|  |  |  |
| --- | --- | --- |
|  | |  | | --- | | **ОБЩЕСТВО С ОГРАНИЧЕННОЙ ОТВЕТСТВЕННОСТЬЮ «ТСС»** |   **средство криптографической защиты информации**  **«Dcrypt 1.0 v.2»**  **Руководство разработчика**  **4012-006-61649217-18 01 92**  Москва  2018 |

**Аннотация**

Настоящий документ является руководством разработчика средств криптографической защиты информации (далее – СКЗИ) на основе библиотечных вызовов СКЗИ «Dcrypt 1.0 v.2» исп. 31, 32, 33, 34, 35 и 36.

**Оглавление**

[1 Основные технические данные и характеристики изделия 4](#_Toc529291732)

[2 Перечень вызовов, использование которых при разработке прикладного ПО с применением СКЗИ «Dcrypt 1.0 v.2» возможно без проведения дополнительных тематических исследований 5](#_Toc529291733)

[2.1 Описание интерфейса CryptoProvider 11](#_Toc529291734)

[2.2 Перечень возвращаемых сообщений об ошибках 12](#_Toc529291735)

[2.3 Интерфейсы вспомогательных объектов. 16](#_Toc529291736)

[3 Пример использования API библиотеки dmcrypt 18](#_Toc529291737)

# Основные технические данные и характеристики изделия

Основные технические данные и характеристики, а также требования к программному и аппаратному обеспечению средства криптографической защиты информации «Dcrypt 1.0 v.2» (далее – СКЗИ «Dcrypt 1.0 v.2», изделие) приведены в документе «Средство криптографической защиты информации «Dcrypt 1.0 v.2. Правила пользования. 4012-006-61649217-18 01 95».

При использовании СКЗИ «Dcrypt 1.0 v.2» в программных продуктах необходимо следовать требованиям эксплуатационной документации, входящей в состав комплекта поставки СКЗИ «Dcrypt 1.0 v.2», включающий следующие эксплуатационные документы: «Средство криптографической защиты информации «Dcrypt 1.0 v.2». Формуляр. 4012-006-61649217-18 01 30», «Средство криптографической защиты информации «Dcrypt 1.0 v.2». Правила пользования. 4012-006-61649217-18 01 95», «Средство криптографической защиты информации «Dcrypt 1.0 v.2». Руководство администратора. 4012-006-61649217-18 01 91», «Средство криптографической защиты информации «Dcrypt 1.0 v.2». Руководство разработчика. 4012-006-61649217-18 01 92», «Средство криптографической защиты информации «Dcrypt 1.0 v.2». Руководство пользователя. 4012-006-61649217-18 01 34».

# Перечень вызовов, использование которых при разработке прикладного ПО с применением СКЗИ «Dcrypt 1.0 v.2» возможно без проведения дополнительных тематических исследований

Библиотеки – условное обозначение СКЗИ «Dcrypt 1.0 v.2» для исполнений 31, 32, 33, 34, 35 и 36.

Библиотеки обеспечивают выполнение следующих целевых функций:

* шифрование файлов и данных, содержащихся в областях оперативной памяти;
* вычисление имитовставки для файлов и данных, содержащихся в областях оперативной памяти;
* вычисление значения хэш-функции для файлов и данных, содержащихся в областях оперативной памяти;
* реализация функций электронной подписи в соответствии с федеральным законом от 6 апреля 2011 г. № 63-ФЗ «Об электронной подписи»: создание электронной подписи, проверка электронной подписи, создание ключа электронной подписи, создание ключа проверки электронной подписи.

Интерфейс криптопровайдера библиотеки dmcrypt используется для выполнения следующих функций:

* сериализация и десериализация ключевых пар и сертификатов X509;
* создание списка отозванных сертификатов (CRL), отзыв сертификатов X509, сохранение и загрузка CRL в формате DER;
* шифрование/расшифрование блоков данных и файлов;
* выработка и проверка ЭЦП для блоков данных и файлов;
* вычисление и проверка значения хэш-функции для блоков данных и файлов.

Перечень вызовов, использование которых при разработке прикладного ПО с применением СКЗИ «Dcrypt 1.0 v.2» возможно без проведения дополнительных тематических исследований приведен в таблице Таблица 1.

