

INPAS EMV Module

Для разработчиков

Содержание

1. Аннотация	3
2. Особенности модуля	4
3. Состав модуля	5
4. Принципы построения	7
5. Проектирование платежного решения на основе EMV модуля «Vx EMV»	9

1. Аннотация

В документе рассматриваются основы построения и использования, комплект поставки программной библиотеки, предназначенной для организации обслуживания EMV электронных карт. Приводится схема проектирования платежного решения с использованием EMV библиотеки «**Vx EMV 1.0**».

EMV библиотека является высоко специализированным продуктом с узкой областью применения, поэтому настоящий документ рассчитан на специалистов, имеющих определенную подготовку, как в части программирования на языке C/C++, так и в области обслуживания EMV карт со знания EMV спецификации, и имеющих представление о проектировании платежных приложений. Вследствие этого и с целью сокращения описания, в документе не приводятся ссылки на руководящие источники, не поясняются термины, сокращения, понятия и процессы EMV.

2. Особенности модуля

1. Соответствие спецификации EMV 4.1.
2. Независимость от операционной системы. Программные компоненты можно интегрировать на различные платформы: Windows, Verix.
3. Программные компоненты модуля не зависят от типа периферийных устройств, установленных в терминале.
4. Независимость от типа обслуживающего терминала (POS, SC, ATM, информационный киоск, домашний терминал, и т.д.)
5. Независимость от способа хранения EMV параметров и EMV конфигураций в терминале.
6. Независимость от типа авторизационного хоста.
7. Независимость модуля от пользовательского интерфейса с держателем карты и продавцом (кассиром).
8. Возможность применения решения для специфических задач, таких как, например, обслуживание loyalty решений на EMV картах. Возможность построения решений, не полностью удовлетворяющих EMV спецификации - реализация так называемых «non - EMV» операций. Возможность изменения стандартной, определяемой EMV спецификацией, последовательности обслуживания чиповых карт.
9. Простота интеграции модуля в прикладное приложение. Архитектура модуля предусматривает несколько вариантов интеграции.
10. Возможность применимости модуля без изменения кода для расширения поддержки платёжных систем.
11. Использование эффективных программных алгоритмов без перераспределения памяти. Интуитивно ясный интерфейс и система функций обратного вызова.
12. Гибкая система управления и параметризации.

3. Состав модуля

В комплект поставки входят:

- программные компоненты, интегрируемые с основным приложением прикладного уровня, в виде динамических библиотек, а для платформы Verix еще и в виде библиотек для статической линковки;
- набор заголовочных файлов на языке C;
- документация;
- пример каркаса модуля сопряжения («обвеса») библиотеки с прикладной программой на языке программирования C.

Состав комплекта поставки приведен в таблице 1.

Таблица 1. Состав комплекта поставки EMV модуля "Vx EMV 1.0"

Каталог	Файлы	Описание
.\Docs	Inpas EMV Module.pdf	Настоящее руководство
.\Include\Base	Cripto.h	Заголовочные файлы низовой логики EMV библиотеки. Данные файлы не предназначены для изучения и документирования в них не поддерживается.
	EMVAss.h	
	EMVCore.h	
	EmvDef.h	
	Helperpr.h	
	Iccr.h	
	Macro.h	
	ParseTlv.h	
	TagSto.	
.\Include ¹	EmvKernel.h	Основной заголовочный файл библиотеки. Описывает интерфейс EMV модуля - машины состояний
	EmvKernelDef.h	Заголовочный файл, описывающий типы, используемые EMV библиотекой
.\Output\VerixV\Shared\Release	Emv.lib	Файл библиотеки для компоновки основного приложения, использующего EMV в виде динамической библиотеки
	Emv.o	Файлы динамической библиотеки под платформу Verix 5XX
	Emv.p7s	

¹ Файлы EmvKernel.h и EmvKernelDef.h хорошо документированы - программисты-разработчики должны руководствоваться комментариями в этих файлах. Не рекомендуется ориентироваться на комментарии в заголовочных файлах низовой логики.

.\Output\VerixV\Static\Release	Emv.a	Объектный файл для статической компоновки EMV библиотеки под платформу Verix 5XX
.\Output\VerixV_ARM11\Shared\Release	Emv.lib	Файл библиотеки для компоновки основного приложения, использующего EMV в виде динамической библиотеки
	Emv.o	Файл динамической библиотеки под платформу Verix с процессором ARM11
.\Output\VerixV_ARM11\Static\Release	Emv.a	Объектный файл для статической компоновки EMV библиотеки под платформу Verix с процессором ARM11
.\Output\Windows\Shared\Release ²	EmvKrnل.dll	Динамическая библиотека под платформу Microsoft Windows
	EmvKrnل.lib	Файл библиотеки для компоновки приложения
.\Simples\Design	EmvProvider.c	Пример каркаса сопряжения EMV библиотеки

² EMV модуль для статической компиляции под платформу Microsoft Windows не поставляется.

