МОСКОВСКОЕ ОТДЕЛЕНИЕ ПЕНЗЕНСКОГО
НАУЧНО-ИССЛЕДОВАТЕЛЬСКОГО ЭЛЕКТРОТЕХНИЧЕСКОГО ИНСТИТУТА

УТВЕРЖДЕН
ЯЦИТ.00020-03 90 01-ЛУ

СРЕДСТВО
КРИПТОГРАФИЧЕСКОЙ ЗАЩИТЫ ИНФОРМАЦИИ
“ВЕРБА-OW”
версия 6.1.2

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

Библиотеки

Инструкция по встраиванию

ЯЦИТ.00020-03 90 01

Листов 80

2008

Литера O_1

АННОТАЦИЯ

Данный документ содержит описание интерфейса библиотек WBOTHO и WSIGNO версии 6.1.2. Библиотеки предназначены для шифрования файлов и областей оперативной памяти (WBOTHO), формирования и проверки электронной цифровой подписи (WBOTHO, WSIGNO).

Библиотеки WBOTHO и WSIGNO входят в состав прикладного программного обеспечения (ППО) СКЗИ "Верба-OW", функционирующего в среде Windows 9x, Windows ME, Windows NT, Windows 2000, Windows XP, Windows 2003, Windows Vista, ОС Windows Server 2008 на ПЭВМ, совместимых с IBM PC с процессором Intel Pentium и выше.

Документ предназначен для системных программистов как руководство по встраиванию библиотек в различное прикладное программное обеспечение.

СОДЕРЖАНИЕ

1. ОСНОВНЫЕ ХАРАКТЕРИСТИКИ БИБЛИОТЕК ШИФРОВАНИЯ И ПОДПИСИ

Библиотеки WBOTHO, WSIGNO входят в состав ППО СКЗИ “Верба-OW”, функционирующего в среде Windows 98, Windows ME, Windows NT4.0, Windows 2000, Windows XP, Windows 2003, Windows Vista, ОС Windows Server 2008 на ПЭВМ, совместимых с IBM PC с процессором Intel Pentium и выше.

Алгоритм шифрования выполнен в соответствии с требованиями ГОСТ 28147-89 “СИСТЕМЫ ОБРАБОТКИ ИНФОРМАЦИИ. ЗАЩИТА КРИПТОГРАФИЧЕСКАЯ”.

Цифровая подпись выполнена в соответствии с требованиями ГОСТ Р 34.10-94 и ГОСТ Р 34.10-2001 “ИНФОРМАЦИОННАЯ ТЕХНОЛОГИЯ. КРИПТОГРАФИЧЕСКАЯ ЗАЩИТА ИНФОРМАЦИИ. СИСТЕМА ЭЛЕКТРОННОЙ ЦИФРОВОЙ ПОДПИСИ НА БАЗЕ АСИММЕТРИЧНОГО КРИПТОГРАФИЧЕСКОГО АЛГОРИТМА”.

Функция хэширования выполнена в соответствии с требованиями ГОСТ Р 34.11-94 “ИНФОРМАЦИОННАЯ ТЕХНОЛОГИЯ. КРИПТОГРАФИЧЕСКАЯ ЗАЩИТА ИНФОРМАЦИИ. ФУНКЦИЯ ХЭШИРОВАНИЯ”.

Размер файла при зашифровании округляется до кратного восьми в большую сторону и увеличивается на (51 + число получателей * 48) байт. Общий размер зашифрованного файла не должен превышать 4Гб (4294967295 байт).

Размер области памяти при зашифровании увеличивается на (37 + число получателей * 48) байт.

Размер файла или области памяти увеличивается на 98 байт при первой подписи и на 87 байта при каждой последующей подписи. Общий размер подписанного файла не должен превышать 2Гб (2147483647 байт).

Производительность СКЗИ "Верба-OW" на различных типах ПЭВМ представлена в документе ЯЦИТ.00020-02 30 01 "СКЗИ "Верба-OW". Формуляр".

Ключевая система для шифрования ориентирована на обеспечение возможности парно-выборочной связи каждого абонента сети друг с другом с использованием открытого распределения ключей.

Длина ключей шифрования: открытого при использовании алгоритма Диффи-Хеллмана на экспоненциальной логике—1024 бита, на эллиптических кривых — 512 бит; закрытого — 256 бит.

Длина ключей электронной цифровой подписи (ЭЦП): закрытый ключ – 256 бит; открытый ключ – 1024 бита для алгоритма ГОСТ Р 34.10-94 и 512 бит для алгоритма ГОСТ Р 34.10-2001.

Ключи для шифрования и формирования подписи хранятся на ключевых носителях — дискетах 3.5" , носителях (таблетках) Touch-Memory, смарт-картах, а также на USB-носителях (iKey 1000, iKey 3000, eToken R2, eToken Pro, ruToken, Шипка). При работе с СКЗИ используется два вида ключевых носителей:

- ключевой носитель для подписи — дискета, таблетка Touch-Memory, смарт-карта с ключами подписи, USB носители;
- совмещенный ключевой носитель — дискета, таблетка Touch-Memory, смарт-карта, USB носители с ключами шифрования и подписи одновременно.

Возможно хранение закрытых ключей шифрования и подписи на ЖМД.

Библиотеки ППО СКЗИ “Верба-OW” предоставляют пользователю следующие дополнительные возможности:

- файл может быть подписан 1-255 корреспондентами;
- файл или область памяти могут быть зашифрованы нескольким корреспондентам;
- пользователь (администратор) может корректировать справочники открытых ключей шифрования и подписи.

Развитая система контроля шифратора и обработка ошибочных ситуаций обеспечивают надежную работу библиотеки и анализ неисправностей.

2. ЗАЩИТА ИНФОРМАЦИИ С ПОМОЩЬЮ ШИФРОВАНИЯ И ЭЛЕКТРОННОЙ ЦИФРОВОЙ ПОДПИСИ

2.1 Защита данных с помощью криптографических преобразований

Защита данных с помощью криптографических преобразований — одно из возможных решений проблемы обеспечения безопасности информации.

СКЗИ "Верба-OW" реализует следующие криптографические преобразования: шифрование, имитозащита и электронно-цифровая подпись (ЭЦП).

Шифрование данных производится с целью скрыть содержание представляемой ими информации. Зашифрованные данные становятся доступными только для того, кто знает соответствующий ключ, с помощью которого можно расшифровать сообщение, и поэтому похищение зашифрованных данных без знания ключа является бессмысленным занятием.

Имитозащита обеспечивает надежное установление фактов случайного или преднамеренного искажения информации в процессе ее хранения или передачи по каналам связи.

Электронно-цифровая подпись подтверждает авторство и целостность электронной документации.

Криптография обеспечивает надежную защиту данных. Однако необходимо понимать, что ее применение не является абсолютным решением всех проблем защиты информации. Для эффективного решения проблемы защиты информации необходим целый комплекс мер, который включает в себя соответствующие организационно-технические и административные мероприятия, связанные с обеспечением правильности функционирования технических средств обработки и передачи информации, а также установление соответствующих правил для обслуживающего персонала, допущенного к работе с конфиденциальной информацией.

Основными компонентами криптографии являются данные, криптографическое преобразование и ключ:

- *данные* — при зашифровании исходными данными будет сообщение, а результирующими — зашифрованное сообщение. При расшифровании они меняются местами. Сообщения могут быть различных типов: текст, видеоизображение и т. п.
- *криптографическое преобразование* — под криптографическим преобразованием понимают преобразование данных при помощи алгоритма шифрования или выработки электронно-цифровой подписи. Считается, что криптографическое преобразование известно всем, но, не зная ключа, с помощью которого пользователь закрыл сообщение, практически невозможно восстановить содержание сообщения или подделать электронно-цифровую подпись.
- *ключ шифрования (ключ связи)* — конкретное секретное состояние некоторых параметров алгоритма криптографического преобразования данных. В СКЗИ "Верба-OW" в качестве ключей используются бинарные последовательности определенной длины.

Термин «шифрование» объединяет в себе два процесса: зашифрование и расшифрование информации.

Если зашифрование и расшифрование осуществляются с использованием одного и того же ключа, то такой алгоритм криптографического преобразования называется симметричным, в противном случае — асимметричным.

В СКЗИ «Верба-OW» используется симметричный алгоритм криптографического преобразования (см. подробнее 2.1.1).

Важно исключить доступ к ключам шифрования посторонних лиц, так как любой, кто обладает ключом шифрования, может прочитать зашифрованное Вами сообщение.

В СКЗИ «Верба-OW» используется механизм открытого распределения ключей, при котором для формирования ключа связи используется пара ключей: открытый и закрытый ключи шифрования (см. подробнее 2.1.2).

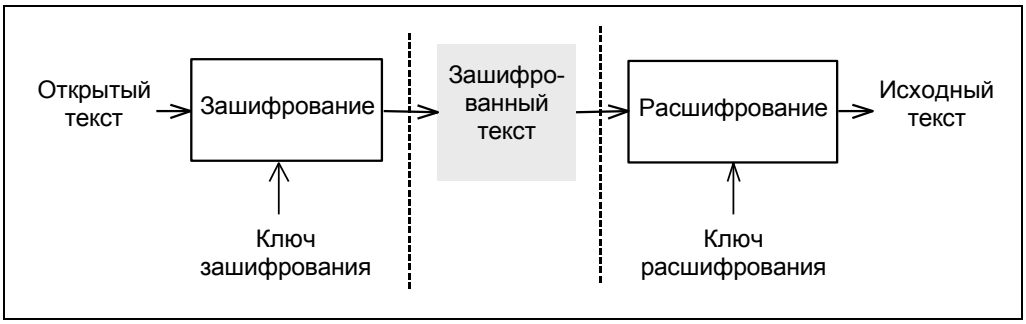

Рис. 1. Шифрование данных

2.1.1 Алгоритм криптографического преобразования

В СКЗИ «Верба-OW» используется симметричный алгоритм криптографического преобразования данных, определенный ГОСТ 28147-89. Данный алгоритм предназначен для аппаратной и программной реализации и удовлетворяет необходимым криптографическим требованиям.

ГОСТ 28147-89 предусматривает несколько режимов работы алгоритма криптографического преобразования. В СКЗИ «Верба-OW» используется алгоритм шифрования, основанный на принципе гаммирования, который подразумевает процесс наложения по определенному закону гаммы шифра на открытые данные (под гаммой понимается псевдослучайная двоичная последовательность, вырабатываемая по заданному алгоритму).

ГОСТ 28147-89 также определяет процесс выработки имитовставки. Имитовставка — это последовательность данных фиксированной длины, которая вырабатывается по определенному правилу из открытых данных и ключа шифрования. Выработка имитовставки обеспечивает защиту информации от случайных или преднамеренных искажений.

Имитовставка передается по каналу связи вместе с зашифрованным сообщением. Поступившие зашифрованные данные расшифровываются, и из полученных блоков данных вырабатывается контрольная имитовставка, которая затем сравнивается с имитовставкой, полученной из канала связи. В случае несовпадения имитовставок все расшифрованные данные считаются ложными.

2.1.2 Пара: открытый и закрытый ключи шифрования

В последнее время широкое распространение получили криптографические системы с открытым распределением ключей. В таких системах каждый пользователь формирует два ключа: открытый и закрытый. Закрытый ключ шифрования должен храниться в тайне. Открытый ключ шифрования не является секретным и может быть опубликован для использования всеми пользователями системы, которые обмениваются сообщениями. Знание открытого ключа шифрования не дает практической возможности определить закрытый ключ.

СКЗИ «Верба-OW» является системой с открытым распределением ключей. Каждый пользователь вырабатывает свой закрытый ключ, из которого затем с помощью некоторой процедуры формируется открытый ключ. Открытые ключи объединяются в справочник.

В СКЗИ «Верба-OW» ключ зашифрования совпадает с ключом расшифрования. При зашифровании сообщения i-ым абонентом для j-ого абонента общий секретный ключ связи вырабатывается на основе закрытого ключа шифрования i-ого абонента и открытого ключа шифрования j-ого абонента. Соответственно, для расшифрования этого сообщения j-ым абонентом формируется секретный ключ связи на основе закрытого ключа шифрования j-ого абонента и открытого ключа шифрования i-ого абонента. Таким образом, для обеспечения связи с другими абонентами каждому пользователю необходимо иметь:

- собственный закрытый ключ шифрования;
- открытые ключи шифрования пользователей сети конфиденциальной связи, объединенные в справочники.

Примем следующее соглашение. *Абонента, который зашифровывает сообщение, будем в дальнейшем называть отправителем; абонента, который расшифровывает закрытое сообщение — получателем.*

2.2 Электронная цифровая подпись

Электронная цифровая подпись — это средство, позволяющее на основе криптографических методов надежно установить авторство и подлинность электронного документа.

Электронная цифровая подпись позволяет заменить при безбумажном документообороте традиционную печать и подпись. При построении цифровой подписи вместо обычной связи между печатью или рукописной подписью и листом бумаги выступает сложная математическая зависимость между электронным документом, закрытым и открытым ключами. Практическая невозможность подделки электронной цифровой подписи опирается на очень большой объем определенных математических вычислений.

Проставление подписи под документом не меняет самого документа, она только дает возможность проверить подлинность и авторство полученной информации.

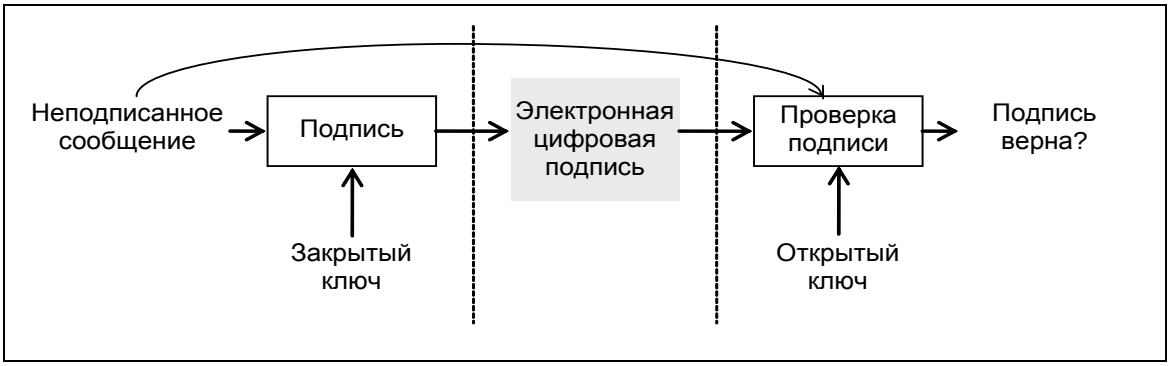

Рис. 2. Электронная цифровая подпись

В СКЗИ «Верба-OW» реализована система электронной цифровой подписи на базе криптографического алгоритма, соответствующего ГОСТ Р 34.10-94 и алгоритма ГОСТ Р 34.10-2001. Выбор алгоритма производится автоматически в зависимости от типа закрытых и открытых ключей.

Закрытый ключ подписи используется для выработки электронной цифровой подписи. Только сохранение пользователем в тайне своего закрытого ключа гарантирует невозможность подделки злоумышленником документа и цифровой подписи от имени заверяющего.

Открытый ключ подписи вычисляется, как значение некоторой функции от закрытого ключа, но знание открытого ключа не дает возможности определить закрытый ключ. Открытый ключ может быть опубликован и используется для проверки подлинности подписанного документа, а также для предупреждения мошенничества со стороны заверяющего в виде отказа его от подписи документа.

При работе с СКЗИ «Верба-OW» каждый пользователь, обладающий правом подписи, самостоятельно формирует личные закрытый и открытый ключи подписи. Открытые ключи подписи всех пользователей объединяются в справочники открытых ключей сети конфиденциальной связи.

Каждому пользователю, обладающему правом подписи, необходимо иметь:

- закрытый ключ подписи;
- справочник открытых ключей подписи пользователей сети.

Электронная цифровая подпись вырабатывается на основе электронного документа, требующего заверения, и закрытого ключа. Вначале производится «сжатие» документа с помощью функции хэширования (ГОСТ Р 34.11-94 «ИНФОРМАЦИОННАЯ ТЕХНОЛОГИЯ. КРИПТОГРАФИЧЕСКАЯ ЗАЩИТА ИНФОРМАЦИИ. ФУНКЦИЯ ХЭШИРОВАНИЯ»). Однонаправленная хэш-функция получает на входе исходное сообщение произвольной длины и преобразует его в хэш-значение фиксированной длины (256 бит согласно ГОСТ Р 34.11-94). Значение хэш-функции сложным образом зависит от содержания документа, но не позволяет восстановить сам документ. Хэш-функция чувствительна к всевозможным изменениям в тексте. Кроме того, для данной хэш-функции практически нельзя подобрать два исходных сообщения, которые могут иметь одно и то же хэш-значение или, что то же самое, одну и ту же цифровую подпись. Далее, к полученному хэш-значению применяется некоторое математическое преобразование, в результате которого и получается собственно цифровая подпись электронного документа.

При проверке подписи проверяющий должен располагать открытым ключом пользователя, поставившего подпись. Проверяющий должен быть полностью уверен в подлинности открытого ключа (а именно в том, что имеющийся у него открытый ключ соответствует открытому ключу конкретного

пользователя). Процедура проверки подписи состоит из вычисления хэш-значения документа и проверки некоторых соотношений, связывающих хэш-значение документа, подпись под этим документом и открытый ключ подписавшего пользователя. Документ считается подлинным, а подпись правильной, если эти соотношения выполняются. В противном случае подпись под документом считается недействительной.

Для разрешения споров между отправителем и получателем информации, связанных с возможностью искажения пересылаемого документа или открытого ключа проверки подписи, достоверная копия этого ключа может выдаваться третьей стороне (арбитру) и применяться им при возникновении конфликта между отправителем и получателем.

Для контроля целостности и подлинности справочников открытых ключей используется процедура выработки имитовставки, определяемая ГОСТ 28147-89.

3. ВАРИАНТЫ ПОСТАВОК БИБЛИОТЕК ШИФРОВАНИЯ И ПОДПИСИ

В зависимости от среды и потребностей пользователя возможны различные варианты поставок библиотек ППО СКЗИ “Верба-OW”.

Примечание. Состав различных поставляемых комплектаций СКЗИ "Верба-OW" описан в документе ЯЦИТ.00020-02 30 01 "СКЗИ "Верба-OW". Формуляр".

Все реализованные криптографические функции по назначению объединяются в две библиотеки:

- WBOTHO — библиотека, содержащая функции шифрования, выработки имитовставки, выработки случайного числа, формирования электронной цифровой подписи, проверки подписи и выработки значения хэш-функции, а также функции работы со справочниками открытых ключей;
- WSIGNO — библиотека, содержащая функции формирования электронной цифровой подписи, проверки подписи и выработки значения хэш-функции, а также функции работы со справочниками открытых ключей.

Дальнейшее описание интерфейса библиотек построено следующим образом. Назначение и состав библиотек указываются в зависимости от набора функций, реализованных в библиотеках, т. е. для WBOTHO, WSIGNO. Описание конкретных функций дается одновременно для всех библиотек.

4. ОСНОВНЫЕ ФУНКЦИИ БИБЛИОТЕК WBOTHO, WSIGNO

4.1 Основные функции библиотеки WBOTHO

Библиотека WBOTHO обеспечивает выполнение следующих функций:

- зашифрование/расшифрование информации на уровне файлов;
- зашифрование/расшифрование области памяти;
- формирование имитовставки для файлов и областей памяти;
- формирование случайного числа;
- подпись файла (ЭЦП добавляется в конец подписываемого файла или записывается в отдельном файле);
- подпись области памяти (ЭЦП добавляется в конец области памяти или сохраняется отдельно);
- проверку ЭЦП файла;
- проверку ЭЦП области памяти;
- удаление подписи;
- хэширование файлов и областей памяти.

Библиотека предоставляет возможность формирования и корректировки справочников открытых ключей шифрования и подписи. Библиотека WBOTHO обеспечивает выполнение следующих функций при работе со справочниками открытых ключей:

- добавление открытого ключа в справочник;
- удаление открытого ключа из справочника (по идентификатору или порядковому номеру);
- получение списка всех открытых ключей указанной серии;
- получение атрибутов открытого ключа по идентификатору и наоборот;
- проверка справочника открытых ключей;
- проверка открытого ключа;
- контроль целостности справочника открытых ключей.

4.2 Основные функции библиотеки WSIGNO

Библиотека WSIGNO обеспечивает выполнение следующих криптографических функций:

- подпись файла (ЭЦП добавляется в конец подписываемого файла или в отдельный файл);
- подпись области памяти (ЭЦП добавляется в конец области памяти или записывается в указанный блок памяти);
- проверку ЭЦП файла;
- проверку ЭЦП области памяти;
- удаление подписи;
- выработка хэш-значения для файла;
- хэширование файлов и областей памяти.

