How Crypto is Used for Finance

Igor Semaev

May 9, 2012

My Story

- Stayed in Brussels 27.11-30.11.2011
- 2 unexpected withdrawals from my account 10 days later
- $>$ 1000 NOK in GBP paid for internet shopping in UK
- $>$ 40 NOK in EUR (train ticket?)
- Card changing is necessary

How Sparebank Vest protects my money

- **ATM, shopping**: Chip VISA Card, PIN(secret)
- **On-line banking**: Social security number(more or less secret),
- Password of my choice(secret),
- One-time password with a calculator
- **Internet shopping**: Card number(not secret), Verified-by-VISA password of my choice(secret)
- Why didn't the latter work?!

Security Tools at Sparebank Vest

- BankAxept for retail shopping
- BankAccess/BankID online shopping and personal authentication
- One-time password calculator is said related to my account number. How?
- Information lack

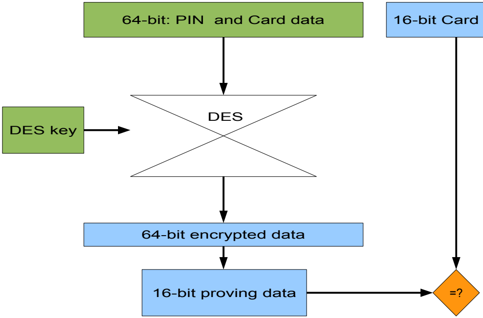
What it is about

- Numerous cases of bank frauds
- Ross Anderson: Why cryptography fails?
- Because not only cryptography matters
- However no cryptography, no modern banking

Why Crypto matters?

- Bank cards were stolen(2001?) from a Norwegian X in Spain
- US$1000 were withdrawn one hour later (correct PIN with the first attempt)
- Bank: ATM is secure, protected with **DES**, likely X had kept the card and the PIN together
- X: PIN was not kept with the card
- Trial: X against bank, lost at a lower court
- Hole et al., 2004: if **DES**, theoretically possible
- Bank(appeals court): **3DES** was used!!!

Hole et al model



- Card magnetic stripe keeps: Account Number, 16-bit verification data(PIN, **DES** key),..
- One **DES** key for all cards from the same Bank
- Transform: 64-bit $\Rightarrow$ 16-bit is not publicly available

Attack

- Available same bank cards(one needs 4) with correct PINs: 4×16-bit constraint on 56-bit key
- Brute force the Bank **DES** key, 2^{56} trials?(offline phase)
- Brute force the stolen card PIN (online phase)
- Withdraw money

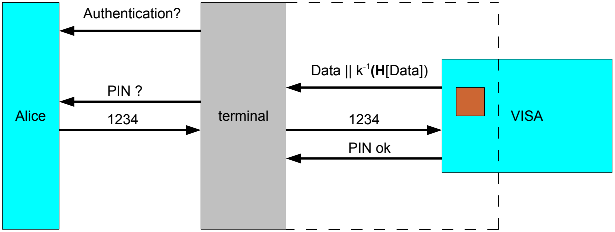
How PIN is generated

- IBM PIN generation method: Account Number $\Rightarrow$ PIN

- symmetric PIN-key K for all cards from the same Bank
- **DES**(K)[Account Number] $\Rightarrow$ 4-decimal digit "Natural PIN"
- Natural PIN + Offset = Customer PIN

- Magnetic stripe contains Account Number, Offset
- ATM possesses K to verify PIN
- if **DES**, then Hole et al. attack works as well

French Card/PIN Authentication in 1998



- Terminal authenticates the Card, Card verifies PIN
- (k, k^{-1}) =(public, private) Bank **RSA** keys
- Faked "yes" card:
  1. A copy of the true Data $|| \, k^{-1}(\mathbf{H}[\text{Data}])$
  2. "PIN ok" answer to any PIN
- Buy and Withdraw, Data pays

Case Humpich, 1998

- Serge Humpich, French engineer, worked in a bank for 12 years
- Factored **RSA** 321-bit modulus($>$ 500-bit that time record)
- Computed k^{-1} from k
- Created several "yes" cards:
  1. xxx $|| \, k^{-1}(\mathbf{H}[\mathrm{xxx}])$
  2. "PIN ok" answer to any PIN
- xxx false Data, so the cards worked off-line
- Bought 10 Paris underground tickets to demonstrate his invention. Got 10 months in jail and considerable fine

EMV standards

- EMV is Europay, MasterCard, VISA
- EMV protocol for smart card payment to fix the above and similar cases:
- Hundreds of millions cards in circulation for ATM, point-of-sell terminals
- called "Chip and PIN" cards

- Generated "liability shift"(according to Ross Anderson):
- Now PIN authorized disputed transaction would be charged to the customer
- Previously, manuscript signature frauds were charged to the merchant

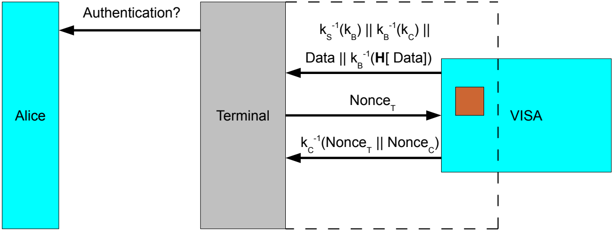
EMV at a Glance

- EMV Specification for Payment Systems, version 4.3, November 2011, based on ISO standards. Three stages:
- Card Authentication assures the terminal: which Bank issued the Card, and that Card data haven't been altered
- Cardholder Verification off-line: entered PIN matches one on the Card. On-line verification is not specified
- Transaction Authorization: Bank authorizes the transaction

What the chip does?

