CODESYS

Библиотеки CmpCrypto и CmpX509Cert

Версия: 1.0
Перевод: oscat.ru

Оглавление

Введение

Криптографические методы играют важную роль при обработке данных.

- Проверка целостности данных обеспечивает полное соответствие отправленной и полученной информации;
- Проверка подлинности позволяет гарантированно установить источник полученных данных;
- Конфиденциальность требуется для предотвращения несанкционированного доступа к данным.

Можно выделить три процесса, в которых обычно применяются эти методы:

- **Обработка данных на локальном устройстве**. В этом случае безопасность данных в основном определяется характеристиками используемого оборудования. Так как обработка данных выполняется в реальном времени, то использование активного шифрования («на лету») обычно неприемлемо. Аппаратное обеспечение в этом случае должно гарантировать, что данные не могут быть восстановлены по косвенным признакам (таким, как энергопотребление процессора);
- **Передача данных от одного устройства к другому.** Пересылка данных осуществляется через транспортные каналы, которые могут быть как защищенными (например, физически изолированными), так и незащищенными. При передаче данных по незащищенным каналам рекомендуется использовать шифрование и цифровую подпись – это позволит защитить их от перехвата и внесения изменений. Шифрование и подпись данных обычно требуют значительных вычислительных ресурсов, поэтому допустимо пренебречь ими при использовании гарантированно защищенного канала связи;
- **Хранение данных**. Как и в случае с каналами связи, хранилища информации можно разделить на защищенные и незащищенные. При использовании незащищенных хранилищ рекомендуется применять шифрование и цифровую подпись, чтобы защитить данные от несанкционированного доступа и изменения.

Библиотеки **CmpCrypto** и **CmpX509Cert** значительно упрощают использование криптографических методов в среде **CODESYS V3.5**. Важно отметить, что сам факт наличия этих библиотек не решает описанные выше проблемы: пользователь должен выбрать подходящие для его задачи функции и корректно их использовать. Кроме того, на работу библиотек оказывают влияние аппаратные характеристики контроллера и его особенности (ОС, набор поддержанных в прошивке компонентов и т.д.).

Криптографические методы основаны на двух основных предположениях:

- На устройстве имеется возможность генерации «хороших», длинных случайных чисел без ущерба производительности. Качество полученных чисел зависит от используемого генератора. В идеальном случае используется аппаратный генератор случайных чисел, а не генератор псевдослучайных чисел. Число разрядов случайных чисел имеет значение, так как относительно небольшие числа можно подобрать методом перебора;
- На устройстве имеется возможность использования односторонних функций, для которых восстановление аргумента (шифруемых данных) на основании результата (хеша и т.п.) невозможно в течение разумного срока при использовании реалистичных ресурсов. Примеры односторонних функций: обратная функция, факторизация.

Второе предложение указывает на одну из ключевых проблем криптографии – с развитием аппаратных средств появляется возможность восстановления шифруемых данных за относительное небольшое время и без существенных финансовых затрат.

Поэтому обеспечение безопасности данных возможно только при условии, что используемые криптографические алгоритмы могут быть оперативно заменены, если появится информация об их «взломе».

Хорошим подходом к разработке ПО является создание функциональных блоков, для которых используемый криптографический алгоритм может быть легко изменен. В библиотеке **CmpCrypto** присутствует перечисление **RtsCryptoID**, которое содержит доступные алгоритмы. Подходящие параметры для каждого алгоритма (например, размер блока, размер ключа и т.д.) могут быть найдены в сети – например, в Википедии. В будущем в библиотеку будет добавлена функция **CryptoGetAlgorithmInfo** (CDS-58920), которая позволит получить рекомендации по параметрам алгоритма прямо в CODEYS.

Дополнительная информация:

- Сравнение производительности PKI и симметричного шифрования
- Механизмы обмена ключами в TLS
- Алгоритм RSA
- Гибридные криптосистемы
- Сравнение алгоритмов SHA-1, SHA-2, SHA-256
- Алгоритм AES
- Криптографический стандарт CMS

1. Случайные числа

Ниже приведен код, который генерирует случайное число в диапазоне 1…10:

```
VAR
        usiNumber : USINT; // случайное число в диапазоне 0..255
        bsNumber : CmpCrypto.RtsByteString := (ui32MaxLen := SIZEOF(usiNumber),
                pByData := ADR(usiNumber) );
        Result : SysTypes.RTS_IEC_RESULT;
        Random : USINT; // случайное число в диапазоне 1..10
END_VAR

Result := CmpCrypto.CryptoGenerateRandomNumber
        (ui32NumOfRandomBytes := SIZEOF(usiNumber),  pRandom := ADR(bsNumber) );
Random := (usiNumber MOD 10) + 1;
```

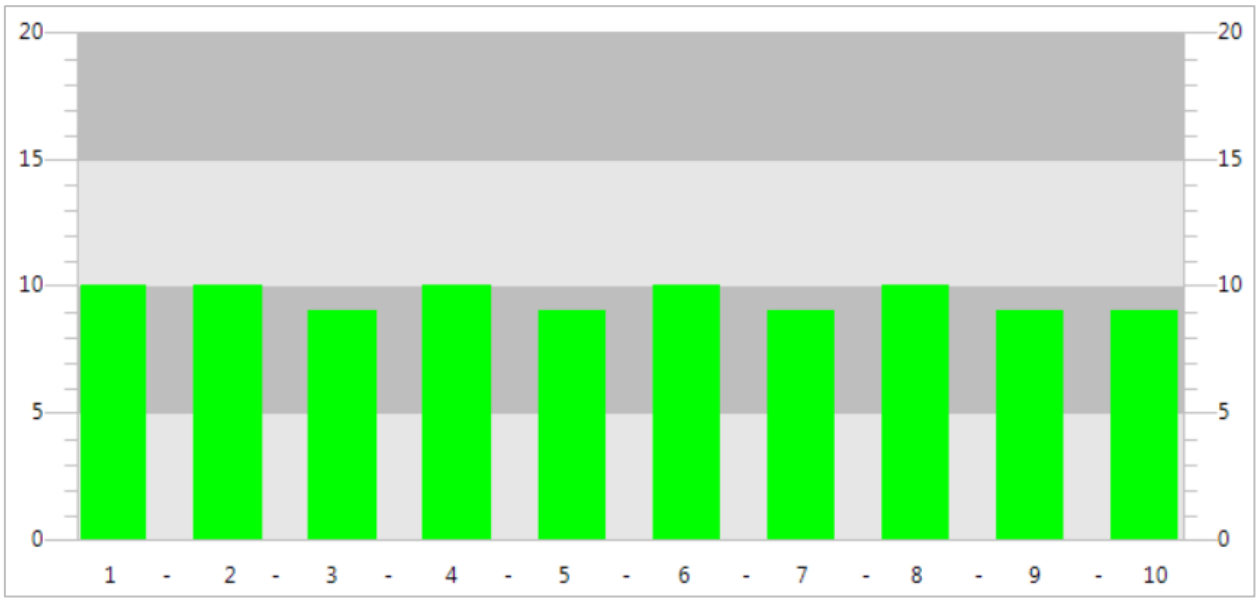

Рис. 1. Распределение сгенерированных случайных чисел

Спустя некоторое время работы функции, распределение получившихся чисел будет близко к равномерному (см. гистограмму). Это ожидаемый результат для «хорошего» генератора случайных чисел. Другим критерием качества является дисперсия начальных значений (зёрен). Чем сложнее подобрать генерируемые последовательности, тем более надежным является криптографический алгоритм, основанный на этом генераторе. См. более подробную информацию в Википедии.

Дополнительная информация:

- https://www.random.org/
- Шармейн Кенни. Генераторы случайных чисел (апрель 2005)
- Луиза Фолей. Анализ онлайн-генераторов случайных чисел (апрель 2001)

2. Хеширование

Хеш-функция преобразует массив входных данных произвольной длины в выходную битовую строку заданной длины. Такие функции, например, часто применяются для того, чтобы определить, не было ли изменено отправленное сообщение во время передачи. Отправитель и получатель заранее должны согласовать, какой алгоритм хеширования будет использоваться при передаче данных. Получатель принимает данные и их хеш, рассчитанный отправителем, после чего заново рассчитывает хеш для сообщения по известному ему алгоритму. Если оба хеша совпадают, то с большой долей вероятности полученное сообщение идентично отправленному. Хеш позволяет проверить подлинность сообщения, но не подлинность адресата – можно представить ситуацию, когда отправленное сообщение будет перехвачено, изменено и дополнено заново рассчитанным хешем.

