Потенциал

Библиотечка программиста

Примеры использования хостовых команд HSM PayShield 9000.

Москва

22 января 2019 г.

Аннотация

Данное руководство адресовано программистам, разработчикам прикладных систем, начинающим или планирующим использовать Host Security Module (HSM) серии PayShield 9000.

Данные модули разработаны и производятся компанией Thales e-Security специально для применения платежных системах в первую очередь в международных платежных системах (МПС) Visa, MasterCard, American Express, JCB, China Union Pay, и т.п.

Предполагается что читатель знаком с предметной областью, основами криптографии, и предварительно ознакомится с документацией по PayShield 9000.

Разработчикам, уже использовавшим PayShield 9000 или HSM предыдущих серий RG6000, RG7000, 8000, могут быть интересны новые алгоритмы, ключевые схемы и вопросы миграции, а также отдельные примеры и подробности параллельной - конвейерной обработки.

Чтение данного руководства ни в коей мере не может заменить использование комплекта оригинальной документации PayShield 9000 для той конкретной версии/разновидности модуля, который вы собираетесь применять. Более того можно определенно утверждать, что данное руководство содержит ошибки и разночтения, вызванные как пробелами в познаниях автора, так и банальным несовпадением использованных в примерах версий и лицензий. Большая часть примеров приведенных примеров не чувствительна к версии функциональности HSM. В примерах, где такая чувствительность присутствует, имеется и упоминание (если автор не забыл). Примеры и рекомендации в основном относятся в версии базовой функциональности не старше 3.4c.

Руководство не претендует на полноту охвата темы. Если вы интересующая вас тема не охвачена, попытайтесь связаться с автором. E-mail anton@ptnl.moscow может быть в следующей редакции…

Компания разработчик HSM - Thales e-Security равно как ее партнеры и поставщики не несут никакой ответственности за корректность и достоверность приведенных здесь примеров, результатов и рекомендаций.

Оглавление

Литература

[1] 1270A541-037 OBKM and CEPS Commands Addendum LIC004 v3.4
[2] 1270A542-037 Host Programmer v3.4
[3] 1270A543-037 Installation v3.4
[4] 1270A544-037 Console V3.4
[5] 1270A545-037 Security Operations v3.4
[6] 1270A546-037 Host Command Reference v3.4.
[7] 1270A547-037 Australian Standards LIC003 V3.4
[8] 1270A548-037 Card & Mobile Issuance LIC011,016,018,023 v3.4
[9] 1270A589-037 Host Command Addendum LIC017 v3.4
[10] 1270A590-037 Host Command Addendum LIC020 v3.4
[11] 1270A591-037 Host Command Addendum LIC031 v3.4
[12] 1270A592-037 Host Command Addendum LIC014 v3.4
[13] 1270A593-037 General Information v3.4
[14] 1270A614-037 Host Command Addendum LIC034 v3.4
[15] 1270A645-037 payShield Manager User Guide v3.4
[16] 1270A646-037 Host Command Examples v3.4
[17] 1270A647-037 Applications Using the payShield 9000 v3.4
[18] PWPR0526-001 payShield 9000 in 3-D Secure
[19] 1270A622-001.8 Card Issuance Firmware 1119-0913 - CIP
[20] 1270A623-001.8 Card Issuance Firmware 1119-0913 - GPS
[21] payShield 9000 and AES
[22] PWPR0537-001 payShield 9000 RSA Booster License
[23] PWPR0543-001 payShield 9000 General Data Protection

Использование интерфейса TCP/IP. Установка соединения, передача сообщений.

Взаимодействие прикладной системы с HSM происходит посредством передачи модулю командного сообщения и получения ответа.

Модуль распознает и по возможности исполняет присланную вами команду, затем возвращает результат исполнения. В ответ на подавляющее большинство команд HSM возвращает одно ответное сообщение. Исключением служат команды, вызывающие печать данных на принтер. Такие команды в ответ порождают два сообщения. Одно по завершении криптографических расчетов в команде, второе по завершении печати данных на принтере. Стоит напомнить, что печать процесс медленный - механический и второй ответ последует спустя приличное время после первого.

Для взаимодействия с payShield9000 разумно использовать интерфейс Ethernet, протокол TCP либо UDP. (Взаимодействие с IBM Mainframe, последовательные протоколы, а также организацию SSL/TLS соединения рассматривать не будем).

При использовании TCP максимальное количество открываемых соединений определяется настройками HSM (но не более 64 соединений на каждый из двух Ethernet портов HSM). Для каждого соединения прием/передачу сообщений можно вести независимо друг от друга. При использовании TCP последовательность передаваемых в каждом соединении команд и ответов будет поддерживаться средствами протокола. При использовании UDP вы сами должны следить за последовательностью отправляемых команд и получаемых ответов. При этом обратите внимание:

- HSM любой модели обеспечивает асинхронное исполнение множества команд в отдельных цепочках. Количество таких цепочек может варьироваться. Его можно положить равным 8, но только приблизительно. Исполнение разных команд занимает разное время и зависит от модели HSM и совершенно нормальна ситуация, когда команда посланная позже, вернет ответ раньше.

IP адреса и номера портов для открытия соединения (well-known port) заданы настройками HSM. См. [4]. По умолчанию этот адрес 1500 и несколько портов следующих за ним см. раздел [Использование нескольких LMK] и аналогичный раздел в [6].

Передавая командное сообщение в HSM, следует отсылать его одним блоком, посредством операции Push. К тому же так будет быстрее. В свою очередь HSM всегда отправляет ответ целым блоком посредством Push.

Отсылаемый блок должен содержать в начале два байта длины командного сообщения затем само командное сообщение. Ответы приходят также предваренные длиной.

Не следует обрамлять командное сообщение знаками «начало - конец текста» как это делалось для последовательных интерфейсов. Забудьте.

Обратите внимание! Длина передаваемого вами блока должна **ТОЧНО** соответствовать суммарной длине командного сообщения! В отличие от предыдущих моделей payShield9000 отрицательно реагирует на такое несовпадение, в том числе, когда в хвост сообщения просто добавлены лишние, невинные с виду байты.

Формат командного сообщения и ответа HSM

Вы найдете подробное описание формата командного сообщения в [2]. Здесь рассмотрим:

Командное сообщение

Заголовок	Код команды	Операнды	«прицеп» к сообщению

Ответ

Заголовок	Код команды +1	Код ошибки	Возвращаемые знач.	«прицеп» к сообщению

Заголовок (Header) – несколько символов в начале командного сообщения. Длина определяется в настройках хостового интерфейса HSM. Заводская установка 4 знака. HSM не использует поле заголовка, не анализирует его, и возвращает неизменным в ответном сообщении. Поле заголовка удобно использовать для отслеживания последовательности команд и ответов. Если вы поместите в это поле счетчик, и будете нумеровать каждую отсылаемую команду, то всегда будете знать, на какую команду пришел ответ. Весьма актуально для UDP.

Код команды – два алфавитно-цифровых символа. Каждой команде соответствует свой код. Описания основного набора команд вы найдете в [6]. Полный список команд исполняемых именно вашим HSM вы получите по консольной команде GETCMDS. В ответном сообщении код команды будет увеличен на 1, например для команды A2 вернется A3, для команды NC вернется ND и т.п. Если код команды вернулся не изменённым, это означает что HSM команду вообще не распознал. Скорее всего, вы неправильно сформировали командное сообщение. Например, длина заголовка не такая как в настройках модуля.

Код ошибки – две шестнадцатеричные цифры. 00 команда успешно выполнилась. Если не ноль смотрите список кодов ошибок в конце [6] или для специфических команд в описании команды. Несколько кодов, которые стоит помнить: 17 – операция не авторизована; 67 - команда не лицензирована; 68 - команда в явном виде запрещена к исполнению офицерами безопасности; 69 – вы пытаетесь использовать пинблок запрещенного формата.

Операнды и возвращаемые значения – зависят от того что это за команда. У каждой команды они свои. Количество, формат, длина произвольно варьируются от команды к команде. Имеется общее ограничение на длину не более размера входного буфера (32 Кбайт для сегодняшних версий)[1]. Но внимательно прочитайте первую главу [2]. Входной буфер выделяется не на каждую команду, а на каждое соединение. Если вы пошлете несколько команд подряд, не дожидаясь обработки, все они должны уместиться в один буфер. HSM не занимается управлением потоком. **Следить, чтобы буфер не переполнился задача программиста!**

«прицеп» к сообщению – совершенно не обязательное поле. Не более 32 знаков. Если присутствует, то должно предваряться символом разделителем с кодом X’19’. Можете помещать в это поле информацию по своему усмотрению, например для идентификации сообщений.

Примеры самых простых команд.

Для рассмотрения примеров мы будем использовать стандартную утилиту Thales e-Secutity HsmScript.exe (Part Number: 1270A418) используя ее в среде MS Windows.

NC – Диагностика HSM.

```
#--
# Test Host Interface
# Perform HSM Diagnostics.
#--
12:32:04 |   to hsm : HEADNC
12:32:04 | from hsm : HEADND00B66AA000000000001362-0902
```

Запускает в модуле процесс самодиагностики. Входных параметров нет. На выходе 00 – ошибок нет; B66AA00000000000 – 6 шестнадцатеричных знаков проверочного значения LMK. Остальные цифры заменены нолями согласно современным требованиям безопасности. 1362-0902 – версия программного обеспечения HSM. Процесс самодиагностики занимает ощутимое время и может вызывать эффекты во внешней среде. **Не следует исполнять эту команду без необходимости!** И уж ни в коем случае не следует исполнять ее просто с цель проверить доступен ли модуль. Гораздо разумнее использовать для этого команду:

NO – Вернуть состояние

```
11:37:26 |   to hsm : 1234NO00
11:37:26 | from hsm : 1234NP0031641119-091319100
```

[1] Узнать размер буфера - см. команду NO – вернуть состояние.

Заодно можно узнать: размер входного буфера: 3 - 32к, протокол связи 1 – TCP, максимальное количество сокетов 64, 1119-0913 – версия программного обеспечения HSM. Остальные знаки ответа пока ничего не значат.