4. Принципы построения

Архитектура модуля включает в себя функции низовой EMV логики, хранилище данных и машину состояний (FSM, конечный автомат - КА³). FSM управляет процессом обслуживания EMV карты, выполняя этапы «**Transaction flow**» в соответствии с EMV спецификацией. Функции низовой логики не доступны прикладному уровню программного обеспечения (ПО), и не рассматриваются в настоящем руководстве. Взаимодействие, доступ и управление функциональными возможностями EMV библиотеки осуществляется только через интерфейс FSM. Под «**EMV ядром**» или, просто, «ядром» по тексту документа понимается совокупность функциональных возможностей FSM, хранилища данных и низовой логики.

FSM обеспечивает:

- запуск процесса обслуживания EMV карты;
- выполнение функций низовой EMV логики, как атомарных элементов, каждый из которых соответствует тому или иному логическому действию или этапу выполнения EMV транзакции;
- последовательность выполнения атомарных функций низовой логики в соответствии со спецификацией EMV;
- управление выбором приложения на карте - приложение, которое не может быть выбрано исключается;
- уведомление прикладного уровня о выполняемом этапе транзакции;
- отправку запросов на получение параметров и данных транзакции и их уточнение⁴;
- данные в запросах группируются по смыслу;
- формирование запросов на получение параметров и данных в те моменты прохождения транзакции, с одной стороны, когда потребность в них возникает, с другой, так, чтобы прикладной уровень мог, как можно раньше повлиять на ход выполнения транзакции;
- завершение транзакции генерацией запроса отказной криптограммы (AAC) в случае возникновения алгоритмической ошибки, вследствие неправильной параметризации, или в случае, прерывания транзакции прикладным уровнем⁵;
- повторное выполнение инициализации EMV транзакции (Initiate Application Processing – Get Processing Options), если PDOL карты содержит параметры транзакции, которые были изменены прикладным уровнем после чтения данных приложения;
- доставку данных от прикладного уровня функциям низовой логики;
- в случае необходимости прерывание (остановку, паузу) исполнения алгоритма FSM с сохранением состояния конечного автомата;
- защиту критических управляющих данных и регистров FSM от несанкционированной модификации их прикладным уровнем на этапах транзакции, где такие изменения могут повлечь возникновение исключительных ситуаций или могут повлечь «повреждение» FSM;
- контроль корректности переданных данных и параметров транзакции;
- трансляцию кодов ошибок прикладного уровня при завершении работы (при выходе) из FSM⁶;

³ В настоящем описании «машина состояний» и «конечный автомат» (КА) используются как синонимы, для обозначения и сокращения используется аббревиатура - FSM – finite state machine.

⁴ Например, после того, как будут прочитаны данные приложения и будет, соответственно, определен РАИ- карты, библиотека сформирует запрос, по которому программа может уточнить сумму авторизации, реализуя программу «лояльности» или политику скидок-наценок. Программа может также уточнить и другие параметры транзакции и сохранить их в хранилище.

⁵ Данный функционал действует только до этапа EMV транзакции «**Terminal Action Analyse**».

⁶ Ошибки прикладного уровня, возникшие при вводе любых типов ПИН, не могут быть транслированы в силу ограничений интерфейса в текущей версии библиотеки.

- формирование достаточного количества признаков для правильной реализации алгоритмов «fallback»⁷;
- механизм управления состояниями, за счет этого обеспечивается возможность реализации не - EMV Transaction flow;
- защиту от вызова функций.

Взаимодействие FSM и прикладного уровня производится посредством функций обратного вызова, которые должны быть реализованы в коде основного приложения. Классическая концепция и потерн проектирования подписки на события в архитектуре EMV библиотеки не заложен.

Совокупность реализаций функций обратного вызова в документе называется кодом или модулем сопряжения («обвесом») EMV библиотеки с прикладной частью программы. FSM имеет интуитивно понятный интерфейс и систему функций обратного вызова, с учетом требований к специалистам, изложенным в аннотации.