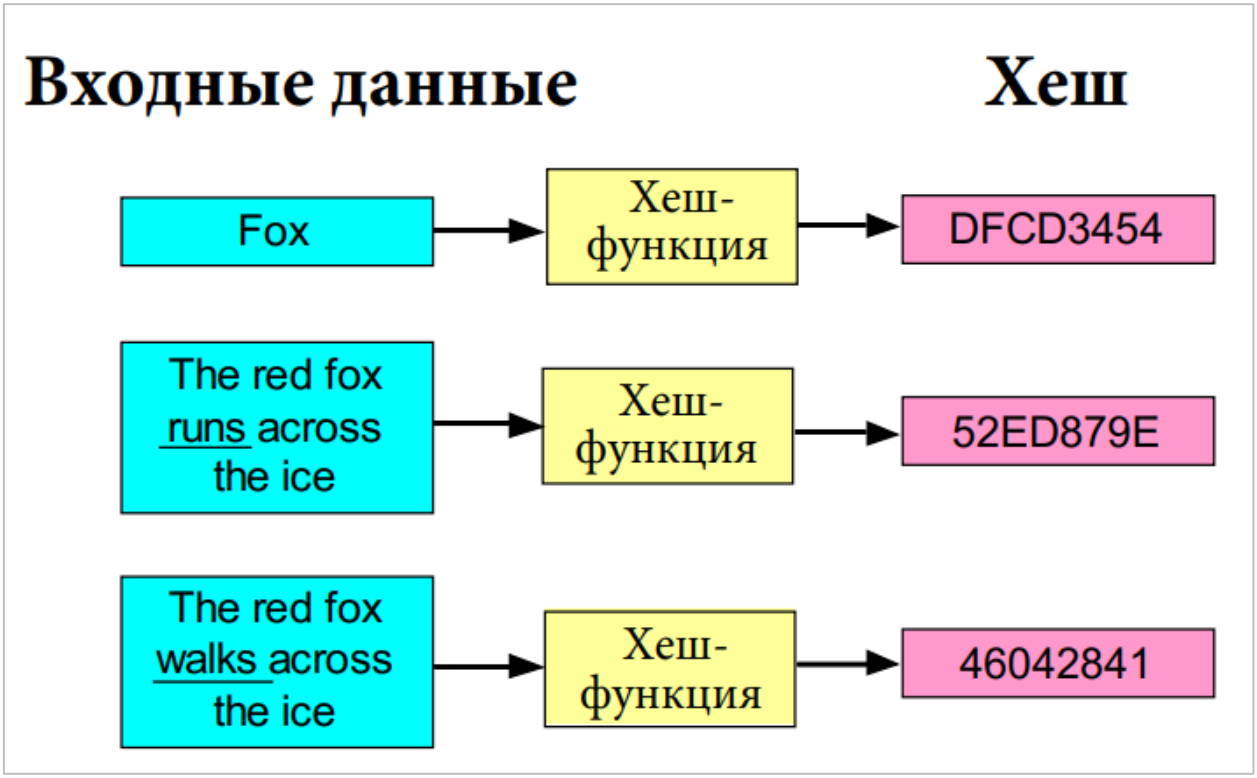

Рис. 2. Пример работы хеш-функции

Выше приведен пример работы хеш-фукции – для каждого сообщения указаны первые четыре байта хеша, рассчитанные по алгоритму SHA-1. Следует отметить, что рассчитанный хеш всегда имеет одинаковый размер независимо от размера массива исходных данных. Также стоит обратить внимание, что даже для незначительно отличающихся данных хеш выглядит совершенно по-разному.

Ниже приведен пример проверки хеша для полученного сообщения. Для расчета хеша используется алгоритм SHA-1. Проверка состоит из следующих этапов:

- Выделение области памяти для исходных данных (sMessage);
- Выделение области памяти для хеша, полученного вместе с сообщением (abyHashCode). Для алгоритма SHA-1 размер хеша составляет 20 байт;
- Расчет хеша для полученного сообщения с помощью функции **CryptoGenerateHash**;
- Сравнение полученного хеша с рассчитанным (в примере для этого используется функция **SysMemCmp** из библиотеки **SysMem**).

```
METHOD CheckMessage : BOOL // TRUE – полученный хеш совпадает с рассчитанным
VAR_INPUT
    sMessage : REFERENCE TO MESSAGE[1];
    szMessage : ULINT;
    abyHashCode : REFERENCE TO HASH_CODE[2];
END_VAR
VAR
    Result : SysTypes.RTS_IEC_RESULT;
    bsMessage : CmpCrypto.RtsByteString := (ui32MaxLen := SIZEOF(sMessage),
         pByData :=ADR (sMessage), ui32Len := TO_UDINT(szMessage));
    bsHashCode : CmpCrypto.RtsByteString := (ui32MaxLen := SIZEOF(abyHashCode),
         pByData :=0 );
    abyNewHashCode : HASH_CODE;
    bsNewHashCode : CmpCrypto.RtsByteString := (ui32MaxLen := SIZEOF(abyNewHashCode),
        pByData := ADR(abyNewHashCode));
    diResult : DINT;
    _hHASH : SysTypes.RTS_IEC_HANDLE := CmpCrypto.CryptoGetAlgorithmById
        (ui32CryptoID := CmpCrypto.RtsCryptoID.HASH_SHA1, pResult := 0);
END_VAR


Result := CmpCrypto.CryptoGenerateHash(hAlgo := _hHASH, pData := ADR(bsMessage),
        pHash := ADR(bsNewHashCode));
diResult := SysMem.SysMemCmp(pBuffer1 := ADR(abyHashCode),
        pBuffer2 := ADR(abyNewHashCode), udiCount := SIZEOF(HASH_CODE));
CheckMessage := (diResult = 0);
```

Примечание: сравнение хешей должно проводиться очень тщательно. В случае ошибки полученное сообщение будет признано корректным (т.е. идентичным отправленному сообщению), даже если оно было изменено во время передачи от отправителя к получателю.

Вычисление хеша – это эффективный метод преобразования большого объема данных в относительно короткую последовательность байт. При использовании подходящего алгоритма даже хеш небольшого размера однозначно идентифицирует исходный набор данных. Размер хеша – крайне важный параметр, так как сравнение большого объема данных является ресурсоемкой и не всегда допустимой операцией. Поэтому ключевой характеристикой алгоритма хеширования является устойчивость к коллизиям – то есть предотвращение ситуаций, когда для разных наборов данных будет рассчитано одинаковое значение хеша.

[1] MESSAGE – это псевдоним (ALIAS) для типа STRING(255)
[2] HASH_CODE – это псевдоним для типа ARRAY[0..19] OF BYTE

3. Код аутентификации сообщений на основе хеша (HMAC)

Код аутентификации сообщений на основе хеша (HMAC) может использоваться для цифровой подписи данных (хранящихся, например, на сервере). При последующем доступе к этим данных подпись можно будет проверить, чтобы верифицировать их подлинность. В отличие от алгоритмов хеширования, рассмотренных в предыдущем пункте, HMAC также использует секретный ключ для формирования подписи. Без наличия ключа проверка подписи будет невозможна.

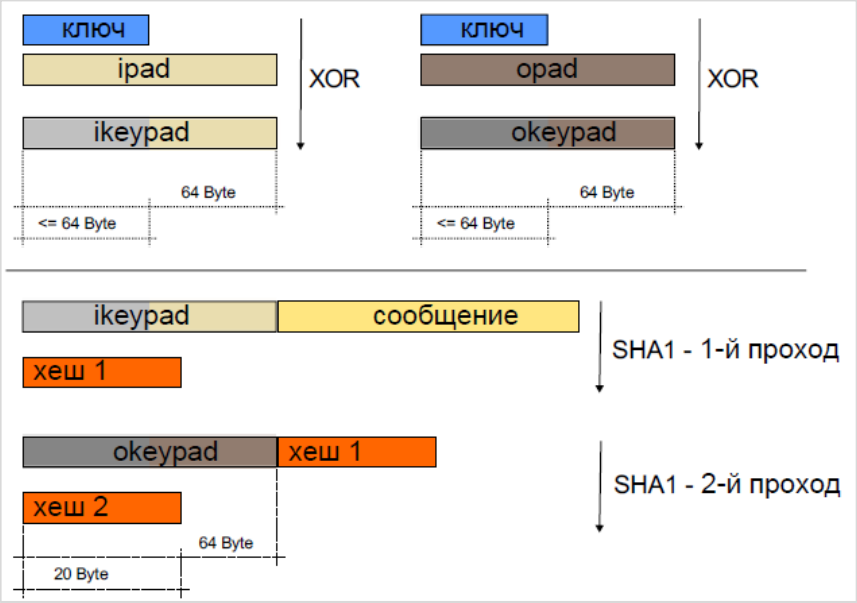

Рис. 3. Реализация HMAC для алгоритма SHA1

На рисунке приведена реализация HMAC для алгоритма SHA1. Более подробная информация доступна в сети – например, в статье на Википедии.