Можно проверить модуль на совместимость с требованиями PCI HSM. Например, настройки данного модуля – 00 не соответствуют требованиям PCI.

```
17:07:20 |   to hsm : 1234NO01
17:07:20 | from hsm : 1234NP000
```

CW – генерация CVV

PAN – 4333322221111222, Exp Date – 9307, Serv code – 141

```
16:54:40 |   to hsm : 1234CW0A61E674E88C6A7EEABC38C2B2BB492F4333322221111222;9307141
16:54:40 | from hsm : 1234CX00382
```

Здесь ключ PVK представлен в виде 2х половинок в нулевой ключевой схеме. Использовать такую схему не следует. Она устарела еще в 2000 году. Сейчас нет никакого оправдания ее применению.

Вот та же самая команда и тот же ключ в вариантной схеме U. И тот же самый результат. Уже лучше.

```
19:35:48 |   to hsm : 1234CWU9B4934384B19946B040CD702B4D58145433333222211111222;9307141
19:35:48 | from hsm : 1234CX00382
```

Однако и вариантная схема ныне не является доверенной. Доверенными схемами ныне считаются только схемы с ключевыми блоками: Thales Key Block (TKB) для локального хранения и обработки, а также TR-31 и GISKE для обмена. Вот тот же ключ в формате TKB.

```
17:14:00 |   to hsm :
1234CWS10096C0TC01N000000F37A8646AD198E61480795CC0AE67AA16AE35301408B4A92C77E96271AFBC
074C749662E8389DB4333322221111222;9307141
17:14:01 | from hsm : 1234CX00382
```

CY – проверка CVV

Очень похоже на генерацию. И те же примеры.

```
16:54:40 |   to hsm :
1234CY0A61E674E88C6A7EEABC38C2B2BB492F66446666655554444111;9206120
16:54:40 | from hsm : 1234CZ00

16:03:57 |   to hsm :
1234CYU9B4934384B19946B040CD702B4D581456644666655554444111;9206120
16:03:57 | from hsm : 1234CZ00

17:14:01 |   to hsm :
1234CYS10096C0TC01N000000F37A8646AD198E61480795CC0AE67AA16AE35301408B4A92C77E96271AFBC
074C749662E8389DB6644666655554444111;9206120
17:14:01 | from hsm : 1234CZ00
```

JA – Генерация случайного ПИН

Команда позволяет сгенерировать случайное число с заданным количеством десятичных знаков (от 4 до 12) и представить результат в виде шифрованном посредством LMK и в привязке к номеру счета (PAN) таким образом, чтобы дешифровка в отсутствии PAN не имела смысла.

Если в настройках безопасности задана проверка слабых пинов (Enable weak PIN checking: YES), команда не будет генерировать значения заданные в глобальной таблице слабых пинов или указанные непосредственно в тексте команды.

То есть команда никогда не генерирует слабых пинов, и не порождает код возврата 86: PIN exists in either global or local Excluded PIN Table.

При использовании TDES LMK (и старого вариантного и современного блокового) максимальная длина и представление шифрованного PIN определяется настройками безопасности (CS). В параметрах максимальная длина чистого пина. Длина шифрованного представления PIN на 1 знак больше.

Предлагается на выбор два алгоритма формирования шифрованного представления: A – предложенный в свое время VISA и B – алгоритм предложенный Thales. Формат предложенный VISA

содержит только десятичные цифры. Алгоритм Thales дает шестнадцатеричный результат и соответственно большую дисперсию и крипто стойкость.

```
Online-AUTH>qs

PIN length: 04
Encrypted PIN length: 05
Echo: OFF
.
.
.
PIN encryption algorithm: A
```

При указанных настройках для TDES LMK и PAN 123456789012, получаются результаты такого вида.

```
19:29:39 |   to hsm : headJA123456789012 ← генерация pin длиной по умолчанию
19:29:39 | from hsm : headJB0018304

19:29:39 |   to hsm : headJA12345678901204 ← генерация pin длиной 4 знака
19:29:39 | from hsm : headJB0030058
```

Попытка генерации пин большей длины дает ошибку с кодом 24.

```
19:29:39 |   to hsm : headJA12345678901208
19:29:39 | from hsm : headJB24
```

Совершено иная картина при использовании AES LMK. Вне зависимости от настроек можно генерировать pin любой длины, а порождаемая величина – pin блок специфического формата с префиксом J и длиной 32 шестнадцатеричных знака.

```
19:28:30 |   to hsm : headJA123456789012
19:28:30 | from hsm : headJB00J2190E0BD1631A765A0F9354A71598D49

19:28:30 |   to hsm : headJA12345678901204
19:28:30 | from hsm : headJB00JF4ED44620162B517FA9820034A0D3A18

19:28:30 |   to hsm : headJA12345678901205
19:28:30 | from hsm : headJB00J5DD8091EB90F66C1EF4224AB2031A8D5

19:28:30 |   to hsm : headJA12345678901206
19:28:30 | from hsm : headJB00J4292F3E5267FFB036B029A4B67F39332

19:28:30 |   to hsm : headJA12345678901207
19:28:30 | from hsm : headJB00J398FE7C86C7BA0FF7A47F5A0FC263D20

19:28:30 |   to hsm : headJA12345678901208
19:28:31 | from hsm : headJB00J535F4393F9993B122F8B3FB7D1E340DF

19:28:31 |   to hsm : headJA12345678901209
19:28:31 | from hsm : headJB00J7807B30AAAC95D77686BA4503F144B6D

19:28:31 |   to hsm : headJA12345678901210
19:28:31 | from hsm : headJB00J8A8DFFD707BD3BC2708EA755034BA6A7

19:28:31 |   to hsm : headJA12345678901211
19:28:31 | from hsm : headJB00J2E8403985BC933773DB4660F727C69C4

19:28:31 |   to hsm : headJA12345678901212
19:28:31 | from hsm : headJB00J67F14DC38F4DAD96BAE94582CDFBE40D
```

Управление ключами

A0 – генерация ключа

Генерация TMK (или TPK), вариантный LMK TDES 2-й длины

```
16:54:40 |   to hsm : 1234A00002U
16:54:40 | from hsm : 1234A100U71ECE1AEF235716073E55DF0B1552C70D93684
```

В ответе криптограмма результата в формате U последние 6 знаков KCV.

Команда умеет генерировать любые допустимые ключи, выдавать их в виде криптограмм под LMK, но она же умеет экспортировать свеже-созданный ключ под ZMK или TMK.

Вот пример генерации и экспорта ключа ZPK в схемах X – ANSI X9.17, U – вариантная схема, R – ANSI TR-31. На входе естественно требуется криптограмма ZMK под LMK. В ответе две криптограммы одного и того же чистого значения. По LMK и под ZMK соответственно. Последний пример требует наличия лицензии LIC006

```
20:21:08 |   to hsm : 1234A01001U;0UD96B47B35CB6BB921B8528A69674E14FX%01
20:21:08 | from hsm :
1234A100UAE9FDE6E0C195C9D94268EE8A94B7C56XB8619020D2FE23762169359D242E26B21B0888

20:21:08 |   to hsm : 1234A01001U;0UD96B47B35CB6BB921B8528A69674E14FU%01
20:21:08 | from hsm :
1234A100UB8C9918528A79B876A69575308D8F66CUD436E724672157ACAA68D1DC63464A0FB6E9AF

21:06:37 |   to hsm : 1234A01001U;0UD96B47B35CB6BB921B8528A69674E14FR%01&P0E01N00
21:06:37 | from hsm :
1234A100U45C6036E96DCC10C7F7F35803CF4EB4FRA0072P0TE01N000081EDA2D4A4E0BD9F28D974577528
49CAD8EE19C3703E8F26E234AA6BFFA327
```

Или аналогично генерация TPK и экспорт под TMK X – ANSI X9.17. Типовая задача при обслуживании ATM.

```
20:21:08 |   to hsm : 1234A01002U;1U1A770CB340D1ABAEBB7823272FBE21D7X%01
20:21:08 | from hsm :
1234A100U2743F25B0D7F64CD6A0B2FC4DAE3DFF4X7E00A0257892D84F637E396AFD59A257FE61A8
```

Генерация TPK и экспорт под TMK, схема R – ANSI TR-31. Этот пример требует наличия лицензии LIC006.

```
21:06:37 |   to hsm : 1234A01002U;1U1A770CB340D1ABAEBB7823272FBE21D7R%01&P0E01N00
21:06:37 | from hsm :
1234A100UE02108A1E348BF8C3FEEE173680E2F1FRA0072P0TE01N000096F4D45BA4344CB2AE34C2168AEC
A436F694BF737442B4FBDC8CD4F4FF392B
```

А вот со схемой GISKE так ловко сделать не получается.

Использование формата GISKE

Кроме общеизвестных блочных форматов обмена ключами TKB и TR-31 существует их предшественник формат GISKE. Он до сих пор широко применяется для передачи ключей в POS-терминалы Verifine. Модули payShield9000 поддерживают формат GISKE исключительно для этой цели. Формату GISKE соответствует ключевая схема **V**. Ее можно применить только к терминальным TDES ключам при экспорте, какого либо терминального ключа под другим терминальным ключом. Получаемы криптограммы будут понимать PIN-клавиатуры Verifone.