Таблица 1 - Перечень вызовов, использование которых при разработке прикладного ПО с применением СКЗИ «Dcrypt 1.0 v.2» возможно без проведения дополнительных тематических исследований

|  |  |  |
| --- | --- | --- |
| **№** | **Функция** | **Описание** |
| **1** | std::string errorMessage() const; | * Возвращает сообщение об ошибке в случае неуспешного завершения последней операции. |
| **2** | bool open(const std::string& seedId, const std::string& serialId); | * Инициализирует внутреннее состояние криптопровайдера и инициализирует внутренний ГПСЧ из источника с идентификатором seedId (например, аппаратный ДСЧ). * Текущий свободный серийный номер для генерации ключевых пар загружается из источника serialId. * Возвращает true в случае успешной инициализации. |
| **3** | void close(); | * Закрывает сессию работы с криптопровайдером и очищает внутреннее состояние. Все несохраненные сертификаты и ключевые пары уничтожаются. |
| **4** | bool serializeKeyPair( const std::string& keyPairId, const char\* passphrase, const std::string& outId ); | * Сериализует ключевую пару с идентификатором keyPairId в выходной контейнер с идентификатором outId, используя пароль passphrase для выработки ключей шифрования данных в контейнере. |
| **5** | bool loadKeyPair( const std::string& keyPairId, const char\* passphrase, const std::string& inId ); | * Загружает ключевую пару с идентификатором keyPairId из входного контейнера с идентификатором inId, используя пароль passphrase для выработки ключей расшифрования данных в контейнере. |
| **6** | bool loadCrl(const std::string& crlId, const std::string& inId, const std::string& caKeyPairId); | * Загружает CRL с идентификатором crlId из контейнера с идентификатором inId в формате DER. Для проверки CRL используется ключевая пара с идентификатором caKeyPairId. |
| **7** | bool serializeCertificate( const std::string& keyPairId, const std::string& outId ); | * Сериализует сертификат X509 из ключевой пары с идентификатором keyPairId в выходной контейнер с идентификатором outId в формате DER. |
| **8** | bool loadCertificate( const std::string& certificateId, const std::string& inId ); | * Загружает сертификат X509 с идентификатором certificateId из контейнера с идентификатором inId в формате DER. |
| **9** | bool encrypt( EncryptionProcessor\* ep, const std::string& senderKeyPairId, const std::string& receiverCertificateId, const std::string& crlId, const std::string& caKeyPairId ); | * Осуществляет шифрование блоков данных, получаемых через объект ep. Объект ep реализует интерфейс EncryptionProcessor, который абстрагирует получение данных для шифрования (блоки памяти, данные из сети, файлы) и сохранение самих зашифрованных данных. * Шифрование осуществляется на ключе, выработанном из ключевой пары отправителя senderKeyPairId и сертификата получателя receiverCertificateId. * Сертификаты отправителя и получателя проходят проверку по списку отозванных сертификатов CRL с идентификатором crlId. Кроме того, проверяются их подписи по ключевой паре с идентификатором caKeyPairId. |
| **10** | bool decrypt( DecryptionProcessor\* dp, const std::string& receiverKeyPairId, const std::string& senderCertificateId, const std::string& crlId, const std::string& caKeyPairId ); | * Осуществляет расшифрование блоков данных, получаемых через объект dp. Объект dp реализует интерфейс DecryptionProcessor, который абстрагирует получение данных для расшифрования (блоки памяти, данные из сети, файлы) и сохранение самих расшифрованных данных. * Шифрование осуществляется на ключе, выработанном из сертификата отправителя senderCertificateId и ключевой пары получателя receiverKeyPairId. * Сертификаты отправителя и получателя проходят проверку по списку отозванных сертификатов CRL с идентификатором crlId. Кроме того, проверяются их подписи по ключевой паре с идентификатором caKeyPairId. |
| **11** | bool calculateMac( CalculateMacProcessor\* ep, const std::string& senderKeyPairId, const std::string& receiverCertificateId, const std::string& crlId, const std::string& caKeyPairId ); | * Вычисляет имитовставку для блоков данных, получаемых через объект ep. * Объект ep реализует интерфейс CalculateMacProcessor, который абстрагирует получение данных для подсчета имитовставки (блоки памяти, данные из сети, файлы) и сохранение контейнера с имитовставкой. * Вычисление имитовставки осуществляется на ключе, выработанном из ключевой пары отправителя senderKeyPairId и сертификата получателя receiverCertificateId. * Сертификаты отправителя и получателя проходят проверку по списку отозванных сертификатов CRL с идентификатором crlId. * Кроме того, проверяются их подписи по ключевой паре с идентификатором caKeyPairId. |
| **12** | bool verifyMac( VerifyMacProcessor\* dp, const std::string& receiverKeyPairId, const std::string& senderCertificateId, const std::string& crlId, const std::string& caKeyPairId ); | * Осуществляет проверку имитовставки для блоков данных, получаемых через объект dp. * Объект dp реализует интерфейс VerifyMacProcessor, который абстрагирует получение данных для подсчета имитовставки (блоки памяти, данные из сети, файлы). * Вычисление имитовставки осуществляется на ключе, выработанном из сертификата отправителя senderCertificateId и ключевой пары получателя receiverKeyPairId. * Сертификаты отправителя и получателя проходят проверку по списку отозванных сертификатов CRL с идентификатором crlId. * Кроме того, проверяются их подписи по ключевой паре с идентификатором caKeyPairId. |
| **13** | bool sign( SignProcessor\* sp, const std::string& senderKeyPairId, const std::string& crlId, const std::string& caKeyPairId ); | * Осуществляет выработку ЭЦП для блоков данных, получаемых через объект sp. Объект sp реализует интерфейс SignProcessor, который абстрагирует получение данных для выработки ЭЦП (блоки памяти, данные из сети, файлы) и сохранение ЭЦП. * Для выработки ЭЦП используется ключевая пара отправителя senderKeyPairId. * Сертификаты отправителя проходит проверку по списку отозванных сертификатов CRL с идентификатором crlId. Кроме того, проверяется его подпись по ключевой паре с идентификатором caKeyPairId. |
| **14** | bool verify( VerifyProcessor\* vp, const std::string& senderCertificateId, const std::string& crlId, const std::string& caKeyPairId ); | * Осуществляет проверку ЭЦП для блоков данных, получаемых через объект vp. Объект vp реализует интерфейс VerifyProcessor, который абстрагирует получение данных для проверки ЭЦП (блоки памяти, данные из сети, файлы). * Для проверки ЭЦП используется сертификат отправителя senderCertificateId. * Сертификаты отправителя проходит проверку по списку отозванных сертификатов CRL с идентификатором crlId. Кроме того, проверяется его подпись по ключевой паре с идентификатором caKeyPairId. |
| **15** | bool calculateHash( HashingProcessor\* hp ); | * Осуществляет вычисление хэш-функции для блоков данных, получаемых через объект hp. Объект hp реализует интерфейс HashingProcessor, который абстрагирует получение данных для вычисления хэш-функции (блоки памяти, данные из сети, файлы) и сохранение ее значения. |
| **16** | bool compareHash( HashCompareProcessor\* hp ). | * Осуществляет вычисление хэш-функции для блоков данных, получаемых через объект hp, и сравнение с ранее вычисленным значением. Объект hp реализует интерфейс HashCompareProcessor, который абстрагирует получение данных для вычисления хэш-функции (блоки памяти, данные из сети, файлы) и получение ранее сохраненного значения. |