Ниже приведен пример создания подписи (SignMessage) и ее проверки (VerifyMessage) для данных sMessage с ключом _sSecret и использованием алгоритма SHA-1.

```
VAR
        _hHMAC : SysTypes.RTS_IEC_HANDLE :=
                CmpCrypto.CryptoGetAlgorithmById(ui32CryptoID :=
                        CmpCrypto.RtsCryptoID.HMAC_SHA1, pResult := 0);
        _sSecret : SECRET[3] := 'MySecret';
        abySignature : SIGNATURE[4];
        xMessageOK : BOOL;
        sMessage : MESSAGE := 'The red fox runs across the ice';

END_VAR

abySignature := SignMessage(sMessage);
xMessageOK := VerifyMessage(sMessage, abySignature);
```

[3] SECRET – это псевдоним для типа STRING(40)
[4] SIGNATURE – это псевдоним для типа ARRAY[0..255] OF BYTE

Для создания подписи и ее проверки используется один и тот же секретный ключ. Пока этот ключ действительно является секретным, то вероятность неавторизованного изменения данных крайне мала. Разумеется, вероятность зависит от степени надежности ключа. Чем проще этот ключ подобрать или взломать перебором, тем ниже его надежность.

```
METHOD SignMessage : ULINT
VAR_INPUT
    sMessage : REFERENCE TO MESSAGE;
    szMessage : ULINT;
    abySignature : REFERENCE TO SIGNATURE;
END_VAR
VAR
    Result : SysTypes.RTS_IEC_RESULT;
    bsSecret : CmpCrypto.RtsByteString := (ui32MaxLen := SIZEOF(SECRET),
        ui32Len := TO_UDINT(_szSecret), pByData := ADR(_sSecret));
    ksStorage : CmpCrypto.RtsCryptoKeyStorage := (byteString:=bsSecret);
    ckSecret : CmpCrypto.RtsCryptoKey := (keyType :=
        CmpCrypto.RtsCryptoKeyType.KeyType_Key, key:=ksStorage);
    bsMessage : CmpCrypto.RtsByteString := (ui32MaxLen := SIZEOF(MESSAGE),
        ui32Len := TO_UDINT(szMessage), pByData := ADR(sMessage));
    bsSignature : CmpCrypto.RtsByteString := (ui32MaxLen := SIZEOF(SIGNATURE),
        ui32Len := 0, pByData := ADR(abySignature));
END_VAR


Result := CmpCrypto.CryptoHMACSign(hAlgo := _hHMAC, pData := ADR(bsMessage),
    key := ckSecret, pSignature := ADR(bsSignature) );
```

```
METHOD VerifyMessage : BOOL
VAR_INPUT
    sMessage : REFERENCE TO MESSAGE;
    szMessage : ULINT;
    abySignature : REFERENCE TO SIGNATURE;
    szSignature : ULINT;
END_VAR
VAR
    Result : SysTypes.RTS_IEC_RESULT;
    bsSecret : CmpCrypto.RtsByteString := (ui32MaxLen := SIZEOF(_sSecret),
        ui32Len := TO_UDINT(_szSecret), pByData := ADR(_sSecret) );
    ksStorage : CmpCrypto.RtsCryptoKeyStorage := (byteString := bsSecret);
    ckSecret : CmpCrypto.RtsCryptoKey := (keyType:=
        CmpCrypto.RtsCryptoKeyType.KeyType_Key, key := ksStorage);
    bsMessage : CmpCrypto.RtsByteString := (ui32MaxLen := SIZEOF(MESSAGE),
        ui32Len := TO_UDINT(szMessage), pByData := ADR(sMessage) );
    abyNewSignature : SIGNATURE;
    bsSignature : CmpCrypto.RtsByteString := (ui32MaxLen := SIZEOF(SIGNATURE),
        ui32Len := 0, pByData := ADR(abyNewSignature) );
    diCmpResult : DINT;
END_VAR

Result := CmpCrypto.CryptoHMACSign(hAlgo := _hHMAC, pData := ADR(bsMessage),
    Key := ckSecret, pSignature := ADR(bsSignature) );
diCmpResult := SysMem.SysMemCmp(pBuffer1 := ADR(abySignature),
        pBuffer2 := ADR(abyNewSignature), udiCount := TO_UDINT(szSignature) );
VerifyMessage := (diCmpResult = 0);
```

Примечание: сравнение подписей должно проводиться очень тщательно. В случае ошибки данные будут признаны корректными (т.е. опубликованными авторизованным лицом), даже если они были изменены за время хранения.

4. Шифрование

Шифрование – это процедура конвертации открытого текста (plaintext) в шифротекст (cipher text). Термины «открытый текст» и «шифротекст» сложились исторически, и в настоящий момент под ними понимаются не только текстовые данные, а любая информация – аудиофайлы, изображения, исходные коды программ и т.д. Для шифрования всех этих типов данных применяются общие криптографические методы.

4.1. Симметричное шифрование

Симметричное шифрование – это способ шифрования, при котором для шифрования и расшифровки данных применяется один и тот же секретный ключ. Почти все методы симметричного шифрования оптимизированы для использования во встраиваемых системах – они не требуют значительных аппаратных ресурсов, характеризуются низким энергопотреблением и просты в реализации. Использование общего ключа позволяет быстро выполнять процедуры шифрования и расшифровки, а применение длинных ключей обеспечивает высокий уровень безопасности.

Важным моментом симметричного шифрования является обмен ключами – если предположить, что отправитель имеет такой ключ по умолчанию, то получатель должен каким-то образом получить его для расшифровки данных. Если этот ключ получит злоумышленник, то он сможет не только расшифровать конфиденциальные данные, но и зашифровать свои собственные – например, с целью дезинформации одной из сторон. Для решения проблемы обмена ключами используются методы ассиметричного шифрования.

Совместное использование обоих методов (симметричных для шифрования данных и ассиметричных для обмена ключами) позволяет сохранить быстродействие криптографических процедур и обеспечить защиту секретных ключей. Более подробная информация приведена в материалах по гибридному шифрованию, а также примере использования библиотек.

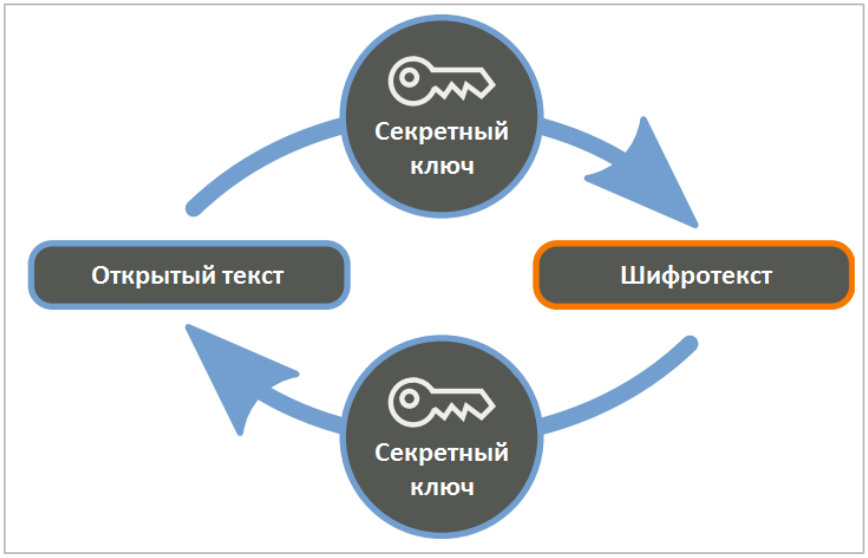

Рис. 4. Принцип симметричной криптографии

Пример использования алгоритма AES-256 в режиме CBC

Дополнительная информация:

- Алгоритм AES
- CBC (Cipher Block Chaining) – режим сцепления блоков шифротекста
- Вектор инициализации
- Процедура дополнения

```
VAR
    _hSymmetricCryptoCipher : SysTypes.RTS_IEC_HANDLE :=
        CmpCrypto.CryptoGetAlgorithmById(ui32CryptoID :=
                CmpCrypto.RtsCryptoID.AES_256_CBC, pResult :=0 );
    _szBlock : ULINT := 16; // Размер блока для "_hSymmetricCryptoCipher" =>
                            // 128 бит (16 байт) для алгоритма AES-256-CBC
    _sKey : SECRET; // 256-битный секретный ключ
    _szKey : ULINT := 32; // длина ключа – 256 бит(32 бит)
    _sInitVector : MESSAGE; // массив случайных чисел для инициализации "_szBlock"
END_VAR
```