Экспорт TPK под TMK и Исходные ключи в вариантной схеме.

```
20:07:05 |   to hsm :
1234A8002;1U0AE93BB1D0CA18201F9395FED569A60BU8326C7D9F06EDE5DF289BF71487C4564V
20:07:05 | from hsm :
1234A900VA0120P0T856F4D03FAB0977805DEAE59DB3EDD42D29E058B110CF819000000000000000000000
00000000000000000000000000032E3D7AC47AD9DA3A87845
```

Экспорт нового TMK под старым TMK с явным указанием назначения и версии ключа.

```
20:20:16 |   to hsm :
____A8002;1U0AE93BB1D0CA18201F9395FED569A60BU8326C7D9F06EDE5DF289BF71487C4564V&K001A
20:20:16 | from hsm :
____A900VA0120K0T722CDE1D93FCF67AADA082D2A9C9AA14784C45A261D46F330000000000000000000000
00000000000000000000000000009F2B721824883CF7A87845
```

Экспорт TAK под TMK - 10 ISO 9797-1 MAC Algorithm 1 – 112 bits (предполагается по умолчанию).

```
20:07:05 |   to hsm :
____A8003;1U0AE93BB1D0CA18201F9395FED569A60BU1AF2E196A2C2A3C07E5C6DA05644DC5AV
20:07:05 | from hsm :
____A900VA012010TB3FEF8B328863B76718AE79EDA3B8263F3FEFAFCBCCA1B1F000000000000000000000
00000000000000000000000000003048CD6450D49B66FB5E2E
```

То же с явным указанием назначения и версии ключа - 30 ISO 9797-1 MAC Algorithm 3 – 112 bits

```
20:20:16 |   to hsm :
____A8003;1U0AE93BB1D0CA18201F9395FED569A60BU1AF2E196A2C2A3C07E5C6DA05644DC5AV&3001A
20:20:16 | from hsm :
____A900VA012030TCF6647F87FD17226C089C52BAEE2DBBD6C387FA05F7A75A3000000000000000000000
0000000000000000000000000005D2DDF74022F1754FB5E2E
```

Для сравнения аналогичный экспорт в формате TR-31

```
21:03:29 |   to hsm :
1234A8002;1U0AE93BB1D0CA18201F9395FED569A60BU8326C7D9F06EDE5DF289BF71487C4564R%01&P0E0
1N00
21:03:29 | from hsm :
1234A900RA0072P0TE01N00009C2F847F9A95314FFA3F345E09B5569994633F5D951B1D2CAC282440A8784
5
```

Экспорт ключей в разных форматах из-под KeyBlock LMK.

После выполнения миграции с вариантного LMK к блочной технологии часто возникает вопрос, - можно ли экспортировать ключи в старых ключевых схемах.

Действительно требуемая согласно PCI PIN Security v2.0 к июню 2019 г. миграция к KeyBlock LMK и блочным технологиям представления ключей при хранении, не затрагивает обмен ключами между хостами и терминалами. Согласно данному документу переход на блочные форматы представления ключей при межхостовом обмене намечен на 2021 г., а при обмене с банкоматами, POS-терминалами на 2023 г. До той поры обмен можно мести в «старых» форматах.

Вот несколько примеров на данную тему:

Будем использовать блоковые AES LMK. Сгенерируем мастер ключ для шифрования других ключей.

```
!txnowait '1234A00FFFS%00#51T2B00S00'
# ZMK-TMK
S1009651TB00S00003F7FA0520BABA0C6A99E88E82D48040FA4826CA256996A532A1E0059A90B472E477C5
926420CA4C7
KCV 1662AE
```

Это будет ZMK – Key Usage 51, алгоритм TDES 2-й длины - T2.

Сгенерируем TPK - Key Usage 71, тоже TDES 2-й длины - T2. Важно разрешить экспорт данного ключа в форматах на усмотрение офицеров безопасности - S

```
!txnowait '1234A00FFFS%00#71T2B00S00'

# TPK
S1009671TB00S0000977BED0F6211179B4E3DED8F5DA415194D4F850124145DA17FCC3DB3B909EA1CB903C
AC2BB870665 KCV 19DD19
```

Настройки параметров безопасности HSM (CS) позволяют экспортировать ключи в данных форматах:

```
….
Enable X9.17 for export? [Y/N]: Y
….
Key export and import in trusted format only? [Y/N] :N
….
```

В формате ANSII x9.17 разрешено и получается:

```
#19:21:22 |   to hsm :
1234A8FFFS1009651TB00S00003F7FA0520BABA0C6A99E88E82D48040FA4826CA256996A532A1E0059A90B
472E477C5926420CA4C7S1009671TB00S0000977BED0F6211179B4E3DED8F5DA415194D4F850124145DA17
FCC3DB3B909EA1CB903CAC2BB870665X%00
#19:21:22 | from hsm : 1234A900X807562C9B1A278FB052358759E54A1B319DD19
```

В «старом» вариантном формате тоже:

```
#19:29:58 |   to hsm :
1234A8FFFS1009651TB00S00003F7FA0520BABA0C6A99E88E82D48040FA4826CA256996A532A1E0059A90B
472E477C5926420CA4C7S1009671TB00S0000977BED0F6211179B4E3DED8F5DA415194D4F850124145DA17
FCC3DB3B909EA1CB903CAC2BB870665U%00
#19:29:58 | from hsm : 1234A900UCEDE11304D5CEA54358892325722O7DE19DD19
```

И конечно традиционный формат GISKE для терминалов Verifone:

```
#19:29:58 |   to hsm :
1234A8FFFS1009651TB00S00003F7FA0520BABA0C6A99E88E82D48040FA4826CA256996A532A1E0059A90B
472E477C5926420CA4C7S1009671TB00S0000977BED0F6211179B4E3DED8F5DA415194D4F850124145DA17
FCC3DB3B909EA1CB903CAC2BB870665V%00
#19:29:58 | from hsm :
1234A900VA0120P0T4807F726B395946ABF2E102FB2747E2BC8BCB460A5F8884362248F6CC2B7C36AEFF50
C720444F9DBC4F9F2F6B854C4E5A739EED2EDBE100319DD19
```

Печать открытых компонент ключей и пинов на присоединенном принтере.

Подключение принтера к payShield9000 не представляет трудностей. Для тестовых целей пригодны любые принтеры кроме т.н. WIN принтеров. Интерфейс USB, параллельный или последовательный через соответствующий переходник. При настройке важно указать параметр Timeout не менее времени реальной печати листа. Параметр задается в миллисекундах. Скажем для принтера производительностью 12 листов в мин. не менее – 5000.

Собственно печать должна предваряться загрузкой шаблона форматов для печати. В памяти payShield9000 имеется отдельная область, в которую следует поместить шаблон описания форматирования теста. Шаблон может содержать до 299 констант и символов управления форматированием. Полный список символов управления и примеры шаблонов вы найдете в [2].

Шаблоны сохраняются в памяти до выключения питания или перезапуска модуля. Память шаблона печати общая, для всех LMK и всех параллельно открытых сессий. В этой связи не пытайтесь использовать HSM для печати нескольких одновременно запускаемых заданий. Ничего путного из этого не выйдет.

A2 - Генерация и печать компонент ключа

Загрузим предварительно формат для печати конверта. Затем сгенерируем и напечатаем компоненту TMK. В программе на хосте мы получим ее шифрованное представление.

```
13:19:44 |   to hsm : 0001PA>L>L>L>L>L>024Clear ^0 component ^1>L>L>010Terminal ID:
^4>037Bank ID: ^2>L>L>028Order ID: ^3>L>L>L>015^P>F
13:19:44 | from hsm : 0001PB00

13:19:44 |   to hsm : 0001A20021UTMK;1;001122334455;987;01234567
13:19:44 | from hsm : 0001A300UAEAA5D9E10C60574D309C20107113BDA
```

А на принтере:

```
                    Clear TMK component 1

        Terminal ID: 01234567      Bank ID: 001122334455

                         Order ID: 987


              C77A FE0B BF1F 98CE 8A73 9246 43D6 9D46
```

OA – Печать ходатайства о присвоении PIN (Print a PIN Solicitation Mailer)

```
20:34:40 |   to hsm : 0000PA>L>L^0^1^T>L^2^R
20:34:40 | from hsm : 0000PB00

20:46:13 |   to hsm : 1000OAC123456789012CARD:;******;Ref No:
20:46:13 | from hsm : 1000OB00566311103423

20:46:13 |   to hsm : 2000RC123456789012566311103423
20:46:13 | from hsm : 2000RD00
```

А на принтере напечатается:

```
CARD:******789012
Ref No:5327 7286 1235
```

HSM в режиме Solicitation batch size: 0001

Припишем к номеру ходатайства собственно PIN – 9999

```
21:25:25 |   to hsm : 2000QC5327728612359999;
21:25:25 | from hsm : 2000QD00345678901255988F
```

Нам вернули правые 10 цифр PAN 3456789012 и шифрованное значение PIN под LMK – 55988.

Использование нескольких LMK

Вы можете загрузить в HSM и одновременно использовать несколько LMK. Сколько и каких определяется лицензией модуля. В настоящее время не более 99 шт. в разновидностях: старая вариантная 112 бит TDES, Thales Key Block TDES 168 бит, Thales Key Block AES до 256 бит.

```
Online-AUTH>vr        ← Версия модуля

Base release: …
Revision:     ….
.
.
Serial Number:
.
.
.
Crypto:              3DES,AES,RSA
LMKs Enabled:        1 Variant, 1 Keyblock LMK
HSM9-LIC001 Base Software
HSM9-LIC002 RSA
HSM9-LIC006 X9 TR-31
HSM9-LIC007 AES Algorithm
HSM9-LIC008 Data Protection
.
.
```

В данном примере присутствуют все три разновидности ключей владельца. Для выполнения операций можно использовать1 Variant, 1 Keyblock LMK. Имеется лицензия HSM9-LIC007 AES Algorithm. Для Keyblock LMK может использоваться алгоритм AES.

```
Online-AUTH>vt        ← Список загруженных LMK

LMK table:

ID Auth     Scheme   Algorithm  Status Check  Comments
00 Yes      KeyBlock AES-256    Test   B66AA0 Test AES
01 Yes      Variant  3DES(2key) Test   268604 Test Variant

Key change storage table:

ID          Scheme   Algorithm  Status Check  Comments
00          KeyBlock 3DES(3key) Test   165126 Old Des Keyblock Test
```

KeyBlock 3DES(3key) загружен в память мены ключей под ID 00. Его можно использовать для перешифрования в AES-256.