## Описание интерфейса CryptoProvider

class DMCRYPT\_EXPORT CryptoProvider

{

public:

CryptoProvider();

~CryptoProvider();

std::string errorMessage() const;

bool open( const std::string& seedId, const std::string& serialId );

void close();

bool generateKeyPair( const Pki::CertificateRequest\* request, const std::string& userKeyPairId, const std::string& caKeyPairId );

bool serializeKeyPair( const std::string& keyPairId, const char\* passphrase, const std::string& outId );

bool loadKeyPair( const std::string& keyPairId, const char\* passphrase, const std::string& inId );

bool loadCrl( const std::string& crlId, const std::string& inId, const std::string& caKeyPairId );

bool serializeCertificate( const std::string& keyPairId, const std::string& outId );

bool loadCertificate( const std::string& certificateId, const std::string& inId );

bool encrypt( EncryptionProcessor\* ep, const std::string& senderKeyPairId, const std::string& receiverCertificateId, const std::string& crlId, const std::string& caKeyPairId );

bool decrypt( DecryptionProcessor\* dp, const std::string& receiverKeyPairId, const std::string& senderCertificateId, const std::string& crlId, const std::string& caKeyPairId );

bool calculateMac( CalculateMacProcessor\* ep, const std::string& senderKeyPairId, const std::string& receiverCertificateId, const std::string& crlId, const std::string& caKeyPairId );

bool verifyMac( VerifyMacProcessor\* dp, const std::string& receiverKeyPairId, const std::string& senderCertificateId, const std::string& crlId, const std::string& caKeyPairId );

bool sign( SignProcessor\* sp, const std::string& senderKeyPairId, const std::string& crlId, const std::string& caKeyPairId );

bool verify( VerifyProcessor\* vp, const std::string& senderCertificateId, const std::string& crlId, const std::string& caKeyPairId );

bool calculateHash( HashingProcessor\* hp );

bool compareHash( HashCompareProcessor\* hp );

private:

CryptoProviderPrivate\* const d;

};

} // namespace DmCrypt

## Перечень возвращаемых сообщений об ошибках

Перед вызовом целевых функций необходимо создать объект класса «CryptoProvider» и вызвать метод «open()». После завершения работы с объектом необходимо вызвать метод «close()».

Каждая целевая функция возвращает значение типа «bool», говорящее о том, успешно ли выполнилась функция. Для получения детального сообщения об ошибке нужно вызвать метод «errorMessage()».

В таблице висимости от вызываемой функций.

Таблица 2 приведен список возвращаемых сообщений об ошибках в зависимости от вызываемой функций.