EMV ядро работает в статическом пространстве памяти - это означает, что модуль использует статические и стековые (автоматические) определения буферов, массивов и других переменных, исключая конкуренцию за системные ресурсы, минимизируя риски возникновения исключительных ситуаций и потери времени на выделение и инициализацию памяти.

Хранилище данных EMV ядра предназначено для хранения данных в формате Ber-TLV, определенных в приложении C, Book 3, EMV 4.2. Для контроля корректности данных в хранилище встроена база знаний тэгов, сопоставляющая идентификаторам тэгов длины данных и правила форматирования. Предусмотрена возможность расширения базы знания, что позволяет использовать EMV хранилище для не специфицированных в EMV тэгов⁸. Интерфейс хранилища содержит так же функции для пакетной записи и чтения данных, что может оказаться полезным при формировании полей протоколов взаимодействия с хостовыми сервисами.

FSM не формирует признак «reversal» для транзакций отправленных на авторизацию в online и завершенных генерацией отказной криптограммы AAC. В FSM не предусмотрено сообщение «Call your bank». FSM не обеспечивает формирование признака «**Advice Required**» для скрипт-процессинга при разблокировке приложения на карте, разблокировке и смене ПИН. В EMV ядре не предусмотрено логирование процессов.

⁷ Однако при проектировании решения по fallback могут возникнуть сложности - рекомендуем следовать решению из примера каркаса приложения, входящего в комплект поставки EMV модуля.

⁸ В силу ограниченности зарезервированной под хранилище памяти, злоупотреблять этой возможностью не стоит: может оказаться так, что при добавлении очередного тэга в базу знаний будет отказано в доступе.

5. Проектирование платежного решения на основе EMV модуля «Vx EMV»

В настоящем разделе рассматривается типовая последовательность разработки модуля сопряжения («обвеса») EMV библиотеки.

1. Реализовать функции для работы с карт-ридером: инициализация устройства, обнаружение присутствия карты в ридере, инициация обмена (открытие сессии), закрытие сессии, закрытие устройства.
2. Реализовать функцию выполнения команды APDU со следующей сигнатурой.

```
/**
 * Косвенно вызываемая функция обмена командами APDU с приложением.
 * Подлежит обязательной реализации в коде управляющего приложения.
 * @return в случае успеха возвращает значение 0 или код ошибки.
 * @param dataOut [in] - указатель на входные данные, включая заголовок APDU-команды (CLA, INS, PI, P2).
 * @param dataOutLen [in] - длина буфера dataOut в байтах.
 * @param dataIn [out] - указатель на буфер выходных данных команды.
 * @param dataInLen [in/out] - длина буфера - входная величина; длина данных, размещенных в dataIn - выходная
 величина.
 */
typedef long (* EmvCallbackExchangeApdu) (EmvApduIsoCase caseNo, const unsigned char * dataOut,
                                         unsigned short dataOutLen, unsigned short * sw,
                                         unsigned char * dataIn, unsigned short bufSize,
                                         unsigned short * dataInLen);
```

3. Реализация пустой функции управления ходом транзакции со следующей сигнатурой.

```
/**
 * Косвенная функция, которая вызывается ядром для уведомления о прохождении того или иного этапа
 транзакции и запроса на переключение в следующее состояние. Функция может быть использована для изменения
 заложённой в ядре последовательности этапов выполнения (состояний FSM) транзакции.
 * Однако, при переходе в новое состояние ядро вызовет функцию EmvSetState и запрошенное состояние может
 быть "отвергнуто" и выполнение транзакции прервется.
 * Ядро устанавливает в exchangeParameters->OldState код пройденного состояния, а в exchangeParameters-
 >NewState - состояние, которое планируется установить.
 * Если функция возвращает 0, то ядро новое состояние считывает из exchangeParameters->NewState. Если функция
 возвращает значение отличное от нуля, выполнение транзакции завершается, и код ошибки транслируется на
 выходе EmvRun.
 * @return 0 - если прикладной уровень согласен с переходом или не анализирует его, или код ошибки, не равный
 нулю, в противном случае.
 * @param exchangeParameters [in/out] - указатель на структуру параметров обмена.
 * @param session [in/out] - указатель на структуру параметров EMV сессии.
 */
typedef long (* EmvCallbackChallengeRule) (EmvStateExchangeParameters * exchangeParameters,
                                         EmvSessionParameters * session);
```