При такой конфигурации доступных LMK HSM слушает три порта:

```
Online-AUTH>netstat

Available Ethernet Ports:
Management Port: 192.168.003.003
Host Interface 1: 172.016.101.002

Host 1 Active Internet connections
```

```
Proto Recv-Q Send-Q  Local Address          Foreign Address        State
tcp        0      0  192.168.101.2.1500     192.168.137.65.25860   ESTABLISHED
tcp        0      0  192.168.101.2.1500     192.168.137.65.25859   ESTABLISHED
tcp        0      0  192.168.101.2.1500     192.168.137.65.25858   ESTABLISHED
tcp        0      0  192.168.101.2.1500     192.168.137.65.25857   ESTABLISHED
tcp        0      0  192.168.101.2.1500     192.168.137.65.25746   ESTABLISHED
tcp        0      0  192.168.101.2.1500     192.168.137.65.25745   ESTABLISHED
tcp        0      0  192.168.101.2.1500     192.168.137.65.25744   ESTABLISHED
tcp        0      0  192.168.101.2.1500     192.168.137.65.25743   ESTABLISHED
tcp        0      0  192.168.101.2.1502     192.168.71.5.47155     ESTABLISHED
tcp        0      0  192.168.101.2.1500     192.168.89.2.1121      ESTABLISHED
tcp        0      0  192.168.101.2.1500     192.168.89.2.58776     ESTABLISHED
tcp        0      0  192.168.101.2.1502     *.*                    LISTEN
tcp        0      0  192.168.101.2.1501     *.*                    LISTEN
tcp        0      0  192.168.101.2.1500     *.*                    LISTEN
```

Порт 1501 адресует ваш запрос - к LMK с ID 00, порт 1502 – к LMK с ID 01 и так далее. Порт 1500 адресует к LMK по умолчанию (в данном случае ID 00). Используя обращения к разным портам можно работать с разными разновидностями и значениями LMK.

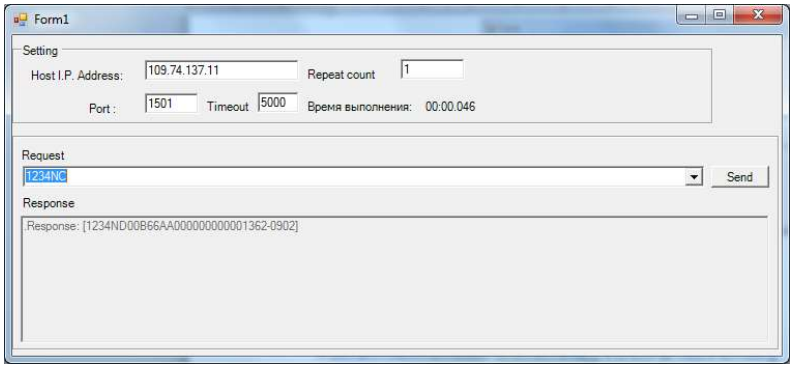


Рис. 1 обращение к порту 1501 в ответе KCV LMK ID 00 = B66AA0

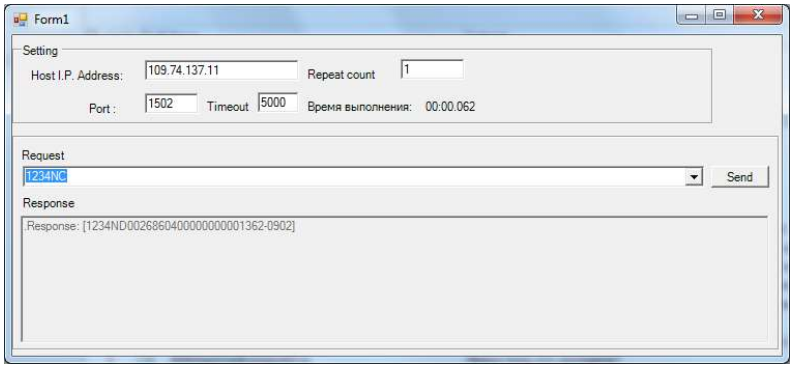


Рис. 2 обращение к порту 1502 в ответе KCV LMK ID 01 = 268604

Подобный механизм удобен, когда к разным LMK (в одном HSM) обращаются разные приложения возможно с разных IP адресов.

Когда требуется из одного приложения обращаться к нескольким LMK заведомо находящихся в одном модуле, удобно использовать механизм модификатора номера LMK в тексте команды.

```
19:28:25 |   to hsm : HEADNC       ← адресовано к LMK по умолчанию
19:28:25 | from hsm : HEADND00B66AA000000000001362-0902

19:28:25 |   to hsm : HEADNC%00    ← адресовано к LMK ID 00
19:28:25 | from hsm : HEADND00B66AA000000000001362-0902

19:28:25 |   to hsm : HEADNC%01    ← адресовано к LMK ID 01
19:28:26 | from hsm : HEADND00268604000000000001362-0902
```

Использование блоковых TDES и AES LMK

Блоковое представление LMK появилось еще в HSM8000. Активное применение AES LMK началось в payShield9000 с выходом версии 2.0 в 2012 году.

Первый вопрос, который возникает при миграции к AES LMK – перешифрование, существующих данных, в первую очередь ключей. Никаких особых сложностей этот процесс не сулит.

Операция выполняется командой BW. Также как при трансляции вариантных и блочных TDES криптограмм.

```
16:26:05 |   to hsm :
1234BWFFFS00072C0TN00N0000E6C60DE02F4C12B5DCF24F19BBFE5C03CB11F61D660B955280CBDF27;FFF
16:26:05 | from hsm :
1234BX00S10096C0TN00N00000B4C869BAE4F239D1998E2C6077B5860F2C1329D22B63BECEA5CAE2F17CA6
00E60E65B50C214A673
```

Для блочных криптограмм тип ключа и его длину указывать нет нужды. В этих полях FFF. Обратите внимание: длина полученной под AES-256 криптограммы несколько больше исходной.

Использование памяти и ключей, заранее загруженных в память.

В payShield 9000 довольно много памяти, которую модуль использует в своих внутренних процессах и которую можно использовать в разнообразных прикладных целях:

Пользовательская память

Так называемая пользовательская память (User storage). Это 98 304 байта незащищенной памяти предназначенной для хранения **не секретных** данных. Эта память разбита на 4096 блока длиной 8, 16 или 24 байта согласно тому, как указано в параметре User storage key length [S/D/T](SINGLE): команды CS. Блоки по длине соответствуют криптограмме DES ключей одинарной, удвоенной или утроенной длины. Первоначально память и предполагалась использовать для хранения криптограмм ключей. Соответственно длина данных загружаемых в эту память должна быть кратна 8, 16, или 24 байтам. Добавлять до целого блока положено заполнителем X'FF'. Таким образом чем большими блоками вы будете оперировать тем больший объем памяти сможете использовать.

Имеются команды Загрузки данных в память – LA и Чтения данных из памяти – LE. Указывается номер начального блока (X'000' – X'FFF' и количество блоков X'01' – X'20'. Сумма количества блоков и номера начального блока не должны выходить за 4096. Читать и писать за границу пользовательской памяти нельзя.

Данные предназначенные для загрузки в пользовательскую память должны быть представлены в распакованной шестнадцатеричной форме. Команда загрузки упакует данные, и в памяти они будут находиться в двоичном представлении. Команда чтения возвращает данные в распакованном шестнадцатеричном представлении. Для модуля с настройкой (User storage key length DOUBLE), чтобы положить в память HSM текст «Non secret value placed in HSM storage» необходимо его представить в виде трех блоков по 32 шестнадцатеричные цифры.

```
4E6F6E20736563726574207661 6C7565
20706C6163656420696E2048534D2073
746F72616765FFFFFFFFFFFFFFFFFFFF
```

LA – загрузка данных в пользовательскую память

```
13:12:44 |   to hsm :
1234LAK004034E6F6E20736563726574207661 6C756520706C6163656420696E2048534D2073746F726167
65FFFFFFFFFFFFFFFFFFFF
13:12:44 | from hsm : 1234LB00
```

LE – Чтение данных из пользовательской памяти

```
13:12:44 |   to hsm : 1234LEK00403
13:12:44 | from hsm :
1234LF004E6F6E20736563726574207661 6C756520706C6163656420696E2048534D2073746F72616765FF
FFFFFFFFFFFFFFFFFF
```

Применение пользовательской памяти

У пользовательской памяти не слишком много полезных применений. Первоначально пользовательская память предназначалась для хранения криптограмм DES ключей под LMK. Загрузив эти криптограммы в память можно затем ссылаться на них по номеру ячейки.

Например загрузим ключ CVK - U9B4934384B19946B040CD702B4D58145 в ячейку номер 1. Префикс типа криптограммы отделяется, а значение 32 шестнадцатеричные цифра загружаются.

```
19:21:50 |   to hsm : 1234LAK00101 9B4934384B19946B040CD702B4D58145
19:21:50 | from hsm : 1234LB00
```

В команде вместо значения криптограммы используется ссылка на нее в виде UKnnn, где U префикс вариантной ключевой схемы для ключей 2-й длины K признак ссылки на ячейку с номером nnn.

```
19:21:50 |   to hsm : 1234CY UK001 6644666655554444111;9206120%01
19:21:51 | from hsm : 1234CZ00
```

Так можно использовать только криптограммы DES, TDES ключей под вариантными LMK. Использовать криптограммы ключей представленных ключевым блоком, к сожалению, в существующих версиях невозможно. Обратите внимание при размещении в памяти, данные переводятся в двоичный вид, но не дешифруются.

Размер блоков памяти должен быть больше или равен длине используемых ключей. То есть если в вашей системе есть хотя бы один ключ утроенной длины, то блоки памяти должны быть утроенной длины, а ключи имеющие меньшую длину должны грузиться в ячейки будучи дополнены до полного блока X’FF’.

Использование ключей, предварительно загруженных в память существенно укорачивает командное сообщение. Это может быть актуально для конвейерной обработки посредством команды NK - Цепочка команд. Скорость обработки в одиночно исполняемых командах почти не увеличивается.

Можно использовать пользовательскую память для передачи значений между серверами. При этом важно помнить, что память не обеспечивает конфиденциальности и содержимое памяти может быть изменено любым процессом. Следует помнить, что выключение питания, перезагрузка HSM уничтожает содержимое пользовательской памяти.