Библиотека предоставляет возможность формирования и корректировки справочников открытых ключей подписи. Библиотека WSIGNO обеспечивает выполнение следующих функций при работе со справочниками:

- добавление открытого ключа в справочник;
- удаление открытого ключа из справочника (по идентификатору или порядковому номеру);
- получение списка всех открытых ключей указанной серии;
- получение атрибутов открытого ключа по идентификатору и наоборот;
- проверка справочника открытых ключей;

- проверка открытого ключа;
- контроль целостности справочника открытых ключей.

5. ОБЩЕЕ ОПИСАНИЕ КЛЮЧЕВОЙ СИСТЕМЫ ППО СКЗИ “ВЕРБА-OW”

5.1 Идентификация абонента в сети обмена конфиденциальной информацией

Каждой ключевой серии, используемой в конфиденциальной связи, защищенной СКЗИ “Верба-OW”, присваивается номер SSSSSS. Размер приватной сети конфиденциальной связи не может превышать 10000 абонентов. СКЗИ “Верба-OW” поддерживает глобальную сеть конфиденциальной связи, которая содержит всех абонентов приватных сетей и абонентов, имеющих в своих идентификаторах номер серии в диапазоне от A00000 до ZZZZZZ.

Абоненты внутри глобальной сети различаются по номерам вида XXXXSSSSSSYY, внутри отдельной подсети — по номерам вида XXXXYY. Номера внутри отдельной сети распределяются Центром управления ключевой системы (ЦУКС). Составляющая XXXX определяет номер ключевого носителя для шифрования и может принимать значение от 0000 до 9999, номер YY называется личным кодом и может принимать значения от 00 до 99.

Таким образом, пользователи, обладающие правом шифрования, идентифицируются внутри подсети по номерам вида XXXX, где XXXX — номер ключевого носителя для шифрования; пользователи, обладающие правом подписи — по номерам вида XXXXYY, где XXXX — номер ключевого носителя для шифрования, YY — личный код.

5.2 Ключевая информация

Дискета, на которой хранятся ключи, называется ключевой дискетой (ключевым диском). Ключи на ключевой дискете хранятся в файлах с фиксированными именами.

При работе ключевой диск, как любой гибкий диск, может быть поврежден. Во избежание потери ключевой информации рекомендуется хранить рабочую копию ключевой дискеты. Ключи и их копии изготавливаются администраторами с использованием АРМ АБ-О.

5.3 Ключевая информация на гибком ключевом диске

На дискете каждый ключ хранится в отдельном файле с фиксированным именем. При дальнейшем описании названия ключей и имена файлов будут определять один и тот же объект.

При работе библиотеки используют следующие ключевые файлы:

Таблица 1. Ключевые файлы, используемые библиотеками

Библиотека	**WBOTHO**	**WSIGNO**
Ключевые файлы	GK.DB3 UZ.DB3 KS KS_XXXX CKD CKDI VERSION.HEX SECRET.KEY SEC_XXXX.KEY XXXX.PUB XXXXYY.FSG XXXXYY.HSG XXXXYY.LFX	GK.DB3 UZ.DB3 CKDI VERSION.HEX XXXXYY.FSG XXXXYY.HSG XXXXYY.LFX

Кратко поясним назначение используемых ключей:

- главный ключ GK.DB3 — служит для инициализации датчика случайных чисел и для шифрования ключей CKD и CKDI;
- узел замены UZ.DB3 — долговременный ключевой элемент;
- закрытый ключ шифрования SECRET.KEY — хранится на ключевом носителе и служит для формирования ключа связи для шифрования сообщения;
- *индивидуальный ключ* шифрования закрытого ключа шифрования CKD - служит для хранения закрытого ключа шифрования на жестком диске, выработки имитовставок на открытые ключи из справочника открытых ключей шифрования;
- закрытый ключ шифрования, подготовленный к хранению на жестком диске SEC_XXXX.KEY — выполняет функции закрытого ключа шифрования и хранится на жестком диске;
- открытый ключ шифрования XXXX.PUB — служит для формирования ключа связи для шифрования открытых данных;
- закрытый ключ подписи XXXXYY.FSG — хранится на ключевом носителе и служит для формирования электронной подписи документа;
- *индивидуальный ключ* шифрования закрытого ключа подписи CKDI — служит для хранения закрытого ключа подписи на жестком диске, выработки имитовставок на открытые ключи подписи из справочника открытых ключей подписи;
- закрытый ключ подписи, подготовленный к хранению на жестком диске XXXXYY.HSG — выполняет функции закрытого ключа подписи и хранится на жестком диске;
- открытый ключ подписи XXXXYY.LFX — служит для проверки электронной цифровой подписи;
- сетевой ключ KS — хранится на ключевом носителе, является общим для ключей одной серии и используется в процессе шифрования;
- сетевой ключ KS_XXXX — хранится на жестком диске. Представляет собой ключ KS, зашифрованный на ключе CKD;
- файл VERSION.HEX — содержит номер версии ключевого носителя.

5.4 СТРУКТУРА КЛЮЧЕВЫХ ДИСКОВ

5.4.1 Структура ключевого диска для подписи

Ключевой диск для подписи имеет следующую структуру:

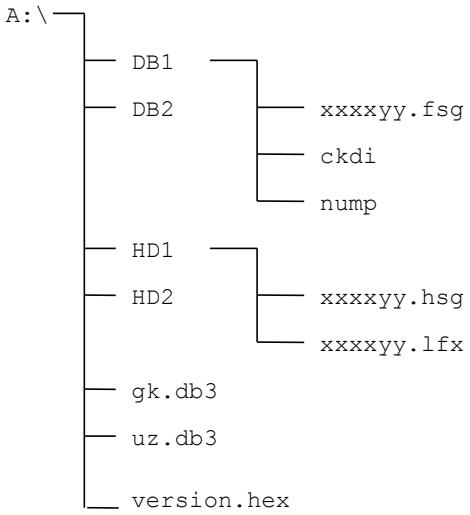

Рис. 3. Структура ключевых данных диска для подписи

Директории DB1 и DB2, а также директории HD1 и HD2 полностью дублируют друг друга. В директории HD1 (HD2) размещаются ключи, предназначенные для хранения на жестком диске. Файл NUMP содержит строку с полным номером ключа подписи в формате XXXXSSSSSSYY.

5.4.2 Структура совмещенного ключевого диска

На совмещенном ключевом диске хранятся ключи шифрования и подписи. Совмещенный ключевой диск имеет структуру:

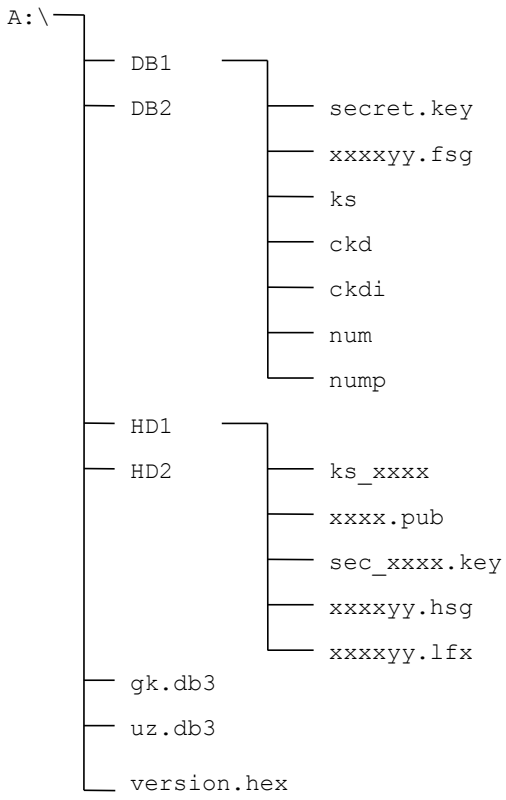

Рис. 4. Структура ключевых данных совмещенного диска

5.5 Использование ключевых носителей Touch-Memory, смарт-карт, USB ключей

5.5.1 Виды ключевых носителей Touch-Memory, смарт-карт, USB ключей используемых СКЗИ “Верба-OW”

Данная библиотека допускает работу с ключевыми носителями типа Touch-Memory, смарт-карта и USB ключами типа iKey и eToken. Перечень имен допустимых ключевых устройств (считывателей ключевых носителей) приведен в файле KEY_DEV.H.

5.5.2 Структура ключевых данных на Touch-Memory и смарт-карте

На совмещенном ключевом носителе Touch-Memory (смарт-карте, USB ключах) хранятся номера закрытых ключей шифрования и подписи (соответствуют файлам NUM и NUMP) и следующие ключи:

- главный ключ;
- узел замены;
- закрытый ключ шифрования;
- закрытый ключ подписи;
- *индивидуальный ключ* шифрования закрытого ключа шифрования;
- *индивидуальный ключ* шифрования закрытого ключа подписи;
- сетевой ключ.

Данные на ключевых носителях Touch-Memory, смарт-картах и USB ключах хранятся в последовательном формате.

5.6 Хранение закрытых ключей на жестком диске

СКЗИ “Верба-OW” предусматривает возможность работы библиотечных функций с ключами, хранящимися на жестком диске. Закрытый ключ шифрования и закрытый ключ подписи хранятся на жестком диске в защищенном виде. Закрытые ключи зашифровываются на специальных, так называемых *индивидуальных ключах* (CKD — для ключей шифрования, CKDI — для ключей подписи), которые в свою очередь зашифровываются на главном ключе (GK.DB3). Подготовленные к хранению на жестком диске закрытые ключи (XXXX.HSG , KS_XXXX и SEC_XXXX.KEY) копируются с ключевого диска на жесткий диск в каталог, имя которого должно удовлетворять правилам указания пути к ключам (см. 5.8). Для получения доступа к закрытым ключам на ЖМД необходимо прогрузить в память (драйвер ASYNCR) *индивидуальные ключи* с помощью программ ASRKEYW.EXE.

Ключевая система СКЗИ "Верба-OW" позволяет также работать с группами из нескольких ключей шифрования на ЖМД, зашифрованными на одном ключе CKD называемом индивидуальным ключом администратора. В этом случае для получения доступа ко всем ключам группы достаточно загрузить индивидуальный ключ администратора. Для формирования групп ключей SEC_XXXX.KEY, зашифрованных на ключе администратора, используются специальные опции программы АРМ АБ-О.

Хранение ключей на жестком диске обеспечивает более быструю работу функций библиотеки при большом потоке обрабатываемых сообщений. Так же допускается загрузка нескольких *индивидуальных ключей* в драйвер ASYNCR (более подробно см. раздел 7 и п. 9.4 "Дополнительные функции для обеспечения доступа к закрытым ключам на ЖМД...").

5.7 Открытые ключи и справочники

Открытый ключ можно использовать только в том случае, если достоверно известны его подлинность и авторство, подтверждаемые сертификатом (здесь: сертификат — бумажный бланк, заверенный печатью и подписью ЦУКС, на котором распечатаны открытый ключ и реквизиты пользователя). Важно защищать открытые ключи от искажений и подделок.

Открытые ключи шифрования и подписи хранятся в справочниках открытых ключей шифрования и подписи, соответственно. Открытые ключи представлены в виде записей, в которых указываются номер ключа, данные об абоненте, тип ключа (действующий, резервный, скомпрометированный) и сам ключ.

На рабочих местах СКЗИ “Верба-OW” с одним и тем же справочником могут работать несколько пользователей с независимой проверкой целостности справочника. Для контроля целостности и подлинности справочников используется процедура выработки имитовставки. Для каждого пользователя создается свой файл с имитовставками, имя которого совпадает с номером закрытого ключа (шифрования или подписи). Имитовставки создаются для каждого открытого ключа из справочника и записываются последовательно в файл с имитовставками. Справочник при этом не меняется. Для справочников открытых ключей шифрования и подписи имитовстакви вырабатываются на ключах CKD и CKDI соответственно. При выполнении функций, требующих наличие справочников происходит автоматическая проверка имитовставки на каждый используемый открытый ключ. Ключи для проверки справочников должны быть загружены в слот 0 драйвера ASYNCR.

Справочник с открытыми ключами шифрования хранится в файле LPUB.SPR, имитовставки на каждый открытый ключ шифрования — в файлах XXXX.IMP, справочник с открытыми ключами подписи хранится в файле LFAX.SPR, имитовставки на каждый открытый ключ подписи — в файлах XXXXYY.IMM.

5.8 Правила указания пути к ключам и справочникам открытых ключей

Для указания пути к закрытым ключам шифрования и подписи, хранящимся на ключевом ГМД, строка полного пути должна иметь вид: "A:\\" или "B:\\", в зависимости от того, в какой дисковод установлена ключевая дискета. Полный путь ("A:\\DB1\\" или "B:\\DB1\\") достроится автоматически.

Для работы с ключевыми носителями на Touch-Memory, смарт-картах или USB ключах необходимо использовать предопределенные строки, описанные в файле KEY_DEV.H (Например, если ключи шифрования и/или подписи берутся с носителя Touch-Memory, поддерживаемого контроллером "Аккорд" модификации 4+, то строка полного пути должна иметь вид: "TM:A4.P").

Закрытые ключи, предназначенные для хранения на жестком диске должны быть переписаны из каталога HD1 на ключевом ГМД в каталог SSSSSS на ЖМД (SSSSSS - номер серии закрытого ключа). Для указания пути к закрытым ключам, хранящимся на жестком диске используется путь к каталогу SSSSSS. Например, если закрытые ключи хранятся в каталоге C:\VERBA\KEY\SSSSSS, то строка полного пути должна иметь вид: "C:\\VERBA\\KEY\\".

Справочники открытых ключей должны находиться в подкаталогах OPENKEY\SSSSSS\ для ключей шифрования и FAXKEY\SSSSSS\ для ключей подписи (SSSSSS - номер серии открытых ключей). Для указания пути к справочнику используется путь к каталогу FAXKEY (OPENKEY). Например, для справочника открытых ключей шифрования серии SSSSSS, находящегося в каталоге C:\VERBA\OPENKEY\SSSSSS\, необходимо указать путь "C:\\VERBA\\"

6. СОСТАВ БИБЛИОТЕК ШИФРОВАНИЯ И ПОДПИСИ

6.1 СОСТАВ БИБЛИОТЕК

Состав библиотек описан в следующей таблице:

Таблица 2. Состав библиотек шифрования и подписи

Библиотека	Состав	Комментарии
библиотека WSIGNO		
WSIGNO	WSIGNO.DLL	Динамическая библиотека.
	WSIGNO.LIB	Библиотека импорта (Microsoft Visual C++ 6.0).
	VERBA.H	Заголовочный файл с описанием функций и типов данных.
	KEY_DEV.H	Предопределенные имена для работы с ключевыми носителями на Touch-Memory и смарт-картах.
	ASRKEYW.EXE	Программа, реализующая операции чтения и загрузки ключей с ключевых носителей в драйвер ASYNCR.
	ASYNCR.VXD (ASYNCR.SYS)	Драйвер хранения ключей для Windows 95/9x (Windows NT), реализует функции хранения ключевой информации и программного датчика случайных чисел.
	ASRDLL.DLL	Динамическая библиотека интерфейса драйвера хранения ключей.
		Динамические библиотека для работы со сменными ключевыми носителями .
библиотека WBOTHO		
WBOTHO	WBOTHO.DLL	Динамическая библиотека.
	WBOTHO.LIB	Библиотека импорта (Microsoft Visual C++ 6.0).
	VERBA.H	Заголовочный файл с описанием функций и типов данных.
	KEY_DEV.H	Предопределенные имена для работы с ключевыми носителями на Touch-Memory и смарт-картах.
	ASRKEYW.EXE	Программа, реализующая операции чтения и загрузки ключей с ключевых носителей в драйвер ASYNCR.
	ASYNCR.VXD (ASYNCR.SYS)	Драйвер хранения ключей для Windows 95/9x (Windows NT), реализует функции хранения ключевой информации и инициализации программного датчика случайных чисел.
	ASRDLL.DLL	Динамическая библиотека интерфейса драйвера хранения ключей.
		Динамические библиотека для работы со сменными ключевыми носителями .

6.2 ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ДОСТУПА К КЛЮЧЕВОЙ ИНФОРМАЦИИ

6.2.1 Программа ASRKEYW - программа установки датчика случайных чисел и доступа к ключам

Программа **AsrkeyW.exe** предназначена для загрузки и выгрузки ключей в драйвер **Asyncr** с ключевых носителей: дискет, таблеток Touch-Memory, смарт-карт и инициализации датчика случайных чисел.

Для работы программы необходимо наличие установленных библиотек СКЗИ "Верба-OW". (Программа загружает необходимые ей библиотечные модули из *системного каталога* Windows, куда их помещает инсталлятор).

При первом (после загрузки ОС) запуска программы, высвечивается окно, в котором пользователю предлагается инициализировать датчик случайных чисел путем перемещения мыши в окне программы или нажатия на произвольные клавиши. Процесс инициализации отображается на экране (Рис. 5).

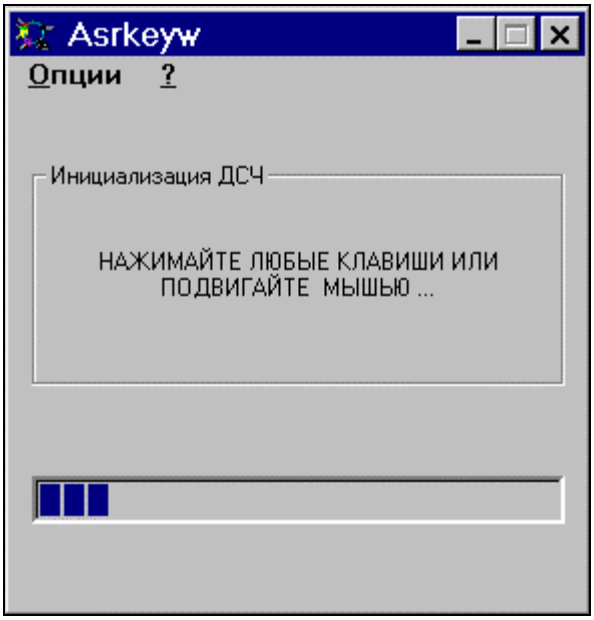

Рис. 5. Инициализация ДСЧ

Примечание. В пункте меню **Опции\Автоинициализация ДСЧ** имеется возможность устанавливать режим автоматический инициализации ДСЧ через программно-аппаратный комплекс (АПК) Аккорд 4+. Окно инициализации ДСЧ (Рис. 5) при этом высвечиваться не будет.

После выполнения описанных выше действий пользователь должен завершить инициализацию ДСЧ. Для этого необходимо выбрать устройство для чтения ключей, установить ключевой носитель и нажать кнопку "**Загрузить**" (Рис. 6).

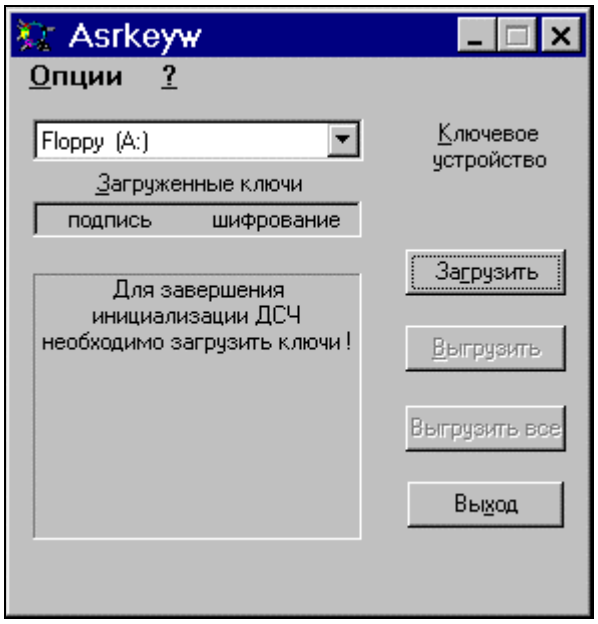

Рис. 6. Загрузка ключей

Примечание. Завершить инициализацию ДСЧ можно также, воспользовавшись библиотечной функцией **InitKey** (см. п. 9.4.2).

В меню программы представлены пункты "**Опции**" и "**?**".

В пункте "**?**" можно получить информацию о программе и инсталлированной СКЗИ.

В пункте "**Опции**" предлагаются следующие возможности:

— **Обновление экрана** - обновление окна со списком загруженных ключей;

— **Автоинициализация ДСЧ** – инициализация ДСЧ с использованием случайной последовательности считанной с АПК Аккорд 4+.