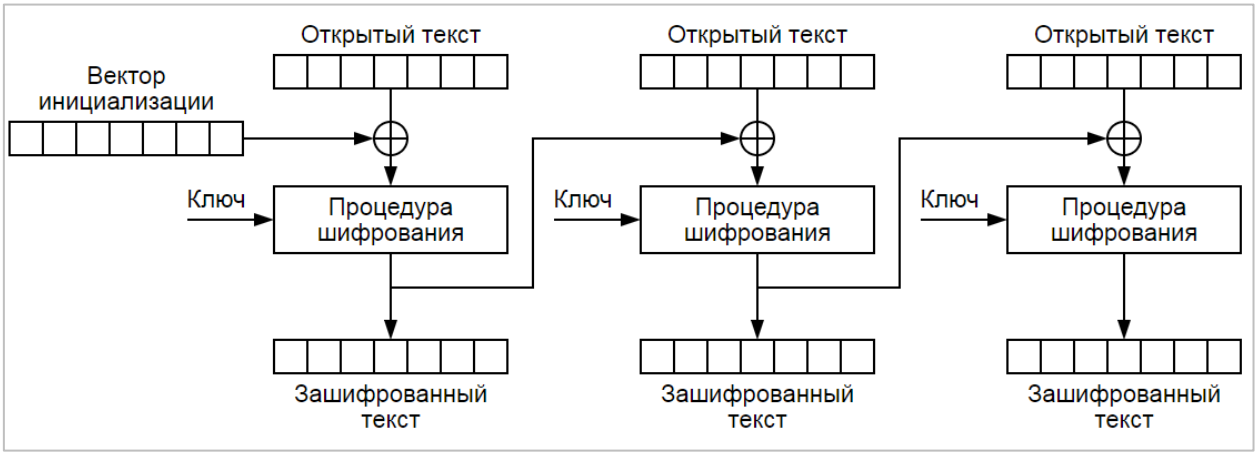

Рис. 5. Шифрование в режиме CBC (режиме сцепления блоков шифротекста)

Для шифрования первого блока данных используется вектор инициализации, для всех последующих блоков – предыдущий зашифрованный блок. Каждый зашифрованный блок уникален, поэтому даже последовательно расположенные повторяющиеся блоки данных будут зашифрованы различным образом. Секретный ключ, используемый для шифрования и расшифровки, не должен быть доступен третьим лицам. Вектор инициализации должен быть известен и отправителю, и получателю, но хранить его в секрете не требуется.

Инициализация ключа и вектора случайными числами крайне затрудняет попытки взлома шифра в случае отсутствия ключа.

```
METHOD SymmetricEncryptMessage : ULINT
VAR_INPUT
    sPlainText : REFERENCE TO MESSAGE;
    szPlainText : ULINT;
    sCipherText : REFERENCE TO BUFFER[5];
END_VAR
VAR
    Result : SysTypes.RTS_IEC_RESULT;
    bsKey : CmpCrypto.RtsByteString := (ui32MaxLen := SIZEOF(_sKey),
        ui32Len := TO_UDINT(_szKey), pByData := ADR(_sKey));
    ksStorage : CmpCrypto.RtsCryptoKeyStorage := (byteString := bsKey);
    ckKey : CmpCrypto.RtsCryptoKey := (keyType :=
        CmpCrypto.RtsCryptoKeyType.KeyType_Key, key := ksStorage);
    bsInitVector : CmpCrypto.RtsByteString := (ui32MaxLen := SIZEOF(_sInitVector),
        ui32Len := TO_UDINT(_szBlock), pByData := ADR(_sInitVector));
    bsPlainText : CmpCrypto.RtsByteString := (ui32MaxLen := SIZEOF(MESSAGE),
        ui32Len := TO_UDINT(szPlainText), pByData := ADR(sPlainText));
    bsCipherText : CmpCrypto.RtsByteString := (ui32MaxLen := SIZEOF(MESSAGE),
        ui32Len := 0, pByData := ADR(sCipherText));
END_VAR

Result := CmpCrypto.CryptoGenerateRandomNumber(
        ui32NumOfRandomBytes := TO_UDINT(_szKey), pRandom := ADR(bsKey));
Result := CmpCrypto.CryptoGenerateRandomNumber(
        ui32NumOfRandomBytes := TO_UDINT(_szBlock),
                pRandom := ADR(bsInitVector));

Result := CryptoSymmetricEncrypt(
        hAlgo := _hSymmetricCryptoCipher,
        pPlainText := ADR(bsPlainText),
        key := ckKey,
        pInitVector := ADR(bsInitVector),
        xEnablePadding := TRUE,
        pCipherText := ADR(bsCipherText)
);

IF Result = CmpErrors.Errors.ERR_OK THEN
    SymmetricEncryptMessage := bsCipherText.ui32Len;
END_IF
```

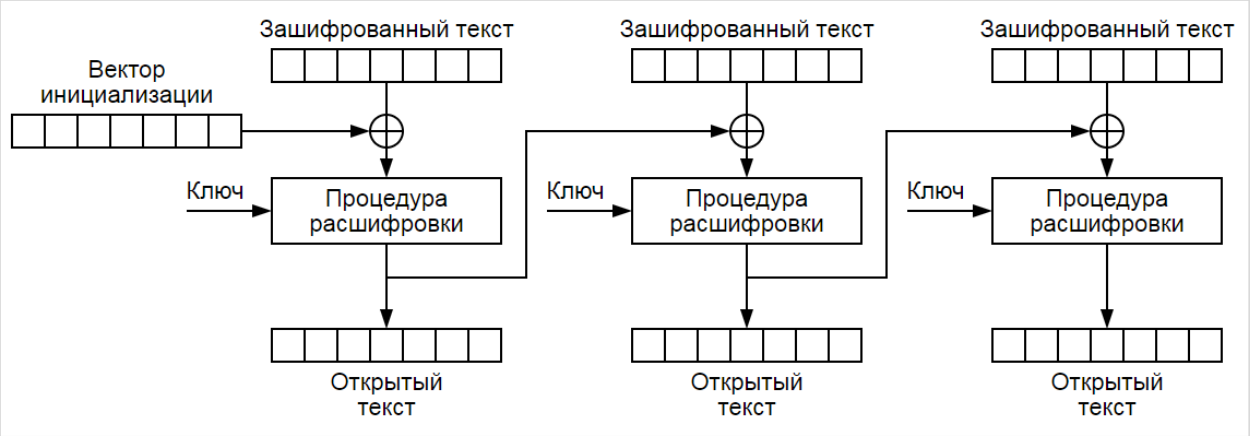

Рис. 6. Расшифровка в режиме CBC (режиме сцепления блоков шифротекста)

[5] BUFFER – это псевдоним для типа ARRAY[0..4096] OF BYTE

```
METHOD SymmetricDecryptMessage : ULINT
VAR_INPUT
    sCipherText : REFERENCE TO BUFFER;
    szCipherText : ULINT;
    sPlainText : REFERENCE TO MESSAGE;
END_VAR
VAR
    Result : SysTypes.RTS_IEC_RESULT;
    bsKey : CmpCrypto.RtsByteString := (ui32MaxLen := SIZEOF(_sKey),
        ui32Len := TO_UDINT(_szKey), pByData := ADR(_sKey));
    ksStorage : CmpCrypto.RtsCryptoKeyStorage := (byteString := bsKey);
    ckKey : CmpCrypto.RtsCryptoKey := (keyType :=
        CmpCrypto.RtsCryptoKeyType.KeyType_Key, key:=ksStorage);
    bsInitVector : CmpCrypto.RtsByteString := (ui32MaxLen := SIZEOF(MESSAGE),
        ui32Len := TO_UDINT(_szBlock), pByData := ADR(_sInitVector));
    bsCipherText : CmpCrypto.RtsByteString := (ui32MaxLen := SIZEOF(MESSAGE),
        ui32Len := TO_UDINT(szCipherText), pByData := ADR(sCipherText));
    bsPlainText : CmpCrypto.RtsByteString := (ui32MaxLen := SIZEOF(MESSAGE),
        ui32Len:=0, pByData := ADR(sPlainText));
END_VAR

Result := CmpCrypto.CryptoSymmetricDecrypt(
        hAlgo := _hSymmetricCryptoCipher,
        pCipherText := ADR(bsCipherText),
        key := ckKey,
        pInitVector := ADR(bsInitVector),
        xEnablePadding := TRUE,
        pPlainText := ADR(bsPlainText)
);
IF Result = CmpErrors.Errors.ERR_OK THEN
    SymmetricDecryptMessage := bsPlainText.ui32Len;
END_IF
```