Аналогично криптограммами вариантных ключей можно загружать в пользовательскую память PIN блоки и аналогично ключам ссылаться на них по номеру Knnn.

Сходное использование памяти предусмотрено для таблиц децимализации в алгоритме генерации и проверки IBM. На предварительно загруженные таблицы можно ссылаться по номеру Knnn в командах, таких как BK, DE, EE, GO.

Пользовательская память используется для загрузки таблицы метода Diebold и последующей генерации и верификации с ее использованием. См. описание команд GA, CE,CG, EG, LC в [6]. Таблица Diebold может быть введена и с консоли командой R. Поскольку таблица имеет ощутимую длину, номер блока ее начала не может превышать 5FF. См. [4].

Пользовательская память также используется payShield9000 при пакетной обработке ходатайств о присвоении PIN (PIN Solicitation). См. описание команд QA, QC в [6]. Исполнение этого процесса полностью затирает пользовательскую память.

Память хранения секретных ключей для алгоритма RSA

Пользовательская память мало пригодна для хранения RSA ключей. Они могут иметь разную и довольно большую дину. Поэтому в HSM предусмотрено 21 отдельная ячейка для хранения криптограмм секретных RSA ключей под LMK. Ячейки нумеруются с 00 до 20.

Размер самих ячеек никак не специфицируется. Предполагается, что он достаточен для ключей поддерживаемых данной моделью. (До 2048 в настоящее время). Не пытайтесь использовать на

старых моделях HSM ключи, сгенерированные на новых моделях. С высокой степенью вероятности, не получится. В обратную сторону – полная преемственность.

Загрузка приватных RSA ключей шифрованных LMK производится командой EK. Затем в командах EW, GI, GK можно ссылаться на такие ключи по номеру. (Указание номера 99 сигнализирует о присутствии криптограммы ключа в теле самой команды).

Нахождение RSA ключа в памяти заметно ускоряет процесс подписания.

В противоположность пользовательской памяти память RSA ключей защищена от вмешательства. При загрузке в память криптограммы ключей дешифруются и в памяти находятся в чистом виде, пригодные к использованию. Ключи могут загружаться как из старого вариантного, так и из современного блочного формата. Контроль корректности ключа производится в момент загрузки. И хотя в HSM нет команды выгрузки секретного ключа из памяти, тем не менее, потенциально возможно нарушение регламента безопасности ключа.

Обратите на это особое внимание! Память RSA ключей общая и для подписания данных ключом уже загруженным в память не требуется никакой особенной санкции. Модуль с загруженными в память секретными ключами сам становится чувствительным устройством! HSM подпишет любые данные поданные в соответствующей команде, вне зависимости от источника команды. Если вы загружаете секретные ключи в память HSM, вся сеть хостового интерфейса должна находится в зоне с ограниченным доступом!

Если вы используете несколько LMK, и загружаете секретные ключи в память HSM вполне возможна ситуация когда объединяя несколько систем с разными LMK, вы нарушите регламент безопасности сами того не заметив. Помните также, что загруженный в память ключ не исчезнет, пока модуль не будет выключен или перезагружен. Команды стирания ключа в памяти модуля нет!

Ниже примеры использования команд генерации, загрузки и подписания данных секретным RSA ключом.

Генерация RSA пары длина ключа 512 бит. Секретный ключ в блочной схеме под AES LMK.

```
12:19:19 |   to hsm :___EI0051201#0000
12:19:20 | from hsm :
45.4A.30.30.30.47.02.40.B9.C4.D0.78.EC.03.E3.2A   EJ000G.@...x...*
93.B7.26.DC.DE.B5.05.61.0C.33.9B.72.85.DD.F6.58   ..&....a.3.r...X
BE.C8.21.8C.8E.5D.4D.AB.74.5F.1F.1F.80.98.8F.3C   ..!..]M.t_.....<
F2.45.C0.00.DD.D4.D1.6A.61.F9.07.DB.07.99.0A.C3   .E.....ja.......
D2.94.7F.A2.E7.C3.45.6B.02.03.01.00.01.46.46.46   ......Ek.....FFF
46.53.31.30.32.30.38.30.33.52.53.30.30.4E.30.30   FS1020803RS00N00
30.30.D0.A8.FD.80.AE.2A.7F.6B.9D.6E.57.91.B7.52   00.....*.k.nW..R
42.45.28.FD.4D.3A.FD.88.70.41.08.18.2B.42.42.26   BE(.M:..pA..+BB&
DB.16.DD.C9.1C.20.90.C2.8B.55.13.83.1F.B2.62.98   .........U....b.
49.BE.4B.4D.A0.28.C2.2F.9F.14.C9.29.08.68.99.3F   I.KM.(./...).h.?
E8.B3.59.8E.67.29.E2.5D.B0.5C.39.FD.B6.DA.5D.CE   ..Y.g).].\9...].
A9.55.CA.23.43.E1.01.FF.41.06.FB.72.EE.1E.1B.5C   .U.#C...A..r...\
22.C7.FD.65.76.BD.37.49.42.06.F5.E3.2C.3E.FA.63   "..ev.7IB...,>.c
61.85.4C.F3.DB.E2.6A.07.5D.C1.F7.E4.EE.F4.28.96   a.L...j.]......(.
58.0A.BC.A9.82.F2.8E.94.27.59.1B.73.BE.4C.07.C1   X.......'Y.s.L..
8B.90.26.4F.4D.14.89.10.82.9F.C6.8E.C1.4F.57.C4   ..&OM........OW.
D8.19.9C.DB.60.77.FB.17.A4.69.FE.80.64.89.EB.0B   ....`w...i..d...
C8.73.39.39.41.34.45.32.36.39.30.45.39.45.34.38   .s99A4E2690E9E48
36.33                                             63
# Response (Hex):
454a303030470240b9c4d078ec03e32a93b726dcdeb505610c339b7285ddf658bec8218c8e5d4dab
745f1f1f80988f3cf245c000ddd4d16a61f907db07990ac3d2947fa2e7c3456b0203010001464646
4653313032303830333525330304e30303030d0a8fd80ae2a7f6b9d6e5791b752424528fd4d3afd88
704108182b424226db16ddc91c2090c28b5513831fb2629849be4b4da028c22f9f14c9290868993f
e8b3598e6729e25db05c39fdb6da5dcea955ca2343e101ff4106fb72ee1e1b5c22c7fd6576bd3749
4206f5e32c3efa6361854cf3dbe26a075dc1f7e4eef42896580abca982f28e9427591b73be4c07c1
8b90264f4d148910829fc68ec14f57c4d8199cdb6077fb17a469fe806489eb0bc873393941344532
36393045394534383633
```

Загрузка ключа в память. Ячейка 11.

```
12:19:20 |   to hsm :
45.4B.31.31.46.46.46.46.53.31.30.32.30.38.30.33   EK11FFFFS1020803
```

```
52.53.30.30.4E.30.30.30.30.D0.A8.FD.80.AE.2A.7F   RS00N0000.....*.
6B.9D.6E.57.91.B7.52.42.45.28.FD.4D.3A.FD.88.70   k.nW..RBE(.M:..p
41.08.18.2B.42.42.26.DB.16.DD.C9.1C.20.90.C2.8B   A..+BB&.........
55.13.83.1F.B2.62.98.49.BE.4B.4D.A0.28.C2.2F.9F   U....b.I.KM.(./.
14.C9.29.08.68.99.3F.E8.B3.59.8E.67.29.E2.5D.B0   ..).h.?..Y.g).].
5C.39.FD.B6.DA.5D.CE.A9.55.CA.23.43.E1.01.FF.41   \9...]..U.#C...A
06.FB.72.EE.1E.1B.5C.22.C7.FD.65.76.BD.37.49.42   ..r...\"..ev.7IB
06.F5.E3.2C.3E.FA.63.61.85.4C.F3.DB.E2.6A.07.5D   ...,>.ca.L...j.]
C1.F7.E4.EE.F4.28.96.58.0A.BC.A9.82.F2.8E.94.27   .....(.X.......'
59.1B.73.BE.4C.07.C1.8B.90.26.4F.4D.14.89.10.82   Y.s.L....&OM....
9F.C6.8E.C1.4F.57.C4.D8.19.9C.DB.60.77.FB.17.A4   ....OW.....`w...
69.FE.80.64.89.EB.0B.C8.73.39.39.41.34.45.32.36   i..d....s99A4E26
39.30.45.39.45.34.38.36.33                        90E9E4863
# Request (hex):
454b313146464646533130323038303335253330304e30303030d0a8fd80ae2a7f6b9d6e5791b75242
4528fd4d3afd88704108182b424226db16ddc91c2090c28b5513831fb2629849be4b4da028c22f9f
14c9290868993fe8b3598e6729e25db05c39fdb6da5dcea955ca2343e101ff4106fb72ee1e1b5c22
c7fd6576bd37494206f5e32c3efa6361854cf3dbe26a075dc1f7e4eef42896580abca982f28e9427
591b73be4c07c18b90264f4d148910829fc68ec14f57c4d8199cdb6077fb17a469fe806489eb0bc8
7339394134453236393045394534383633
12:19:20 | from hsm :___EL00
```

Подписание 64 байт данных ключом из 11-й ячейки. Обратите внимание: никаких ссылок на LMK нет!

```
12:19:20 |   to hsm :
45.57.30.31.30.31.30.31.30.30.36.34.30.31.32.33   EW010101006400123
34.35.36.37.38.39.61.62.63.64.65.66.30.31.32.33   456789abcdef0123
34.35.36.37.38.39.61.62.63.64.65.66.30.31.32.33   456789abcdef0123
34.35.36.37.38.39.61.62.63.64.65.66.30.31.32.33   456789abcdef0123
34.35.36.37.38.39.61.62.63.64.65.66.3B.31.31      456789abcdef;11
# Request (hex):
455730313031303130303634303132333435363738396162636465663031323334353637383961626
36465663031323334353637383961626364656630313233343536373839616263646566 3b3131
12:19:20 | from hsm :
45.58.30.30.30.30.36.34.21.D1.51.2A.C0.35.40.B4   EX000064!.Q*.5@.
56.8F.68.33.4B.76.E7.18.5E.5B.93.C6.FB.1E.C4.7B   V.h3Kv..^[.....{
DF.CE.C6.0D.F4.73.67.63.38.9C.B2.66.90.69.76.B1   .....sgc8..f.iv.
2A.DD.01.E3.A6.48.03.F3.F8.A6.0E.B2.9A.5C.04.58   *....H.......\.X
F8.8B.27.A8.34.39.AC.D0                           ..'.49..
```

Смена PIN. Таблица слабых пинов.