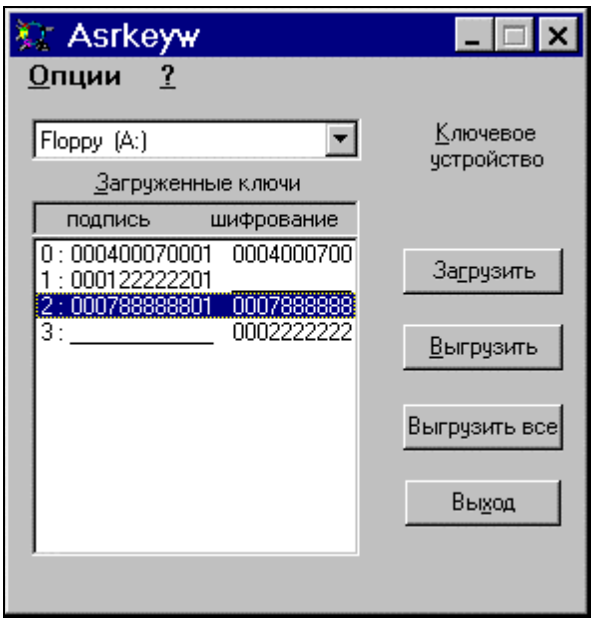

Рис. 7. Отображение загруженных ключей

Кнопка "**Загрузить**" загружает ключи с выбранного ключевого устройства в первый свободный слот драйвера **Asyncr** (см. п. 9.4.2). Свободные слоты ищутся по возрастанию номеров от 0 до 15. Загружаться могут ключевые носители для шифрования, подписи и совмещенные. Номера слотов и идентификаторы загруженных ключей отображаются на экране (см. Рис. 7).

Кнопка "**Выгрузить**" выгружает всю информацию из выделенного слота.

Кнопка "**Выгрузить все**" выгружает ключи из всех слотов драйвера Asyncr.

Кнопка "**Выход**" завершает работу программы.

Запуск программы **AsrkeyW.exe** может производиться с параметрами. Например: **AsrkeyW.exe param**, где **param** может быть одним из символов: I, U, R или S. Коды возврата программы соответствуют кодам, возвращаемым библиотечными функциями СКЗИ "Верба-OW".

Внимание. При запуске программы с параметрами вывод диалоговых окон производится только при необходимости "ручной" инициализации ДСЧ, иначе на дисплей ничего не выводится.

Список параметров программы **AsrkeyW.exe**:

I — загрузка ключа в слот 0. Если слот уже загружен, то код возврата равен E_REDEFINE;

U — загрузка ключа в слот 0 с предварительной очисткой слота;

R — выгрузка ключа из слота 0;

S — выполняется только инициализация ДСЧ без загрузки ключа.

7. ВСТРАИВАНИЕ БИБЛИОТЕК

Для того чтобы использовать криптографические функции библиотек шифрования и подписи, необходимо предварительно:

- подготовить необходимую ключевую информацию;
- установить программное обеспечение хранения и доступа к ключевым структурам.

7.1 КАТАЛОГ С СКЗИ

Для простоты и удобства использования СКЗИ “Верба-OW” рекомендуется создать каталог с СКЗИ, в котором будут храниться файлы ППО СКЗИ “Верба-OW”. Каталог с СКЗИ может иметь следующую структуру: каталог со справочниками открытых ключей, каталог с закрытыми ключами, подготовленными для хранения на жестком диске, модули, входящие в состав СКЗИ “Верба-OW”:

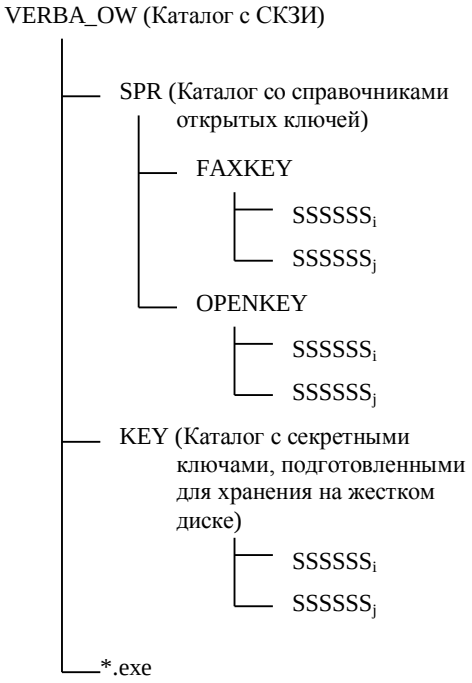

Рис. 8. Возможная структура каталога с СКЗИ

7.2 ПОДГОТОВКА КЛЮЧЕВОЙ ИНФОРМАЦИИ

7.2.1 Каталог со справочниками открытых ключей

Если Вы формируете справочник самостоятельно, создайте подкаталог в каталоге с СКЗИ, в котором Вы будете хранить справочники открытых ключей шифрования и (или) подписи (см. 5.7, 9). Формируйте справочник, добавляя записи открытых ключей с помощью функций библиотек (см. 9).

7.2.2 Каталог для хранения закрытых ключей

В каталоге с СКЗИ создайте подкаталог для закрытых ключей, подготовленных для хранения на жестком диске. В подкаталоге для хранения закрытых ключей создайте подкаталоги с именами, совпадающими с номерами серий закрытых ключей (SSSSSS). Скопируйте в подкаталог SSSSSS содержимое директории HD1 с соответствующего гибкого ключевого диска.

7.3 Установка ПО для хранения и доступа к ключевой информации

7.3.1 библиотеки WSIGNO и WBOTHO для Windows 9x и Windows ME

ПО для хранения и организации доступа к ключевой информации состоит из программы AsrKeyW, драйвера ASYNCR.VXD (с интерфейсной библиотекой ASRDLL.DLL) и вспомогательными библиотеками и драйверами для работы с ключевыми носителями типа Touch-Memory и смарт-карта. Установка всего ПО осуществляется автоматически при инсталляции СКЗИ. При этом .exe и .dll модули, входящие в состав СКЗИ записываются в системный каталог Windows 9x/ME (System). В тот же каталог помещается драйвер ASYNCR.VXD

7.3.2 библиотека WSIGNO и WBOTHO для Windows NT, Windows 2000, Windws XP, Windows 2003, Windows Vista, ОС Windows Server 2008

ПО для хранения и организации доступа к ключевой информации состоит из программы AsrKeyW, драйвера ASYNCR.SYS (с интерфейсной библиотекой ASRDLL.DLL) и вспомогательными библиотеками и драйверами для работы с ключевыми носителями типа Touch-Memory и смарт-карта. Установка всего ПО осуществляется автоматически при инсталляции СКЗИ. При этом .exe и .dll модули, входящие в состав СКЗИ записываются в системный каталог Windows NT(2000,XP,2003,Vista, ОС Windows Server 2008) (System32). Драйвер ASYNCR.SYS помещается в подкаталог System32\Drivers основного каталога Windows NT.

7.4 Заголовочные файлы (Include)

Для использования функций библиотек СКЗИ в прикладном программном обеспечении, написанном на C или C++, в исходный текст ПО должны быть включены заголовочные файлы (*.H) в которых описаны библиотечные функции и типы используемых данных. Например:

```
#include    "VERBA.H"
#include    "VERB_GEN.H"
#include    "KEY_DEV.H"
```

В проект пользовательского ПО может включаться соответствующая библиотека импорта обеспечивающая автоматическое подключение динамической библиотеки при загрузке пользовательского приложения.

7.5 Порядок запуска программ, использующих библиотеки

Перед работой с библиотеками для инициализации ДСЧ и загрузки *индивидуальных ключей* необходимо запустить программу Asrkeyw.exe и выполнить предлагаемые программой действия. Программа позволяет работать с ключевыми носителями на ГМД, Touch-memory и смарт-картах. Программа позволяет также производить выгрузку ключей и получать справочную о загруженных ключах.

Внимание. Если *индивидуальные ключи* уже считаны в память, то использовать криптографические функции можно, не имея гибкого ключевого диска. Поэтому для защиты от несанкционированного использования СКЗИ после окончания сеанса работы необходимо выгрузить ключи из памяти драйвера, используя программу Asrkeyw или библиотечные функции. Кроме этого рекомендуется после окончания сеанса работы в Windows NT перезагрузить систему или остановить драйвер ASYNCR (например, командой **net stop asyncr**), а в Windows 95 перезагрузить систему.

8. ОПИСАНИЕ КРИПТОГРАФИЧЕСКИХ ФУНКЦИЙ БИБЛИОТЕК ШИФРОВАНИЯ И ПОДПИСИ

Все описанные ниже функции, входят в библиотеку шифрования и подписи (WBOTHO). В библиотеку подписи (WSIGNO) входят функции, описанные в разделах *Загрузка и удаление ключа подписи из оперативной памяти*, *Подпись и проверка подписи файла*, *Подпись и проверка подписи блока памяти*, *Хэширование файлов и областей памяти*, а также функции инициализации и завершения работы с библиотекой подписи. Функции обработки ошибочных ситуаций и освобождения памяти, используемой при работе СКЗИ, включены во все библиотеки (WBOTHO и WSIGNO). Кроме того, в описании каждой функции явно указывается, в какую из библиотек она входит: (**wbotho**), (**wsigno**) или (**wbotho, wsigno**).

При описании параметров функций библиотек используется следующее соглашение: перед каждым из них стоит находится один из знаков (**i**), (**o**) или (**i/o**), указывающий на семантическое значение данного параметра.

(**i**) — обозначает, что через данный параметр передаются “входные” данные для функции. Например, (**i**)-параметр может быть числом или указателем на блок памяти, который содержит некоторые исходные данные.

(**o**) — обозначает, что через данный параметр осуществляется доступ к результату работы функции, т. е. к ее “выходным” данным. Обычно (o)-параметр — это указатель на область памяти, в которую помещаются вычисленные функцией данные.

(**i/o**) — обозначает комбинацию (**i**) и (**o**).

8.1 ДИАГНОСТИКА ОШИБОЧНЫХ СИТУАЦИЙ

Для диагностики ошибочных ситуаций функции библиотеки возвращают коды возврата. Полный список кодов ошибок приведен в Приложении 1. Также коды ошибок описаны в заголовочном файле verba.h. Функция возвращает нулевой код ошибки, если она успешно завершила работу. Любое ненулевое значение предполагает возникновение ошибочной ситуации, которая должна быть обработана особо.

Пример обработки возвращаемого значения:

```
err_code = CryptoInit(key_path, open_key_path);
switch(err_code)
 {
 case NO_ERROR : break;
 case E_DRIVE : { printf("НЕ ЗАГРУЖЕН ДРАЙВЕР"); return 1;}
 case E_CONTROL : exit(1);
 default : return 1;
 }
```

8.2 ФУНКЦИИ ИНИЦИАЛИЗАЦИИ И ЗАВЕРШЕНИЯ РАБОТЫ С БИБЛИОТЕКОЙ

Перед использованием функций зашифрования/расшифрования или формирования ЭЦП, необходимо выполнить функцию инициализации. Для этого в самом начале программы, использующей библиотеку шифрования и подписи, либо библиотеку подписи, надо вызвать функции для инициализации шифрования и (или) подписи, соответственно: CryptoInit(**wbotho**) и SignInit(**wbotho, wsigno**). Они проверяют работоспособность функций библиотеки, загрузку датчика случайных чисел, устанавливают пути к справочникам открытых ключей и указывают, откуда считывать закрытые ключи. Их прототипы выглядят следующим образом:

```
T16bit WINAPI CryptoInit (char *pathToSecret,
                          char *pathToBase);

T16bit WINAPI SignInit (char *pathToSecret,
                        char *pathToBase);
```

Функции имеют параметры:

(i) pathToSecret — указатель на строку полного пути к закрытым ключам шифрования и подписи;

(i) pathToBase — указатель на строку полного пути к структуре со справочниками открытых ключей шифрования или подписи.

Замечание. Функции инициализации не проверяют корректность задания путей. Соответствующие проверки выполняют функции, непосредственно работающие с ключевой информацией.

Например:

```
error_code = CryptoInit("C:\\VERBA_OW\\KEY\\", "C:\\VERBA_OW\\SPR\\");
if(error_code != NO_ERROR) return(1);
error_code = SignInit("C:\\VERBA_OW\\KEY\\", "C:\\VERBA_OW\\SPR\\");
if(error_code) return(1);
```

Внимание. Если функция инициализации в качестве кода возврата вернула любое ненулевое значение, использовать криптографические функции зашифрования/расшифрования или формирования ЭЦП нельзя.

В случае, когда вызывались функции SignInit или CryptoInit по окончании работы необходимо вызвать функции: CryptoDone(**wbotho**) и (или) SignDone(**wbotho, wsigno**). Функции имеют прототипы:

```
T16bit WINAPI CryptoDone (void);

T16bit WINAPI SignDone (void);
```

Пример обращения к функциям завершения работы с библиотекой:

```
if (CryptoDone()) return 1;
if(SignDone()) return 1;
```

8.3 Шифрование файлов

8.3.1 Зашифрование файлов

Для зашифрования файлов служит функция EnCryptFile(**wbotho**). (Рекомендуется вместо этой функции использовать функцию EnCryptFileEx).

```
T16bit WINAPI EnCryptFile (char *file_in,
                           char *file_out,
                           T16bit node_From,
                           P16bit node_To,
                           char *ser);
```

Функция имеет параметры:

(i) file_in — указатель на строку полного пути к незашифрованному файлу;

(i) file_out — указатель на строку полного пути к зашифрованному файлу;

Внимание. Размер файла при зашифровании округляется до кратного восьми в большую сторону, а затем увеличивается на (51 + число получателей * 48) байт.

(**i**) node_From — номер ключа шифрования отправителя (XXXX), представленный в виде числа типа unsigned short;

(**i**) node_To — список номеров ключей получателей, который представляется строкой элементов, имеющих тип unsigned short, и оканчивается нулем (0);

(**i**) ser — указатель на строку длиной 6 символов с номером серии (SSSSSS) открытых ключей шифрования получателей. Если в качестве значения этого параметра указать последовательность из шести нулевых символов (\0\0\0\0\0\0), т. е. зашифрованный файл предназначен для пользователей с тем же номером серии, что и у отправителя, то номер серии берется из драйвера хранения ключевой информации.

Пример вызова функции зашифрования:

```
char   ser_To[7] = "000005";
node_To[0] = 2;
node_To[1] = 3;
node_To[2] = 0;
node_From = 1;
error_code = EnCryptFile("C:\\AUTOEXEC.BAT", "C:\\TEST.TST",
      node_From, node_To , ser_To);
```

8.3.2 Расшифрование файлов

Для расшифрования файлов служит функция DeCryptFile(**wbotho**). (Рекомендуется вместо этой функции использовать функцию DeCryptFileEx).

```
T16bit WINAPI DeCryptFile (char *file_in,
            char *file_out,
            T16bit abonent);
```

Функция имеет параметры:

(**i**) file_in — указатель на строку полного пути к зашифрованному файлу;

(**i**) file_out — указатель на строку полного пути к расшифрованному файлу;

(**i**) abonent — номер ключа шифрования получателя (XXXX), представленный в виде числа типа unsigned short.

Пример вызова функции расшифрования:

```
error_code = DeCryptFile("C:\\TEST.TST","C:\\AUTOEXEC.BAT", 1);
```

8.3.3 Получение идентификатора ключа абонента, зашифровавшего файл

Для получения идентификатора ключа абонента, зашифровавшего файл, служит функция GetFileSenderID(**wbotho**). Она имеет прототип:

```
T16bit WINAPI GetFileSenderID (char *path, char *sender_id);
```

Функция имеет параметры:

(**i**) path — указатель на строку полного пути к зашифрованному файлу;

(**o**) sender_id — указатель на строку длиной 11 байт, в которую записывается идентификатор зашифровавшего файл абонента в виде XXXXSSSSSS.

Пример вызова функции расшифрования:

```
char my_ID[11];
memset(my_ID, 0, 11);
err_code=GetFileSenderID("tst.tst", my_ID);
```

8.3.4 Получение списка получателей зашифрованного файла

Для получения списка получателей (абонентов, которые могут расшифровать данный закрытый файл), служит функция GetCryptKeysF(**wbotho**). При формировании списка получателей зашифрованного файла расшифрование и проверка правильности этого файла не производятся.

Функция GetCryptKeysF имеет прототип:

```
T16bit WINAPI GetCryptKeysF (char *file_name,
            P16bit abonents,
            P16bit *user_list,
            char *ser);
```

Функция имеет параметры:

(**i**) file_name — указатель на строку полного пути к зашифрованному файлу;

(**o**) abonents — указатель на переменную, в которую возвращается число получателей;

(**o**) user_list — список получателей файла file_name, который имеет такую же структуру, как и в функции EnCryptFile, т. е. представлен строкой элементов, имеющих тип unsigned short, и оканчивающийся нулем (0).

Внимание. Память под этот массив отводится внутри функции GetCryptKeysF и должна быть освобождена функцией FreeMemory (см. п. 8.12), как только user_list обработан.

(**o**) ser — указатель на строку, куда возвращается номер серии (7 байт) ключей отправителей.

Пример использования функции GetCryptKeysF:

```
char ser[7];
error_code = GetCryptKeysF("C:\\TEST.TST", abonents, user_list, ser);
for(i = 0; i < *abonents; i++) printf("%u\n", *user_list[i]);
FreeMemory(user_list);
```

8.3.5 Зашифрование файлов (расширенное)

Функция EnCryptFileEx(**wbotho**) служит для зашифрования файлов. Функция не обращается к справочникам открытых ключей, также допускается шифрование между абонентами из различных ключевых серий . Функция имеет следующий прототип:

```
T16bit WINAPI EnCryptFileEx (char *file_in,
            char *file_out,
            char * From,
            void **open_keys_array,
            unsigned short open_keys_quantity,

            T32bit   flags);
```

Функция имеет параметры:

(**i**) file_in — указатель на строку полного пути к незашифрованному файлу;

(**i**) file_out — указатель на строку полного пути к зашифрованному файлу;

Внимание. Размер файла при зашифровании округляется до кратного восьми в большую сторону, а затем увеличивается на (51 + число получателей * 48) байт.

(i) From — указатель на строку с идентификатором ключа отправителя в формате XXXXSSSSSS;

(i) open_keys_array — указатель на массив указателей на открытые ключи получателей;

(i) open_keys_quantity — количество получателей (количество открытых ключей).

(i) flags — зарезервированный параметр.

Пример вызова функции зашифрования:

```
…
char   From[11] = "0001910000";
static char key1_91[304];
static char key1_92[304];
void   *open_keys_array[]={key1_91,key1_92};
…
err_code=ExtractKey(CRYPT_SPR,"0001910000",key1_91);
if(err_code) goto error;
err_code=ExtractKey(CRYPT_SPR,"0001920000",key1_92);
if(err_code) goto error;
…
err_code=EnCryptFileEx ("C:\\test.org","C:\\test.cry",From
,open_keys_array, 2,0);
…
```

8.3.6 Расшифрование файлов (расширенное)

Функция DeCryptFileEx(**wbotho**) служит для расшифрования файлов . Функция не обращается к справочникам открытых ключей, также функция позволяет расшифровать файл и в случае если ключевые серии отправителя и получателя различны . Функция имеет следующий прототип :

```
T16bit WINAPI DeCryptFileEx (char *file_in,
            char *file_out,
            char *abonent,
            void *pub_key);
```

Функция имеет параметры:

(i) file_in — указатель на строку полного пути к зашифрованному файлу;

(i) file_out — указатель на строку полного пути к расшифрованному файлу;

(i) abonent — указатель на строку с идентификатором ключа получателя в формате XXXXSSSSSS;

(i) pub_key — указатель на буфер с открытым ключом отправителя.

Пример вызова функции расшифрования:

```
…
char   abonent[] = "0001920000";
char   sender_ID[11];
char   sender_key[304];
…
```

```
err_code=GetFileSenderID("C:\\test.cry",sender_ID);
if(err_code) goto error;
err_code=ExtractKey(CRYPT_SPR,sender_ID,sender_key);
if(err_code) goto error;
…
err_code=DeCryptFileEx ("C:\\test.cry","C:\\test",abonent,sender_key);
…
```

8.4 Шифрование блока памяти

8.4.1 Зашифрование блока памяти

Для зашифрования блока памяти служит функция EnCryptMem(**wbotho**). (Рекомендуется вместо этой функции использовать функцию EnCryptMemEx.)

```
T16bit WINAPI EnCryptMem (void *in_mem_buf,
                          TLen leng,
                          void *out_mem_buf,
                          T16bit node_From,
                          P16bit node_To,
                          char *ser);
```

Функция имеет параметры:

(**i**) in_mem_buf — указатель на блок памяти, который должен быть зашифрован;

(**i**) length — длина блока;

(**o**) out_mem_buf — указатель на блок памяти, в котором нужно сохранить зашифрованные данные (отводится пользователем);

Внимание. При зашифровании блока памяти необходимо учесть, что размер зашифрованного блока памяти для каждого получателя увеличивается на длину заголовка (MEM_TITLE_LEN = 37) и длину ключа, используемого при массовой рассылке (KEY_IN_FILE_LEN = 48). Таким образом, длина зашифрованного блока памяти увеличивается на (MEM_TITLE_LEN + KEY_IN_FILE_LEN * n) байт, где n — число получателей.

(**i**) node_From — номер ключа шифрования отправителя (XXXX), представленный в виде числа типа unsigned short;

(**i**) node_To — список номеров ключей шифрования получателей (должен быть представлен строкой элементов, имеющих тип unsigned short и оканчиваться нулем (0));

(**i**) ser — указатель на строку длинной 6 символов с номером серии (SSSSSS) открытого ключа получателя; если в качестве значения этого параметра указать последовательность из шести нулевых символов (\0\0\0\0\0\0), то номер серии берется из драйвера хранения ключевой информации.