4. Реализовать функцию получения параметров терминала со следующей сигнатурой.

```
/**
 * Объявление типа для функции получения параметров и данных терминала/
 * @return в случае успеха функция должна вернуть 0 любое другое значение воспринимается как код ошибки по
 которому выполнение транзакции будет завершено, а код транслирован при выходе из EmvRun.
 * @param classParam [in] - идентификатор класса команды или сообщения.
 * @param session [in/out] - указатель на структуру параметров EMV сессии.
 * @param tags [in] - список запрашиваемых идентификаторов тэгов. В настоящее время не используется,
 зарезервирован для будущего применения (RFU).
 * @param countTags [in] - число элементов в tags, параметр зарезервирован для будущего применения (RFU).
 */
typedef EmvTypeCbKReturnValue (*EmvCallbackGetTerminalParameters) (EmvKrnEngineClass classParam,
                           EmvSessionParameters * session,
                           const EmvTypeTagId * tags, unsigned short countTags);
```

5. Реализовать функцию получения списка поддерживаемого терминалом EMV приложений со следующей сигнатурой

```
/**
 * Объявление типа для функции получения списка EMV приложений, поддерживаемых терминалом.
 * @return в случае успеха функция должна вернуть 0 любое другое значение воспринимается как код ошибки по
 * которому выполнение транзакции будет завершено, а код транслирован при выходе из EmvRun.
 * @param termApplItemsArray [out] - указатель, который необходимо проинициализировать списком приложений,
 * каждый элемент списка описывается структурой EmvApplicationItem.
 * @param termApplItemsCount [out] - количество элементов в списке termApplItemsArray.
 * @param terminalLanguages [out] - список поддерживаемых терминалом языков в формате "ruen". Не более
 * четырех языков. Неиспользуемые позиции массива terminalLanguages необходимо занулить.
 */
typedef EmvTypeCbkJReturnValue (* EmvCallbackGetTerminalApplicationsList) (count EmvApplicationItem
** termApplItemsArray, EmvTypeLongValue * termApplItemsCount, char terminalLanguages [8]);
```

6. Реализовать функцию получения параметров EMV приложения со следующей сигнатурой

```
/**
 * Объявление типа для функции получения параметров и данных EMV приложения, выбранного на карте.
 * @return в случае успеха функция должна вернуть 0 любое другое значение воспринимается как код ошибки по
 * которому выполнение транзакции будет завершено, а код транслирован при выходе из EmvRun.
 * @param classParam [in] - идентификатор класса команды или сообщения.
 * @param session [in/out] - указатель на структуру параметров EMV сессии.
 * @param tags [in] - список запрашиваемых идентификаторов тэгов. В настоящее время не используется,
 * зарезервирован для будущего применения (RFU).
 * @param countTags [in] - число элементов в tags, параметр зарезервирован для будущего применения (RFU).
 */
typedef EmvTypeCbkJReturnValue (*EmvCallbackGetApplicationParameters) (EmvKrnEngineClass classParam,
EmvSessionParameters * session,
const EmvTypeTagId * tags, unsigned short countTags);
```

7. Реализовать функцию получения параметров транзакции со следующей сигнатурой

```
/**
 * Объявление типа для функции получения параметров и данных транзакции (сумма авторизации, валюта, тип
 * счета, тип операции и т. д.).
 * @return в случае успеха функция должна вернуть 0 любое другое значение воспринимается как код ошибки по
 * которому выполнение транзакции будет завершено, а код транслирован при выходе из EmvRun.
 * @param classParam [in] - идентификатор класса команды или сообщения.
 * @param session [in/out] - указатель на структуру параметров EMV сессии.
 * @param tags [in] - список запрашиваемых идентификаторов тэгов. В настоящее время не используется,
 * зарезервирован для будущего применения (RFU).
 * @param countTags [in] - число элементов в tags, параметр зарезервирован для будущего применения (RFU).
 */
typedef EmvTypeCbkJReturnValue (*EmvCallbackGetTransactionParameters) (EmvKrnEngineClass classParam,
EmvSessionParameters * session,
const EmvTypeTagId * tags, unsigned short countTags);
```

8. Реализовать функцию получения публичного ключа со следующей сигнатурой

```
/**
 * Проверка номера сертификата в списке отозванных (Revocation List).
 * @return 0 - сертификат нормальный, 1 - сертификат отозван.
 * @param CertificateId - Идентификатор сертификата.
 */
typedef EmvTypeCbkJReturnValue (*EmvCallbackCheckRevocationList) (const EmvKeyRevocationListEntry *
capkListEntry);
```

9. Реализовать функции ввода offline открытого и шифрованного ПИН и ввода online шифрованного ПИН.

