Security Model

Parties involved in smart card–based system:

- Cardholder
- Data owner
- Terminal
- Card issuer
- Card manufacturer
- Software manufacturer

Smart card threat models:

- attacks by the terminal against the cardholder
- attacks by the cardholder against the terminal
- attacks by the cardholder against the data owner
- attacks by the cardholder against the issuer
- attacks by the cardholder against the software manufacturer
- attacks by the terminal owner against the issuer
- attacks by the issuer against the cardholder
- attacks by the (software)manufacturer against the data owner

Estonian ID card

- Used for:
  - Protected RSA private key storage
  - Perform on-card signing/decryption
  - Authorize cryptographic operations (using PIN)

- Cardholder / Data owner / Terminal / Card issuer / Card manufacturer / Software manufacturer

- Attacks:
  - by the terminal against the cardholder
  - by the cardholder against the terminal
  - by the cardholder against the data owner
  - by the cardholder against the issuer
  - by the issuer against the cardholder
  - by the (software)manufacturer against the data owner
Mobile phones (SIM card)

- Used for:
  - Store phone book contacts and SMS messages
  - Store settings (operator information)
  - Store 128-bit symmetric subscriber authentication key
  - Perform RUN GSM ALGORITHM
  - Authorize operations (using PIN)
  - Mobile-ID

- Attacks:
  - by the cardholder against the data owner
  - by the terminal owner against the issuer
  - by the issuer against the cardholder
Payments (EMV)

EMV stands for Europay, MasterCard and Visa

• Used for:
  • Store symmetric MAC key
  • Authentication of credit card transactions (using PIN)

• Attacks:
  • by the terminal against the cardholder
  • by the cardholder against the data owner
  • by the cardholder against the issuer
  • by the terminal owner against the issuer
  • by the issuer against the data owner
Other Payment Cards

• Used for:
  • Store credit value
  • Store account number

• Attacks:
  • by the cardholder against the terminal
  • by the cardholder against the data owner/issuer
Pay TV

- Used for:
  - TV signal decryption
  - Store channel filters

- Attacks:
  - by the cardholder against the data owner/issuer
  - by the terminal owner against the issuer
Tachograph

- **Used for:**
  - Record driving activities

- **Attacks:**
  - by the cardholder against the data owner/issuer
  - by the terminal owner against the issuer
Attacks Against Smart Cards

- Side channel attacks:
  - Timing analysis
  - Power analysis
  - EM signal analysis

- Introducing glitches, faults (voltage, clock rate)
  - Induce bit errors

- Physical attacks:
  - Chemical etching
  - Chip re-wiring
  - Addition of a track
  - Cutting of a track

- Countermeasures
  - Metal layers
  - Onboard sensors (temp, light, frequency)
  - ...
Task 1

Implement utility that performs signing on ID card.

$ .esteid_sign.py somefile somefile.signature
[+] Selected reader: Gemalto PC Twin Reader 01 00
[+] EstEID v1.1 on MultiOS (DigiID)
[=] Signing file somefile
[+] Calculated SHA1 digest: 9498115e5475693cd7f32184a22190ab6606642c
[+] Making DigestInfo object...
[+] Sending DigestInfo to smart card for signing...
[?] Enter PIN2: 02611
[+] Signature saved into somefile.signature

$ .esteid_getcert.py --cert sign --out sign.pem
$ openssl x509 -in sign.pem -pubkey -noout > signpub.pem
$ openssl dgst -verify signpub.pem -sha1 -signature somefile.signature somefile
Verified OK

$ openssl rsautl -certin -inkey sign.pem -in somefile.signature -verify -raw -hexdump
Task 1: Signing (spec page 40)

- SELECT FILE MF/EEEEE
- With MANAGE SECURITY ENVIRONMENT restore security environment #1
- With VERIFY verify PIN2
  - Obtain PIN2 using python’s `raw_input()`
  - PIN has to be sent as list of chars `[ord(c) for c in pin2]`
  - Wrong PIN decreases PIN retry counter (!!!)
    - Successfully entered PIN resets retry counter
    - If PIN blocks, it can be reset by using PUK (id.ee)
    - If PUK blocks, PINs can be reset in bank or PBGB
- With PERFORM SECURITY OPERATION compute digital signature (spec page 131)
Task 2

Implement utility that performs decryption on ID card.

$ echo -n "secret hello to you..." > plain.txt
$ openssl rsautl -encrypt -certin -inkey sign.pem -in plain.txt -pkcs -out ciphertext

$ ./esteid_decrypt.py ciphertext
[+] Selected reader: Gemalto PC Twin Reader 01 00
[+] EstEID v1.1 on MultiOS (DigiID)
[?] Enter PIN2: 02611
[+] Decrypted text: secret hello to you...

• Decrypting (spec page 48):
  • SELECT FILE MF/EEEE
  • With MANAGE SECURITY ENVIRONMENT restore security environment #6
  • With MANAGE SECURITY ENVIRONMENT delete reference to auth and sign keys
  • With MANAGE SECURITY ENVIRONMENT tell that we want to decrypt with sign key
  • With VERIFY verify PIN2
  • With PERFORM SECURITY OPERATION decipher ciphertext (spec page 131)
If the data in APDU is larger than 255 bytes, data must be sent to the card in several blocks.

- Required for task 2 if your card has 2048-bit RSA keys (data will be 257 bytes (256 bytes ciphertext + 0x00 byte prefix))

Split the data in smaller blocks and send to the card using 0x10 CLA byte (except for the last block).
Bonus (+2 points)

Provide utilities with --measure argument which measures the time needed to perform 100 operations:

$ ./esteid_decrypt.py ciphertext --measure
[+] Selected reader: Gemalto PC Twin Reader 01 00
[+] EstEID v1.1 on MultiOS (DigiID)
[+] Measuring the time required for 100 decryptions...
[?] Enter PIN2: 02611
[+] Time: 100.336151

$ ./esteid_sign.py somefile somefile.signature --measure
[+] Selected reader: Gemalto PC Twin Reader 01 00
[+] EstEID v1.1 on MultiOS (DigiID)
[=] Signing file somefile
[+] Calculated SHA1 digest: 9498115e5475693cd7f32184a22190ab6606642c
[+] Making DigestInfo object...
[+] Measuring the time required for 100 signings...
[+] Sending DigestInfo to smart card for signing...
[?] Enter PIN2: 02611
[+] Time: 100.647671
• Measurement loop must contain only minimum operations required

• The time can be measured using:

```python
import datetime
s = datetime.datetime.now()
print "[+] Time:" , (datetime.datetime.now()-s).total_seconds()
```

• Run each experiment 3 times and provide min/max measurement output into esteid_sign.out and esteid_decrypt.out on your repository

• Give your educated guess why one operation is faster than the other