджест сообщения одним из подписавших. Набор предназначен для описания всех использовавшихся при выработке подписей алгоритмов хэширования, в любом порядке, для того чтобы позволить проверить все подписи в один проход.

- `contentInfo` – это подписываемое содержимое сообщения. Оно должно иметь любой из определенных типов содержимого (*данные, подписанные данные* или *электронный конверт*).

- `certificates` – это множество *PKCS #6* расширенных сертификатов и *X.509* сертификатов[2]. Данный элемент структуры предназначен для включения необходимых для проверки ЭЦП цепочек сертификатов для всех подписавших сообщение лиц. Данное поле может включать больше сертификатов, чем необходимо. Оно может включать сертификаты любого количества цепочек независимых центров сертификатов. Также сообщение может включать не все необходимые сертификаты. В этом случае ожидается что получатель имеет иные возможности для получения необходимых для проверки ЭЦП данного сообщения сертификатов.

- `crls` – это множество списков отозванных сертификатов. Он может содержать СОС центров сертификатов, выдавших сертификаты содержащиеся в поле `certificates`, для проверки того, что данные сертификаты не были отозваны их издателями. Но данное соответствие необязательно. Множество СОС может содержать больше списков чем необходимо, также как и меньшее их количество.

- `signerInfos` – набор информации по выработанным ЭЦП. Данный набор может содержать любое количество элементов, включая 0.

Примечания

[2]Вопрос о поддерживаемых типах сертификатов в настоящий момент открыт.

1. Тот факт что поле `digestAlgorithms` находится раньше поля `contentInfo` и поле `signerInfos` находится позже поля `contentInfo` позволяет обрабатывать структуру `SignedData` в один проход.

2. Библиотека AvCryptMail поддерживает только сообщения формата `SignedData` только версии 1и 0 и генерирует сообщения только формата версии 1. Различия между версиями 1 и 0:

- поля `digestAlgorithms` и `signerInfos` могут содержать 0 элементов только в версии 1.

- поле `crls` разрешено только в версии 1.

Следствием этих отличий является то, что сообщения версии 0 являются подмножеством возможных сообщений версии 1 и могут обрабатываться теми же алгоритмами обработки. При этом генерируются библиотекой только сообщения версии 1.

3. В случае отсутствия подписей в сообщении рекомендуется чтобы содержимое поля `ContentInfo` также отсутствовало, поскольку проверить его корректность невозможно.

Тип данных SignerInfo

Информация о каждой из подписей представляется типом `SignerInfo`:

```
SignerInfo ::= SEQUENCE {
  version Version,
  issuerAndSerialNumber IssuerAndSerialNumber,
  digestAlgorithm DigestAlgorithmIdentifier,
  authenticatedAttributes
    [0] IMPLICIT Attributes OPTIONAL,
  digestEncryptionAlgorithm
    DigestEncryptionAlgorithmIdentifier,
  encryptedDigest EncryptedDigest,
  unauthenticatedAttributes
    [1] IMPLICIT Attributes OPTIONAL }

EncryptedDigest ::= OCTET STRING
```

Поля типа `SignerInfo` имеют следующее назначение:

- `version` – версия синтаксиса типа данных. В настоящий момент поддерживается только версия 1.

- `issuerAndSerialNumber` – определяет сертификат подписавшего лица. Данное поле состоит из уникального имени издателя и серийного номера сертификата, присвоенного ему издателем.

- `digestAlgorithm` – определяет алгоритм хэширования и необходимые параметры алгоритма хэширования, использовавшиеся при выработки цифрового дайджеста сообщения при выработке подписи данным подписывающим лицом. Этот алгоритм хэширования и его параметры должен присутствовать в списке алгоритмов хэширования в поле `SignerInfo`, описанном ранее.

- `authenticatedAttributes` – множество подписываемых атрибутов, то есть атрибутов, чью подлинность заверяет выработанная ЭЦП. Это поле не является обязательным, но оно обязательно должно присутствовать если тип содержимого сообщения `ContentInfo` не *данные* (`data`). Если поле присутствует, оно должно содержать как минимум два атрибута:

  1. *PKCS #9* атрибут типа содержимого сообщения, значение которого должно быть таким же как и тип содержимого в поле `ContentInfo`, описанном ранее.

