REPORT, IMRICH NAGY, DANIEL RYCHLÝ

PIN INSTALLATION

The PIN is installed using the applet installation parameter. With a real card, it would be printed on paper and distributed to the end-user. Our demo program prints the PIN to the standard output during installation. The PIN is required from the user to establish a secure channel.

WRONG PIN HANDLING

User has three attempts to insert the right PIN. Both the card and the PC maintain their counter. The PC counter might be incorrect if the execution procedure is incorrect. OwnerPIN object is used on the side of the card. As we are never comparing actual PIN, the counter is decremented on each suspicious PIN attempt by using intentionally wrong PIN constant 0000 (with the assertion on install that this PIN is never actually installed on the card). Counter decrementing is called every time a new ECDH init session starts and reset only when the authentication is successful. For each incoming APDU request, there is a check whether the card is blocked and the procedure continues only if not.

SESSION ESTABLISHMENT

The session establishment is initiated by the client PC and so is explicit session closing.

EPHEMERAL ECDH

The keys are established using elliptic curve Diffie-Hellman key exchange protocol using ephemeral ECDH keys. We used the NIST-224 (`secp224r1`) elliptic curve to achieve good support in Java Cards (following the information available at [1]). The public ECDH shares are transmitted in ANSI X9.62 encoding. Java Card implementation of ECDH produces a 20-byte hash of the established secret, so we did the same on the PC client side.

EKE - AUTHENTICATION OF THE ECDH SHARES USING THE PIN

As a PAKE protocol, we are using Encrypted Key Exchange as described in [4] with PIN hashes and based on elliptic curves. We decided not to implement augmented EKE, as in case of PIN hash compromise the PIN can be found easily anyway, utilizing dictionary attack.

PINs are hashed using the SHA-1 algorithm and truncated to 16 bytes and used as AES keys. The subsequent exchange has this structure:

1. PC prompts the card to init the exchange
2. Card decrements PIN then attempts to create a DH key pair and sends DH public key encrypted with the shared secret
3. PC creates own DH key pair, decrypts Card public key, computes session key K, creates a challenge
4. and encrypts it with K, then encrypts both the encrypted challenge and own public key with the shared secret and sends it.
5. Card decrypts both PC public key and challenge, computes session key K, creates own challenge and
6. send both, solved PC challenge and own challenge, encrypted by K
7. PC authenticates Card by checking own challenge, solves Card challenge and sends it encrypted by K
8. Card authenticates PC by checking own challenge

PIN attempts are reset and session counter set. For each step on the card, it is verified that preceding steps were made.

SECURE CHANNEL

KEY ESTABLISHMENT

From the established shared ECDH secret – keys are derived in the following way. SHA-256 hash of the secret is computed. The first 128 bits are used as the AES key and the second half is used as a 128-bit initialization vector for PC-to-Card channel and the other way around for Card-to-PC communication. Key derivation function was not used – it would be slow on the card and it does not provide additional security.

ENCRYPTION SETUP

128-bit AES in ECB mode with PKCS#5 padding is used to encrypt all data transmitted between PC client and the card. The parameters were chosen so that the majority of Java Cards would support it with good performance. The PKCS#5 Padding needed to be implemented by us on the card's side because the JCardSim does not support it. Because of troubles with JCardSim, each message has to start from the same initial state of the Cipher object, because using Cipher.update() instead of Cipher.doFinal() did not work on the card's side (Java Card Exceptions probably caused by the JCardSim). Therefore, the same message in one way is encrypted to the same ciphertext, but this would not be the case if we were able to implement the secure channel on the real Java Card. The message limit is 20 and there would not be the same messages sent in real-case scenarios.

SESSION CONTROL

Each session can consist of up to 20 messages encrypted with the established keys. In cases of exhausting the limit, the explicit command to end the session or power loss, the session is ended, established secrets and keys are wiped, and the counter is set to zero. Any further attempts to process instructions that should be protected by the secure channel are rejected by the card until the new session is explicitly started again. The provided client program can do this session establishment automatically when needed, requiring the user to type the PIN again.

IMPLEMENTATION

Only instructions used to establish the session can be executed by default. Sending APDU with any other instruction initiates secure channel handling. If the state of the applet is not "valid session", the instruction is denied and request code for starting a new session is sent back. If the applet is currently in the state of valid secure channel session, the APDU data is automatically decrypted before handing the APDU over. After the APDU is serviced by the appropriate method, the APDU data is automatically encrypted and sent back.

THE FINAL PROGRAM

There are several methods implemented to provide API to our implementation – for installing the applet with the PIN, starting and closing ECDH session and for sending and receiving data through a secure channel. Instructions on the Card's side are wrapped with the secure channel processing code so that any new methods could be easily added without any need to worry about the secure channel. Three example instructions/methods are implemented to show and use the secure channel. The demo program (the main function) uses our implementation to establish ECDH session, exchange data with the card using the secure channel and shows how to maintain a secure connection.

We used CardTools package from the 4th seminar to manage the connection with the JCardSim. There are only two members in our team and implementing our own "Card Manager" classes would require additional hours of work we simply did not have at our disposal. Any taken-over code is credited either directly in the documentation or as a whole in the repository readme.

We tried to use as little ROM as possible. However, Cipher.OneShot did not work in the JCardSim.