Таблица 2 – Сообщения об ошибках в зависимости от вызываемой функции

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Функция** | **Ошибка** | **Пояснение** |
| **1** | open(): | "CryptoProvider already opened" | «CryptoProvider уже открыт» |
| "Failed to parse DSC file: " | «Не удалось проанализировать файл DSC:» |
| "Failed to read seed: " | «Не удалось прочитать seed:» |
| "Unknown seed source" | «Неизвестный источник seed» |
| "Failed to allocate PRNG" | «Не удалось выделить PRNG» |
| "Failed to initialize PRNG" | «Не удалось инициализировать PRNG» |
| "Unknown serial source" | «Неизвестный серийный источник» |
| **2** | generateKeyPair(): | "CryptoProvider is not opened" | «CryptoProvider не открывается» |
| "Specified key pair already exists" | «Указанная пара ключей уже существует» |
| "Specified CA key pair doesn't exist" | «Указанная пара ключей CA не существует» |
| **3** | serializeKeyPair(): | "CryptoProvider is not opened" | «CryptoProvider не открывается» |
| "Specified key pair doesn't exist" | «Указанная пара ключей не существует» |
| "Unsupported I/O string format" | «Неподдерживаемый формат строки ввода / вывода» |
| "Specified output container is not supported" | «Указанный входной контейнер не поддерживается» |
| **4** | loadKeyPair(): | "CryptoProvider is not opened" | «CryptoProvider не открывается» |
| "Specified key pair already exist" | «Указанная пара ключей уже существует» |
| "Unsupported I/O string format" | «Неподдерживаемый формат строки ввода / вывода» |
| "Specified input container not supported" | «Указанный входной контейнер не поддерживается» |
| **5** | loadCrl(): | "CryptoProvider is not opened" | «CryptoProvider не открывается» |
| "Specified CRL already exist" | «Указанный CRL уже существует» |
| "Unsupported I/O string format" | «Неподдерживаемый формат строки ввода / вывода» |
| "Specified input container not supported" | «Указанный входной контейнер не поддерживается» |
| **6** | serializeCertificate(): | "CryptoProvider is not opened" | «CryptoProvider не открывается» |
| "Specified key pair doesn't exist" | «Указанная пара ключей не существует» |
| "Unsupported I/O string format" | «Неподдерживаемый формат строки ввода / вывода» |
| "Specified output container not supported" | «Указанный выходной контейнер не поддерживается» |
| **7** | loadCertificate | "CryptoProvider is not opened" | «CryptoProvider не открывается» |
| "Specified certificate already exist" | «Указанный сертификат уже существует» |
| "Unsupported I/O string format" | «Неподдерживаемый формат строки ввода / вывода» |
| "Specified input container not supported" | «Указанный входной контейнер не поддерживается» |
| **8** | encrypt(): | "CryptoProvider is not opened" | «CryptoProvider не открывается» |
| "EncryptionProcessor not specified" | «EncryptionProcessor не указан» |
| "Specified sender key pair doesn't exist" | «Указанная пара ключей отправителя не существует» |
| "Specified receiver certificate doesn't exist" | «Указанный сертификат получателя не существует» |
| "Specified CRL doesn't exist" | «Указанный CRL не существует» |
| "Specified CA certificate doesn't exist" | «Указанный сертификат CA не существует» |
| **9** | decrypt() | "CryptoProvider is not opened" | «CryptoProvider не открывается» |
| "DecryptionProcessor not specified" | «DecryptionProcessor не указан» |
| "Specified receiver key pair doesn't exist" | «Указанная пара ключей приемника не существует» |
| "Specified sender certificate doesn't exist" | «Указанный сертификат отправителя не существует» |
| "Specified CRL doesn't exist" | «Указанный CRL не существует» |
| "Specified CA certificate doesn't exist" | «Указанный сертификат CA не существует» |
| **10** | calculateMac(): | "CryptoProvider is not opened" | «CryptoProvider не открывается» |
| "CalculateMacProcessor not specified" | «CalculateMacProcessor не указан» |
| "Specified sender key pair doesn't exist" | «Указанная пара ключей отправителя не существует» |
| "Specified receiver certificate doesn't exist" | «Указанный сертификат получателя не существует» |
| "Specified CRL doesn't exist" | «Указанный CRL не существует» |
| "Specified CA certificate doesn't exist" | «Указанный сертификат CA не существует» |
| **11** | verifyMac(): | "CryptoProvider is not opened" | «CryptoProvider не открывается» |
| "VerifyMacProcessor not specified" | «VerifyMacProcessor не указан» |
| "Specified receiver key pair doesn't exist" | «Указанная пара ключей приемника не существует» |
| "Specified sender certificate doesn't exist" | «Указанный сертификат отправителя не существует» |
| "Specified CRL doesn't exist" | «Указанный CRL не существует» |
| "Specified CA certificate doesn't exist" | «Указанный сертификат CA не существует» |
| **12** | sign(): | "CryptoProvider is not opened" | «CryptoProvider не открывается» |
| "Specified sender key pair doesn't exist" | «Указанная пара ключей отправителя не существует» |
| "Specified CRL doesn't exist" | «Указанный CRL не существует» |
| "Specified CA certificate doesn't exist" | «Указанный сертификат CA не существует» |
| **13** | verify(): | "CryptoProvider is not opened" | «CryptoProvider не открывается» |
| "VerifyProcessor not specified" | «VerifyProcessor не определен» |
| "Specified sender certificate doesn't exist" | «Указанный сертификат отправителя не существует» |
| "Specified CRL doesn't exist" | «Указанный CRL не существует» |
| "Specified CA certificate doesn't exist" | «Указанный сертификат CA не существует» |
| **14** | calculateHash(): | "CryptoProvider is not opened" | «CryptoProvider не открывается» |
| "HashingProcessor not specified" | «HashingProcessor не указан» |
| **15** | compareHash(): | "CryptoProvider is not opened" | «CryptoProvider не открывается» |
| "HashCompareProcessor not specified" | «HashCompareProcessor не указан» |