Протестировать функции шифрования и расшифровки можно с помощью такого кода:

```
szCipherText := SymmetricEncryptMessage(sPlainText, TO_ULINT(LEN(sPlainText)),
        sCipherText);
szDecryptedText := SymmetricDecryptMessage(sCipherText, szCipherText, sDecryptedText);

IF szDecryptedText > 0 THEN
    diCmpResult := SysMem.SysMemCmp(pBuffer1 := ADR(sPlainText),
        pBuffer2 := ADR(sDecryptedText, udiCount := TO_UDINT(szDecryptedText));
END_IF
```

Примечание: эффективность и надежность шифрования зависит не только от выбранного алгоритма, но и от его параметров – режима шифрования, длины ключа, значения ключа и т.д. При неправильных значениях этих параметров злоумышленнику не составит труда получить доступ к данным. По возможности следует использовать уже известные и зарекомендовавшие себя технологии. Например, при передаче данных по TCP/IP следует использовать протокол TLS. Такой канал связи можно считать безопасным, и передаваемые по нему данные шифровать уже не обязательно.

Совместное использование цифровой подписи и шифрования

Ранее мы уже упоминали код аутентификации сообщений на основе хеша (HMAC), который используется для проверки целостности данных и верификации их автора. При использовании этого метода важно соблюдать правильную последовательность операций: отправитель сначала выполняет шифрование данных, а потом формирует подпись. Получатель сначала проверяет подпись, и производит расшифровку только в том случае, если подпись соответствует ожидаемой. Если подпись не проходит проверку, то получатель не должен расшифровывать данные, так как результат расшифровки специально подготовленных сообщений позволит злоумышленнику получить определенную информацию об использующихся криптографических алгоритмах и их параметрах.

4.2. Асимметричное шифрование

При ассиметричном шифровании отправитель создает два ключа – открытый и закрытый. Закрытый ключ должен храниться в секрете. Открытый ключ может свободно распространяться, так как по его значению невозможно получить какую-либо информацию о закрытом ключе.

Набор этих ключей позволяет выполнять следующие операции:

- открытый ключ может применяться любым пользователем для шифрования данных, но расшифровка будет возможна только с использованием закрытого ключа;
- закрытый ключ может применяться для создания цифровой подписи, но ее проверка будет возможна только с использованием открытого ключа.

Асимметричное шифрование выглядит идеальным средством обеспечения информационной безопасности. Но необходимо обратить внимание на следующие моменты:

- для генерации ключей должно использоваться надежное, регулярно обновляемое ПО;
- доступ к закрытому ключу должен быть только у его владельца;
- владелец открытого ключа должен быть идентифицирован. Обычно это производится в помощью сертификата, выдаваемого специальными организациями (центрами сертификации или уполномоченными ими компаниями). Крайне важно проверить сертификат перед началом выполнения операций с открытым ключом;
- операции ассиметричного шифрования довольно ресурсоемкие и не подходят для обработки большого количества данных.

Использование асимметричного шифрования для обмена ключами (что является уже гораздо менее ресурсоемким) и симметричного шифрования для передачи данных позволяет обеспечить как надежность, так и быстродействие.

Более подробная информация приведена в материалах по гибридному шифрованию, а также примере использования библиотек.

Генерация ключей

В среде CODESYS для работы с сертификатами и ключами используется вкладка **Оболочка ПЛК**, расположенная в узле **Device**. В первую очередь пользователю надо создать хранилище сертификатов с помощью программного кода:

```
_sComponentName : STRING := 'MyTestApplication';
_dwComponentVersion : DWORD := 16#0001;
_dwComponentId : DWORD := 16#7000;
_hComponent : SysTypes.RTS_IEC_HANDLE := SysTypes.HandleConstants.RTS_INVALID_HANDLE;
_hCertStore : SysTypes.RTS_IEC_HANDLE := SysTypes.HandleConstants.RTS_INVALID_HANDLE;
_hCert : SysTypes.RTS_IEC_HANDLE := SysTypes.HandleConstants.RTS_INVALID_HANDLE;
Result : SysTypes.RTS_IEC_RESULT;

hComponent := Component_Manager.CMAddComponent(sComponentName, dwComponentId,
        dwComponentVersion, ADR(Result) );

IF Result = CmpErrors.Errors.ERR_OK THEN
        hCertStore := CmpX509Cert.X509CertStoreOpen(dwComponentId, ADR(Result) );
END_IF
```

После этого хранилище будет отображаться в списке компонентов, который можно увидеть через оболочку ПЛК (команда **cert-getapplist**):

```
> cert-getapplist

Nr.    ComponentName              CommonName                 CertAvailable DateNotBefore
-------------------------------------------------------------------------------------------
0      CmpOPCUAServer             OPCUAServer@DOLLWONB       FALSE         --
1      CmpSecureChannel           DOLLWONB                   FALSE         --
2      CmpApp                     DOLLWONB                   FALSE         --
3      CmpWebServer               DOLLWONB                   FALSE         --
4      MyTestApplication          MyTestApplication          FALSE         --
```

Рис. 7. Список компонентов

В первом столбце списка отображается ID компонента (для созданного хранилища ID=4). Зная его, можно создать новый самоподписанный сертификат командой **cert-genselfsigned**. Аргументами команды являются ID хранилища и число дней действия сертификата:

```
> cert-genselfsigned 4 expdays=10

Generate self-signed certificate with given index with valid time of 10 day(s). Check logger to see
```

Рис. 8. Команда создания самоподписанного сертификата

Генерация сертификата займет некоторое время. Информация о завершении операции будет доступна на вкладке **Журнал** в узле **Device**:

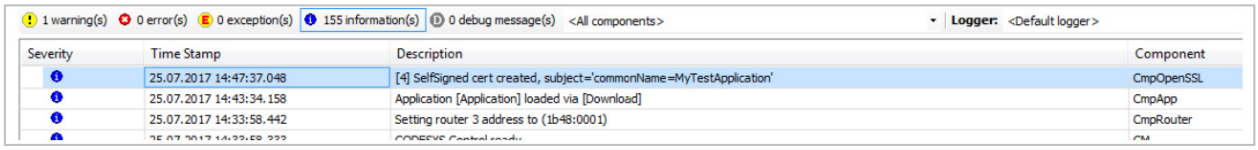

Рис. 9. Журнал ПЛК

Список сертификатов можно посмотреть с помощью команды **cert-getcertlist own**:

```
> cert-getcertlist own

-----------------------------------------------------
List of own certificates
-----------------------------------------------------
Number: 0
Thumbprint: 132245455ef12cf27ee94bd5125350bdc84fb4e5
Subjects:
 - commonName: MyTestApplication
Valid from: 25.7.2017 12:47:36
Valid until: 4.8.2017 12:47:36
```

Рис. 10. Список доступных сертификатов

Для каждого сертификата в хранилище содержаться открытый и закрытый ключ. Как уже упоминалось выше, должна быть возможность идентификации открытого ключа. Для этого сертификат подписывается центром сертификации. Выше был сгенерирован так называемый «самоподписанный» сертификат, который не вызовет доверия у пользователей.

Рис. 11. Генерация ключей при создании сертификата

Для получения подписи необходимо сформировать запрос (CSR) с помощью команды **cert-createcsr**, которая в качестве аргумента принимает ID хранилища:

```
> cert-createcsr 4

Create CSR for application with given index. Check logger to see when finished.
```

Рис. 12. Команда формирования запроса на подпись (CSR)

Информация о завершении создания CSR будет доступна в журнале ПЛК.