Пример вызова функции зашифрования:

```
char   ser_To[7] = "000005";
node_To[0] = 2;
node_To[1] = 3;
node_To[2] = 0;
in_mem_buf = (char*)malloc(10);
out_mem_buf = (char*)malloc(10 + MEM_TITLE_LEN + KEY_IN_FILE_LEN * 2);
```

```
for(i = 0; i < 10; in_mem_buf[i++] = 0);
error_code = EnCryptMem (in_mem_buf, 10, out_mem_buf, 1,
                         node_To, ser_To);
```

8.4.2 Расшифрование блока памяти

Для расшифрования блока памяти служит функция DeCryptMem(**wbotho**). (Рекомендуется вместо этой функции использовать функцию EnCryptFileEx). которая имеет следующий прототип:

```
T16bit WINAPI DeCryptMem (void *buffer,
                          PLen leng,
                          T16bit abonent);
```

Функция имеет параметры:

(**i/o**) buffer — указатель на зашифрованный блок памяти, в который будут помещены расшифрованные данные при успешном выполнении функции;

(**i/o**) leng — указатель на переменную, в которой задается длина зашифрованного блока; после расшифрования — длина открытых данных;

(**i**) abonent — номер ключа шифрования получателя (XXXX), представленный в виде числа типа unsigned short.

Пример вызова функции расшифрования:

```
leng = 10 + MEM_TITLE_LEN + KEY_IN_FILE_LEN * 2;
error_code = DeCryptMem(out_mem_buf, &leng, 1);
if(leng != 10) return 1;
```

8.4.3 Получение идентификатора абонента, зашифровавшего блок памяти

Для получения идентификатора абонента, зашифровавшего блок памяти, служит функция GetMemSenderID(**wbotho**). Она имеет прототип:

```
T16bit WINAPI GetMemSenderID (void *mem_block,
                              Tlen leng,
                              LPSTR sender_id);
```

Функция имеет параметры:

(**i**) mem_block — указатель на зашифрованный блок памяти;

(**i**) leng — длина зашифрованного блока памяти;

(**o**) sender_id — указатель на строку длиной 10 байт, в которую записывается идентификатор абонента.

Пример вызова функции расшифрования:

```
char my_ID[11];
memset(my_ID, 0, 11);
code=GetMemSenderID(buff, buff_leng, my_ID);
```

8.4.4 Получение списка получателей зашифрованного блока памяти

Для зашифрованных блоков памяти, так же как и для зашифрованных файлов, определена функция GetCryptKeysM(**wbotho**), позволяющая получить список получателей (абонентов, которые могут расшифровать данный блок памяти). При формировании списка получателей зашифрованного блока памяти расшифрование и проверка правильности этого блока не производятся.

Функция GetCryptKeysM имеет прототип:

```
T16bit WINAPI GetCryptKeysM (void *block,
                TLen leng,
                P16bit abonents,
                P16bit * user_list, char *ser);
```

Функция имеет параметры:

(**i**) buffer — указатель на зашифрованный блок памяти;

(**i**) leng — длина зашифрованного блока;

(**o**) abonents — указатель на переменную, в которую записывается количество получателей;

(**o**) user_list — список получателей зашифрованного блока, представленный строкой элементов, имеющих тип unsigned short, и оканчивающийся нулем (0).

Внимание. Память под этот массив отводится внутри функции GetCryptKeysM и должна быть освобождена функцией FreeMemory (см. п. 8.12), как только user_list будет обработан.

(**o**) ser — указатель на строку, в которую записывается номер серии (7 байт) открытых ключей отправителей.

Пример использования функции GetCryptKeysM:

```
char ser[7];
error_code = GetCryptKeysM(out_mem_buf, leng, abonents, user_list,ser);
for(i = 0; i < *abonents; i++) printf("%u\n", *user_list[i]);
FreeMemory(user_list);
```

8.4.5 Зашифрование блока памяти (расширенное)

Функция EnCryptMemEx(**wbotho**) служит для зашифрования блока памяти. Функция не обращается к справочникам открытых ключей, также допускается шифрование между абонентами из различных ключевых серий . Функция имеет следующий прототип:

```
T16bit WINAPI EnCryptMemEx (void *in_mem_buf,
                TLen *leng,
                void *out_mem_buf,
                char *From,
                void **open_keys_array,
                T16bit open_keys_quantity,
                DWORD flags);
```

Функция имеет параметры:

(**i**) in_mem_buf — указатель на блок памяти, который должен быть зашифрован;

(**i/o**) length — указатель на переменную в которой передаётся длина блока памяти, в случае успешного завершения функциия записывает в эту переменную результирующую длину зашифрованного блока памяти ;

(**o**) out_mem_buf — указатель на блок памяти, в котором нужно сохранить зашифрованные данные (отводится пользователем);

Внимание. При зашифровании блока памяти необходимо учесть, что размер зашифрованного блока памяти для каждого получателя увеличивается на длину заголовка (MEM_TITLE_LEN = 37) и длину ключа, используемого при массовой рассылке (KEY_IN_FILE_LEN = 48). Таким образом, длина зашифрованного блока памяти увеличивается на (MEM_TITLE_LEN + KEY_IN_FILE_LEN * n) байт, где n — число получателей.

(i) From — указатель на строку с идентификатором ключа шифрования отправителя в формате XXXXSSSSSS;

(i) open_keys_array — указатель на массив указателей на открытые ключи получателей;

(i) open_keys_quantity — количество получателей (количество откырых ключей);

(i) flags — зарезервированный параметр

Пример вызова функции зашифрования (см. также п. 8.3.5):

```
…
char   From[11] = "0001910000";
void   *open_keys_array[2];
DWORD leng;
…
err_code=EnCryptMemEx (in_mem_buf,&leng,out_mem_buf,From
,open_keys_array, 2,0);
…
```

8.4.6 Расшифрование блока памяти (расширенное)

Функция DeCryptMemEx(**wbotho**) служит для расшифрования блока памяти . Функция не обращается к справочникам открытых ключей, также функция позволяет расшифровать блок памяти и в случае если ключевые серии отправителя и получателя различны . Функция имеет следующий прототип :

```
T16bit WINAPI DeCryptMemEx (void *buffer,
            TLen *leng,
            char *abonent
            void *pub_key);
```

Функция имеет параметры:

(i/o) buffer — указатель на зашифрованный блок памяти, в который будут помещены расшифрованные данные при успешном выполнении функции;

(i/o) leng — указатель на переменную, в которой задается длина зашифрованного блока; после расшифрования — длина открытых данных;

(i) abonent — указатель на строку с идентификатором ключа получателя в формате XXXXSSSSSS

(i) pub_key — указатель на буфер с открытым ключом отправителя.

Пример вызова функции расшифрования (см. также п. 8.3.6):

```
char   abonent[11] = "0001920000";
char   sender_key[304];
DWORD leng;
…
err_code=DeCryptMemEx (buffer,&leng,abonent,sender_key);
…
```

8.5 Получение имитовставки

8.5.1 Получение имитовставки для файла на ключе связи

Для получения имитовставки для файла на ключе связи служит функция ImitoFile(**wbotho**), прототип которой имеет вид:

```
T16bit WINAPI ImitoFile (char *path, T16bit From,
            T16bit To,
            T32bit * imit_out,
            char *ser);
```

Параметрами функции являются:

(**i**) path — указатель на строку полного пути к файлу, для которого вырабатывается имитовставка;

(**i**) From — номер ключа шифрования отправителя (XXXX), представленный в виде числа типа unsigned short; используется для формирования ключа связи;

(**i**) To — номер ключа шифрования получателя (XXXX), представленный в виде числа типа unsigned short; используется для формирования ключа связи;

(**o**) imit_out — указатель на переменную типа unsigned long, в которую записывается имитовставка;

(**i**) ser — указатель на строку с номером серии (SSSSSS) открытого ключа отправителя.

Пример вызова функции:

```
err_code=ImitoFile("test.tst", 1, 2, immita, "999999");
```

8.5.2 Получение имитовставки для файла на пароле

Для получения имитовставки для файла на пароле служит функция ImitoFileOnPassword(**wbotho**), прототип которой имеет вид:

```
T16bit WINAPI ImitoFileOnPassword (char *path,
            void *key_adr,
            T32bit * imit_out);
```

Функция имеет параметры:

(**i**) path — указатель на строку полного пути к файлу, для которого вырабатывается имитовставка;

(**i**) key_adr — указатель на область памяти с паролем (длина пароля должна равняться 32 байтам); в качестве пароля можно использовать случайное число, полученное с помощью функции Rndm (см. 8.6).

(**o**) imit_out — указатель на переменную типа unsigned long, в которую записывается имитовставка.

Пример вызова функции:

```
err_code=ImitoFPassword("test.tst", passw, immita);
```

8.5.3 Получение имитовставки для файла на главном ключе

Для получения имитовставки для файла на главном ключе служит функция ImitoFileOnGK(**wbotho**), прототип которой имеет вид:

```
T16bit WINAPI ImitoFileOnGK (char *path,
                char *my_ID,
                T32bit * imit_out);
```

Функция имеет параметры:

(**i**) path — указатель на строку полного пути к файлу, для которого вырабатывается имитовставка;

(**i**) my_ID — указатель на строку с номером ключевой дискеты в формате XXXXSSSSS или XXXXSSSSSSYY

Если my_ID[0] = '\0', то главный ключ (GK) считывается с ключевой дискеты. В противном случае указанный идентификатор ключа сравнивается с идентификатором ключа, прогруженного в память драйвера, и при их совпадении главный ключ берется из драйвера;

(**o**) imit_out — указатель на переменную типа unsigned long, в которую записывается имитовставка.

Пример вызова функции:

```
unsigned long immita;
err_code=ImitoFileOnGk("test.tst", "000199999901", &immita);
```

8.5.4 Получение имитовставки для блока памяти на ключе связи

Для получения имитовставки для блока памяти на ключе связи служит функция ImitoMem(**wbotho**), прототип которой имеет вид:

```
T16bit WINAPI ImitoMem (void *block,
                TLen leng,
                T16bit From,
                T16bit To,
                T32bit * imit_out,
                char *ser);
```

Параметрами функции являются:

(**i**) block — указатель на блок памяти, для которого вырабатывается имитовставка;

(**i**) leng — длина блока памяти, для которого вырабатывается имитовставка;

(**i**) From — номер ключа шифрования отправителя (XXXX), представленный в виде числа типа unsigned short; используется для формирования ключа связи;

(**i**) To — номер ключа шифрования получателя (XXXX), представленный в виде числа типа unsigned short; используется для формирования ключа связи;

(**o**) imit_out — указатель на переменную типа unsigned long, в которую записывается имитовставка;

(**i**) ser — строка с номером серии (SSSSSS) открытого ключа отправителя.

Пример вызова функции:

```
unsigned long immita;
mem_buff=(char*)malloc(10);
err_code=ImitoMem(mem_buff, 10, 1, 2, &immita, "999999");
```

8.5.5 Получение имитовставки для блока памяти на пароле

Для получения имитовставки для блока памяти на пароле служит функция ImitoMemOnPassword(**wbotho, wsigno**), прототип которой имеет вид:

```
T16bit WINAPI ImitoMemOnPassword (void *block,
                TLen leng,
                void *key_adr,
                P32bit imit_out);
```

Функция имеет параметры:

(**i**) block — указатель на блок памяти, для которого вырабатывается имитовставка;

(**i**) leng — длина этого блока памяти;

(**i**) key_adr — указатель на область памяти, где хранится пароль (длина пароля — 32 байта); в качестве пароля можно использовать случайное число, полученное с помощью функции Rndm (см. 8.6).

(**o**) imit_out — указатель на переменную типа unsigned long, в которую записывается имитовставка.

Пример вызова функции:

```
mem_block =(char*)malloc(10);
passw=(char*)malloc(32);
immita=(char*)malloc(4);
 if(Rndm((BYTE*)passw,32)
 return;
err_code=ImitoMemPassword(mem_block, 10, password, &immita);
```

8.6 ФОРМИРОВАНИЕ СЛУЧАЙНОГО ЧИСЛА

Для получения случайного числа служит функция Rndm(**wbotho, wsigno**), прототип которой имеет вид:

```
T16bit WINAPI Rndm (void *buff,
                TLen rnd_size);
```

Функция имеет параметры:

(**o**) buff — указатель на область памяти, в которую записывается случайное число (отводится пользователем);

(**i**) rnd_size — длина случайной последовательности в байтах.

Пример вызова функции:

```
unsignrd char random_buff[32];
 ...
if(Rndm(random_buff,32)
return;
```

8.7 ЗАГРУЗКА И УДАЛЕНИЕ КЛЮЧА ПОДПИСИ ИЗ ОПЕРАТИВНОЙ ПАМЯТИ

8.7.1 Загрузка ключа подписи в оперативную память

Для того чтобы ускорить процесс выполнения криптографических функций, можно предварительно считать закрытый ключ подписи в память (при использовании библиотеки разрешается подписывать файлы и блоки памяти на закрытом ключе подписи, хранящемся в памяти). Для этого предназначена функция SignLogIn(**wbotho, wsigno**), прототип которой имеет вид:

```
T16bit WINAPI SignLogIn (char *path);
```

Параметром этой функции является указатель на строку полного пути к закрытому ключу подписи (path).

Пример обращения к функции загрузки ключа подписи в память:

```
error_code = SignLogIn("A:\\");
if(error_code) return(error_code);
```

Внимание. Перед завершением работы программы, использующей библиотеку, закрытые ключи должны быть удалены из памяти (см. 8.7.2).

8.7.2 Удаление ключа подписи из оперативной памяти

Закрытый ключ подписи можно (а по окончании работы библиотеки, нужно) удалить из оперативной памяти. Для этого служит функция SignLogOut(**wbotho, wsigno**), прототип которой имеет вид:

```
T16bit WINAPI SignLogOut (void);
```

8.8 Подпись и проверка подписи файла

В состав библиотеки входят функции, которые позволяют подписывать информацию, хранящуюся в виде файлов. Подпись файла может выполняться в двух режимах:

- с добавлением подписи в конец подписываемого файла (см. 8.8.1);
- с сохранением подписи в отдельном файле (см. 8.8.2).

Внимание. Нельзя путать режимы подписи файла, так как при проставлении подписи с сохранением ЭЦП в отдельном файле структура подписываемой информации предварительно не проверяется.

8.8.1 Подпись файла с добавлением подписи в конец подписываемого файла

Перед выполнением подписи файла с добавлением ЭЦП в конец подписываемого файла всегда проверяется, был ли этот файл ранее подписан или нет. Если файл ранее подписан не был, формируется заголовок, в котором прописывается служебная информация. Если файл уже был подписан, в конец заголовка добавляется запись о новой ЭЦП. После этого формируется электронная цифровая подпись, которая дописывается в конец файла.

Для подписи файлов предназначена функция SignFile(**wbotho, wsigno**), которая имеет следующий прототип:

```
T16bit WINAPI SignFile (char *src_file_name,
                        char *dst_file_name,
                        char *name);
```

Функция имеет следующие параметры:

(**i**) src_file_name — указатель на строку полного пути к подписываемому файлу;

(**i**) dst_file_name — указатель на строку полного пути к подписанному файлу;

Внимание. При проставлении первой подписи размер подписанного файла увеличивается на 98 байт, при проставлении каждой последующей — на 87 байт.

(**i/o**) name — указатель на строку с идентификатором ключа подписи в формате XXXXSSSSSSYY. Под строку должно быть отведено не менее 13 байт; в случае, когда идентификатор ключа не задан (name[0]='\0';) или задан не полностью(XXXXSSSSSS или XXXX), информация об идентификаторе берётся из нулевого слота драйвера (или со сменного ключевого носителя). При завершении функции полный идентификатор ключа записывается в строку name.

Пример вызова функции подписи:

```
char name[]="000112345601";
error_code = SignFile("C:\\AUTOEXEC.BAT", "C:\\TEST.TST", name);
```

8.8.2 Подпись файла с сохранением ЭЦП в отдельном файле

При подписи файла с сохранением ЭЦП в отдельном файле вся информация, содержащаяся в исходном файле, подписывается целиком. Перед выполнением подписи проверяется только структура и длина файла, в котором хранятся ЭЦП. Новая ЭЦП добавляется в конец файла, в котором хранятся подписи. Файл для подписей создается автоматически при формировании первой подписи.

Для подписи файлов служит функция SignFileSeparate(**wbotho, wsigno**), которая имеет следующий прототип:

```
T16bit WINAPI SignFileSeparate (char *src_file_name,
            char *name,
            char *sign_file);
```

Функция имеет следующие параметры:

(**i**) src_file_name — указатель на строку полного пути к подписываемому файлу;

(**i/o**) name — указатель на строку с идентификатором ключа подписи в формате XXXXSSSSSSYY. Под строку должно быть отведено не менее 13 байт; в случае, когда идентификатор ключа не задан (name[0]='\0';) или задан не полностью(XXXXSSSSSS или XXXX), информация об идентификаторе берётся из нулевого слота драйвера (или со сменного ключевого носителя). При завершении функции полный идентификатор ключа записывается в строку name.

(**i**) sign_file — указатель на строку полного пути к файлу, в котором хранятся подписи;

Внимание. Размер файла, в котором хранятся подписи, равен 98 байтам при проставлении первой подписи и увеличивается на 87 байт при проставлении каждой последующей.

Пример вызова функции:

```
char   name[]="000199999901";
err_code=SignFileSeparate("test.tst",name , "tst_sign.tst");
```

8.8.3 Проверка подписи, добавленной в конец исходного файла

Для проверки подписи под файлом служит функция check_file_sign(**wbotho, wsigno**), которая имеет следующий прототип:

```
T16bit WINAPI check_file_sign (char *file_name,
            P8bit count,
            Check_Status_Ptr *stat_array);
```

Функция имеет следующие параметры:

(**i**) file_name — указатель на строку полного пути к файлу;

(**o**) count — указатель на переменную, в которую записывается количество обнаруженных подписей;

(**o**) stat_array — массив результатов проверки каждой подписи, элементами которого являются структуры (структура описана в файле verba.h):

```
#define NAME_LEN  12
#define NAME_LEN  120
typedef struct tagCheck_Status {
    char Name[NAME_LEN+1];
    char Alias[ALIAS_LEN+1];
```

```
    T8bit Position;
    T8bit Status;
    T32bit Date;
} Check_Status;
typedef Check_Status *Check_Status_Ptr;
#define CHECK_STATUS_LEN    sizeof(Check_Status)
```

- Каждая структура содержит:
- идентификатор ключа;
- описание (текстовые атрибуты);
- порядковый номер подписи;
- гринвическое время (UTC) формирования подписи (time_t);

Внимание. Данный формат для времени формирования подписи используется начиная с версии библиотек СКЗИ 5.0 .

- результат проверки:

  0 = CORRECT — подпись верна;

  1 = NOT_CORRECT — подпись не верна;

  2 = OKEY_NOT_FOUND — открытый ключ для проверки подписи не найден.

Внимание. Память под этот массив отводится внутри функции check_file_sign и должна быть освобождена функцией FreeMemory (см. п. 8.12), как только stat_array будет обработан.

Пример использования функции check_file_sign:

```
unsigned char count;
Check_Status_Ptr status_array;
...
error_code = check_file_sign("C:\\TEST.TST", &count, &status_array);
if(error_code) return(error_code);
for(i = 0; i < count; i++)
  printf("%u-%u\n", status_array[i].Position, status_array[i].Status);
FreeMemory(status_array);
```

8.8.4 Проверка подписи, добавленной в конец исходного файла (расширенная)

Функция check_file_sign_ex (**wbotho, wsigno**) служит для проверки подписи под файлом. Функция не обращается к справочникам открытых ключей. Прототип функции:

```
T16bit WINAPI check_file_sign_ex (char *file_name,
            void **open_keys_array, T32bit open_keys_quantity,
            P8bit count, Check_Status_Ptr *stat_array);
```

Функция имеет следующие параметры:

(**i**) file_name — указатель на строку полного пути к файлу;

(**i**) open_keys_array — указатель на массив указателей на открытые ключи, которые будут использоваться для проверки подписи (подписей) файла

(**i**) open_keys_quantity — количество открытых ключей ;

(**o**) count — указатель на переменную, в которую записывается количество обнаруженных подписей;

(**o**) stat_array — массив результатов проверки каждой подписи, элементами которого являются структуры (структура описана в файле verba.h , см. также описание функции check_file_sign):

Внимание. Память под этот массив отводится внутри функции check_file_sign и должна быть освобождена функцией FreeMemory (см. п. 8.12), как только stat_array будет обработан.