## Интерфейсы вспомогательных объектов.

При вызове функций шифрования/расшифрования, вычисления и проверки ЭЦП, вычисления хэш-функции используются вспомогательные объекты для чтения/записи данных из памяти или из файла. Интерфейсы этих объектов представлены в таблице Таблица 3.

Таблица 3 – Описание функций интерфейса

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Название интерфейса** | **Функция** | **Описание** |
| **1** | DataProcessor | virtual bool readBlock( u8\* ptr, size\_t& size, bool& isLast ) = 0;  // ptr is pre-allocated | * Читает блок данных в массив ptr и указать его размер в size. * Если блок данных последний, то выставить флаг isLast. |
| virtual bool writeBlock( const u8\* ptr, size\_t size ) = 0; | * Записывает блок данных из массива ptr размера size |
| virtual std::string error() const = 0; | * Возвращает сообщение об ошибке, если операцию чтения или записи дала сбой. |
| **2** | EncryptionProcessor | virtual bool startHeaderReady( const u8\* ptr, size\_t size ) = 0; | * Сообщает, что начальный заголовок зашифрованного контейнера готов. * Размещает заголовок по адресу ptr и имеет длину size. |
| virtual bool finalHeaderReady( const u8\* ptr, size\_t size ) = 0; | * Сообщает, что финальный заголовок зашифрованного контейнера готов. * Размещает заголовок по адресу ptr и имеет длину size. |
| **3** | DecryptionProcessor | virtual bool headerReady( const u8\* ptr, size\_t size ) = 0; | * Сообщает, что заголовок контейнера ЭЦП готов. * Размещает заголовок по адресу ptr и имеет длину size. |
| virtual bool readHeader( u8\* ptr, size\_t size ) = 0; | * Читает заголовок контейнера ЭЦП в массив ptr длины size. |
| **4** | CalculateMacProcessor | virtual bool headerReady( const u8\* ptr, size\_t size ) = 0; | * Сообщает, что финальный заголовок зашифрованного контейнера готов. * Размещает заголовок по адресу ptr и имеет длину size. |
| **5** | VerifyMacProcessor | virtual bool getLength( size\_t& size ) = 0; | * Сообщает, что заголовок контейнера имитовствавки готов. * Размещает заголовок по адресу ptr и имеет длину size. |
| virtual bool readHeader( u8\* ptr, size\_t size ) = 0; | * Читает заголовок зашифрованного контейнера в массив ptr длины size. |
| **6** | SignProcessor | virtual bool headerReady( const u8\* ptr, size\_t size ) = 0; | * Сообщает, что заголовок контейнера со значением готов. * Размещает заголовок по адресу ptr и имеет длину size. |
| **7** | VerifyProcessor | virtual bool readHeader( u8\* ptr, size\_t size ) = 0. | * Читает заголовок контейнера со значением в массив ptr длины size. |
| **8** | HashingProcessor | virtual bool headerReady( const u8\* ptr, size\_t size ) = 0; | * Сообщает, что заголовок контейнера со значением хэш-фукнции готов. * Размещает заголовок по адресу ptr и имеет длину size. |
| **9** | HashCompareProcessor | virtual bool readHeader( u8\* ptr, size\_t size ) = 0; | * Читает заголовок контейнера со значением хэш-фукнции в массив ptr длины size. |

# Пример использования API библиотеки dmcrypt

В приведенном примере показано, как использовать API CryptoProvider для зашифрования и расшифрования файла на диске.

#include "types.h"

#include "cryptoprovider.h"

#include <cerrno>

#include <cstdio>

#include <cstring>

#include <iostream>

using namespace std;

*\* Интерфейс чтения и записи блоков данных из файла*

struct FileProcessor

{

public:

FileProcessor()

: m\_inFile( NULL ), m\_outFile( NULL )

{

}

~FileProcessor()

{

close();

}

*\* Открывает файл inPath для чтения блоков данных*

bool openInputFile( const string& inPath )

{

m\_inFile = fopen( inPath.c\_str(), "rb" );

if( !m\_inFile ) {

m\_error = string( "Failed to open input file: " ) + strerror( errno );

return false;

}

return true;

}

*\* Открывает файл outPath для записи блоков данных*

bool openOutputFile( const string& outPath )

{

m\_outFile = fopen( outPath.c\_str(), "wb" );

if( !m\_outFile ) {

m\_error = string( "Failed to open output file: " ) + strerror( errno );

return false;

}

return true;