После этого в папке **/cert/import** (см. узел **Device** – вкладка **Файлы**) появится новый файл формата **.csr**, который следует отправить в центр сертификации:

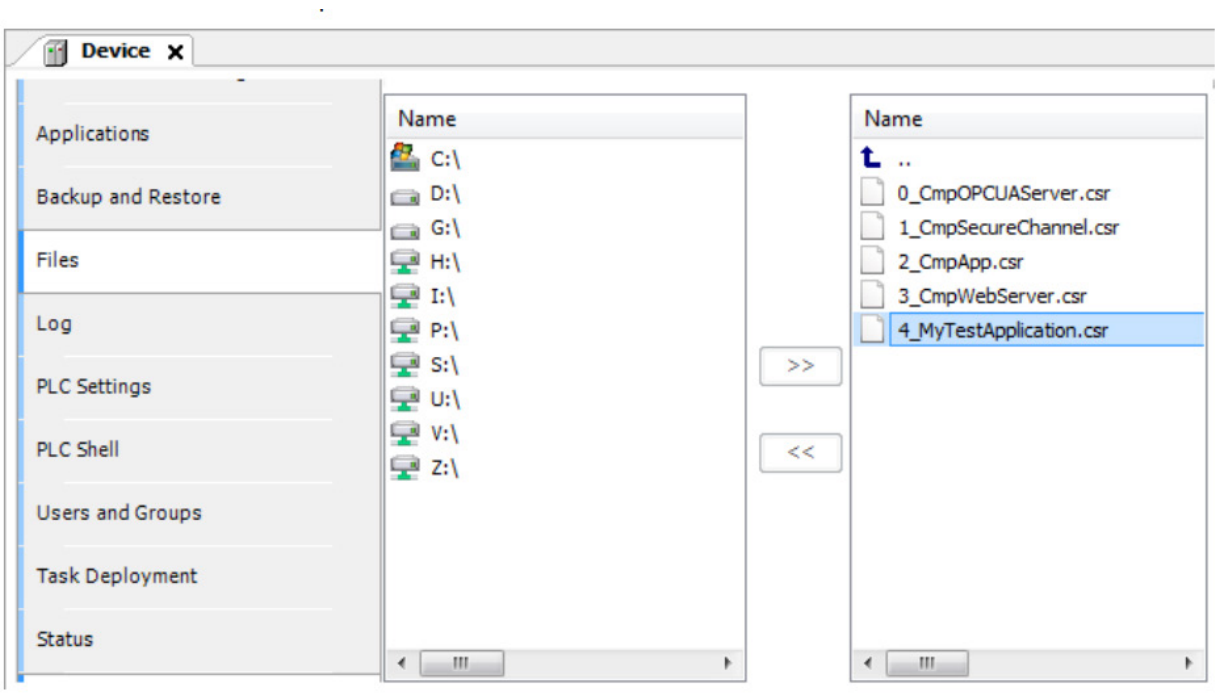

Рис. 13. Файл созданного запроса на подпись

Переменные программы

```
_hAsymmetricCryptingCipher : SysTypes.RTS_IEC_HANDLE :=
        CmpCrypto.CryptoGetAlgorithmById(ui32CryptoID:=
                CmpCrypto.RtsCryptoID.RSA_PKCS1_V15_PADDING, pResult:=0);
_hAsymmetricSigningCipher  : SysTypes.RTS_IEC_HANDLE :=
        CryptoGetAlgorithmById(ui32CryptoID:=RtsCryptoID.RSA_OAEP_SHA256,
                pResult:=0);
_sComponentName : STRING := 'MyTestApplication';
_dwComponentVersion : DWORD := 16#0001;
_dwComponentId : DWORD := 16#7000;
_hComponent : SysTypes.RTS_IEC_HANDLE := SysTypes.HandleConstants.RTS_INVALID_HANDLE;
_hCertStore : SysTypes.RTS_IEC_HANDLE := SysTypes.HandleConstants.RTS_INVALID_HANDLE;
_hClaim : SysTypes.RTS_IEC_HANDLE := SysTypes.HandleConstants.RTS_INVALID_HANDLE;
_hCert : SysTypes.RTS_IEC_HANDLE := SysTypes.HandleConstants.RTS_INVALID_HANDLE;
abyCipherText : BUFFER;
xCertOk : BOOL;
```

Доступ к сертификату

```
METHOD ProvideCertificate : BOOL
VAR
    certInfo : CmpX509Cert.RtsX509CertInfo;
    subject : CmpX509Cert.RtsX509NameEntry;
END_VAR

IF _hComponent = SysTypes.HandleConstants.RTS_INVALID_HANDLE THEN
    _hComponent := Component_Manager.CMAddComponent(_sComponentName, _dwComponentId,
         _dwComponentVersion, ADR(Result) );
    IF Result = CmpErrors.Errors.ERR_OK THEN
        _hCertStore := CmpX509Cert.X509CertStoreOpen(_dwComponentId, 0);
        certInfo.subject.numOfEntries := 1;
        certInfo.subject.entries := ADR(subject);
        RtsOIDCreate(ADR(CmpX509Cert.KnownOIDs.RTS_OID_COMMON_NAME), ADR(subject.id));
        subject.value := ADR(_sComponentName);
        _hClaim := CmpX509Cert.X509CertStoreRegister(_hCertStore, _dwComponentId,
                  ADR(certInfo), ADR(Result));
        IF Result = CmpErrors.Errors.ERR_OK THEN
            _hCert := CmpX509Cert.X509CertStoreGetRegisteredCert
                 (_hCertStore, _hClaim, 0);
        END_IF
    END_IF
END_IF

ProvideCertificate := (_hCert <> SysTypes.HandleConstants.RTS_INVALID_HANDLE);
```

Удаление сертификата

```
METHOD ReleaseCertificate

_hCert := SysTypes.HandleConstants.RTS_INVALID_HANDLE;
IF _hCertStore <> SysTypes.HandleConstants.RTS_INVALID_HANDLE THEN
    CmpX509Cert.X509CertStoreUnregister(_hCertStore, _hClaim);
    _hClaim := SysTypes.HandleConstants.RTS_INVALID_HANDLE;
    CmpX509Cert.X509CertStoreClose(_hCertStore);
    _hCertStore := SysTypes.HandleConstants.RTS_INVALID_HANDLE;
END_IF
IF _hComponent <> SysTypes.HandleConstants.RTS_INVALID_HANDLE THEN
    Component_Manager.CMRemoveComponent(_hComponent);
    _hComponent := SysTypes.HandleConstants.RTS_INVALID_HANDLE;
END_IF
```

Шифрование

```
METHOD AsymmetricEncryptMessage : ULINT
VAR_INPUT
    sPlainText : REFERENCE TO MESSAGE;
    szPlainText : ULINT;
    abyCipherText : REFERENCE TO BUFFER;
END_VAR
VAR
    Result : SysTypes.RTS_IEC_RESULT;
    ksPublicKey : CmpCrypto.RtsCryptoKey;
    bsPlainText : CmpCrypto.RtsByteString := (ui32MaxLen := SIZEOF(MESSAGE),
        ui32Len := TO_UDINT(szPlainText), pByData: = ADR(sPlainText));
    bsCipherText : CmpCrypto.RtsByteString := (ui32MaxLen := SIZEOF(BUFFER),
        ui32Len := 0, pByData := ADR(abyCipherText) );
END_VAR

Result := CmpX509Cert.X509CertGetPublicKey(
    hCert:=_hCert,
    pPublicKey:=ADR(ksPublicKey)
);

Result := CmpCrypto.CryptoAsymmetricEncrypt(
    hAlgo := _hAsymmetricCryptingCipher,
    pPlainText := ADR(bsPlainText),
    publicKey := ksPublicKey,
    pCipherText := ADR(bsCipherText)
);

AsymmetricEncryptMessage := bsCipherText.ui32Len;
```