  2. *PKCS #9* атрибут *message-digest*, значением которого должен быть цифровой дайджест сообщения.

  3. *PKCS #9* Другие необходимые атрибуты, подлинность которых необходимо заверять определены в *PKCS #9*. Наиболее важным из них является дата и время выработки ЭЦП.

- `digestEncryptionAlgorithm` – определяет алгоритм шифрования цифрового дайджеста (и необходимые параметры), при помощи которого цифровой дайджест и ассоциированная информация были зашифрованы личным ключом подписывающего лица, то есть определяет использованный алгоритм выработки ЭЦП.

- `encryptedDigest` – это результат зашифрования цифрового дайджеста сообщения и ассоциированных атрибутов на личном ключе подписавшего лица, то есть электронная цифровая подпись.

- `unauthenticatedAttributes` – множество неподписываемых атрибутов, то есть множество атрибутов которые не нуждаются в удостоверении подлинности. Это поле опционально. Типы атрибутов, которые могут быть использованы в данном поле определены в *PKCS #9*. Одним из возможных типов, который может быть использован, является *countersignature*[3].

Примечания

1. Рекомендуется не использовать поле `authenticatedAttributes`, если тип содержимого сообщения `ContentInfo` *данные* (`data`) и не требуется заверять подлинность каких-либо дополнительных атрибутов. Библиотека использует в обязательном порядке дополнительный аутентифицируемый атрибут *дата и время подписи*, поэтому следовать данной рекомендации нет необходимости.

[3] Смысл термина в настоящий момент не ясен, похоже что это подпись подписи.

2. Разница между версиями 1 и 0 поля `SignerInfo` заключается в алгоритме зашифрования цифрового дайджеста сообщения. Предлагается что все реализации PKCS генерируют и обрабатывают только поля версии 1.

Процесс генерации цифрового дайджеста сообщения

Процесс генерации цифрового дайджеста сообщения вычисляет дайджест либо от подписываемого содержимого сообщения, либо от содержимого вместе с авторизуемыми атрибутами подписывающего лица. В любом случае первоначально на вход функции хэширования подается "значение" подписываемого сообщения. То есть на вход функции хэширования подаются октеты в кодировке *DER* поля `content` структуры `ContentInfo`. Только октеты в кодировке *DER* этого поля хэшируются, то есть октеты идентификатора и длины не хэшируются.

Результат процесса генерации цифрового дайджеста сообщения, то есть хэширования, зависит от того, присутствует ли поле `authenticatedAttributes`. Если поле отсутствует то результатом является только результат функции хэширования содержимого сообщения. Если же поле присутствует, то результатом является результат хэширования полного значения `Attributes` поля `authenticatedAttributes` в DER кодировке[4]. Поскольку значение `Attributes`, в том случае, когда поле присутствует, должно содержать атрибуты типа вложенного сообщения и его цифровой дайджест, эти поля косвенно включаются в результат.

Хотя октеты идентификатора содержимого и октеты длины не хэшируются, они защищены иным способом. Октеты длины защищены природой функции хэширования, поскольку невозможно найти два различных сообщения любой длины, чтобы их цифровой дайджест был одинаков. Октеты идентификатора содержимого защищены следующим образом: либо идентификатор должен быть *данные* `(data)`, либо он должен быть включен в поле авторизуемых (подписываемых) атрибутов.

Примечание

Тот факт, что хэш-функция вычисляется от части DER кодировки, не означает что DER является необходимым методом представления этой части для передачи данных. В действительности, некоторые реализации могут сохранять объекты в отличной от DER кодировки и это не отразится на вычисление хэш-функции[5].