Со стороны МПС имеется настойчивое пожелание предоставить клиенту возможность менять пин код карты и при этом не допускать установки заведомо слабых значений. Какие значения являются слабыми, следует определить самостоятельно.

Задача распадается на три части: определение какие значения пин считать слабыми их проверка и отбрасывание; смена пин для магнитной полосы (online pin); смена пин для чипа (offline pin).

После того как вы решили какие значения пинов «плохие» можно загрузить эти значения в глобальную таблицу слабых пинов и активировать проверку слабых пинов в настройках безопасности.

BM - Загрузка глобальной таблицы слабых пинов.

Таблица действует для всех LMK в модуле и оказывает влияние на операции генерации и смены PIN.

Объявим плохими пины 1234 1111 8888

```
!txnowait '1234BM0304123411118888'
```

Объявим плохими пины длиной 6 знаков 123456 123123

```
!txnowait '1234BM0206123456123123'
```

HSM по своей инициативе более не будет генерировать указанные значения. А при выполнении операций приводящих к смене PIN появление указанных значений в качестве нового PIN HSM вернет ошибку с кодом 86.

Можно подать список слабых пинов непосредственно в качестве операндов команды затрагивающих обработку PIN. Например в качестве операндов команд DU,CU служащих для проверки старого PIN и пересчета offset или PVV соответственно под новый PIN. Аналогично можно поступить с командой FW – расчет PVV под новый PIN.

В нижеследующем примере рассмотрен ряд подобных операций.

В расчетах используем пару ключей:

```
##PVK тип 002
##Key under LMK: U53CD 6E27 7EE9 A9A2 1373 860C 8785 2476
##Key check value: BF074D

##ZPK тип 001
##Key under LMK: U9489 6837 1768 518A E9EF C901 CAF1 F200
##Key check value: 5B89E5
```

Для PAN 123456789012 сгенерируем случайный PIN 4 знака, но только не: 1111 2222 3333 4444 5555 6666 7777 8888 9999 1234 0000

```
!txnowait 'headJA12345678901204*1104111122223333444455556666777788889999123400000'
## 16:46:14 | from hsm : headJB0035664

##List of enabled PIN Block formats:
##01 - ISO 9564-1 & ANSI X9.8 format 0
##02 - Docutel ATM format
##03 - Diebold & IBM ATM format
##04 - PLUS Network format
##05 - ISO 9564-1 format 1
##34 - Standard EMV 1996 format
##35 - MasterCard Pay Now and Pay Later format
##41 - Visa/Amex new PIN only format
##42 - Visa/Amex new & old PIN format
##46 - AS2805.3 Format 8 PIN block
##47 - ISO 9564-1 & ANSI X9.8 format 3
```

Превратим наш случайный пин в пинблок как-бы введенный пользователем.

```
!txnowait 'headJGU948968371768518AE9EFC901CAF1F2000112345678901235664'
##16:48:14 | from hsm : headJH006978B4F7F3531E5B
## PIN Block  6978B4F7F3531E5B
```

Вычислим PVV для этого сочетания PIN – PAN - PVK

```
!txnowait
'headFW001U948968371768518AE9EFC901CAF1F200U53CD6E277EE9A9A21373860C878524766978B4F7F3
531E5B011234567890120'
##16:48:14 | from hsm : headFX000867
## PVV 0867
```

Посмотрим, а какое собственно было значение PIN

```
## Decrypt PIN
!txnowait 'headNG12345678901235664'
##16:48:14 | from hsm : headNH002212F532772861235

## Clear PIN 2212
```

Теперь именно это значение объявим слабым

```
!txnowait
'headFW001U948968371768518AE9EFC901CAF1F200U53CD6E277EE9A9A21373860C878524766978B4F7F3
531E5B011234567890120*0142212'
##16:59:22 | from hsm : headFX86
```

Для исполнения третьей части задачи используются команды KU, KY –Secure messaging для разных версий EMV-стандарта.

Работа с PIN блоками и чистым PIN.

Пусть для PAN 123456789012

Клиент выбрал значение PIN 1234

```
12:18:36 |   to hsm : ____BA1234F123456789012
12:18:36 | from hsm : ____4BB0094415
```

Получилось шифрованное значение PIN под этим LMK - 94415

Преобразуем в PIN блок для отправки …

ZPK UCF091C61654DEBE6BA5C06950A39BC74

PIN блок формата 05

```
12:18:36 |   to hsm : ____JGUCF091C61654DEBE6BA5C06950A39BC740512345678901294415
12:18:36 | from hsm : ____JH009F6E9C2D6BD4FC60
```

Получился PIN блок 9F6E9C2D6BD4FC60

Теперь обратно

```
12:18:36 |   to hsm :
1234JEUCF091C61654DEBE6BA5C06950A39BC749F6E9C2D6BD4FC6005123456789012
12:18:36 | from hsm : ____JF0094415
```

Для проверки дешифруем PIN

```
12:18:36 |   to hsm : ____NG12345678901294415
12:18:36 | from hsm : ____NH001234F532772861235
```

Наши искомые 1234

Обработка чиповых транзакций

В стандартной части системы команд payShield9000 имеется 7 команд для обработки чиповых транзакций.

```
KQ – Проверка ARQC и/или генерация ARPC (EMV 3.1.1)
KS – Проверка DAC и DN (EMV 3.1.1)
KU – Создание безопасных сообщений (EMV 3.1.1)
KY – Создание безопасных сообщений (EMV 4.x)
KW – Проверка ARQC и/или генерация ARPC (EMV 4.x)
K0 – Дешифровка значений зашифрованных счетчиков (EMV 4.x)
K2 – Проверка укороченных криптограмм (MasterCard CAP)
```

Обычно их применение не вызывает особенных сложностей. Рассмотрим здесь только 2 момента.

Соотношение CVN и параметров команд

Часто задают вопрос , -«как зная Cryptogram Version Number – CVN понять какие команды HSM и с какими параметрами употребить». Действительно в описаниях команд Thales HSM в том числе в документации по payShield9000 до версии 2.0 отсутствовали подобные указания. Авторы

документов наивно полагали, что все и так это знают?! Ныне в описаниях команд вся эта информация присутствует! См. [6].

Посмотрим эти соотношения на примере функции проверки криптограмм:

Приложение/CVN	**Команда**	**Параметр Scheme ID**
Visa VIS (CVN 10 или 17)	KQ	‘0’
MasterCard M/Chip (CVN 10 или 11)	KQ	‘1’
American Express AEIPS (CVN 01 или 02)	KQ	‘2’
Visa VIS (CVN 14)	KW	‘0’
Visa VIS (CVN 18 или 22)	KW	‘2’ или ‘3’
Visa VCP (CVN 43)	KW	'4'
Visa VCP QR Code (CVN 44)	KW	'8'
MasterCard M/Chip (CVN 12 или 13) (EMV 2000)	KW	‘0’
MasterCard M/Chip (CVN 14 или 15) (EMV CSK)	KW	‘2’
Mastercard MCBP	KW	'5'
American Express (CVN 05, 07 и MPVV)	KW	'6'
Discover D-PAS (05 или 06)	KW	‘2’
Индонезийская национальная платежная система (Indonesian National Standard Chip Card Specification - NSICCS)	KW	‘2’
Discover HCE	KW	'7
EMV карты с ключами сгенерированнми с опцией 'C' и EMV CSK	KW	'9'
JCB (CVN 01)	KW	'A'
JCB (CVN 02)	KW	'B'
JCB (CVN 04)	KW	'2'
Union Pay (Vers 4.2)	KW	'C'

Не забудьте о различиях в нотации принятых в различных МРС и вперед. (В Visa VIS CVN 10 десятичное, в MasterCard M/Chip CVN 10 – шестнадцатеричное, т.е. x’10’ и x’11’ ‘ это 16 и 17 в десятичном представлении.)

Обратите внимание что начиная с версии 3.3 функциональности HSM нет необходимости в использовании специальных команд (LIC031) для работы с чипами Union Pay.

Отладка проверки криптограмм

Иногда отладка частей системы проверяющих криптограммы чиповх карт вызывает сложности особенно когда она идет параллельно с отладкой самих приложений на чиповой карте и ее персонализацией.

Бывает сложно определить, где ошибка при вычислении криптограммы в чиповом приложении, в параметрах его персонализации или в параметрах вызова команд HSM. В обычных условиях команда возвращает результат проверки 00 – OK, 01 – нет совпадения. Для облегчения такой отладки имеется специальный режим авторизации DIAGNOSTICS. Если команда проверки криптограмм выполняется в доверенном состоянии, и совпадения нет, то помимо результата проверки, команда возвращает значение криптограммы, посчитанное по исходным данным.

Например: для произвольных исходных данных попытаемся проверить бессмысленное значение криптограммы.

```
Online-AUTH>
21:21:16 |   to hsm :
1234KQ00UE28A9F98B48B509A62C3B8D853B89DBA01234567010123100123456789abcdef;01234567
21:21:16 | from hsm : 1234KR01\I.fVL..
# Response (Hex):
4b523031 5c49fb66564c8601
```

Конечно, не совпадает, зато в ответе есть значение криптограммы соответствующе этим исходным данным. Если ее подставить в запрос естественно будет совпадение.

```
21:22:20 |   to hsm :
1234KQ00UE28A9F98B48B509A62C3B8D853B89DBA01234567010123100123456789abcdef;
HEX:49fb66564c8601
21:22:20 | from hsm : 1234KR00
```

После снятия авторизации диагностические данные не появляются.

```
Online-AUTH>c
NOT AUTHORISED
Online>
21:23:33 |   to hsm :
1234KQ00UE28A9F98B48B509A62C3B8D853B89DBA01234567010123100123456789abcdef;01234567
21:23:33 | from hsm : 1234KR01
```

Абсолютно аналогично ведут себя команды KY и K2.