}

*\* Открывает файл inPath для чтения, а outPath - для записи*

bool open( const string& inPath, const string& outPath )

{

if( !openInputFile( inPath ) ) {

return false;

}

if( !openOutputFile( outPath ) ) {

return false;

}

return true;

}

void close()

{

if( m\_inFile ) {

fclose( m\_inFile );

m\_inFile = NULL;

}

if( m\_outFile ) {

fclose( m\_outFile );

m\_outFile = NULL;

}

}

*\* Читает блок данных из файла в массив ptr, размер указывается в size, а в переменной «isLast» указывается, является ли блок данных последним в файле (достигнут конец файла)*

bool readDataBlock( u8\* ptr, size\_t& size, bool& isLast )

{

if( !m\_inFile ) {

m\_error = string( "Input file not opened" );

return false;

}

const size\_t readn = fread( ptr, 1, size, m\_inFile );

if( ferror( m\_inFile ) ) {

m\_error = string( "Failed to read data block: " ) + strerror( errno );

return false;

}

size = readn;

isLast = feof( m\_inFile );

return true;

}

*\* Записывает блок данных из массива ptr размера size в открытый файл*

bool writeDataBlock( const u8\* ptr, size\_t size )

{

if( !m\_outFile ) {

m\_error = string( "Output file not opened" );

return false;

}

if( fwrite( ptr, 1, size, m\_outFile ) != size ) {

m\_error = string( "Failed to write data block: " ) + strerror( errno );

return false;

}

return true;

}

*\* Возвращает последнее сообщение об ошибке*

string error() const

{

return m\_error;

}

protected:

FILE\* m\_inFile;

FILE\* m\_outFile;

string m\_error;

};

*\* Реализует интерфейс EncryptionProcessor для зашифрования файл*

struct FileEncryptionProcessor : public DmCrypt::EncryptionProcessor, public FileProcessor

{

public:

virtual bool readBlock( u8\* ptr, size\_t& size, bool& isLast )

{

return readDataBlock( ptr, size, isLast );

}

virtual bool writeBlock( const u8\* ptr, size\_t size )

{

return writeDataBlock( ptr, size );

}

virtual std::string error() const

{

return m\_error;

}

virtual bool startHeaderReady( const u8\* ptr, size\_t size )

{

if( !m\_outFile ) {

m\_error = string( "Output file not opened" );

return false;

}

if( fwrite( ptr, 1, size, m\_outFile ) != size ) {

m\_error = string( "Failed to write encrypted file header: " ) + strerror( errno );

return false;

}

return true;

}

virtual bool finalHeaderReady( const u8\* ptr, size\_t size )

{

if( !m\_outFile ) {

m\_error = string( "Output file not opened" );

return false;

}

rewind( m\_outFile );

if( fwrite( ptr, 1, size, m\_outFile ) != size ) {

m\_error = string( "Failed to write encrypted file header: " ) + strerror( errno );

return false;

}

return true;

}

};

*\* Реализует интерфейс DecryptionProcessor для расшифрования файла*

struct FileDecryptionProcessor : public DmCrypt::DecryptionProcessor, public FileProcessor

{

public:

virtual bool readBlock( u8\* ptr, size\_t& size, bool& isLast )

{

return readDataBlock( ptr, size, isLast );

}

virtual bool writeBlock( const u8\* ptr, size\_t size )

{

return writeDataBlock( ptr, size );

}

virtual std::string error() const

{

return m\_error;

}

virtual bool getLength( size\_t& size )

{

if( !m\_inFile ) {

m\_error = string( "Input file not opened" );

return false;

}

fseek( m\_inFile, 0 , SEEK\_END );

//size = ftell( m\_inFile ) - sizeof( DmCrypt::EncryptionHeader );

size = ftell( m\_inFile );

rewind( m\_inFile );

return true;

}

virtual bool readHeader( u8\* ptr, size\_t size )

{

if( !m\_inFile ) {

m\_error = string( "Input file not opened" );

return false;

}

if( fread( ptr, 1, size, m\_inFile ) != size ) {

m\_error = string( "Failed to read header: " ) + strerror( errno );

return false;

}

return true;

}

};

*\* Реализует интерфейс CalculateMacProcessor для вычисления имитовставки файла*

struct FileCalculateMacProcessor : public DmCrypt::CalculateMacProcessor, public FileProcessor

{

public:

virtual bool readBlock( u8\* ptr, size\_t& size, bool& isLast )

{

return readDataBlock( ptr, size, isLast );

}

virtual bool writeBlock( const u8\* ptr, size\_t size )

{

return writeDataBlock( ptr, size );

}

virtual std::string error() const

{

return m\_error;

}

virtual bool headerReady( const u8\* ptr, size\_t size )

{

if( !m\_outFile ) {

m\_error = string( "Output file not opened" );

return false;

}

if( fwrite( ptr, 1, size, m\_outFile ) != size ) {

m\_error = string( "Failed to write MAC file header: " ) + strerror( errno );

return false;

}

return true;

}

};