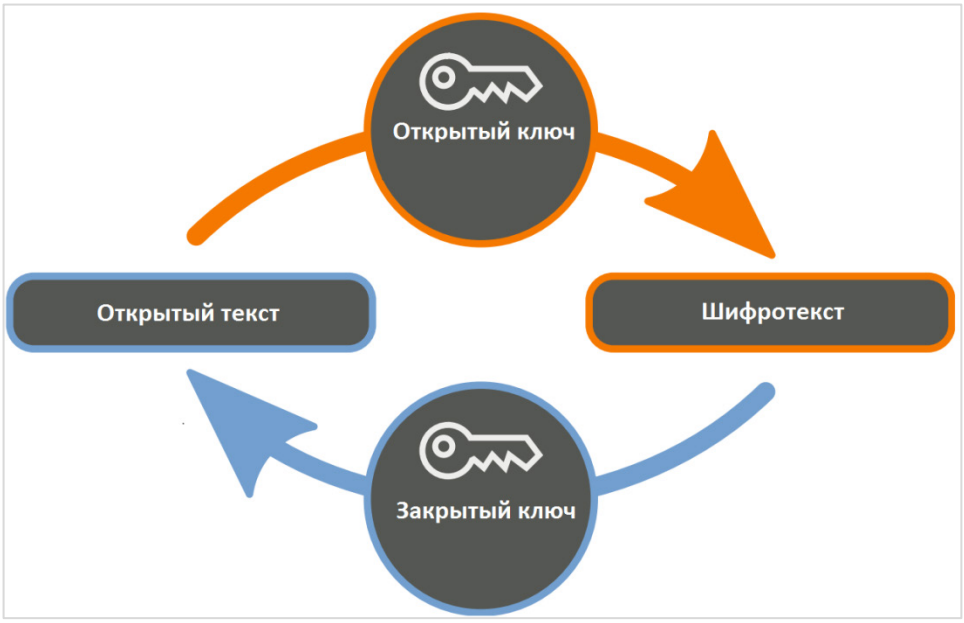

Рис. 14. Алгоритм ассиметричного шифрования

Расшифровка

```
METHOD AsymmetricDecryptMessage : ULINT
VAR_INPUT
    abyCipherText : REFERENCE TO BUFFER;
    szCipherText : ULINT;
    sPlainText : REFERENCE TO MESSAGE;
END_VAR
VAR
    Result : SysTypes.RTS_IEC_RESULT;
    ksPrivateKey : CmpCrypto.RtsCryptoKey;
    bsCipherText : CmpCrypto.RtsByteString := (ui32MaxLen := SIZEOF(BUFFER),
         ui32Len := TO_UDINT(szCipherText), pByData := ADR(abyCipherText));
    bsPlainText : CmpCrypto.RtsByteString := (ui32MaxLen:=SIZEOF(MESSAGE),
        ui32Len := 0, pByData := ADR(sPlainText) );
END_VAR

Result := CmpX509Cert.X509CertGetPrivateKey(
    hCertStore := _hCertStore,
    hCert := _hCert,
    pPrivateKey := ADR(ksPrivateKey)
);

Result := CmpCrypto.CryptoAsymmetricDecrypt(
    hAlgo := _hAsymmetricCryptingCipher,
    pCipherText := ADR(bsCipherText),
    privateKey := ksPrivateKey,
    pPlainText := ADR(bsPlainText)
);

AsymmetricDecryptMessage := bsPlainText.ui32Len;
```

Создание подписи

```
METHOD AsymmetricSignMessage : ULINT
VAR_INPUT
    sMessage : REFERENCE TO MESSAGE;
    szMessage : ULINT;
    abySignature : REFERENCE TO SIGNATURE;
END_VAR
VAR
    Result : SysTypes.RTS_IEC_RESULT;
    ksPrivateKey : CmpCrypto.RtsCryptoKey;
    bsPlainText : CmpCrypto.RtsByteString := (ui32MaxLen := SIZEOF(MESSAGE),
        ui32Len := TO_UDINT(szMessage), pByData := ADR(sMessage));
    bsSignature : CmpCrypto.RtsByteString := (ui32MaxLen:=SIZEOF(abySignature),
        ui32Len := 0, pByData := ADR(abySignature));
END_VAR

Result := CmpX509Cert.X509CertGetPrivateKey(
    hCertStore := _hCertStore,
    hCert := _hCert,
    pPrivateKey := ADR(ksPrivateKey)
);

Result := CmpCrypto.CryptoSignatureGenerate(
    hAlgo := _hAsymmetricSigningCipher,
    pData := ADR(bsPlainText),
    privateKey := ksPrivateKey,
    pSignature := ADR(bsSignature)
);

AsymmetricSignMessage := bsSignature.ui32Len;
```

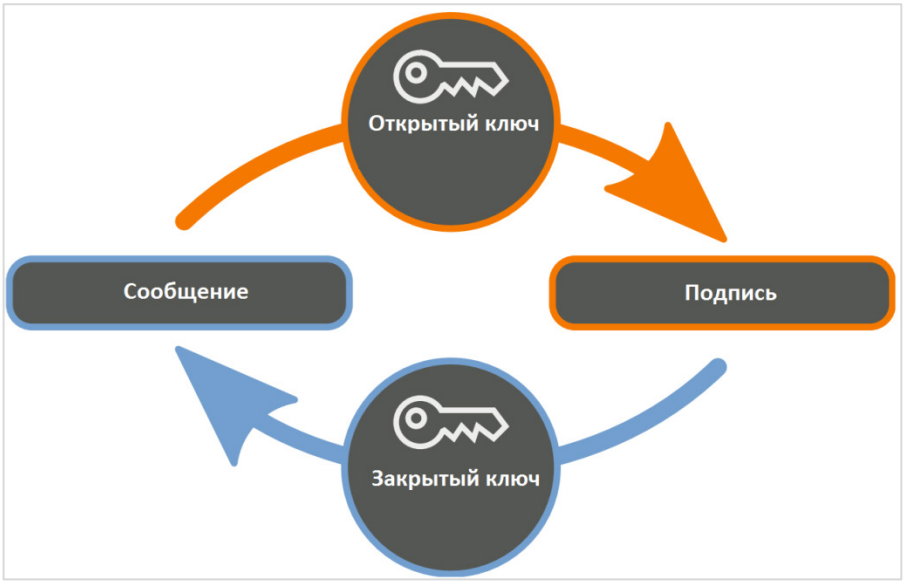

Рис. 15. Алгоритм создания подписи

Проверка подписи

```
METHOD AsymmetricVerifyMessage : BOOL
VAR_INPUT
    sMessage : REFERENCE TO MESSAGE;
    szMessage : ULINT;
    abySignature : REFERENCE TO SIGNATURE;
    szSignature : ULINT;
END_VAR
VAR
    Result : SysTypes.RTS_IEC_RESULT;
    ksPublicKey : CmpCrypto.RtsCryptoKey;
    bsMessage : CmpCrypto.RtsByteString := (ui32MaxLen := SIZEOF(MESSAGE),
        ui32Len := TO_UDINT(szMessage), pByData := ADR(sMessage));
    bsSignature : CmpCrypto.RtsByteString := (ui32MaxLen:=SIZEOF(SIGNATURE),
        ui32Len := TO_UDINT(szSignature), pByData := ADR(abySignature));
END_VAR

Result := CmpX509Cert.X509CertGetPublicKey(
    hCert := _hCert,
    pPublicKey := ADR(ksPublicKey)
);

Result := CmpCrypto.CryptoSignatureVerify(
    hAlgo := _hAsymmetricSigningCipher,
    pData := ADR(bsMessage),
    publicKey := ksPublicKey,
    pSignature := ADR(bsSignature)
);

AsymmetricVerifyMessage := (Result = CmpErrors.Errors.ERR_OK);
```

5. Пример использования библиотек

Информационные системы (в том числе АСУ ТП) выполняют различные функции. Некоторые из них доступны всем пользователям. Другие требуют авторизации и доступны только в течение ограниченной по времени сессии. Важно заранее понимать, что процесс авторизации может происходить по незащищенному каналу связи. Перед началом авторизации клиентское ПО должно идентифицировать сервер (чтобы, например, не передать пароль на сервер злоумышленника).

Подпись и шифрование документов позволяют подтвердить их подлинность и ограничить доступ. В метаинформации документа может быть указана дата его создания. Цифровая подпись позволяет верифицировать автора документа и определить, вносились ли в него какие-либо изменения за время хранения. Шифрование позволяет обеспечить доступ к содержимому документа только ограниченному кругу лиц.

На рисунке ниже приведена схема шифрования и расшифровки сообщения при использовании гибридной криптографии:

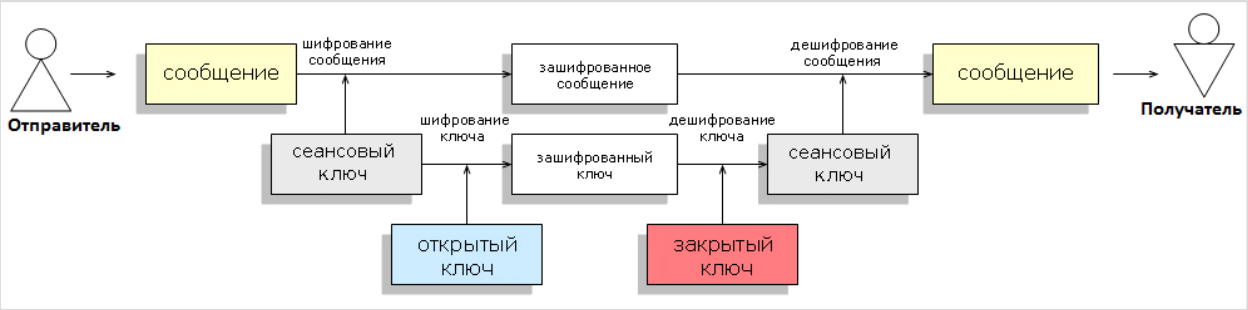

Рис. 16. Шифрование и расшифровка данных при использовании гибридной криптографии

Этап отправки:

- отправитель генерирует случайный сеансовый ключ;
- сообщение отправителя шифруется сеансовым ключом (с помощью симметричного алгоритма);
- сеансовый ключ шифруется открытым ключом получателя (асимметричным алгоритмом);
- отправитель посылает получателю зашифрованное сообщение и зашифрованный сеансовый ключ.