Пример использования функции check_file_sign:

```
unsigned char count;
Check_Status_Ptr status_array;
static char key1_91[304];
static char key1_92[304];
void   *open_keys_array[]={key1_91,key1_92};
…
err_code=ExtractKey(SIGN_SPR,"000191000011",key1_91);
if(err_code) goto error;
err_code=ExtractKey(SIGN_SPR,"000192000012",key1_92);
if(err_code) goto error;
...
error_code = check_file_sign_ex("C:\\TEST.TST", open_keys_array,2,
&count, &status_array);
if(error_code) return(error_code);
for(i = 0; i < count; i++)
  printf("%u-%u\n", status_array[i].Position, status_array[i].Status);
FreeMemory(status_array);
...
```

8.8.5 Проверка подписи, сохраненной в отдельном файле

Для проверки подписи, сохраненной в отдельном файле, служит функция CheckFileSeparate(**wbotho, wsigno**), которая имеет следующий прототип:

```
T16bit WINAPI CheckFileSeparate (char *src_file,
             P8bit count,
             Check_Status_Ptr *stat_array,
             char *sign_file);
```

Функция имеет следующие параметры:

(**i**) src_file — указатель на строку полного пути к подписанному файлу;

(**o**) count — указатель на переменную, в которую записывается количество обнаруженных подписей;

(**o**) stat_array — массив результатов проверки каждой подписи, структура которого описана в предыдущем пункте;

(**i**) sign_file — указатель на строку полного пути к файлу, в котором хранятся подписи.

Внимание. Память под массив stat_array отводится внутри функции CheckFileSeparate и должна быть освобождена функцией FreeMemory (см. п. 8.12), как только stat_array будет обработан.

Пример вызова функции:

```
unsigned char count=0;
Check_Status_Ptr status_array;
...
err_code=CheckFileSeparate("test.tst",&count,&status_array,
                           "tst_sign.tst")
if(err_code)
  return(err_code);
for(i=0;i<count;i++)
  printf("%u-%u\n",status_array[i].Position,status_array[i].Status);
FreeMemory(status_array);
```

8.8.6 Удаление подписи, добавленной в конец исходного файла

Для удаления подписи файла служит функция DelSign(**wbotho, wsigno**), которая имеет следующий прототип:

```
T16bit WINAPI DelSign (char *file_name, T8bit count);
```

Функция имеет параметры:

(i) file_name — указатель на строку полного пути к подписанному файла;

(i) count — количество удаляемых подписей; если параметр count = –1, то удаляются все подписи.

Подписи в файле удаляются, начиная с последней. При этом проверка правильности подписей не производится.

Пример обращения к функции удаления подписи:

```
error_code = DelSign("C:\\TEST.TST", -1);
```

8.8.7 Получение информации о подписанном файле

Функция GetFileSignInfo(**wbotho, wsigno**) имеет прототип:

```
T16bit WINAPI GetFileSignInfo (char *name,
                SIGN_INFO **info,
                unsigned long *sign_quantity);
```

Параметры:

(i) name —указатель на строку с именем подписанного файла;

(i/o) info — указатель на массив структур каждая из которых состоит из двух строк: строки с идентификатором ключа, на котором была сформирована подпись и строки с регистрационным номером библиотеки СКЗИ "Верба-OW" :

```
typedef struct _sign_info {
    char nump[13];
    char reg_num[8];
} SIGN_INFO;
```

Внимание. Память под этот массив отводится внутри функции GetMemSignInfo и должна быть освобождена, как только массив info будет обработан.

(i/o) sign_quantity -указатель на переменную, в которую возвращается число обнаруженных подписей.

Пример обращения к функции:

```
SIGN_INFO *info;
ULONG      sign_quantity;
...
error_code = GetFileSignInfo("C:\\TEST.TST", &info, &sign_quantity);
...
FreeMemory(info);
```

8.9 Подпись и проверка подписи блока памяти

В состав библиотеки входят функции, которые позволяют подписывать информацию, хранящуюся в виде блоков памяти. Подпись блока памяти может выполняться в двух режимах:

- с добавлением подписи в конец подписываемого блока памяти (см. 8.9.1);
- с сохранением подписи в отдельном блоке (см. 8.9.2).

Внимание. Нельзя путать режимы подписи блока памяти, так как при проставлении подписи с сохранением ЭЦП в отдельном блоке памяти структура подписываемой информации предварительно не проверяется.

8.9.1 Подпись блока памяти с добавлением ЭЦП в конец исходного блока

Для подписи блока памяти служит функция SignMem(**wbotho, wsigno**), которая имеет следующий прототип:

```
T16bit WINAPI SignMem (void *block,
                       TLen leng,
                       char *name);
```

Функция имеет параметры:

(i) block — указатель на блок подписываемой памяти;

Внимание. При подписи блока памяти с добавлением подписи в конец блока необходимо учитывать изменение размера исходного блока памяти. Длина подписанного блока памяти увеличивается на длину заголовка (SIGN_TAIL_LEN = 11 байт) и длину подписи (FILE_SIGN_LEN = 87 байт) при проставлении первой подписи и на длину подписи (FILE_SIGN_LEN = 87 байт) при каждом последующем.

(i) leng — длина подписываемого блока памяти;

(i/o) name — указатель на строку с идентификатором ключа подписи в формате XXXXSSSSSSYY. Под строку должно быть отведено не менее 13 байт; в случае, когда идентификатор ключа не задан (name[0]='\0';) или задан не полностью(XXXXSSSSSS или XXXX), информация об идентификаторе берётся из нулевого слота драйвера (или со сменного ключевого носителя). При завершении функции полный идентификатор ключа записывается в строку name.

Пример обращения к функции подписи блока памяти:

```
char name[]="000112345601";
block = (char*)malloc(10 + SIGN_TAIL_LEN + FILE_SIGN_LEN);
error_code = SignMem(block, 10,name);
```

Имеется модификация функции SignMemEx(**wbotho, wsigno**):

```
T16bit WINAPI SignMemEx (void *block,
                TLen *leng,
                char *name);
```

Её отличие состоит в том, что параметр leng является указателем и после успешного завершения операции через него возвращается результирующая длина подписанного блока памяти.

8.9.2 Подпись блока памяти с сохранением ЭЦП в отдельном блоке

Для подписи блока памяти служит функция SignMemSeparate(**wbotho, wsigno**), которая имеет следующий прототип:

```
T16bit WINAPI SignMemSeparate (void *block,
                TLen leng,
                TLen sign_block_leng,
                char *name,
                void *sign_block);
```

Функция имеет следующие параметры:

(i) block — указатель на блок подписываемой памяти;

(i) leng — длина подписываемого блока памяти;

(i) sign_block_leng — длина блока памяти с подписями, в который добавляется новая подпись;

При проставлении подписи в первый раз значение параметра sign_block_leng должно быть равно нулю (0), при повторной подписи — (SIGN_TAIL_LEN + FILE_SIGN_LEN*n), где n — количество проставленных под блоком подписей.

(i/o) name — указатель на строку с идентификатором ключа подписи в формате XXXXSSSSSSYY. Под строку должно быть отведено не менее 13 байт; в случае, когда идентификатор ключа не задан (name[0]='\0';) или задан не полностью(XXXXSSSSSS или XXXX), информация об идентификаторе берётся из нулевого слота драйвера (или со сменного ключевого носителя). При завершении функции полный идентификатор ключа записывается в строку name.

(o) sign_block — указатель на блок памяти, в котором будет сохранена подпись.

В блок памяти, предназначенный для хранения подписей, записывается информация длиной (SIGN_TAIL_LEN + FILE_SIGN_LEN) для первой подписи и FILE_SIGN_LEN для каждой последующей. Как и в предыдущем пункте: SIGN_TAIL_LEN — длина заголовка подписываемого файла (11 байт), FILE_SIGN_LEN — длина подписи (87 байт).

Пример вызова функции:

```
char   name0[]="000199999901";
char   name1[]="000299999901";
/* подписываемый блок памяти */
mem_block =(char*)malloc(10);
/* буфер на две подписи */
sign_block=(char*)malloc(SIGN_TAIL_LEN+(2*FILE_SIGN_LEN));
 ...
if(SignInit("A:\\", "C:\\VERBA_OW\\SPR\\")
 return;
/* первая подпись */
if(SignMemSeparate(mem_block, 10, 0, name0, sign_block))
 return;
 ...
```

```
/* вторая подпись */
if(SignMemSeparate(mem_block, 10, SIGN_TAIL_LEN+FILE_SIGN_LEN,
name1, sign_block))
 return;
 ...
```

Имеется модификация функции SignMemSeparateEx(**wbotho, wsigno**):

```
T16bit WINAPI SignMemSeparateEx (void *block,
             TLen leng,
             TLen *sign_block_leng,
             char *name,
             void *sign_block);
```

Её отличие состоит в том, что параметр sign_block_leng является указателем и после успешного завершения операции через него возвращается результирующая длина блока с подписями.

8.9.3 Подпись хэша

Для вычисления ЭЦП от значения хэш-функции служит Функция SignHash , которая имеет следующий прототип:

```
T16bit WINAPI SignHash (HASHCONTEXT hHash,
             char     *name,
             T8bit    *pbSignatyre,
             P32bit   pdwSigLen,
             T32bit   dwFlags);
```

Функция имеет следующие параметры:

(**i**) hHash — в зависимости от значения параметра dwFlags может интерпретироваться либо как описатель объекта хэширования (см. п.), либо как указатель на значение хэш-функции (значение вычисленное функциями GetHash,HashMem или HashFile).;

(**i/o**) name — указатель на строку с идентификатором ключа подписи в формате XXXXSSSSSSYY. Под строку должно быть отведено не менее 13 байт; в случае, когда идентификатор ключа не задан (name[0]='\0';) или задан не полностью(XXXXSSSSSS или XXXX), информация об идентификаторе берётся из нулевого слота драйвера (или со сменного ключевого носителя). При завершении функции полный идентификатор ключа записывается в строку name.

(**i/o**) pdwSigLen — указатель на переменную, в которой передается длина буфера для ЭЦП. Если размер буфера недостаточен, то функция возвращает код ошибки E_MEM_LENGTH и записывает в переменную требуемый размер буфера. длина блока памяти с подписями, в который добавляется новая подпись;

(**i**) dwFlags — флаги с помощью которых задается тип вычисляемой подписи. Параметр может принимать сследующие значения:

SIGN_DEF - для формировании "чистого" значения ЭЦП длиной 64 байта (SIGN_LEN). hHash при этом интерпретируется как описатель объекта хэширования;

SIGN_DEF | DATA_HASH - аналогично предыдущему, за исключением того, что hHash интерпретируется как указатель на буфер со значением хэш-функции;

SIGN_VER - для формирования ЭЦП в формате "Вербы". Размер буфера при этом должен быть не менее чем FILE_SIGN_LEN + SIGN_TAIL_LEN. hHash интерпретируется как описатель объекта хэширования.

SIGN_VER | ADD_SIGN - для формирования ЭЦП в формате "Вербы". Если в буфере уже есть ЭЦП в формате "Вербы" , то новая подпись добавляется к существующим (аналогично функции SignMemSeparate), размер блока с подписями при этом увеличивается на FILE_SIGN_LEN байт.Функция возвращает в pdwSigLen результирующий размер буфера с подписями. HHash интерпретируется как описатель объекта хэширования. В pdwSigLen должно передаваться значение не меньшее чем FILE_SIGN_LEN + SIGN_TAIL_LEN.

Замечание. После выполнения функции SignHash возможно продолжение процесса хэширования для объекта hHash.

8.9.4 Проверка подписи блока памяти, добавленной в конец исходного блока

Для проверки подписи под блоком памяти служит функция check_mem_sign(**wbotho, wsigno**), которая имеет следующий прототип:

```
T16bit WINAPI check_mem_sign (void *block,
                              TLen leng,
                              P8bit count,
                              Check_Status_Ptr *stat_array);
```

Функция имеет следующие параметры:

(**i**) block — указатель на подписанный блок памяти;

(**i**) leng — длина блока;

(**o**) count — указатель на переменную, в которую записывается количество обнаруженных подписей;

(**o**) stat_array — массив результатов проверки каждой подписи (подробнее см. 8.8.3).

Внимание. Память под этот массив отводится внутри функции check_mem_sign и должна быть освобождена функцией FreeMemory (см. п. 8.12), как только stat_array будет обработан.

Пример использования функции check_file_sign:

```
unsigned char count;
Check_Status_Ptr status_array;
...
error_code = check_mem_sign(block, 10, &count, &status_array);
if(error_code) return(error_code);
for(i = 0; i < count; i++)
  printf("%u-%u\n", status_array[i].Position, status_array[i].Status);
FreeMemory(status_array);
```

8.9.5 Проверка подписи блока памяти, сохраненной в отдельном блоке

Для проверки подписи под блоком памяти служит функция CheckMemSeparate(**wbotho, wsigno**), которая имеет следующий прототип:

```
T16bit WINAPI CheckMemSeparate (void *block,
                                TLen leng,
                                TLen sign_block_leng,
                                P8bit count,
                                Check_Status_Ptr *stat_array,
                                void *sign_block);
```

Функция имеет следующие параметры:

(**i**) block — указатель на подписанный блок памяти;

(**i**) leng — длина блока памяти;

(**i**) sign_block_leng — длина блока с подписями;

(**o**) count — указатель на переменную, в которую записывается количество найденных подписей;

(**o**) stat_array — массив результатов проверки каждой подписи (подробнее см. 8.8.3);

Внимание. Память под этот массив отводится внутри функции CheckMemSeparate и должна быть освобождена функцией FreeMemory (см. п. 8.12), как только stat_array будет обработан.

(**i**) sign_block — указатель на блок памяти, в котором хранятся подписи.

Пример вызова функции:

```
/* проверить после двух операций подписи */
unsigned char count=0;
Check_Status_Ptr status_array;
...
if(CheckMemSeparate(block,10,SIGN_TAIL_LEN+(2*FILE_SIGN_LEN),
                  &count, &status_array, sign_block))
    return;
if(count!=2)
    return;
...
FreeMemory(status_array);
```

8.9.6 Проверка подписи блока памяти, сохраненной в отдельном блоке (расширенная)

Функция CheckMemSeparateEx(**wbotho, wsigno**) служит для проверки подписи под блоком памяти . Функция не обращается к справочникам открытых ключей. Прототип функции:

```
T16bit WINAPI CheckMemSeparateEx (void *block,
                TLen leng,
                void * sign_block,
                TLen sign_block_leng,void
                void **open_keys_array,
                T32bit open_key_quantity,
                P8bit count,
                Check_Status_Ptr *stat_array);
```

Функция имеет следующие параметры:

(**i**) block — указатель на подписанный блок памяти;

(**i**) leng — длина блока памяти;

(**i**) sign_block — указатель на блок памяти, в котором хранятся подписи;

(**i**) sign_block_leng — длина блока с подписями;

(**i**) open_keys_array —указатель на массив указателей на открытые ключи , которые будут использоваться при проверке подписи (подписей) блока памяти ;

(**i**) open_keys_quantity — количество открытых ключей;

(**o**) count — указатель на переменную, в которую записывается количество найденных подписей;

(**o**) stat_array — массив результатов проверки каждой подписи (подробнее см. 8.8.3).

Внимание. Память под этот массив отводится внутри функции CheckMemSeparate и должна быть освобождена функцией FreeMemory (см. п. 8.12), как только stat_array будет обработан.

Пример вызова функции (см. также п. 8.8.4):

```
/* проверить после двух операций подписи */
unsigned char count=0;
Check_Status_Ptr status_array;
static char key1_91[304];
static char key1_92[304];
void   *open_keys_array[]={key1_91,key1_92};
...
if(CheckMemSeparateEx(block, 10, sign_block,
SIGN_TAIL_LEN+(2*FILE_SIGN_LEN), open_keys_array, 2, &count,
&status_array))
    return;
if(count!=2)
    return;
...
FreeMemory(status_array);
...
```

8.9.7 Проверка подписи хэша

Функция VerifySignature предназначена для проверки ЭЦП, вычисленной от значения хэш-функци. Прототип функции:

```
T16bit WINAPI VerifySignature (HASHCONTEXT hHash,
            const T8bit   *pbSignature,
            T32bit   dwSigLen,
            void     **open_keys_array,
            T32bit   open_key_quantity,
            Check_Status_Ptr    *stat_array
            P32bit   sig_count,
            T32bit   dwFlags);
```

Функция имеет следующие параметры:

(i) hHash — в зависимости от значения параметра dwFlags может интерпретироваться либо как описатель объекта хэширования (см. п.), либо как указатель на значение хэш-функции (значение вычисленное функциями GetHash, HashMem или HashFile);

(i) pbSignature — указатель на буфер с ЭЦП;

(i) dwSigLen — длина буфера в байтах;

(i) open_keys_array —указатель на массив указателей на открытые ключи , которые будут использоваться при проверке ЭЦП;

(o) stat_array — массив результатов проверки каждой подписи (подробнее см. 8.8.3).

Внимание. Память под этот массив отводится внутри функции VerifySignature и должна быть освобождена функцией FreeMemory (см. п. 8.12), как только stat_array будет обработан.

(i) open_keys_quantity — количество открытых ключей;

(**o**) sig_count — указатель на переменную, в которую записывается количество найденных подписей;

(**i**) dwFlgas — флаги с помощью которых задается тип проверяемой подписи. Параметр может принимать ссследующие значения:

SIGN_DEF - содержимое буфера pbSignature интерпретируется как "чистая" ЭЦП длиной 64 байта (SIGN_LEN), hHash - описатель объекта хэширования. Значение dwSigLen должно быть равно SIGN_LEN;

SIGN_DEF | DATA_HASH - аналогично предыдущему, за исключением того, что hHash интерпретируется как указатель на буфер со значением хэш-функции.Значение dwSigLen должно быть равно SIGN_LEN;

SIGN_VER - pbSignature указывает на буфер с подписями в формате Вербы (аналогично CheckMemSeparate), Значение dwSigLen должно быть не меньше чем FILE_SIGN_LEN + SIGN_TAIL_LEN; hHash интерпретируется как описатель объекта хэширования .

8.9.8 Проверка подписи под сообщением в формате PKCS#7 с использованием сертификата в формате X.509 .

Для проверки подписи под сообщением PKCS#7 служит функция PKCSVerifyMessageSignature , которая имеет следующий прототип:

```
BOOL WINAPI PKCSVerifyMessageSignature(

            void *data, T32bit szdata,

            void *PKCSsign, T32bit szPKCSsign,

            void *cert, T32bit szsert,

            void *plaindata,

            T32bit     *szplaindata);
```

Функция имеет параметры:

(i) data – указатель на блок памяти с открытыми данными, от которых проверяется подпись. Если подписываемые данные содержатся внутри сообщения PKCS#7, этот параметр может быть равен NULL.

(i) szdata – размер данных для которых проверяется подпись. Если парамети data – NULL, то значение szdata должно быть равно 0.

(i) PKCSsign – указатель на блок памяти с сообщением в формате PKCS#7, содержащим проверяемое значениеЭЦП.

(i) szPKCSsign – размер сообщения PKCS#7.

(i) cert – указатель блок памяти с сертификатом открытого ключа в формате X.509.

(i) szsert – размер сертификата открытого ключа.

(i) plaindata – указатель на буфер в который будут записаны открытые подписанные данные(без аутентифицируемых атрибутов) из сообщения PKCS#7. Если параметр data не равен NULL, то значение plaindata должно быть установлено в NULL.

(io) szplaindata – указатель на переменную в которую будет записана длина открытых подписанных данных при условии, что значения параметров plaindata и data – NULL. Если plaindata – NULL , то szplaindata должно быть равно 0.

В случае если подпись верна, то код возврата не нулевой (TRUE). Если подпись не верна, то функция возвращает 0 (FALSE) при этом дополнительную информацию об ошибках можно получить вызвав функциюGetLastError.

Пример:

```
/* проверить подпись под данными, содержащимися в сообщении PKCS#7 */
void *plaintext;
T32bit szplaindata;
…
/* определить размер буфера для открытых данных и выделить память*/
    code=PKCSVerifyMessageSignature(NULL, 0, PKCSsign, szPKCSsign,
cert, szcert, NULL, &szplaindata);
if(!code)
 code=GetLastError();
plaintext=malloc(szplaindata);
if(!plaintext)
 return;
/* проверить подпись */
    code=PKCSVerifyMessageSignature(NULL, 0, PKCSsign, szPKCSsign,
cert, szcert, plaintext, &szplaindata);
if(!code)
 code=GetLastError();
```

8.9.9 Удаление подписи, добавленной в конец блока памяти

Для удаления подписей из подписанного блока служит функция Del_Mem_Sign(**wbotho, wsigno**), которая имеет следующий прототип:

```
T16bit WINAPI Del_Mem_Sign (void *block,
                            PLen leng,
                            T8bit count);
```

Функция имеет параметры:

(i) block — указатель на блок памяти;

(i/o) leng — указатель на переменную, в которую задается длина подписанного блока памяти; после удаления подписей — длина исходного текста;

(i) count — количество удаляемых подписей; если count = -1, то удаляются все подписи.

При удалении подписей из подписанного блока функцией Del_Mem_Sign проверка правильности подписей не производится.