Помните! работа в доверенном состоянии на авторизационном хосте эмитента НЕДОПУСТИМА. Думаю после приведенного примера дальнейшие пояснения «почему?» не требуются!

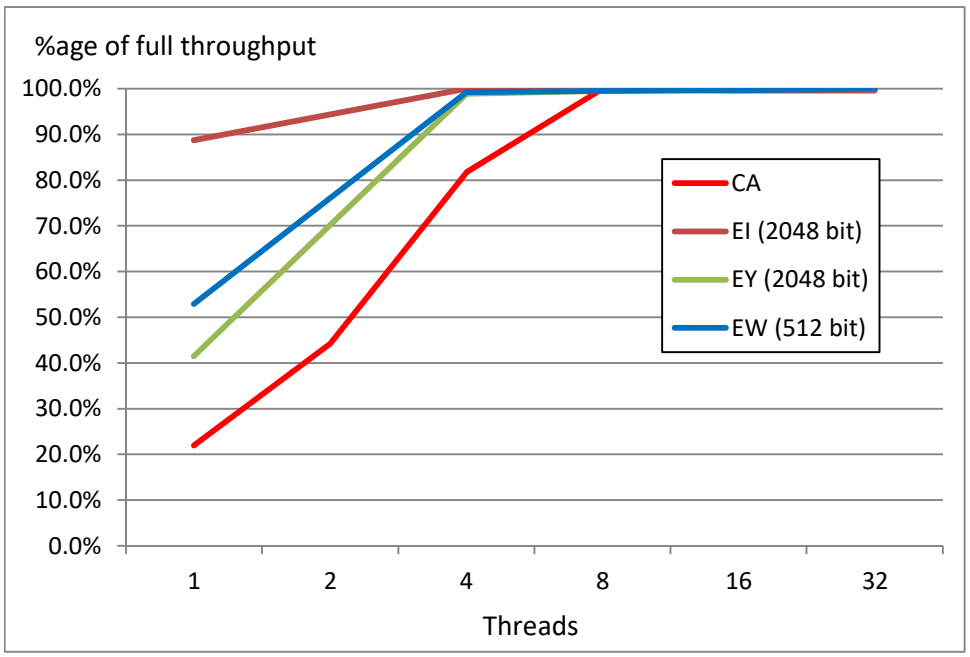
Производительность, параллельная и конвейерная обработка.

По скорости исполнения можно условно разделить все команды на три группы:

- Команды исполняющиеся приблизительно с той же скоростью что и титульная производительность HSM. С этой скоростью выполняются почти все команды связанные с симметричной криптографией. С этой же скоростью выполняются команды проверки ЭЦП вовлекающие RSA алгоритм. Исключение составляют модули с установленной лицензией RSA Booster. См. [22]. Параллельное исполнение таких команд допустимо.
- Команды вызывающие печать. (печать пинов, печать компонентов ключей,…) Расчеты в этих командах выполняются быстро, примерно с той же скоростью что и прочие команды с симметричной криптографией. Но печать процесс механический – медленный. Не пытайтесь исполнять эти команды в параллельных потоках. Результат может вас удивить.
- Команды генерации RSA пар ключей в соответствии со строгими требованиями платежных систем. Эти команды выполняются медленно – 10 пар 1024битовых ключей в секунду, - совсем неплохая производительность. Параллельное исполнение таких команд также вполне допустимо.

Параллельная обработка

Thales e-Security дает следующую картину зависимости пропускной способности модуля от количества цепочек параллельно запушенных команд.



Для операций с использованием RSA алгоритма зависимость линейна, и производительность модуля полностью выбирается четырьмя параллельно запущенными цепочками команд. Для операций с TDES полная производительность выбирается в 8 цепочках одновременно исполняемых команд.

График приведен в процентах от титульной производительности и справедлив для всех моделей с интерфейсом Ethernet.

Таким образом, применение параллельной обработки запросов в нескольких потоках повышает общую пропускную способность системы. И с этой точки зрения весьма эффективно, но естественно время исполнения любого отдельного запроса в потоке больше, нежели если бы он выполнялся один.

NK - Цепочка команд, конвейерная обработка

Команда позволяет объединить в одном запросе несколько произвольных хостовых команд. Команды отсылаются в одном командном сообщении. Ответы также приходят одним блоком. Количество команд в блоке не может быть больше 99, при этом длина каждой отдельной команды не может превышать 9999 байт и общая длина блока запроса не должна превысить размер входного буфера (32кб), а длина ответного сообщения не должна превышать 11 000 байт.

Команда существовала и ранее на HSM8000, однако в payShield9000 команда позволяет не только существенно сократить затраты времени на ввод/вывод, но и исполнять все команды в цепочке одновременно и параллельно. В нижеследующих примерах обратите внимание на время исполнения цепочки из 10 команд генерации ключа, и таких же точно команд по отдельности.

На примере payShield9000 модели Low Speed (20 tps) выглядит весьма наглядно:

```
12:19:19 |   to hsm :
_01_NK0100010A00002U%010010A00002U%010010A00002U%010010A00002U%010010A00002U%010010A00
002U%010010A00002U%010010A00002U%010010A00002U%010010A00002U%01
12:19:19 | from hsm :
_01_NL00100043A100U94CF0C02F364484628EC2ADDE89CC1206C3EFE0043A100U9C6AE98EFD223A0C03F7
7A2DBA54956E99EA610043A100U4F3120A7CE1E6AC2803BB454C5ACA20692C80B0043A100U50946A41603D
F9C0CBC2DA8A7DE3D8E7C9265A0043A100UB3827FC3984FFC296C34C19FF0F54701F400D50043A100U3545
41C6BD18EA712545C0A1D11AA32D6EFDDF0043A100U3CD8B27479072669C8A96BBBBBF0B0DFF69D180043A
100UE0ED93017D01B3BAA85BEC7E83C4C555951CB80043A100UF6EC107BCAA813E5A6E38D7C903676B6D96
8F50043A100UE22DD87BB2A1CB37DD1E062E3FEB9F44738EC8
```

… И так 20 раз подряд ….

```
12:19:21 |   to hsm :
_20_NK0100010A00002U%010010A00002U%010010A00002U%010010A00002U%010010A00002U%010010A00002U%010010A00
002U%010010A00002U%010010A00002U%010010A00002U%010010A00002U%01
12:19:21 | from hsm :
_20_NL00100043A100U2572C957FE54162F0B3AAB8C4A4265E7920A7D0043A100U11764940BC5A53A2A991
7E3A3C886DB9D639210043A100U4DA7740DE5FF4DDD1B3D41FAEEC31246E5E9A20043A100U2B237ECB2A57
CABDA680F0B1E43C102DDFB0620043A100U18159EFC33335D5DE1723B175CD14E09B513F40043A100U43BA
BC8A93E53A7B7DE6D30C60CAB3EF0B613D0043A100U5C2A66929BD79DDEEFE63515A03F74C147B0D60043A
100U92E64D672B4D3E4F26D04A0F4D3F22F0ACD8A90043A100U06B0459475C160921B23CD24315F2BC4930
EE70043A100UD31B9459EE39FA48726615F11FE82B6E9C97E9
```

Заняло около 2 сек.

Теперь тоже самое, но по одной команде.

```
12:32:57 |   to hsm : _01_A00002U%01
12:32:58 | from hsm : _01_A100UDCC23F8BEC8EC5B88292C62665D4059381CB9B
```

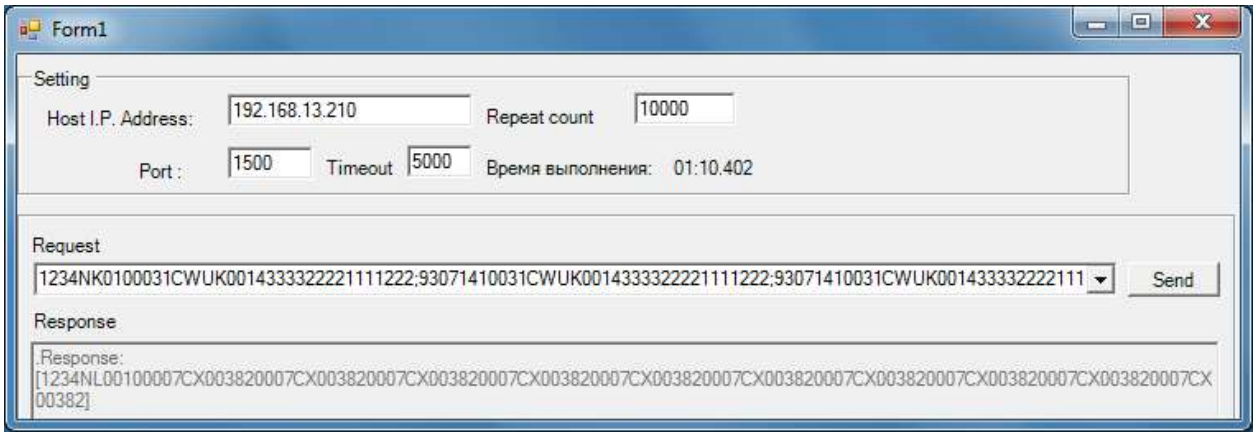
… И так 20 раз подряд ….

```
12:33:00 |   to hsm : _20_A00002U%01
12:33:00 | from hsm : _20_A100U0776958669CA3F2E3D699192C899240903E213
```

Заняло примерно одно и то же время, только в первом случае было сгенерировано 200 ключей, а во втором только 20.