Этап приёма:

- получатель принимает зашифрованное сообщение отправителя и зашифрованный сеансовый ключ;
- получатель расшифровывает сеансовый ключ своим закрытым ключом;
- при помощи полученного таким образом сеансового ключа получатель расшифровывает зашифрованное сообщение отправителя.

Некоторые комментарии по описанной процедуре:

- использование случайного ключа для симметричного шифрования повышает безопасность, поскольку его значение непредсказуемо и практически не поддается подбору;
- данные шифруются симметричными алгоритмами, так как они нетребовательны к ресурсам и позволяют осуществить шифрование больших объемов информации за приемлемое время;
- ассиметричные алгоритмы, напротив, требовательны к ресурсам, поэтому используются только для шифрования ключа симметричных алгоритмов;
- подпись данных позволяет верифицировать их автора (при условии, что закрытый ключ не скомпрометирован).

Ниже приведен пример кода для создания зашифрованного сообщения и его проверки.

```
METHOD CreateDocument : BOOL;
VAR_INPUT
    ckPublicKey : CmpCrypto.RtsCryptoKey;
    bsData : CmpCrypto.RtsByteString;
    bsDocument : REFERENCE TO CmpCrypto.RtsByteString;
END_VAR
VAR
    bsEncryptedData : CmpCrypto.RtsByteString := (ui32MaxLen := 32, ui32Len := 0,
       pByData := bsDocument.pByData);
    bsEncryptedKey : CmpCrypto.RtsByteString := (ui32MaxLen := 256, ui32Len := 0,
       pByData := bsDocument.pByData + 32);
    bsSignature : CmpCrypto.RtsByteString := (ui32MaxLen := 256, ui32Len:=0,
       pByData := bsDocument.pByData + 32 + 256);
    bsInitVector : CmpCrypto.RtsByteString := (ui32MaxLen := TO_UDINT(_szBlock),
       ui32Len := TO_UDINT(_szBlock), pbyData := bsDocument.pByData + 32 + 256 + 256);

    Result : SysTypes.RTS_IEC_RESULT;

    abyKey : ARRAY[0..31] OF BYTE; // 256 Bit
    bsKey : CmpCrypto.RtsByteString := (ui32MaxLen := SIZEOF(abyKey),
        ui32Len := TO_UDINT(_szKey), pbyData := ADR(abyKey) );
    ksStorage : CmpCrypto.RtsCryptoKeyStorage := (byteString := bsKey);
    ckKey : CmpCrypto.RtsCryptoKey :=
        (keyType := CmpCrypto.RtsCryptoKeyType.KeyType_Key, key := ksStorage);

    ckPrivateKey : CmpCrypto.RtsCryptoKey;
    bsCollection : CmpCrypto.RtsByteString := (ui32MaxLen := 32 + 256,
        ui32Len := 32 + 256, pByData := bsDocument.pByData);
END_VAR

// генерируем случайный сеансовый ключ
Result := CmpCrypto.CryptoGenerateRandomNumber(ui32NumOfRandomBytes :=
        TO_UDINT(_szKey), pRandom := ADR(bsKey) );

// генерируем вектор инициализаци
Result := CmpCrypto.CryptoGenerateRandomNumber
        (ui32NumOfRandomBytes := TO_UDINT(_szBlock), pRandom := ADR(bsInitVector) );

// шифруем данные симметричным алгоритмом с использованием сеансового ключа
Result := CmpCrypto.CryptoSymmetricEncrypt(
        hAlgo := _hSymmetricCryptoCipher,
        pPlainText := ADR(bsData),
        key := ckKey,
        pInitVector := ADR(bsInitVector),
        xEnablePadding := TRUE,
        pCipherText := ADR(bsEncryptedData)
);
```

```
// шифруем сеансовый ключ ассиметричным алгоритмом с использованием открытого ключа…
// …получателя
Result := CmpCrypto.CryptoAsymmetricEncrypt(
    hAlgo := _hAsymmetricCryptingCipher,
    pPlainText := ADR(bsKey),
    publicKey := ckPublicKey,
    pCipherText := ADR(bsEncryptedKey)
);

// подписываем зашифрованные данные и ключ ассиметричным алгоритмом с использованием…
// …закрытого ключа отправителя
Result := CmpX509Cert.X509CertGetPrivateKey(
    hCertStore := _hCertStore,
    hCert := _hCert,
    pPrivateKey := ADR(ckPrivateKey)
);
Result := CmpCrypto.CryptoSignatureGenerate(
    hAlgo := _hAsymmetricSigningCipher,
    pData := ADR(bsCollection),
    privateKey := ckPrivateKey,
    pSignature := ADR(bsSignature)
);
```

```
METHOD UtilizeDocument
VAR_INPUT
    ckPublicKey : CmpCrypto.RtsCryptoKey;
    bsDocument : CmpCrypto.RtsByteString;
    bsData : REFERENCE TO CmpCrypto.RtsByteString;
END_VAR
VAR
    bsEncryptedData : CmpCrypto.RtsByteString := (ui32MaxLen := 32, ui32Len := 32,
       pByData := bsDocument.pByData);
    bsEncryptedKey : CmpCrypto.RtsByteString := (ui32MaxLen := 256, ui32Len := 256,
       pByData := bsDocument.pByData + 32);
    bsSignature : CmpCrypto.RtsByteString := (ui32MaxLen := 256, ui32Len := 256,
       pByData:=bsDocument.pByData + 32 + 256);
    bsInitVector : CmpCrypto.RtsByteString := (ui32MaxLen := TO_UDINT(_szBlock),
       ui32Len := TO_UDINT(_szBlock), pbyData := bsDocument.pByData + 32 + 256 + 256);

    Result : SysTypes.RTS_IEC_RESULT;

    bsCollection : CmpCrypto.RtsByteString := (ui32MaxLen := 32 + 256,
        ui32Len := 32 + 256, pByData := bsDocument.pByData);
    ckPrivateKey : CmpCrypto.RtsCryptoKey;
    abyKey : ARRAY[0..255] OF BYTE;
    bsKey : CmpCrypto.RtsByteString := (ui32MaxLen := SIZEOF(abyKey), ui32Len := 0,
        pbyData := ADR(abyKey) );
    ksStorage : CmpCrypto.RtsCryptoKeyStorage;
    ckKey : CmpCrypto.RtsCryptoKey := (keyType :=
        CmpCrypto.RtsCryptoKeyType.KeyType_Key);
END_VAR

// проверяем подпись ассиметричным алгоритмом с использованием открытого ключа…
// …отправителя
Result := CmpCrypto.CryptoSignatureVerify(
    hAlgo := _hAsymmetricSigningCipher,
    pData := ADR(bsCollection),
    publicKey := ckPublicKey,
    pSignature := ADR(bsSignature)
);
IF Result <> CmpErrors.ERRORS.ERR_OK THEN
    RETURN;
END_IF
```

```
Result := CmpX509Cert.X509CertGetPrivateKey(
    hCertStore:=_hCertStore,
    hCert:=_hCert,
    pPrivateKey:=ADR(ckPrivateKey)
);

// расшифровываем сеансовый ключ ассиметричным алгоритмом с использованием…
// …закрытого ключа  получателя
Result := CmpCrypto.CryptoAsymmetricDecrypt(
    hAlgo := _hAsymmetricCryptingCipher,
    pCipherText := ADR(bsEncryptedKey),
    privateKey := ckPrivateKey,
    pPlainText := ADR(bsKey)
);
ksStorage.byteString := bsKey;
ckKey.key := ksStorage;

// расшифровываем данные симметричным алгоритмом с использованием сеансового ключа
Result := CmpCrypto.CryptoSymmetricDecrypt(
         hAlgo := _hSymmetricCryptoCipher,
         pCipherText := ADR(bsEncryptedData),
         key := ckKey,
         pInitVector := ADR(bsInitVector),
         xEnablePadding := TRUE,
         pPlainText := ADR(bsData)
);
```

Примечание: приведенные методы должны использоваться при передаче данных по незащищенному каналу связи. При использовании защищенного канала (например, TLS-соединения) их применение является необязательным.