10. Реализовать функцию проверки карты в стоп листе, если терминал не поддерживает работу со стоп листами, необходимо реализовать пустую функцию, возвращающую '0'.

```
/**
 * Проверка номера сертификата в списке отозванных (Revocation List).
 * @return 0 - сертификат нормальный, 1 - сертификат отозван.
```

```
* @param CertificateId - Идентификатор сертификата.
*/
typedef EmvTypeCbKReturnValue (*EmvCallbackCheckRevocationList) (const EmvKeyRevocationListEntry *
capkListEntry);
```

11. Реализовать функцию проверки публичного ключа в списке отозванных сертификатов, если данный функционал терминалом не поддерживается, необходимо реализовать пустую функцию, возвращающую '0'.

```
/**
 * Проверка вхождения карты в стоп-лист.
 * @return возвращает 0, если ключ не отзывался или данная проверка не реализована терминалом, или -1, если
был отозван.
 * @return 0 - карты нет в стоп-листе, 1 - карта входит в стоп-лист.
 * @param PAN Номер карты.
 * @param PANSequenceNumber раздел номера карты.
 */
typedef EmvTypeCbKReturnValue (* EmvCallbackCheckHostList) (const char * applicationPAN, unsigned char
panSequenceNumber);
```

12. Реализовать функцию реализующую опПпе-процессинг

```
/**
 * Объявление типа для функции для выполнения online процессинга.
 * @return в случае успеха функция должна вернуть 0 любое другое значение воспринимается как код ошибки по
которому выполнение транзакции будет завершено, а код транслирован при выходе из EmvRun.
 * @param issuerARC [out] – ARC.
 * @param hostData [out] - данные, пришедшие от хоста.
 * @param hostDataLen [out] - длина данных.
 * @param acquirerTransDecision [out] - решение эквайера о результатах прохождения транзакции.
 */
typedef EmvTypeCbKReturnValue (* EmvCallbackOnlineProcissing) (char issuerARC[2], count unsigned char **
hostData, unsigned short * hostDataLen, EmvCoreResults * acquirerTransDecision);
```

13. Если планируется вывод диалога подтверждения ввода суммы операции⁹, динамическое изменение параметров транзакции (как правило, в зависимости от PAN карты), реализация программ лояльности, учета скидок и наценок и или с какой-либо другой целью¹⁰, необходимо реализовать так же общую функцию получения данных EMV сессии, которая получает дополнительные сообщения, т.е. обрабатывает подуровни машины состояний.
14. Реализовать функцию инициализации вектора функций обратного вызова.
15. Реализовать процедуру инициализации EMV библиотеки. Необходимо помнить, что объект EMV-ядра создается как синглтон (singleton).
16. Реализовать общую логику запуска EMV сессии и обработку кодов ошибок.
17. Реализовать процесс перевода транзакции в режим fallback.
18. Некоторые операции, поддерживаемые терминалом, могут проходить не в соответствии с EMV. Примерами таких операций могут служить, где карта выступает не как платежный инструмент, а больше как идентификационная карточка. Другими примерами являются операции разблокировки приложения на карте, разблокировки

⁹ EMV спецификация обязывает предусматривать возможность отказа держателя карты от проведения транзакции. Отказ от проведения транзакции для online терминалов можно совместить, например, вводом ПИН, если он является обязательным. Так же можно совместить диалог отказа от выполнения транзакции с вводом ПИН в конфигурациях, в которых не поддерживается «обход ввода ПИН» (bypass PIN). В других конфигурациях необходимо выводить отдельный диалог, который лучше совместить с показом ордера операции, содержащим сумму операции.

¹⁰ Еще одним примером для реализации этой функции является дополнительное логирование процесса обслуживания карты.

ПИН, смены ПИН. Для поддержки этих операций необходимо управлять состояниями FSM, с этой целью используется функция обратного вызова¹¹.

19. Реализовать режим «голосовой авторизации», если планируется его поддержка.

Для уточнения деталей проектирования программного модуля сопряжения EMV библиотеки необходимо ознакомиться с прилагаемым каркасом EmvProvider.c.

¹¹ Сравнительный анализ различных решений по обслуживанию EVM карт, различных поколений программного обеспечения показал, что библиотеки, где EVM функционал представлен набором функций, выигрывают перед централизованным управлением FSM, только в конечных решениях. Для универсальных и гибких архитектур, в лучшем случае, такой «центр управления» приходится воспроизводить на прикладном уровне, в худшем - создается стохастический код, обвешанный командами управления компиляцией или состоящий из кучи конфигураций сборки проектов.