Пример обращения к функции удаления подписи:

```
leng = 10 + SIGN_TAIL_LEN + FILE_SIGN_LEN:
error_code = Del_Mem_Sign(block, &leng, 1);
if(leng != 10) return 1;
```

8.9.10 Получение информации о подписанном блоке памяти

Функция GetMemSignInfo(**wbotho, wsigno**) имеет прототип:

```
T16bit WINAPI GetMemSignInfo (void *block,
                TLen leng,
                SIGN_INFO **info,
                unsigned long sign_quantity);
```

Параметры:

(**i**) block — указатель на блок памяти;

(**i**) leng — длина подписанного блока памяти;

(**i/o**) info — указатель на массив структур каждая из которых состоит из двух строк: строки с идентификатором ключа, на котором была сформирована подпись и строки с регистрационным номером библиотеки СКЗИ "Верба-OW"

```
typedef struct _sign_info {
    char nump[13];
    char reg_num[8];
} SIGN_INFO;
```

Внимание. Память под этот массив отводится внутри функции GetMemSignInfo и должна быть освобождена, как массив info будет обработан.

(**i/o**) sign_quantity - указатель на переменную, в которую возвращается число обнаруженных подписей.

Пример обращения к функции:

```
SIGN_INFO *info;
unsigned long sign_quantity;
...
error_code = GetMemSignInfo(block, leng, &info, &sign_quantity);
...
FreeMemory(info);
```

8.10 Хэширование файлов и областей памяти

Кроме функций формирования и проверки подписи в библиотеку включены также функции, которые позволяют получить значения функции хэширования для данных, представленных в виде файлов или блоков памяти. Процедура получения хэш-значения для открытых данных используется в алгоритме формирования электронной цифровой подписи. Напомним, что значение хэш-функции однозначно определяется исходными данными, не позволяет восстановить исходную информацию и имеет фиксированную длину (256 бит).

8.10.1 Вычисление значения хэш-функции для файла

Для нахождения хэш-значения для данных, хранящихся в файле, служит функция HashFile(**wbotho, wsigno**), прототип которой имеет следующий вид:

```
T16bit WINAPI HashFile (char *path,
                P32bit out);
```

Параметрами этой функции являются:

(**i**) path — указатель на строку полного пути к файлу, для которого вырабатывается хэш-значение;

(**o**) out — буфер под хэш-значение (32 байта).

Пример вызова функции:

```
hash_buff=(char*)malloc(32);
err_code=HashFile("test.tst", hash_buff);
```

8.10.2 Вычисление хэш-значения для блока памяти

Для нахождения хэш-значения для блока памяти служит функция HashMem(**wbotho, wsigno**), прототип которой имеет вид:

```
T16bit WINAPI HashMem (void *block,
                       TLen leng,
                       P32bit out);
```

Функция HashMem имеет параметры:

(**i**) block — указатель на блок хэшируемой памяти;

(**i**) leng — длина блока хэшируемой памяти;

(**o**) out — буфер под хэш-значение (32 байта).

Пример вызова функции:

```
/* хэшируемый блок памяти */
mem_block =(char*)malloc(10);
/* буфер для хэша */
hash_block=(char*)malloc(32);
err_code=HashMem(mem_block,10,hash_block);
```

8.11 Потоковые функции хэширования

8.11.1 Создание и инициализация объекта хэширования

Функция CreateHash (**wbotho, wsigno**)создает и инициализирует объект(контекст) "хэш" и возвращает его описатель в параметре phHash. Этот описатель служит входным параметром для функций связанных с хэшированием "поточных" данных,а также для функций SignHash и VerifySignature. Прототип функции имеет вид:

```
T16bit WINAPI CreateHash (HASHCONTEXT *phHash,
                          const char *alg_id,
                          T32bit dwFlags);
```

Функция CreateHash имеет параметры:

(**o**) phHash — указатель переменную, куда будет записан описатель объекта хэширования;

(**i**) alg_id — параметр определяет алгоритм и константы, используемые при вычислении

хэша. Параметр может принимать значния ALG_VERBA или ALG_VERBA_O;

(**i**) dwFlags — зарезервированный параметр.

8.11.2 Уничтожение объекта хэширования

Функция DestroyHash (**wbotho, wsigno**) уничтожает объект "хэш", определенный параметром hHash. После уничтожения объект хеширования не может более использоваться, а его описатель более не указывает на этот объект. После завершения работы приложения с объектами хеширования, все они должны быть уничтожены функцией DestroyHash. Прототип функции имеет вид:

```
T16bit WINAPI DestroyHash (HASHCONTEXT hHash,
                           T32bit dwFlags);
```

Функция DestroyHash имеет параметры:

(**i**) hHash — описатель уничтожаемого объекта хэширования;

(**i**) dwFlags — зарезервированный параметр.

8.11.3 Хэширование потоковых данных

Функция HashData (**wbotho, wsigno**) служит для хэширования поточных данных. Перед вызовом этой функции должен быть вызвана функция CreateHash для создания объекта(контекста) хеширования. Функция может вызываться многократно для одного объекта хэширования. Прототип функции имеет вид:

```
T16bit WINAPI HashData (HASHCONTEXT hHash,
                        const T8bit *pbData,
                        T32bit dwDataLen,
                        T32bit dwFlags);
```

Функция HashData имеет параметры:

(**i**) hHash — описатель объекта хэширования;

(**i**) pbData — указатель на блок хэшируемых данных;

(**i**) dwDataLen — число байт в хэшируемом блоке ;

(**i**) dwFlags — зарезервированный параметр.

8.11.4 Получение значения хэш-функции от потоковых данных

Функция GetHash (**wbotho, wsigno**) служит для получения значения хэш-функции от данных прошедших через объект хэширования. (От блоков данных обработанных вызовами функциями HashData). После вызова функции возможно продолжение процесса хэширования (HashData) для объекта hHash. Прототип функции имеет вид:

```
T16bit WINAPI GetHash (HASHCONTEXT hHash,
                       T8bit *pbData,
                       P32bit dwDataLen,
                       T32bit dwFlags);
```

Функция HashData имеет параметры:

(**i**) hHash — описатель объекта хэширования;

(**i**) pbData — указатель на буфер, отведенный пользователем, куда будет записано значение хэш-функции

(**i/o**) dwDataLen — указатель на переменную, куда записывается длина буфера в байтах. Длина должна быть равна HASH_LEN.

(**i**) dwFlags — должен быть равен EXPORT_DATA..

Пример:

```
{
...
char Data1[DATA1_LEN];
char Data2[DATA2_LEN];
char pbHash1[HASH_LEN];
```

```
char pbHashAll[HASH_LEN];
HASHCONTEXT hHash;
DWORD dwDataLen=HASH_LEN;
...
if(CreateHash(&hHash,ALG_VERBA_O,0))
 goto error;
...
/* хэшируем первый блок данных */
if(HashData(hHash,Data1,DATA1_LEN,0)
 goto error;
/* если нужно получить значение хэш-функции отдельно для первого блока*/
if(GetHash(hHash,pbHash1,&dwDataLen,EXPORT_HASH))
 goto error;
        ...
/* хэшируем следующий блок данных */
if(HashData(hHash,Data2,DATA2_LEN,0)
 goto error;
/* получаем хэш-функцию от двух  блоков Data1 и Data2 */
if(GetHash(hHash,pbHashAll,&dwDataLen, EXPORT_HASH))
 goto error;
       ...
DestroyHash(hHash);
       ...
}
```

8.11.5 Дублирование объекта хэширования

Функция DuplicateHash (**wbotho, wsigno**) служит для дублирования объекта хэширования. Прототип функции имеет вид:

```
T16bit WINAPI DuplicateHash (HASHCONTEXT hHash,
                             HASHCONTEXT *phHash,
                              T32bit dwFlags);
```

Функция DuplicateHash имеет параметры:

(i) hHash — описатель дублируемого объекта хэширования;

(i) phHash — указатель на описатель нового(продублированного) объекта хэширования;

(i) dwFlags — зарезервированный параметр.

8.12 Освобождение памяти, используемой при работе СКЗИ

Для освобождения памяти, которая была выделена для работы некоторых функций 16-разрядныой библиотеки, служит функция FreeMemory(**wbotho, wsigno**), которая имеет прототип:

```
void WINAPI FreeMemory (void *lpMemory);
```

Параметр:

(i) lpMemory — указатель на блок памяти, который необходимо освободить.

Пример использования функции FreeMemory:

```
Check_Status_Ptr status_array;
...
FreeMemory (status_array);
```

8.13 Удаление файла

Для удаления файлов с уничтожением находящейся в них информации, служит функция WipeFile(**wbotho, wsigno**), которая имеет прототип:

```
T16bit WINAPI WipeFile (char *file_name);
```

Параметры:

(**i**) file_name — имя файла, который необходимо удалить.

9. ОПИСАНИЕ ФУНКЦИЙ ДЛЯ РАБОТЫ СО СПРАВОЧНИКАМИ ОТКРЫТЫХ КЛЮЧЕЙ

В справочниках открытых ключей хранятся записи открытых ключей пользователей, называемые сертификатами. Справочники открытых ключей составляются администраторами безопасности при формировании ключевой информации. Пользователь (администратор) может редактировать справочники в процессе работы с СКЗИ.

9.1 Структура сертификата открытого ключа

В сертификате открытого ключа указываются:

- номер ключа (для ключа шифрования — XXXX, для ключа подписи — номер вида XXXXYY);
- тип ключа (действующий, резервный, скомпрометированный);
- текстовые атрибуты корреспондента (имя корреспондента);
- дата формирования ключа;
- открытый ключ;
- значение хэш-функции на все данные сертификата.

9.2 Структура справочников

9.2.1 Структура справочника открытых ключей шифрования

Справочник открытых ключей шифрования размещается в каталоге OPENKEY. Этот каталог создается в каталоге с СКЗИ (см. 7.1). Каталог OPENKEY разбит на подкаталоги по номерам серий ключей, в которых хранятся справочники на каждую серию. Справочники с открытыми ключами шифрования одной серии хранятся в файле LPUB.SPR.

В дальнейшем именно эти файлы будем называть справочниками открытых ключей шифрования.

Опишем подробнее структуру каталога OPENKEY.

Файлы в каталоге OPENKEY хранятся в подкаталогах согласно номерам серий открытых ключей (SSSSSS). В указанных подкаталогах размещаются файлы:

- LPUB.SPR — содержит справочник открытых ключей шифрования;
- XXXX.IMP — содержит имитовставки на все открытые ключи, которые вырабатываются на ключе CKD ключевой дискеты с номером XXXX (эта информация используется при проверке целостности справочника).

Каталог OPENKEY может иметь следующую структуру:

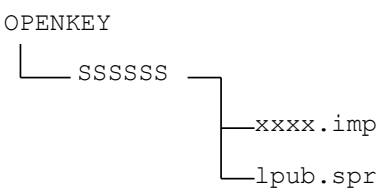

Рис. 9. Структура каталога OPENKEY

Каталог OPENKEY с описанной структурой создается автоматически при первом добавлении ключа шифрования в справочник.

9.2.2 Структура справочника открытых ключей подписи

Справочник открытых ключей подписи размещается в каталоге FAXKEY. Этот каталог создается в каталоге с СКЗИ (см. 7.1). Каталог FAXKEY разбит на подкаталоги по номерам серий ключей, в которых

хранятся справочники на каждую серию. Справочники с открытыми ключами подписи одной серии хранятся в файле LFAX.SPR.

В дальнейшем именно эти файлы будем называть справочниками открытых ключей подписи.

Опишем подробнее структуру каталога FAXKEY.

Файлы в каталоге FAXKEY хранятся в подкаталогах согласно номерам серий открытых ключей (SSSSSS). В указанных подкаталогах размещаются файлы:

- LFAX.SPR — содержит справочник открытых ключей подписи;
- XXXXYY.IMM — содержит имитовставку, которая вырабатывается на ключе CKDI ключевой дискете с номером XXXX и личным кодом YY (эта информация используется при проверке целостности справочника).

Каталог FAXKEY может иметь следующую структуру:

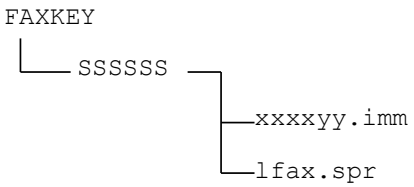

Рис. 10. Структура каталога FAXKEY

Каталог FAXKEY создается автоматически при первом добавлении открытого ключа подписи в справочник.

9.3 Функции для работы со справочниками

9.3.1 Получение атрибутов открытого ключа по его идентификатору

Для получения атрибутов открытого ключа по его идентификатору служит функция GetAlias(**wbotho**,**wsigno**), которая имеет прототип:

```
T16bit WINAPI GetAlias (char *base_dir,
                        char *open_key_ID,
                        char *alias);
```

Параметрами функции являются:

(**i**) base_dir — указатель на строку пути к каталогу OPENKEY или FAXKEY (файл со справочником LPUB.SPR ищется в подкаталоге: base_dir\OPENKEY\SSSSSS\; файл со справочником LFAX.SPR ищется в подкаталоге: base_dir\FAXKEY\SSSSSS\);

(**i**) open_key_ID — указатель на строку с идентификатором ключа.

Для получения атрибутов открытого ключа шифрования идентификатор открытого ключа шифрования должен быть указан в формате XXXXSSSSSS или XXXX (отсутствующие поля дополняются из строки, хранящейся в файле NUM).

Для получения атрибутов открытого ключа подписи идентификатор открытого ключа подписи должен быть указан в формате XXXXSSSSSSYY или XXXXYY (отсутствующие поля дополняются из строки, хранящейся в файле NUMP).

(**o**) alias — указатель на строку длиной 121 байт, в которую записываются атрибуты открытого ключа.

Пример вызова функции:

```
/* текстовые атрибуты открытого ключа подписи */
char Alias[121];
```

```
err_code=GetAlias("open_dir", "000199999901", Alias);
```

9.3.2 Получение идентификатора открытого ключа по его текстовым атрибутам

Для получения идентификатора открытого ключа по его атрибутам служит функция GetID(**wbotho, wsigno**), которая имеет прототип:

```
T16bit WINAPI GetID (char *base_dir,
                     char *alias,
                     char *ser,
                     char S_or_E,
                     char *open_key_ID);
```

Параметрами функции являются:

(**i**) base_dir — указатель на строку пути к каталогу OPENKEY или FAXKEY (файл со справочником LPUB.SPR ищется в подкаталоге: base_dir\OPENKEY\SSSSSS\; файл со справочником LFAX.SPR ищется в подкаталоге: base_dir\FAXKEY\SSSSSS\);

(**i**) alias — указатель на строку атрибутов открытого ключа (длина = 121 символ, последний символ — 0);

(**i**) ser — указатель на строку с номером серии открытого ключа;

(**i**) S_or_E — байт, указывающий на то, в каком справочнике находится ключ: в справочнике ключей шифрования (S_or_E = 'E') или подписи (S_or_E = 'S');

(**o**) open_key_ID — указатель на строку для идентификатора ключа.

Пример вызова функции:

```
char Key_ID[13];
char Alias[]={"Подпись абонента 1 "};
err_code=GetID("open_dir",Alias,"999999",'S', Key_ID);
```

9.3.3 Получение идентификатора ключа, прогруженного в драйвере

Для получения идентификатора ключа, соответствующего считанному в драйвер индивидуальному ключу CKD или CKDI, служит функция GetCurrID(**wbotho**,**wsigno**). Она имеет прототип:

```
T16bit WINAPI GetCurrID (LPSTR curr_ID, char S_or_E);
```

Параметрами функции являются:

(**o**) curr_ID — указатель на строку длиной 11 байт (для ключа шифрования) или 13 байт (для ключа подписи), в которую записывается идентификатор ключа в формате XXXXSSSSSS или XXXXSSSSSSYY;

(**i**) S_or_E — байт, указывающий на тип ключа, идентификатор которого нужно получить: шифрования (S_or_E = 'E') или подписи (S_or_E = 'S').

Пример вызова функции:

```
char sign_id[13];
err_code=GetCurrID(sign_id, 'S');
```

Идентификаторы считываются из слота 0 драйвера(см. **Ошибка! Источник ссылки не найден.**).

9.3.4 Получение идентификатора ключа, хранящегося на сменном носителе

Для чтения и проверки идентификатора ключа, хранящегося на ГМД, служит функция GetFlopID(**wbotho**,**wsigno**), которая имеет прототип:

```
T16bit WINAPI GetFlopID (LPSTR flop_ID, char S_or_E);
```

Параметрами функции являются:

(**o**) flop_ID — указатель на строку длиной 11 байт (для ключа шифрования) или 13 байт (для ключа подписи), в которую записывается идентификатор ключа в формате XXXXSSSSSS или XXXXSSSSSSYY;

(**i**) S_or_E — байт, указывающий тип ключа, идентификатор которого необходимо получить: шифрования (S_or_E = 'E') или подписи (S_or_E = 'S').

Пример вызова функции:

```
char sign_id[13];

err_code=GetFlopID(sign_id, 'S');
```

Имеется дополнительная функция(**wbotho**,**wsigno**), позволяющая получать идентификаторы ключей, хранящихся на различных сменных носителях, поддерживаемых СКЗИ "Верба-OW":

```
T16bit WINAPI GetKeyID (LPSTR key_dev, LPSTR key_ID, char
                S_or_E);
```

Параметрами функции являются:

(**i**) key_dev - строка с именем ключевого носителя (имена поддерживаемых сменных ключевых носителей приведены в файле KEY_DEV.H).

(**o**) key_ID — указатель на строку длиной 11 байт (для ключа шифрования) или 13 байт (для ключа подписи), в которую записывается идентификатор ключа в формате XXXXSSSSSS или XXXXSSSSSSYY;

(**i**) S_or_E — байт, указывающий тип ключа, идентификатор которого необходимо получить: шифрования (S_or_E = 'E') или подписи (S_or_E = 'S').

Пример вызова функции:

Получение идентификатора ключа шифрования, хранящегося на touch-memory, через считыватель Dallas, подключенный к COM1:

```
char crypt_id[11];

err_code=GetKeyID(TM_DS_COM1,crypt_id, 'E');
```

9.3.5 Добавление открытого ключа в справочник

Перед добавлением ключа в справочник проверяется целостность справочника. При обнаружении искажений добавление открытого ключа в справочник не производится.

Если операция добавления открытого ключа в справочник выполняется впервые, т. е. каталог OPENKEY или FAXKEY не существует, он будет создан автоматически со структурой, как указано выше. Если открытый ключ с таким идентификатором уже присутствует в справочнике, то при добавлении он будет заменен.

Для добавления открытого ключа в справочник служит функция AddOpenKey(**wbotho**,**wsigno**), которая имеет прототип:

```
T16bit WINAPI AddOpenKey (LPSTR base_dir,
                LPVOID open_key,
                LPSTR my_ID, char S_or_E);
```

Функция AddOpenKey имеет параметры:

(**i**) base_dir — указатель на строку пути к каталогу OPENKEY или FAXKEY (файл со справочником LPUB.SPR ищется в подкаталоге: base_dir\OPENKEY\SSSSSS\; файл со справочником LFAX.SPR ищется в подкаталоге: base_dir\FAXKEY\SSSSSS\);

(i) open_key — указатель на область памяти, в которой содержится открытый ключ; например, открытый ключ может быть считан в память из файла *.pub, *.lfx или *.spr (см. пример ниже);

(i) my_ID — указатель на строку с идентификатором ключа, соответствующего индивидуальному ключу CKD или CKDI, на котором будет выработана имитовставка для обновленного файла со справочником.

Длина строки с идентификатором ключа шифрования — 11 байт, с идентификатором ключа подписи — 13 байт; формат записи идентификатора: XXXXSSSSSS или XXXXSSSSSSYY соответственно. Если my_ID[0] = '\0', то *индивидуальный ключ* (CKD или CKDI) считывается с ключевой дискеты. В противном случае указанный идентификатор ключа сравнивается с идентификатором ключа, прогруженного в память драйвера, и при их совпадении *индивидуальный ключ* (CKD или CKDI) берется из драйвера.

(i) S_or_E — байт, указывающий на то, в каком справочнике находится ключ: в справочнике ключей шифрования (S_or_E = 'E') или подписи (S_or_E = 'S').

Пример вызова функции:

```
open_key_buff=malloc(304);
FILE * open_key_file=NULL;
 open_key_file=fopen("000101.lfx","rb");
   if(open_key_file==NULL)
    return;
 if(fread(open_key_buff,1,304,open_key_file)!=304)
   {
    fclose(open_key_file);
    return;
   }
 err_code=AddOpenKey("open_dir",open_key_buff,"000199999901",'S');
 ...
```

9.3.6 Удаление открытого ключа из справочника по идентификатору

Перед удалением открытого ключа проверяется целостность справочника открытых ключей и корректность задаваемого идентификатора ключа. При обнаружении искажений функция возвращает соответствующий код ошибки, и удаление не производится.