Процесс зашифрования цифрового дайджеста

На вход процесса зашифрования цифрового дайджеста подаются результат генерации цифрового дайджеста сообщения, описанный ранее, и идентификатор алгоритма хэширования (ил идентификатор объекта). Результат зашифрования цифрового дайджеста это зашифрованное на личном ключе подписывающего лица BER кодировка значения типа `DigestInfo`[6]:

```
DigestInfo ::= SEQUENCE {
  digestAlgorithm DigestAlgorithmIdentifier,
  digest Digest }

Digest ::= OCTET STRING
```

Поля типа `DigestInfo` имеют следующее назначение:

---

[4] Тэг `IMPLICIT [0]` поля `authenticatedAttributes` не является частью значения `Attributes`. Тэг значения `Attributes – SET OF`, и поэтому DER кодировка тэга `SET OF`, а не тэга `IMPLICIT [0]` подается на вход функции хэширования вместе с длиной и октетами содержимого значения `Attributes`.

[5] Это дословный перевод The fact that the message digest is computed on part of a DER encoding does not mean that DER is the required method of representing that part for data transfer. Indeed, it is expected that some implementations of this standard may store objects in other than their DER encodings, but such practices do not affect message-digest computation.

[6] Данное описание процесса подписывания не ясно, и идет в разрез с алгоритмами генерации подписи.

- `digestAlgorithm` – идентифицирует алгоритм хэширования (и его параметры) при помощи которого подписываемое содержание было хэшировано. Он должен совпадать со значением поля `digestAlgorithm`.

- `digest` – это результат процесса генерации цифрового дайджеста сообщения.

Примечания[7]

Выдержки из PKCS #9: Типы атрибутов, используемые в PKCS #7

Некоторые из описанных типов определены также в *R. Housley. RFC 2630: Cryptographic Message Syntax CMS. IETF, June 1999.*

Тип атрибута тип содержимого (contentType)

Тип атрибута *тип содержимого* (`contentType`) определяет тип подписываемого содержимого. В таких данных данный атрибут обязателен при наличии аутентифицируемых атрибутов.

```
contentType ATTRIBUTE ::= {
WITH SYNTAX ContentType
EQUALITY MATCHING RULE objectIdentifierMatch
SINGLE VALUE TRUE
ID pkcs-9-at-contentType
}
ContentType ::= OBJECT IDENTIFIER
```

Данный тип атрибутов должен иметь единственное значение атрибута. Для проверки того, что два значения атрибута совпадают, их представления в виде строк октетов должны быть одинаковы по длине и соответствующие октеты идентичны, как определяет `objectIdentifierMatch`[8].

Цифровой дайджест сообщения

Тип атрибута *Цифровой дайджест сообщения* (`messageDigest`) определяет цифровой дайджест (хэш) октетов содержания в DER кодировке поля `content` подписываемого значения `ContentInfo`. Данный атрибут обязателен при наличии аутентифицируемых атрибутов.

```
messageDigest ATTRIBUTE ::= {
WITH SYNTAX MessageDigest
EQUALITY MATCHING RULE octetStringMatch
SINGLE VALUE TRUE
ID pkcs-9-at-messageDigest
}
MessageDigest ::= OCTET STRING
```

Данный тип атрибутов должен иметь единственное значение атрибута. Для проверки того, что два значения атрибута совпадают, их представления в виде строк октетов должны быть одинаковы по длине и соответствующие октеты идентичны, как определяет `octetStringMatch`[9].

---

[7] Примечания к данному пункту опущены в силу их неактуальности для алгоритмов выработки ЭЦП, отличных от RSA.

[8] ISO/IEC 9594-2:1997: Information technology – Open Systems Interconnection – The Directory: Models. 1997.

[9] ISO/IEC 9594-6:1997: Information technology – Open Systems Interconnection –

Дата и время подписи (signingTime)

Тип атрибута *Дата и время подписи* (`signingTime`) определяет дату и время подписания документа (предположительно).

```
signingTime ATTRIBUTE ::= {
WITH SYNTAX SigningTime
EQUALITY MATCHING RULE signingTimeMatch
SINGLE VALUE TRUE
ID pkcs-9-at-signingTime
}
SigningTime ::= Time -- imported from ISO/IEC 9594-8
```

Данный тип атрибутов должен иметь единственное значение атрибута. Правило сравнения `signingTimeMatch` возвращает **TRUE** если значения даты и времени совпадают. Правило сравнения описано в следующем разделе.