*\* Реализует интерфейс* VerifyMacProcessor *для проверки имитовставки файла*

struct FileVerifyMacProcessor : public DmCrypt::VerifyMacProcessor, public FileProcessor

{

public:

~FileVerifyMacProcessor()

{

close();

}

bool open( const string& inPath, const string& macPath )

{

if( !openInputFile( inPath ) ) {

return false;

}

m\_macFile = fopen( macPath.c\_str(), "rb" );

if( !m\_macFile ) {

m\_error = string( "Failed to open MAC file: " ) + strerror( errno );

return false;

}

return true;

}

void close() {

FileProcessor::close();

if( m\_macFile ) {

fclose( m\_macFile );

m\_macFile = NULL;

}

}

virtual bool readBlock( u8\* ptr, size\_t& size, bool& isLast )

{

return readDataBlock( ptr, size, isLast );

}

virtual bool writeBlock( const u8\* ptr, size\_t size )

{

return writeDataBlock( ptr, size );

}

virtual std::string error() const

{

return m\_error;

}

virtual bool readHeader( u8\* ptr, size\_t size )

{

if( !m\_macFile ) {

m\_error = string( "MAC file not opened" );

return false;

}

if( fread( ptr, 1, size, m\_macFile ) != size ) {

m\_error = string( "Failed to read header: " ) + strerror( errno );

return false;

}

return true;

}

virtual bool getLength( size\_t& size )

{

if( !m\_inFile ) {

m\_error = string( "Input file not opened" );

return false;

}

fseek( m\_inFile, 0 , SEEK\_END );

size = ftell( m\_inFile );

rewind( m\_inFile );

return true;

}

private:

FILE\* m\_macFile;

};

int main()

{

*\* Открывает криптопровайдер с начальным заполнением из файла seed.dsc*

CryptoProvider cp;

if( !cp.open( "file://seed.dsc", "file://serial.txt" ) ) {

cerr << "failed to open: " << cp.errorMessage() << endl;

return 0;

}

*\* Создает запрос на создание корневой ключевой пары*

Pki::CertificateRequest caRequest;

caRequest.addSubjectEntry( Pki::SubjectEntry::CN, "TEST CA" );

caRequest.addSubjectEntry( Pki::SubjectEntry::C, "RU" );

caRequest.addSubjectEntry( Pki::SubjectEntry::L, "Moscow" );

caRequest.addSubjectEntry( Pki::SubjectEntry::ST, "MSK" );

caRequest.addSubjectEntry( Pki::SubjectEntry::O, "TSS" );

caRequest.addSubjectEntry( Pki::SubjectEntry::OU, "Dev" );

caRequest.addSubjectEntry( Pki::SubjectEntry::Email, "ivanov@tssltd.ru" );

caRequest.setValidityNotBefore( "140820101505Z" );

caRequest.setValidityNotAfter( "200820101505Z" );

*\* Генерирует ключевую пару по сформированному запросу*

if( !cp.generateKeyPair( &caRequest, "CA\_KEY\_PAIR", "CA\_KEY\_PAIR" ) ) {

cerr << "failed to generate CA key pair: " << cp.errorMessage() << endl;

return 0;

}

*\* Сохраняет ключевую пару в файл*

if( !cp.serializeKeyPair( "CA\_KEY\_PAIR", "1111", "file://ca.dkpc" ) ) {

cerr << "failed to serialize CA key pair: " << cp.errorMessage() << endl;

return 0;

}

*\* Создает запрос на создание пользовательской ключевой пары*

Pki::CertificateRequest user1Request;

user1Request.addSubjectEntry( Pki::SubjectEntry::CN, "TEST USER 1" );

user1Request.addSubjectEntry( Pki::SubjectEntry::C, "RU" );

user1Request.addSubjectEntry( Pki::SubjectEntry::L, "Moscow" );

user1Request.addSubjectEntry( Pki::SubjectEntry::ST, "MSK" );

user1Request.addSubjectEntry( Pki::SubjectEntry::O, "TSS" );

user1Request.addSubjectEntry( Pki::SubjectEntry::OU, "Dev" );

user1Request.addSubjectEntry( Pki::SubjectEntry::Email, "ivanov@tssltd.ru" );

user1Request.setValidityNotBefore( "140820101505Z" );

user1Request.setValidityNotAfter( "200820101505Z" );

*\* Генерирует пользовательскую ключевую пару по сформированному запросу*

if( !cp.generateKeyPair( &user1Request, "USER1\_KEY\_PAIR", "CA\_KEY\_PAIR" ) ) {

cerr << "failed to generate user1 key pair: " << cp.errorMessage() << endl;

return 0;

}

*\* Сохраняет ключевую пару в файл*

if( !cp.serializeKeyPair( "USER1\_KEY\_PAIR", "1111", "file://user1.dkpc" ) ) {

cerr << "failed to serialize user1 key pair: " << cp.errorMessage() << endl;

return 0;