Для удаления открытого ключа из справочника служит функция DelOpenKey(**wbotho**,**wsigno**), которая имеет прототип:

```
T16bit WINAPI DelOpenKey (LPSTR base_dir,
                          LPSTR open_key_ID,
                          LPSTR my_ID, char S_or_E);
```

Функция DelOpenKey имеет параметры:

(i) base_dir — указатель на строку пути к каталогу OPENKEY или FAXKEY (файл со справочником LPUB.SPR ищется в подкаталоге: base_dir\OPENKEY\SSSSSS\; файл со справочником LFAX.SPR ищется в подкаталоге: base_dir\FAXKEY\SSSSSS\);

(i) open_key_ID — указатель на строку с идентификатором удаляемого ключа.

Для открытого ключа шифрования длина строки с идентификатором должна быть равна 11 байт. Идентификатор ключа должен быть указан в формате XXXXSSSSSS.

Для открытого ключа подписи длина строки с идентификатором должна быть равна 13 байт. Идентификатор ключа должен быть указан в формате XXXXSSSSSSYY.

(**i**) my_ID — указатель на строку с идентификатором ключа, соответствующего индивидуальному ключу CKD или CKDI пользователя; формат записи идентификатора ключа шифрования — XXXXSSSSSS и идентификатора ключа подписи — XXXXSSSSSSYY.

(**i**) S_or_E — байт, указывающий на то, в каком справочнике находится ключ: в справочнике ключей шифрования (S_or_E = 'E') или подписи (S_or_E = 'S').

При удалении открытого ключа из справочника с помощью функции DelOpenKey из файла с имитовставками удаляется запись, соответствующая данному ключу.

Пример вызова функции:

```
err_code=DelOpenKey("open_dir", "000199999901", "000199999901", 'S');
```

9.3.7 Удаление открытого ключа из справочника по его порядковому номеру

Замечание. Порядковый номер открытого ключа можно определить, используя функцию SprList (см. 9.3.9).

Для удаления открытого ключа подписи из справочника по порядковому номеру служит функция DelKeyByNum(**wbotho**,**wsigno**), которая имеет прототип:

```
T16bit WINAPI DelKeyByNum (LPSTR base_dir,
            LPSTR ser,
            T16bit num, char S_or_E);
```

Функция DelKeyByNum имеет параметры:

(**i**) base_dir — указатель на строку пути к каталогу OPENKEY или FAXKEY (файл со справочником LPUB.SPR ищется в подкаталоге: base_dir\OPENKEY\SSSSSS\; файл со справочником LFAX.SPR ищется в подкаталоге: base_dir\FAXKEY\SSSSSS\);

(**i**) ser — указатель на строку с номером серии открытого ключа, который удаляется из справочника;

(**i**) num — порядковый номер открытого ключа в справочнике;

(**i**) S_or_E — байт, указывающий на то, в каком справочнике находится ключ: в справочнике ключей шифрования (S_or_E = 'E') или подписи (S_or_E = 'S').

При удалении открытого ключа из справочника по его порядковому номеру идентификатор открытого ключа не проверяется на наличие искажений. Для того чтобы корректно удалить ключ из справочника, нужно дополнительно удалить из файла имитовставок имитовставку на данный открытый ключ (см. 9.3.8).

Пример вызова функции:

```
err_code=DelKeyByNum("open_dir", "999999", 1, 'S');
```

9.3.8 Удаление имитовставки на открытый ключ по его порядковому номеру в справочнике

Для удаления имитовставки на открытый ключ по его порядковому номеру в справочнике служит функция DelImitByNum(**wbotho**,**wsigno**). Прототип имеет вид:

```
T16bit WINAPI DelImitByNum (LPSTR base_dir,
            LPSTR ser,
            T16bit num, LPSTR my_ID,
            char S_or_E);
```

Функция DelImitByNum имеет параметры:

(**i**) base_dir — указатель на строку пути к каталогу OPENKEY или FAXKEY (файл со справочником LPUB.SPR ищется в подкаталоге: base_dir\OPENKEY\SSSSSS\; файл со справочником LFAX.SPR ищется в подкаталоге: base_dir\FAXKEY\SSSSSS\);

(**i**) ser — указатель на строку с номером серии открытого ключа, который удаляется из справочника;

(**i**) num — порядковый номер открытого ключа в справочнике;

(**i**) my_ID — указатель на строку с идентификатором ключа, соответствующего индивидуальному ключу CKD или CKDI пользователя; формат записи идентификатора ключа шифрования — XXXXSSSSSS и ключа подписи — XXXXSSSSSSYY.

Если my_ID[0] = '\0', то *индивидуальный ключ* (CKD или CKDI) считывается с ключевой дискеты. В противном случае указанный идентификатор ключа сравнивается с идентификатором ключа, прогруженного в память драйвера, и при их совпадении *индивидуальный ключ* (CKD или CKDI) берется из драйвера.

(**i**) S_or_E — байт, указывающий на то, в каком справочнике находится ключ: в справочнике ключей шифрования (S_or_E = 'E') или подписи (S_or_E = 'S').

Пример вызова функции:

```
err_code=DelImitByNum("open_dir", "999999", 1, "000199999901", 'S');
```

9.3.9 Получение списка открытых ключей справочника

Для получения списка открытых ключей справочника (файл LPUB.SPR для открытых ключей шифрования или файл LFAX.SPR для открытых ключей подписи) служит функция SprList(**wbotho**,**wsigno**), которая имеет прототип:

```
T16bit WINAPI SprList (LPSTR base_dir,
                       LPSTR ser,
                       Spr_List_Ptr * list,
                       P16bit num, char S_or_E);
```

Функция SprList имеет параметры:

(**i**) base_dir — указатель на строку пути к каталогу OPENKEY или FAXKEY (файл со справочником LPUB.SPR ищется в подкаталоге: base_dir\OPENKEY\SSSSSS\; файл со справочником LFAX.SPR ищется в подкаталоге: base_dir\FAXKEY\SSSSSS\);

(**i**) ser — указатель на строку с номером серии справочника открытых ключей, совпадающий с именем подкаталога SSSSSS;

(**o**) list — массив с информацией об открытых ключах, элементами которого являются структуры(структура описана в файле verba.h):

```
typedef struct tagSpr_List
 { char key_id[13];
        char key_type;
        char key_status;
 } Spr_List;
```

Элементами этого массива являются:

- идентификатор открытого ключа;
- тип ключа;
- результат проверки открытого ключа :

KEY_OK (0)

BAD_HASH (1) значение хэш-функции не совпадает с вычисленным

BAD_SER (2) серия открытого ключа не совпадает с серией справочника

Внимание. Память под этот массив отводится внутри функции SprList и должна быть освобождена функцией FreeMemory (см. п. 8.12).

(**o**) num — указатель на переменную, в которую записывается количество ключей, обнаруженных в справочнике;

(**i**) S_or_E — байт, указывающий на то, список какого справочника необходимо получить: справочника ключей шифрования (S_or_E = 'E') или подписи (S_or_E = 'S').

Пример вызова функции:

```
T16bit num;
Spr_List *list;
err_code= SprList("open_dir", "999999", &list, &num, 'S');
```

9.3.10 Проверка открытого ключа

При проверке открытого ключа проверяются имитовставка, значение хэш-функции для данного ключа и его структура.

Для проверки открытого ключа служит функция CheckOpenKey(**wbotho**,**wsigno**), которая имеет прототип:

```
T16bit WINAPI CheckOpenKey (LPSTR base_dir,
                LPSTR open_key_ID,
                LPSTR my_ID, char S_or_E);
```

Функция CheckOpenKey имеет параметры:

(**i**) base_dir — указатель на строку пути к каталогу OPENKEY или FAXKEY (файл со справочником LPUB.SPR ищется в подкаталоге: base_dir\OPENKEY\SSSSSS\; файл со справочником LFAX.SPR ищется в подкаталоге: base_dir\FAXKEY\SSSSSS\);

(**i**) open_key_ID — указатель на строку с идентификатором проверяемого ключа.

Для открытого ключа шифрования длина строки с идентификатором должна быть равна 11 байтам. Идентификатор ключа должен быть указан в формате XXXXSSSSSS.

Для открытого ключа подписи длина строки с идентификатором должна быть равна 13 байтам. Идентификатор ключа должен быть указан в формате XXXXSSSSSSYY.

(**i**) my_ID — указатель на строку с идентификатором ключа, соответствующего индивидуальному ключу CKD или CKDI, на котором будет выработана имитовставка для обновленного файла со справочником.

Длина строки с идентификатором ключа шифрования — 11 байт, с идентификатором ключа подписи — 13 байт; формат записи идентификатора: XXXXSSSSSS или XXXXSSSSSSYY соответственно. Если my_ID[0] = '\0', то *индивидуальный ключ* (CKD или CKDI) считывается с ключевой дискеты. В противном случае указанный идентификатор ключа сравнивается с идентификатором ключа, прогруженного в память драйвера, и при их совпадении *индивидуальный ключ* (CKD или CKDI) берется из драйвера.

(**i**) S_or_E — байт, указывающий на то, в каком справочнике находится ключ: в справочнике ключей шифрования (S_or_E = 'E') или подписи (S_or_E = 'S').

Пример вызова функции:

```
err_code=CheckOpenKey("open_dir","000199999901","000199999901",'S');
```

9.3.11 Проверка открытого ключа по порядковому номеру

Для проверки открытого ключа подписи по порядковому номеру служит функция CheckKeyByNum(**wbotho**,**wsigno**), которая имеет прототип:

```
T16bit WINAPI CheckKeyByNum (LPSTR base_dir,
                LPSTR ser,
                T16bit num, LPSTR my_ID,
                char S_or_E);
```

Функция CheckKeyByNum имеет параметры:

(i) base_dir — указатель на строку пути к каталогу OPENKEY или FAXKEY (файл со справочником LPUB.SPR ищется в подкаталоге: base_dir\OPENKEY\SSSSSS\; файл со справочником LFAX.SPR ищется в подкаталоге: base_dir\FAXKEY\SSSSSS\);

(i) ser — указатель на строку с номером серии проверяемого открытого ключа;

(i) num — порядковый номер проверяемого открытого ключа в справочнике;

(i) my_ID — указатель на строку с идентификатором ключа пользователя, соответствующего индивидуальному ключу CKD или CKDI, на котором будет проверена имитовставка для обновленного файла со справочником.

Длина строки с идентификатором ключа шифрования — 11 байт, с идентификатором ключа подписи — 13 байт; формат записи идентификатора: XXXXSSSSSS или XXXXSSSSSSYY соответственно. Если my_ID[0] = '\0', то *индивидуальный ключ* (CKD или CKDI) считывается с ключевой дискеты. В противном случае указанный идентификатор ключа сравнивается с идентификатором ключа, прогруженного в память драйвера, и при их совпадении *индивидуальный ключ* (CKD или CKDI) берется из драйвера.

(i) S_or_E — байт, указывающий на то, в каком справочнике находится ключ: в справочнике ключей шифрования (S_or_E = 'E') или подписи (S_or_E = 'S').

Пример вызова функции:

```
err_code=CheckKeyByNum("open_dir","999999",1,"000199999901",'S');
```

9.3.12 Проверка хэш-значения на открытый ключ

Для проверки хэш-значения открытого ключа служит функция CheckKeyHash(**wbotho**,**wsigno**), которая имеет прототип:

```
T16bit WINAPI CheckKeyHash (LPVOID open_key,
                LPVOID hash_adr);
```

Функция CheckKeyHash имеет параметры:

(i) open_key — указатель на область памяти, в которой содержится открытый ключ; например, открытый ключ может быть считан в память из файла *.pub или *.lfx;

(i) hash_adr — указатель на область памяти длиной 32 байта, где находится значение хэш-функции, или NULL (в этом случае считается, что параметр open_key — указывает на сертификат открытого ключа, и хэш-значение находится внутри сертификата).

Пример вызова функции:

```
/* функция может быть использована при вводе открытого ключа в справочник с клавиатуры */
open_key_buff=(char*)malloc(304);
/* ввод сертификата открытого ключа в open_key_buff*/
  ...
/* хэш-функция находится внутри сертификата */
err_code=CheckKeyHash(open_key_buff, NULL)
```

9.3.13 Выработка имитовставки для справочника

Целостность справочника контролируется с помощью файла с имитовставками, которые формируются для каждого открытого ключа. Для выработки имитовставок для открытых ключей справочника служит функция SignSpr(**wbotho**,**wsigno**), которая имеет прототип:

```
T16bit WINAPI SignSpr (LPSTR base_dir,
                       LPSTR ser,
                       LPSTR my_ID, char S_or_E);
```

Функция SignSpr имеет параметры:

(**i**) base_dir — указатель на строку пути к каталогу OPENKEY или FAXKEY (файл со справочником LPUB.SPR ищется в подкаталоге: base_dir\OPENKEY\SSSSSS\; файл со справочником LFAX.SPR ищется в подкаталоге: base_dir\FAXKEY\SSSSSS\);

(**i**) ser — указатель на строку с номером серии открытого ключа, на котором подписывается справочник;

(**i**) my_ID — указатель на строку с идентификатором ключа, соответствующего индивидуальному ключу CKD или CKDI, на котором будет выработана имитовставка для обновленного файла со справочником.

Длина строки с идентификатором ключа шифрования — 11 байт, с идентификатором ключа подписи — 13 байт; формат записи идентификатора: XXXXSSSSSS или XXXXSSSSSSYY соответственно. Если my_ID[0] = '\0', то *индивидуальный ключ* (CKD или CKDI) считывается с ключевой дискеты. В противном случае указанный идентификатор ключа сравнивается с идентификатором ключа, прогруженного в память драйвера, и при их совпадении *индивидуальный ключ* (CKD или CKDI) берется из драйвера.

(**i**) S_or_E — байт, указывающий на то, для какого справочника формируется файл с имитовставками: справочника ключей шифрования (S_or_E = 'E') или подписи (S_or_E = 'S').

Пример вызова функции:

```
err_code=SignSpr("open_dir", "999999", "000199999901, 'S');
```

9.3.14 Проверка целостности справочника

Для проверки целостности справочника открытых ключей служит функция CheckSpr(**wbotho**,**wsigno**), которая имеет прототип:

```
T16bit WINAPI CheckSpr (LPSTR base_dir,
                        LPSTR ser,
                        Spr_List_Ptr * list, P16bit num,
                        LPSTR my_ID, char S_or_E);
```

Функция CheckSpr имеет параметры:

(**i**) base_dir — указатель на строку пути к каталогу OPENKEY или FAXKEY (файл со справочником LPUB.SPR ищется в подкаталоге: base_dir\OPENKEY\SSSSSS\; файл со справочником LFAX.SPR ищется в подкаталоге: base_dir\FAXKEY\SSSSSS\);

(**i**) ser — указатель на строку с номером серии справочника открытых ключей, совпадающий с именем подкаталога SSSSSS;

(**o**) list — массив с информацией об открытых ключах, элементами которого являются структуры:

```
typedef struct tagSpr_List
 { char key_id[13];
       char key_type;
       char key_status;
 } Spr_List;
```

Элементами этого массива являются:

- идентификатор открытого ключа;
- тип ключа;
- результат проверки открытого ключа:

KEY_OK	(0)
BAD_HASH	(1) значение хэш-функции не совпадает с вычисленным
BAD_SER	(2) серия открытого ключа не совпадает с серией справочника
BAD_IMM	(4) неверная имитовставка на открытый ключ
NO_IMM	(8) нет имитовставки на открытый ключ.

Внимание. Память под этот массив отводится внутри функции SprList и должна быть освобождена функцией FreeMemory (см. п. 8.12).

(**i**) my_ID — указатель на строку с идентификатором ключа, соответствующего индивидуальному ключу CKD или CKDI пользователя.

Длина строки с идентификатором ключа шифрования — 11 байт, с идентификатором ключа подписи — 13 байт; формат записи идентификатора: XXXXSSSSSS или XXXXSSSSSSYY соответственно. Если my_ID[0] = '\0', то *индивидуальный ключ* (CKD или CKDI) считывается с ключевой дискеты. В противном случае указанный идентификатор ключа сравнивается с идентификатором ключа, прогруженного в память драйвера, и при их совпадении *индивидуальный ключ* (CKD или CKDI) берется из драйвера.

(**o**) num — указатель на переменную, в которую записывается количество ключей, обнаруженных в справочнике;

(**i**) S_or_E — байт, указывающий на то, целостность какого справочника проверяется: справочника ключей шифрования (S_or_E = 'E') или подписи (S_or_E = 'S').

Пример вызова функции:

```
T16bit num;
Spr_List *list;
err_code=CheckSpr("open_dir","999999",&list,&num,"000199999901",'S');
```

9.3.15 Считывание открытого ключа из справочника в память

Для считывания открытого ключа из справочника в память служит функция ExtractKey(**wbotho**,**wsigno**), прототип которой имеет вид:

```
T16bit WINAPI ExtractKey (LPSTR base_dir, LPSTR open_key_ID,
                          LPVOID key);
```

Параметрами функции являются:

(**i**) base_dir — указатель на строку пути к каталогу OPENKEY или FAXKEY (файл со справочником LPUB.SPR ищется в подкаталоге: base_dir\OPENKEY\SSSSSS\; файл со справочником LFAX.SPR ищется в подкаталоге: base_dir\FAXKEY\SSSSSS\);

(**i**) open_key_ID — указатель на строку с идентификатором ключа.

Идентификатор открытого ключа шифрования должен быть указан в формате XXXXSSSSSS.

Идентификатор открытого ключа подписи должен быть указан в формате XXXXSSSSSSYY.

В данной функции явно не указывается, в справочнике подписи или шифрования хранится открытый ключ, это определяется по идентификатору ключа;

(**o**) key —указатель на блок памяти размером 304 байт, в который считывается данный открытый ключ (отводится пользователем).

Пример вызова функции:

```
open_key_buff=(char*)malloc(304);
```

```
err_code=ExtractKey("open_dir", "000199999901", open_key_buff);
```

9.3.16 Перевод открытого ключа из резервного в действующий

Для смены типа ключа с резервного на действующий служит функция Res2Work(**wbotho**,**wsigno**), которая имеет прототип:

```
T16bit WINAPI Res2Work (LPSTR base_dir,
            LPSTR open_key_ID,
            LPSTR my_ID, char S_or_E);
```

Параметрами данной функции являются:

(i) base_dir — указатель на строку пути к каталогу OPENKEY или FAXKEY (файл со справочником LPUB.SPR ищется в подкаталоге: base_dir\OPENKEY\SSSSSS\; файл со справочником LFAX.SPR ищется в подкаталоге: base_dir\FAXKEY\SSSSSS\);

(i) open_key_ID — указатель на строку с идентификатором ключа.

Идентификатор открытого ключ шифрования должен быть указан в формате XXXXSSSSSS.

Идентификатор открытого ключа подписи должен быть указан в формате XXXXSSSSSSYY.

(i) my_ID — указатель на строку с идентификатором ключа пользователя, соответствующего индивидуальному ключу CKD или CKDI, на котором будет выработана имитовставка для нового действующего ключа.

Длина строки с идентификатором ключа шифрования — 11 байт, с идентификатором ключа подписи — 13 байт; формат записи идентификатора: XXXXSSSSSS или XXXXSSSSSSYY соответственно. Если my_ID[0] = '\0', то *индивидуальный ключ* (CKD или CKDI) считывается с ключевой дискеты. В противном случае указанный идентификатор ключа сравнивается с идентификатором ключа, прогруженного в память драйвера, и при их совпадении *индивидуальный ключ* (CKD или CKDI) берется из драйвера.

(i) S_or_E — байт, указывающий на то, в каком справочнике находится ключ: в справочнике ключей шифрования (S_or_E = 'E') или подписи (S_or_E = 'S').

Пример вызова функции:

```
err_code=Res2Work("open_dir","000199999901","000199999901",'S');
```

9.3.17 Проверка целостности открытого ключа

Для проверки целостности открытого ключа и получения его типа служит функция CheckOpenKeyBuff (**wbotho**,**wsigno**), которая имеет прототип:

```
T16bit WINAPI CheckOpenKeyBuff( void *open_key,void *hash,char
            *key_type);
```

Параметрами данной функции являются:

(i) open_key — указатель на буфер с открытым ключом или регистрационной записью открытого ключа.

(i) hash — указатель на буфер с хэш-функцией открытого ключа. Если в этом параметре передаётся NULL, то считается, что параметр open_key указывает на регистрационную запись открытого ключа длиной 240 или 304 байта и хэш-функция ищется в конце регистрационной записи. В случае несовпадения рассчитанного и переданного значений хэш-функции возвращается код ошибки ERR_HASH.

(o) key_type — указатель на байт, в котором возвращается тип открытого ключа: ‘E’ – рабочий ключ шифрования, ‘S’ – рабочий ключ подписи, ‘C’ – скомпрометированный ключ.

9.4 Дополнительные функции для обеспечения доступа к закрытым ключам на ЖМД

Как было отмечено выше, ключевая система СКЗИ "Верба-OW" обеспечивает возможность хранения закрытых ключей шифрования и подписи в файлах на ЖМД. При этом сами файлы с ключами хранятся в зашифрованном виде. Ключи для их расшифрования находятся на сменных ключевых носителях (ГМД, "таблетках" Touch-Memory, смарт-картах) и для обеспечения доступа к ключа на ЖМД должны быть загружены в драйвер ASYNCR (см. 5.6 и 6.2.1). В драйвере ASYNCR зарезервировано 16 "слотов", в каждый из которых может быть загружена ключевая информация с одного ключевого носителя. Загрузка и выгрузка ключевой информация может производиться либо посредством программы ASRKEYW.EXE, либо с помощью библиотечных функций.