COMMENTED APDU TRACES (DEMO CODE OUTPUT)

Installing applet with PIN to card.
Connecting to card... Done.
User PIN: 1666 (Generated by program randomly, stored on the card, printed to the user, never stored to PC)

Starting new secure channel session.
Please enter 4-digit PIN: 1666 (typed in by user)
--> [B0620000] 4
<--
CADF92535B1242B811186D1EB4FF3B0F4E2C1E188C29A89CB592492DE34DECADF49C16AB1F10CFD6EDDDCDB791F16E0AC0EE900ED965
91B534191A59C59472B2 9000 (64) [47 ms]
-->
[B063000060102292828 5EEC0A77AB126CBDF461AB0C77FF6FC8C1C2483FF3FB26818CABE4554F1054770FE42D959C0460F17F86AB3704B
699F733EC3E6D617DB8C67DF09F10E74C710E516E2739197720E865E99FD8A06CB36E4EE92BEED697245E124CAAF] 101
<--
86D306D8CCC4AC20563A1D1F092BF4B311CCCA1BED0EA85DA61226EBD0760EC1833DE614D48CA04C55E0EA0471792C2C86EDE37C4071
978FCC26C2C1406077D3 9000 (64) [15 ms]
--> [B06400002090A3AC532F0A42F103103F76B31CD3610775FCC3F6A42C4ECB2CAB2F19062629] 37
<-- 01 9000 (1) [0 ms]
Authentication was successful!

Communicating with the card using secure channel.
Marco-Polo:
--> [B070000010A8834AD41454AD99AD8943EF4FA23987] 21 **("marco" encrypted using secure channel)**
<-- B9EFAF1CCEC130BAB070670782654C5B 9000 (16) [0 ms] **("polo" encrypted using secure channel)**
Received '706F6C6F', which is 'polo' in ASCII.

Storing 'J.E. did not commit suicide. Also, the cake is a lie.' on card... (The message)
-->
[B071000040A478426BB44372E970DC599C1E3B1CD86D12D3207CF6E50961692EE75A3839B6FE08B586A2CB434E7155AB1E5C72AC68AC6
11756E6C0A58EFC0BB3F8B4A97B7D] 69 **(Store encrypted message)**
<-- F002841D34313865D42AC54779E79EB9 9000 (16) [0 ms] **(Empty response)**
--> [B072000010B36A7D234E9AAB88C8CEC5F5BF711B71] 21 **(Command to load previously stored message)**
<--
B6EED977DC2E4305E38FE7E2C5C5D7C0EA145ABA65C68EA0FB4B55ED784F143A65FD8A1C0594BFAC5EBED15EBF57D81D886E7BB234E71
36AB323DE67BD689CE8 9000 (64) [0 ms] **(Encrypted message)**
...and receiving 'J.E. did not commit suicide. Also, the cake is a lie.' back. (Decrypted the original message)

Now, Marco-Polo will be performed 17 times to exhaust the session message limit
--> [B070000010A8834AD41454AD99AD8943EF4FA23987] 21
<-- B9EFAF1CCEC130BAB070670782654C5B 9000 (16) [0 ms]
Received '706F6C6F', which is 'polo' in ASCII.
...Marco-Polo traces 16 more times edited out…

Now, the secret message will be read, but new session is required
--> [B072000010A456E7C6FED77E10094EEA28CE2987CE] 21 **(Attempt to load previously stored message)**
<-- 6901 [0 ms] **(Return code for invalid session)**
Session needs to be reestablished.
Please enter 4-digit PIN: 1666 (Typed in by user)
--> [B0620000] 4 **(The whole EKE ECDH is done again, thus beginning a new session)**
<--
08A8FB0601C284AFF0A112FB568C3F620803D05088A31592BC593CAB8FFC6B42FA2DAB76F2B88C9A35F0DA89ED92B0174391F780AF678
63E6DC8E5C4F4D5A697 9000 (64) [16 ms]

-->
[B0630000609B84A5553AB1DBED81EFC316B72C824181CB6218D8D13108FBA090311B2425C9E208B02FFBC6D0B7E4E9393B2406A51B12D02E53D0E405144810C1CEE340FB23242D35DB261047615D86B5745E62C9EEB3DE6F850A472DF88D002A6D487531AF] 101
<--
968F8502041C7B5F98D6D9673A4963450465936D59FF2D45F2A8B574A2FE5C0CBEEA95A3FD59E3A94A42DB02EA02C4EE072F65CB38D3E7A06ED236F8D4F5F4A2 9000 (64) [0 ms]
--> [B064000020D82B236473E5946837E802A67C430E6707407D7F841563D581A1323A48B4016A] 37
<-- 01 9000 (1) [0 ms]

Authentication was successful!

--> [B07200001049D75C8C882F240682098820D20A393E] 21 **(Automatically resend failed APDU encrypted with new session key – load previously stored message)**

<--
92FBF8BB38BE392B3C87A11938E14FFAA6818F010E05955D08E74729CF9FB104E8F7EB52298CF85A33495C04ACFBBC1A35D395EF7452E1270E63A8042A536444 9000 (64) [0 ms] **(Now, the stored message is successfully received)**

Received 'J.E. did not commit suicide. Also, the cake is a lie.'.

Closing channel.

--> [B065000000] 5 **(Session closing command. The result is the same as exhausting the session message limit or disconnecting the card – no more secured APDUs can be processed until a new ECDH session is initiated)**

<-- 9000 [0 ms]

Channel closed.

GITHUB & TEAM WORKFLOW

We used the GitHub Project feature for the whole semester [2] while creating a separate milestone for each phase of the project. Issues were used to distribute the work and keep track of the implementation. The implementation was done in separate branches to keep the repository clean and organized [3].

The project took about 50 hours of work per team member so far. We sure could have used the third-guy-who-never-does-anything, who could at least maintain the repository, write the report and prepare a more detailed demo program or some simple tests.

SOURCES

[1] https://www.fi.muni.cz/~xsvenda/jcalgtest/
[2] https://github.com/NimRo97/pv204_in_dr/projects/1
[3] https://github.com/NimRo97/pv204_in_dr/network
[4] https://www.cs.columbia.edu/~smb/papers/neke.pdf