Цитата из *R. Housley. RFC 2630: Cryptographic Message Syntax CMS. IETF, June 1999*:
«Даты между 1 января и 31 декабря 2049 (включительно) ДОЛЖНЫ быть закодированы как `UTCTime`. Любые даты ранее 1950 года или позднее 2049 года ДОЛЖНЫ быть закодированы в формате `GeneralizedTime`. [Далее] Значения `UTCTime` должны быть представлены в GMT (время по Гринвичу) и ДОЛЖНО включать секунды (то есть быть в формате YYMMDDHHMMSSZ), даже если количество секунд 0. Полночь по Гринвичу должна иметь вид “YYMMDD000000Z”. Информация о столетии неявна, и определяется следующим правилом:

- Когда YY больше или равно 50, год следует трактовать как 19YY
- Когда YY меньше 50, год должен быть интерпретирован как 20YY.

Значения `GeneralizedTime` должны быть представлены в GMT (время по Гринвичу) и должно включать секунды, даже если количество секунд 0. Значения `GeneralizedTime` не должно включать дробную часть секунд. Никаких проверок корректности даты и времени не предусматривается и оставлено на рассмотрение приложениями.

Сравнение даты и времени

Правило `signingTimeMatch` определяет идентичность двух атрибутов типа *дата и время* (`SigningTime`).

```
signingTimeMatch MATCHING-RULE ::= {
SYNTAX SigningTime
ID pkcs-9-mr-signingTimeMatch
}
```

Правило возвращает **TRUE** если значения атрибутов представляют одно и тоже дату и время. Если сравниваемые строки имеют различный *ASN.1* синтаксис, сравнение выполняется следующим образом:
Where the strings being matched are of different *ASN.1* syntax, the comparison proceeds as follows:
1. Оба значения конвертируются в *DER* кодированные значения типа `GeneralizedTime`. Если это невозможно, сравнение завершается неудачно.
2. Две сконвертированные строки сравниваются. Правило возвращает **TRUE** если, и только если, строки имеют одинаковую длину и соответствующие их октеты идентичны.

The Directory: Selected attribute types. 1997.

Тип атрибута Countersignature

Тип атрибута `counterSignature` специфицирует одну или более ЭЦП октетов содержимого DER кодированного поля `encryptedDigest` значения `SignerInfo`**.**
Поэтому атрибут типа `countersignature` предписывает другую подпись. Атрибут данного типа должен быть не аутентифицируемым атрибутом, он не может входить во множество аутентифицируемых атрибутов.

```
counterSignature ATTRIBUTE ::= {
WITH SYNTAX SignerInfo
ID pkcs-9-at-counterSignature
}
```

Значения этого атрибута имеют такое же содержание, как и обычные значения `SignerInfo`, за исключением:
1. Поле `authenticatedAttributes` должно содержать атрибут `messageDigest` если оно содержит любые другие атрибуты, но не должно содержать атрибут `contentType`, поскольку никакого типа содержания для атрибута типа `counterSignature` нет.
2. Входом функции хэширования являются октеты содержания в DER кодировке поля `signatureValue` значения `SignerInfo`, с которым данный атрибут ассоциирован.

Атрибут типа `counterSignature` может иметь много значений.

Примечания

1. Тот факт, что *countersignature* вычисляется для подписи означает что процесс выработки *countersignature* не должен знать оригинальное сообщение. Преимущества данного подхода заключаются в ускорении процесса выработки ЭЦП и конфиденциальности.
Поскольку атрибут *countersignature* имеет тип `SignerInfo`, он может сам содержать атрибут типа *countersignature*. Это позволяет конструировать цепочки подписей *countersignature* любой длины.

Другие типы атрибутов

Стандарт PKCS #9 определяет также типы атрибутов `randomNonce` и `sequenceNumber`. Описание этих атрибутов выходит за рамки данного документа.

Лист регистрации изменений									
Номер листов (страниц)					Всего листов (страниц) в документе	№ документа	Входящий № сопроводительного документа и дата	Подп.	Дата
Изм	Изменен-ных	Заменен-ных	Новых	Анну-лиро-ванных					