Примечание. Пользователю не требуется явно задавать номера слотов. Функции библиотеки автоматически ищут нужный слот, основываясь на информации об идентификаторах ключей.

Наличие нескольких слотов и функций работы с ними обеспечивает работу одного пользователя в нескольких сетях и/или работу нескольких пользователей с одним и тем же приложением, а также оперативное управление доступом к ключам. Данные свойства является полезными при построении приложений типа абонентских пунктов или серверов.

В пользовательском приложении вместе с каждой функцией загрузки ключей в некоторый слот в обязательном порядке должна присутствовать парная ей функция выгрузки ключей из данного слота. Выполнение данного требования является мерой защиты от несанкционированных действий, направленных на получение информации о ключах.

Аналогичное требование должно выполняться при использовании программы AsrkeyW. Это означает, что после завершения работы приложения следует выгрузить индивидуальные ключи из памяти с помощью опций программы.

9.4.1 Пользовательские функции

Для работы этих функций необходимо предварительно с помощью программы ASRKEYW.EXE инициализировать ДСЧ.

9.4.2 Загрузка ключей

Загрузка ключей осуществляется функцией(**wbotho**,**wsigno**):

```
T16bit WINAPI InitKey (char *Key_Dev,char *Key_ID);
```

Параметры:

(**i**) Key_Dev - строка с именем ключевого носителя ("A:\\" или "B:\\" - для ключевого ГМД, имена остальных поддерживаемых сменных ключевых носителей приведены в файле KEY_DEV.H).

(**i**) Key_ID - строка, содержащая идентификатор ключа шифрования или подписи, либо пустая строка (Key_ID[0] == '\0'). Функция ищет пустой "слот" в драйвере и загружает в него информацию с ключевого носителя. Поиск свободного слота осуществляется по возрастанию номера.

Если идентификатор ключа задан явно, то в зависимости от типа идентификатора (XXXXSSSSSS или XXXXSSSSSSYY) загружается информация, обеспечивающая доступ либо к ключам шифрования, либо к ключам подписи, хранящимся на ЖМД. Загружаемые ключи могут быть также и *ключами администраторов*, т.е. обеспечивать доступ к группам зашифрованных на них ключей шифрования.

Если задана пустая строка, то загружается ключевая информация и для шифрования, и для подписи (в случае совмещенного ключевого носителя).

Функция запрещает повторную загрузку ключей с одинаковыми идентификаторами, возвращая в этом случае код ошибки E_REDEFINE. Если все 16 слотов уже заняты, то возвращается код ошибки E_NO_FREE_SLOTS. Функция InitKey не требует предварительного выполнения функции CryptoInit или SignInit и может быть использована для завершения инициализации ДСЧ (см. п. 6.2.1)

9.4.3 Выгрузка ключей из драйвера

Выгрузка ключей из драйвера осуществляется функциями(**wbotho**,**wsigno**):

```
T16bit WINAPI ResetKey(char *Key_ID)

T16bit WINAPI ResetKeyEx(char *Key_ID,BOOL flag);
```

Параметры:

(**i**) Key_ID — строка, содержащая идентификатор ключа шифрования или подписи.

Функции выгружают всю информацию из слота, в котором найден ключ с заданным идентификатором. С помощью функции **ResetKey** могут быть выгружены слоты 1-15. Для выгрузки информации из любого слота (0-15) используется функции **ResetKeyEx** с параметром flag установленным в TRUE. Если параметр flag установлен в FALSE и выгружаемый ключ находится в слоте 0, то функция **ResetKeyEx** вернет код E_KEY_NOT_FOUND и выгрузка произведена не будет.

9.4.4 Получение списка ключей, загруженных в драйвер

Получение списка ключей, загруженных в драйвер, производится при помощи функции(**wbotho**,**wsigno**):

```
T16bit WINAPI GetDrvInfo(USR_KEYS_INFO *Keys_Info,
                         unsigned long *n_Key_Slot);
```

Параметры:

(**o**) Keys_Info - указатель на массив из 16 структур USR_KEYS_INFO, куда будет возвращена информация о слотах драйвера, в которые прогружены ключи для шифрования и/или подписи (CKD и/или CKDI). Память под массив резервируется пользователем.

(**o**) n_Key_Slot - возвращаемое количество загруженных слотов (может быть равно нулю, в том случае, когда была произведена выгрузка всех ключей).

Функция в начале работы обнуляет весь массив структур Keys_Info, а затем последовательно заполняет его информацией о найденных загруженных слотах в порядке возрастания их номеров. (В Keys_Info[0] записывается информация о первом найденном загруженном слоте, в Keys_Info[1] — о втором, и т. д.)

При работе с несколькими ключами следует учитывать, что:

- контроль справочников открытых ключей подписи и шифрования в операциях проверки подписи и шифрования осуществляется на ключах из слота 0. (Имитовставки на справочники открытых ключей подписи и шифрования рассчитываются на ключах CKDI и CKD из слота 0).
- Под текущей серией ("\0\0\0\0\0\0") в функциях шифрования подразумевается серия ключей из слота 0.
- если в функциях подписи задан неполный идентификатор ключа подписи "nnnn\0\0\0\0\0\0\0\0" или "nnnnssssss\0\0", то недостающая информация о серии и/или личном коде берется из слота 0.

Пример.

```
{
...
USER_KEYS_INFO Keys_Info[16];
unsigned long nKeySlot,i;
short err_code;
...
    err_code=CryptoInit("c:\\key\\",NULL);
...
    err_code=SignInit("c:\\key\\","c:\\sertif\\”);
```

```
...
/* Загрузка ключевой информации для шифрования и подписи */
   printf("Insert key_diskette and press any key...\n");
   getch();
   err_code=InitKey("A:\\","");
...
/* Загрузка ключевой информации для шифрования */
   printf("Insert key_diskette and press any key...\n");
   getch();
   err_code=InitKey("A:\\","0001222222");
...
/* Загрузка ключевой информации для подписи */
   printf("Insert key_diskette and press any key...\n");
   getch();
   err_code=InitKey("A:\\","000344444401");
...
   err_code=GetDrvInfo(Keys_Info,&nKeySlot);
...
/* Выгрузка всех ключей, прогруженных функциями InitKey */
   for(i=1;i<nKeySlot;i++)
    {
     if(Keys_Info[i].num[0])   /* если в слоте есть ключи для
шифрования */
       err_code=ResetKeyEx(Keys_Info[i].num,TRUE);
     else
      {
       if(Keys_Info[i].nump[0]) /* если в слоте есть ключи для подписи
*/
        err_code=ResetKeyEx(Keys_Info[i].nump,TRUE);
      }
     if(err_code)
      break;
    }
...
    CryptoDone();
    SignDone();
    return;
}
```

10. ФУНКЦИИ ГЕНЕРАЦИИ КЛЮЧЕЙ.

В библиотеках СКЗИ «Верба –OW» версии 6 имеется возможность генерации закрытых и открытых ключей на основе информации, считанной со специального ключевого ГМД с лицензией, определяющей диапазон номеров генерируемых ключей и задающей ключевую серию. Лицензия на генерацию ключей приобретается отдельно. Лицензионный ГМД имеет формат аналогичный формату лицензионного ГМД АРМ АБ-О для DOS.

10.1 Загрузка лицензии

Для загрузки лицензионного ГМД и получении информации о диапазоне допустимых номеров генерируемых ключей служит функция LoadLicense (**wbotho**,**wsigno**). Эту функцию необходимо вызвать перед функцией генерации ключей. Перед вызовом функции должен быть проинициализирован ДСЧ. Прототип функции имеет следующий вид:

```
T16bit WINAPI    LoadLicense(char *dev, LIC_INFO **info,
                             T32bit *nrecords, char *seria,
                             T32bit reserved);
```

Параметрами данной функции являются:

(i) dev – ключевое устройство с лицензией (“a:” или “b:”)

(o) info – указатель на массив структур с информацией о диапазоне допустимых номеров генерируемых ключей. Каждый элемент массива задаёт начальный и конечный номера в группе ключей шифрования и количество ключей подписи, для каждого ключа шифрования из данной группы. В зависимости от лицензии диапазон номеров ключей шифрования для заданной ключевой серии может быть от 1 до 9999. Память под массив выделяется самой функцией и освобождается при выгрузке лицензии.

(o) nrecords – количество записей в массиве (количество групп ключей шифрования)

(o) seria – серия ключей

(io) reserved – зарезервированный параметр, должен быть равен 0.

10.2 Выгрузка лицензии

Для выгрузки лицензии и освобождения памяти служит функция UnloadLicense (**wbotho**,**wsigno**):

```
T16bit WINAPI UnloadLicense();
```

10.3 Генерация ключей

Для генерации на ключевой носитель закрытых ключей подписи и шифрования и формирования соответствующих им открытых ключей предназначена функция GenKeys (**wbotho**, **wsigno**):

```
T16bit WINAPI    GenKeys(char *media,
                         T32bit,
                         char *key_num,
                         BYTE *pub_sign,
                         BYTE *pub_crypt,
                         char *pub_text,
                         char *alg_id,
                         void *reserved);
```

Функция генерирует ключ шифрования XXXXSSSSSS и(или) ключ подписи XXXXSSSSSSYY

Параметрами данной функции являются:

(**i**) media - Имя ключевого носителя на который будут записаны закрытые ключи

(**io**) flags – Флаги, задающие тип генерируемых ключей. Данный параметр может принимать следующие значения: VERBA_KEY_GEN_SIGN - генерация ключа подписи; VERBA_KEY_GEN_CRYPT - генерация ключа шифрования; VERBA_KEY_GEN_BOTH - генерация ключа подписи и ключа шифрования.

(**o**) pub_sign – Указатель на буфер, в который нужно записать открытый ключ подписи.

(**o**) pub_crypt - Указатель на буфер, куда в который нужно записать открытый ключ шифрования.

(**i**) pub_text - Указатель на буфер с текстовой информацией, которая будет записана в поле атрибутов открытого ключа.

(**i**) pkey_id - Номер генерируемого ключ Указатель на 13-байтовый буфер с номером генерируемого ключа в формате XXXXSSSSSSYY.

(**i**) key_syst Тип ключевой системы. Параметр должен иметь значение ALG_VERBA_O для генерации ключей с использованием алгоритма ГОСТ Р 34.10-94 и ALG_3410_01 для ГОСТ Р 34.10-2001.

Пример

См пп.10.4

10.4 Генерация ключей, предназначенных для хранения на ЖМД.

Для формирования ключей, предназначенных для хранения на ЖМД нужно использовать функцию (**wbotho**, **wsigno**):

```
T16bit WINAPI verba_key_set(
    char* media, /*+ (i) Путь к носителю. +*/
    NULL,
    int* flags, /*+ (io) Флаги установки ключей. +*/
    char* path_hd, /*+ (i) Путь для устанавливаемых ключей. +*/
    char* encr_id); /*+ (i) Номер ключа перешифрования. +*/
```

Параметры функции:

(i) media – Путь к ключевому носителю с закрытыми ключами, на основе которых будут сформированы ключи для хранения на ЖМД.

(io) flags – Флаги установки ключей. Должен быть установлен флаг VERBA_KEY_SET_BOTH_ENCRYPTED.

(i) path_hd – Путь к каталогу на ЖМД в который будут записаны сформированные ключи.

(i) encr_id – номер ключа, на котором будут перешифрованы ключи для хранения на ЖМД. Ключ для перешифрования должен быть загружен в драйвер Asyncr.

Пример:

```
…
char key_num[20];
char seria[7];
char pub_sign[304];
char pub_crypt[304];
char   pub_text[120]="test key";
int nrecords,flags;
unsigned short    err_code;
LIC_INFO *info;
…
 err_code = LoadLicense("a:\\", &info, &nrecords, seria, 0);
 if(err_code)
       return;
//сформировать ключ с первым из допустимых номеров
 memset(key_num,0, sizeof(key_num));
 sprintf(key_num,"%04d",info[0].first_crypt);
 strcat(key_num, seria);
 sprintf(&key_num[strlen(key_num)],"%02d",info[0].sign_nums);

 flags = VERBA_KEY_GEN_CRYPT |VERBA_KEY_GEN_SIGN;
 err_code = GenKeys("TM:COM0", flags, key_num,
                            pub_sign,pub_crypt,pub_text,
                            ALG_VERBA_O,NULL);
 UnLoadLicense();
```

```
 if(err_code)
       return ;
/* сформировать ключи для хранения на ЖМД и записать их в каталог KEYS
*/
 err_code=InitKey("a:\\",key_num);
 if(err_code)
       return ;
flags=VERBA_KEY_SET_BOTH_ENCRYPTED;
err_code=verba_key_set(
    "TM:COM0",
    NULL,
    &flags,
    "KEYS",
    key_num,
  );
…
```

10.5 Получение параметров алгоритмов формирования/проверки ЭЦП.

Для получения значений криптографических констант, используемых в процедурах формирования/проверки ЭЦП нужно использовать функцию (**wbotho**,**wsigno**):

```
T16bit WINAPI GetAlgParam(
    BYTE* pbData, /* (i) Входные данные, для идентификации
            параметров */
    T32bit dwDataLen, /* (i) Длина ввходных данных.*/
    T32bit dwDtype, /* (i) Тип входных данных. */
    BYTE* pbParam, /* (o) Возвращаемые параметры алгоритма. */
    T32bit* pdwParamLen, /*+ (io) Длина блока параметров. */
    T32bit dwFlags); /* (i) Флаги. */
```

Параметры функции:

(**i**) pbData – Указатель на блок данных для идентификации параметров.

(**io**) dwDataLen – Длина блока данных.

(**i**) dwDtype – Тип данных для идентификации параметров. Может принимать следующие значения:

VERBA_ALG_ID – входные данные – структура типа ALG_PARAM_ID;

VERBA_PUB_KEY – входные данные – открытый ключ (в формате *.lfx)

(**i**) pbParam – Указатель на блок данных с параметрами алгоритма. Память под данные распределяется пользователем. Для определения размера памяти можно передать в этом параметре NULL, тогда в параметре dwParamLen будет возвращён необходимый размер.

(i) pdwParamLen – Указатель на переменную с длиной блока данных с параметрами.

(i) dwFlags – Флаги.

Пример:

…

```
ALG_PARAM   param; // структура для параметров алгоритма
BYTE pub_sign[304];
DWORD  pdwParamLen = sizeof(param);


err_code=ExtractKey(CRYPT_SPR,"0001910000",pub_sign);
if(err_code) goto error;
err_code = GetAlgParam(pub_sign, sizeof(pub_sign), VERBA_PUB_KEY,
(BYTE*)param, &pdwParamLen, 0);
```

ПРИЛОЖЕНИЕ 1. СПИСОК КОДОВ ВОЗВРАТА

СПИСОК КОДОВ ВОЗВРАТА ФУНКЦИЙ БИБЛИОТЕК WSIGNO, WBOTHO

Таблица 3. Коды возврата

Код возврата	**Обозначение**	**Описание**
0	NO_ERROR	Нет ошибки. Функция завершилась успешно.
1	E_NO_MEM	Не хватает динамической памяти.
2	E_CONTROL	Сбой криптографической функции или искажение тела библиотеки.
3	E_DRIVER	Ошибка датчика случайных чисел.
4	E_IMMITO	Не совпадает имитовставка — файл (блок памяти) искажен.
6	E_KEY_NOT_FOUND	Ключ не найден (или искажен).
7	E_PARAM	Ошибка параметра обращения к функции.
8	E_INIT	Ошибка инициализации.
10	E_MEM_LENGTH	Неверная длина блока памяти.
11	E_MEM_NOT_ENCRYPTED	Попытка расшифровать незашифрованный блок памяти.
12	E_MEM_NOT_SIGNED	Попытка проверить подпись неподписанного блока памяти.
21	E_OPEN_IN_FILE	Ошибка открытия входного файла.
22	E_OPEN_OUT_FILE	Ошибка открытия выходного файла.
23	E_WRITE_FILE	Ошибка записи файла.
24	E_READ_FILE	Ошибка чтения файла.
25	E_RENAME_FILE	Ошибка переименования файла.
26	E_FILE_LENGTH	Неверная (например, нулевая) длина файла.
27	E_SRC	Несовпадение контрольной суммы зашифрованного файла.
29	E_FILE_NOT_ENCRYPTED	Попытка расшифрования незашифрованного файла.
30	E_FILE_NOT_SIGNED	Попытка проверки подписи неподписанного файла.
31	E_SEEK	Ошибка смещения файлового указателя.
32	E_CLOSE	Ошибка закрытия файла.
33	E_DELETE_FILE	Ошибка удаления файла.
34	E_GK	Ошибка при чтении GK.
35	E_KC	Ошибка при чтении KC.
36	E_DEVICE	Ошибка при обращении к сменному ключевому устройству.
37	E_REDEFINE	Попытка перезаписи ключа в драйвер ASYNCR.
38	E_NO_FREE_SLOTS	В драйвере ASYNCR нет свободных "слотов".
39	E_KEY_NOT_SET	Ошибка при загрузке ключа в драйвер ASYNCR.

101	ERR_NUMP	Номер ключа NUM или NUMP не соответствует считанному из драйвера Asyncr
102	ERR_HASH	Значение хэш-функции не совпало.
103	ERR_OPEN_SPR	Ошибка при открытии файла со справочником открытых ключей.
104	ERR_OPEN_IMM	Ошибка при открытии файла с имитовставками.
105	ERR_UZ	Ошибка чтения UZ.
106	ERR_CKD	Ошибка чтения CKD или CKDI.
107	ERR_IMM_SPR	Длина файла со справочником не соответствует длине файла с имитовставками.
108	ERR_READ_SPR	Ошибка чтения файла со справочником открытых ключей.
109	ERR_WRITE_SPR	Ошибка записи в файл со справочником открытых ключей.
110	ERR_READ_IMM	Ошибка чтения файла с имитовставками.
111	ERR_IMM	Имитовставка неверна.
112	ERR_COMPROM	Открытый ключ скомпрометирован.
113	ERR_CRE_DIR	Ошибка при создании каталога.
114	ERR_CRE_FILE	Ошибка при создании файла *.imm, *.imp, или *.spr.
115	ERR_EXIST_SPR	В заданном каталоге уже существует файл *.spr.
116	ERR_WRITE_IMM	Ошибка записи в файл имитовставок.
117	ERR_NO_KEY	Указанный открытый ключ отсутствует в справочнике.
118	ERR_LENGTH	Неверная длина файла *.imm, *.imp, или *.spr.
119	ERR_OPEN_TMP	Ошибка открытия временного файла.
120	ERR_SPR_EMPTY	Справочник открытых ключей пуст.
121	ERR_KEY_HEAD	Заголовок открытого ключа искажен.
122	ERR_FIND_SPR	Справочник открытых ключей не найден.
123	ERR_NO_RES	Открытый ключ не является резервным.
124	ERR_IMM_HEAD	Заголовок файла с имитовставками искажен.
125	ERR_NO_SIGN	Нет имитовставки на открытый ключ.
126	ERR_NO_IMM	Нет имитовставки на открытый ключ.
127	ERR_FLOP	Ошибка при обращении к гибкому диску.

ПРИЛОЖЕНИЕ 2. ПРИМЕР ИСПОЛЬЗОВАНИЯ БИБЛИОТЕКИ В КОНСОЛЬНОМ ПРИЛОЖЕНИИ

Программа позволяет производить следующие операции с файлами:

- шифрование/расшифрование;
- получение списка абонентов, для которых зашифрован файл;
- формирование/проверка ЭЦП;
- удаление ЭЦП;
- загрузку ключей в драйвер ;
- выгрузку ключей из драйвера;
- получение информации о прогруженных в драйвер ключах .

Аргументы задаются в командной строке программы.

Текст консольного приложения находится в файле WFTEST.C

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ

ПО	Программное обеспечение
ППО	Прикладное программное обеспечение
СКЗИ	Средство криптографической защиты информации
НСД	Несанкционированный доступ
ОС	Операционная система
ПАК	Программно-аппаратный комплекс
АРМ	Автоматизированное рабочее место
ЭЦП	Электронная цифровая подпись
АРМ	АРМ Администратора безопасности.
АБ	ПО изготовления ключевой информации
КС	Ключевая система
ДСЧ	Датчик случайных чисел
СЗИ	Средства защиты информации
ТМ	Устройство хранения информации на таблетке touch-memory
ЖМД	Жесткий магнитный диск
ГМД	Гибкий магнитный диск

Лист регистрации изменений									
	Номера листов (страниц)				Всего листов (страниц) в документе	№ документа	Входящий № сопроводи-тельного документа и дата	Подпись	Дата
Изм	изменен-ных	заменен-ных	новых	аннулиро-ванных					