}

*\* Создает запрос на создание еще одной пользовательской ключевой пары*

Pki::CertificateRequest user2Request;

user2Request.addSubjectEntry( Pki::SubjectEntry::CN, "TEST USER 2" );

user2Request.addSubjectEntry( Pki::SubjectEntry::C, "RU" );

user2Request.addSubjectEntry( Pki::SubjectEntry::L, "Moscow" );

user2Request.addSubjectEntry( Pki::SubjectEntry::ST, "MSK" );

user2Request.addSubjectEntry( Pki::SubjectEntry::O, "TSS" );

user2Request.addSubjectEntry( Pki::SubjectEntry::OU, "Dev" );

user2Request.addSubjectEntry( Pki::SubjectEntry::Email, "ivanov@tssltd.ru" );

user2Request.setValidityNotBefore( "140820101505Z" );

user2Request.setValidityNotAfter( "200820101505Z" );

*\* Генерирует пользовательскую ключевую пару по сформированному запросу*

if( !cp.generateKeyPair( &user2Request, "USER2\_KEY\_PAIR", "CA\_KEY\_PAIR" ) ) {

cerr << "failed to generate user2 key pair: " << cp.errorMessage() << endl;

return 0;

}

*\* Сохраняет ключевую пару в файл*

if( !cp.serializeKeyPair( "USER2\_KEY\_PAIR", "1111", "file://user2.dkpc" ) ) {

cerr << "failed to serialize user2 key pair: " << cp.errorMessage() << endl;

return 0;

}

*\* Для шифрования файла понадобятся сертификаты, поэтому выделим их в отдельные сущности*

if( !cp.serializeCertificate( "USER1\_KEY\_PAIR", "file://user1.crt" ) ) {

cerr << "failed to serialize user1 certificate: " << cp.errorMessage() << endl;

return 0;

}

if( !cp.loadCertificate( "USER1\_CERTIFICATE", "file://user1.crt" ) ) {

cerr << "failed to load user1 certificate: " << cp.errorMessage() << endl;

return 0;

}

if( !cp.serializeCertificate( "USER2\_KEY\_PAIR", "file://user2.crt" ) ) {

cerr << "failed to serialize user2 certificate: " << cp.errorMessage() << endl;

return 0;

}

if( !cp.loadCertificate( "USER2\_CERTIFICATE", "file://user2.crt" ) ) {

cerr << "failed to load user2 certificate: " << cp.errorMessage() << endl;

return 0;

}

*\* Загружает CRL*

if( !cp.loadCrl( "CRL", "file://crl.crl", "CA\_KEY\_PAIR" ) ) {

cerr << "failed to load CRL: " << cp.errorMessage() << endl;

return 0;

}

*\* Шифрует тестовый файл*

FileEncryptionProcessor ep;

if( !ep.open( "test.txt", "test.enc" ) ) {

cerr << "failed to open EP: " << ep.error() << endl;

return 0;

}

if( cp.encrypt( &ep, "USER1\_KEY\_PAIR", "USER2\_CERTIFICATE", "CRL", "CA\_KEY\_PAIR" ) ) {

cerr << "CRL test when encrypting failed: " << cp.errorMessage() << endl;

return false;

}

ep.close();

*\* Расшифровывает тестовый файл*

*\* Файлы test.txt и test.dec должны совпадать.*

FileDecryptionProcessor dp;

if( !dp.open( "test.enc", "test.dec" ) ) {

cerr << "failed to open DP: " << dp.error() << endl;

return 0;

}

if( !cp.decrypt( &dp, "USER2\_KEY\_PAIR", "USER1\_CERTIFICATE", "CRL", "CA\_KEY\_PAIR" ) ) {

cerr << "failed to decrypt test file: " << cp.errorMessage() << endl;

return false;

}

dp.close();

*\* Вычисление имитовставки для файла cp\_test.txt.*

FileCalculateMacProcessor cmp;

if( !cmp.open( "cp\_test.txt", "cp\_test.imita" ) ) {

cerr << "CryptoProvider\_test: failed to open CMP: " << cmp.error() << endl;

return 0;

}

if( !cp.calculateMac( &cmp, "USER1\_KEY\_PAIR", "USER3\_KEY\_PAIR", "CRL", "CA\_KEY\_PAIR" ) ) {

cerr << "CryptoProvider\_test: failed to calculate mac for test file: " << cp.errorMessage() << endl;

return false;

}

cmp.close();

*\* Проверка имитовставки для файла cp\_test.txt.*

FileVerifyMacProcessor vmp;

if( !vmp.open( "cp\_test.txt", "cp\_test.imita" ) ) {

cerr << "CryptoProvider\_test: failed to open VMP: " << vmp.error() << endl;

return 0;

}

if( !cp.verifyMac( &vmp, "USER3\_KEY\_PAIR", "USER1\_KEY\_PAIR", "CRL", "CA\_KEY\_PAIR" ) ) {

cerr << "CryptoProvider\_test: failed to verify mac for test file: " << cp.errorMessage() << endl;

return false;

}

vmp.close();

cp.close();

return 0;

}