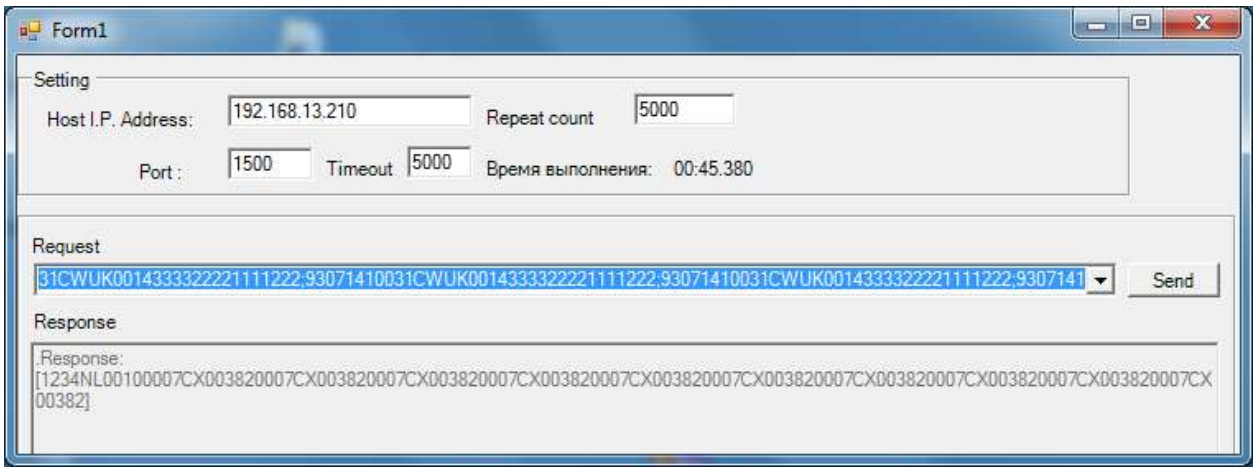
Для проведения более точных и масштабных экспериментов надо использовать утилиту подсчитывающую время исполнения.

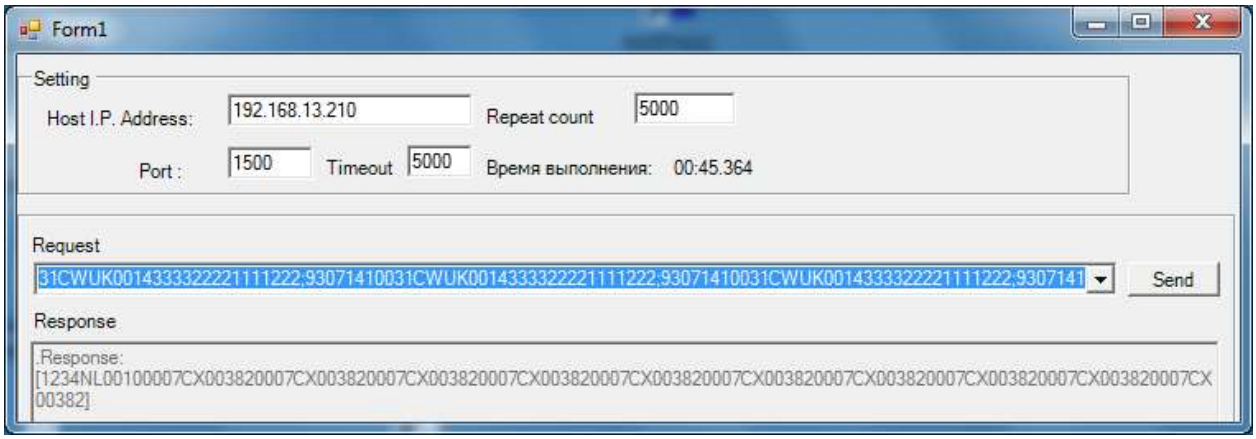
Выполним 10000 раз цепочку из 10 команд генерации CVV. Используя payShield9000 с титульной производительностью (220 tps).



Процесс генерации CVV для 100 000 карт занял чуть больше 70 сек. То есть 1420 команд в сек. Это в несколько раз больше титульной производительности HSM.

Проведем тот же самый эксперимент в два параллельно исполняемых потоков в каждом по 5000 раз цепочку из 10 команд генерации CVV.





То есть те же 100 000 CVV. Только заняло это чуть больше 45 сек. То есть 2205 команд в сек. В 10 раз больше титульной производительности.

Проведя нехитрые эксперименты можно получить табличку реально достижимых скоростей исполнения команд для модулей разной производительности. Например, с цепочкой из 10 команд CW – генерация CVV можно получить следующие значения.

Количество потоков	Титульная/Потоковая производительность (tps)				
	1500	**800**	**220**	**50**	**20**
1	3260	3000	1420	420	174
2	5150	5605	2205	487	201
3	6320	5505	2310	491	202
4	7315	6100	2424	495	203
5	7440	6300			
6	7930	6410			
7	8650	7240			
8	8140	6560			
9	8100	6565			

Таким образом, сочетание параллельной и конвейерной обработки позволяет не просто повысить скорость такой обработки, но и легко превзойти т.н. титульную производительность модуля.

Сходные результаты получаются при использовании любых команд использующих симметричную криптографию.

Разные полезные команды

B2 – Эхо

Возвращает ту двоичную последовательность, которую вы задали в операндах команды.

Зачем может быть нужна такая команда? Например, для проверки канала связи: его прозрачности и быстродействия на пакетах разной длины.

```
20:02:28 |   to hsm : 1234B200101234567890abcdef
20:02:28 | from hsm : 1234B3001234567890abcdef
```

Получение серийного номера HSM

В системе команд payShield9000 есть ряд команд возвращающих информацию собранной статистики: J2, J4 и «состоянии здоровья» модуля: J8,JK. Все эти команды возвращают серийный номер HSM к которому происходит обращение.

```
20:02:28 |   to hsm : 1234J4
20:02:29 | from hsm :
1234J500B4665277385A1311291449311404181607191404181607190012100514A0000000000868A60000
00000112A8000000000258AE000000000003B0000000000009B2000000000003BA000000000011BI000000
000001BM000000000003BU000000003198BW000000001072BY000000000003CC000000000001CW00000000
1285CY000000000800DC000000000001DG000000000330EC000000000014EI000000000062EK0000000000
02EO000000000029EW000000000006G0000000000019GM000000000009GW000000000030HA000000000001
HC000000000029IA000000000005JA000000000450JE000000000693JG000000000254KA000000000001KQ
000000000015KU000000000006KW000000000004LA000000000034LE000000000001LI000000000031M000
0000000106M2000000000079MA000000000003NC000000000779NG000000000357NK000000009608NO0000
00000016PA000000000031PE000000000007PM000000000031RY000000000010SC000000000054SE000000
000010
```

NI - Информация о сетевых соединениях

Вернуть информацию о хостовых соединениях.

```
21:28:45 |   to hsm : headNIH0
21:28:45 | from hsm :
headNJ000017005DC6D4A8929DBE9000000000005DC6D4A89416ADA000000000005DC6D4A89416AD500000
0000005DC6D4A89416AD4000000000005DC6D4A89416AD3000000000005DC6D4A89416AD2000000000005D
C6D4A89416ACD000000000005DC6D4A89416ACC000000000005DC6D4A89416ACB000000000005DE5FD372B
1BF9A000000000005DE5FD372B1C803000000000005DCBCE35902933E000000000005DE000000000000200
000000005DD0000000000002000000000005DC0000000000020000000010000000000000000000000100
000000000000000000000
```

Информация возвращается в распакованном шестнадцатеричном представлении. Для знакомых с устройством IP стеков все вполне очевидно.

Шифрование/дешифрование сообщений

Для шифрования/дешифрования (произвольных) сообщений HSM должен быть снабжен лицензией LIC008 Data Protection license.

Выполнение операций производится командой M0 – шифрование и M2 – дешифрование. Могут использоваться алгоритмы TDES или AES (при наличии лицензии LIC007 AES Data Encryption license). Использовать алгоритм AES для шифрования данных можно только при использовании блоковых AES LMK. (Причина очевидна.)

Шифрование может производиться в режиме электронной книги - ECB либо с зацеплением в режимах CBC, CFB8, CFB64, OFB. Режимы Format Preserving Encryption – BPS (Ingenico), FF1 (NIST approved FPE mode), Visa FPE рассматривать не будем.

Длина шифруемых в рамках одной команды данных не может превышать размеров буфера HSM и должна быть кратна блоку шифрования используемого алгоритма T-DES -8 байт, AES – 16 байт.

Если требуется шифровать сообщения большей длины, их следует разбить на блоки меньшего размера. Для режима без «зацепления» это не представляет проблем, а в режиме с зацеплением используется начальный вектор IV.

Для алгоритма TDES с вариантными LMK шифрование/дешифрование можно проводить ключами ZEK, DEK и TEK, а также BDK типов 1, 2 и 3.

Для блочных LMK можно использовать ключи ‘B0', ‘41’, ‘42’ только с алгоритмом TDES и ключи ‘D0’, ‘21’, ‘22’, ‘23’ для любого из алгоритмов DES, TDES и AES. Режим использования ключа естественно должен допускать шифрование/дешифрование сообразно применению.

Для ключей ZEK и TEK в настройках безопасности модуля (CS) может быть установлено ограничение либо запрет применения (по умолчанию). Параметр:

```
Enable ZEK/TEK encryption of ASCII data or Binary data or None: NONE
```

Простейшие примеры: Шифрование предварительно сгенерированным ключом DEK в режиме ECB данных представленных в шестнадцатеричном коде, шестнадцатеричное представление результата.

```
12:17:56 |   to hsm :
headM0001100BU7A2215B2F4F3C1F9A38D14AEDA31CB7E0020769C5C1A0C50E25891EE6A90B571798B
12:17:56 | from hsm : headM1000020E7868987417A80698248F97673D58BCB
```

Дешифрование возвращает первоначальное значение.

```
12:31:46 |   to hsm :
headM2001100BU7A2215B2F4F3C1F9A38D14AEDA31CB7E0020E7868987417A80698248F97673D58BCB
12:31:46 | from hsm : headM3000020769C5C1A0C50E25891EE6A90B571798B
```

Ключ DEK примечателен тем, что его невозможно экспортировать/импортировать, только генерировать. Этим он сходен с LMK и весьма полезен для шифрования локально хранящихся данных.

Множество подробностей и примеров применений вы найдете в [23] и в [16].