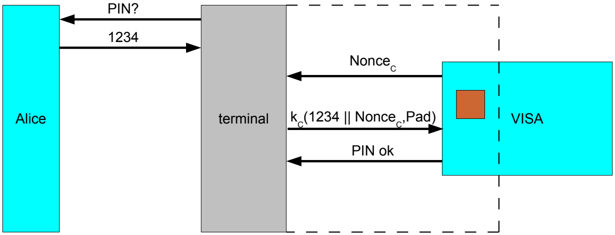
- Tiny computer keeps
  1. cardholder's PIN
  2. symmetric encryption keys
  3. card private asymmetric key
- computes(verifies) digital signatures, encryptions with e.g.
- **RSA** cryptosystem
- **3DES**, **AES**
- Message Authentication Code(**MAC**) for integrity and authentication

Dynamic Card Authentication off-line



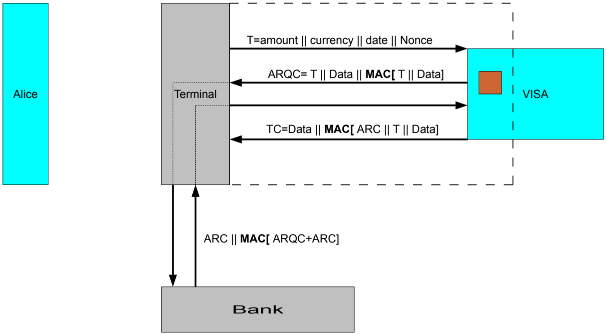
- Card provides:
  1. $k_S^{-1}(k_B)$ certified Bank public key
  2. $k_B^{-1}(k_C)$ certified Card public key
- Terminal holds k_S, recovers k_B and k_C. Verifies
  1. Data $||\ k_B^{-1}(\mathbf{H}[\text{Data}])$ with k_B
  2. $k_C^{-1}(\text{Nonce}_T || \text{Nonce}_C)$ with k_C

Cardholder Verification off-line



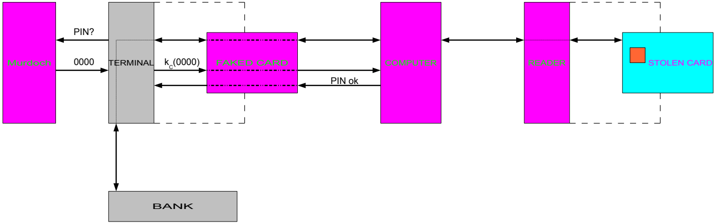
- Specific for points-of-sell
- Alice introduces the PIN
- Terminal encrypts the PIN with the card publ. key k_C
- Card decrypts the PIN and compares with what on the chip

Transaction authorization



- ARQC authorization request cryptogram
- ARC authorization response code
- TC transaction certificate
- **MAC** depends on a symmetric key shared by Card and Bank

Steven Murdoch Attack



- Stolen card does Card Authentication
- Cardholder Verification: "yes" to any PIN
- Stolen card does Transaction Authorization
- A protocol flaw is exploited, does not seem fixed in EMV 4.3

Security Mechanisms

- Block ciphers and **MAC**
- Symmetric Key Management
- Hash-functions and **HMAC**
- **RSA**
- Elliptic Curve Crypto(**ECC**)

Block Ciphers and **MAC**

- Block Ciphers
- **MAC** Computation
- How **MAC** works

Block Ciphers

- For Message Authentication Code (**MAC**) and encrypting in on-line PIN-verification

- **ALG** n-byte block cipher algorithm
- $n = 8$ for **3DES** and $n = 16$ for **AES**128(256)

- plain-text $X_1, \ldots, X_B$ to cipher-text $Y_1, \ldots, Y_B$
- K encryption key

- ECB: $Y_i = \mathbf{ALG}(K)[X_i]$
- CBC: $Y_0 = 0$ and $Y_i = \mathbf{ALG}(K)[X_i \oplus Y_{i-1}]$

Key Management

- Session Key Derivation
- Card Master Key Derivation

Card Master Key Derivation

- Bank(Issure) master key IK_M
- Y padded Account Number
- $Z_L = \mathbf{3DES}(IK_M)[Y]$
- $Z_R = \mathbf{3DES}(IK_M)[Y \oplus C]$
- C constant
- **3DES**: $K_M = Z_L, Z_R$ after setting parity for **DES** keys
- Similar for **AES**

- K_M put on the Card chip

Session Key Derivation

- Card and Bank share Card master key K_M
- Diversification 8(16)-byte value: $R = \text{TransactionData}, 0.., 0$
- **MAC** session key:
- K_S =truncate(**ALG**$(K_M)[R] \| $ **ALG**$(K_M)[R \oplus C]$)

MAC Computation

- 8-byte ciphers, **MAC** of $4 \leq s \leq 8$ bytes

- **3DES**: K_S 128-bit session key(with parity redundancy)
- $H_0 = 0$ and $H_i = \textbf{ALG}(K_S)[X_i \oplus H_{i-1}]$ for $i \leq B$, **MAC**=truncate(H_B)

- **DES**: $K_S = K_{SL}, K_{SR}$ session key
- $H_0 = 0$ and $H_i = \textbf{ALG}(K_{SL})[X_i \oplus H_{i-1}]$
- **MAC**=truncate($\textbf{ALG}(K_{SL})[\textbf{ALG}^{-1}(K_{SR})[H_B]]$)

- Similar with 16-byte **AES**

How **MAC** works?

- Data $= X_1, \ldots, X_B$ message from Card to Bank
- Card sends Data$||$ **MAC**(K_S)[Data]
- This session key K_S depends on the transaction and master key K_M
- Bank shares K_M, it computes K_S, **MAC**(K_S)[Data]
- Thus Bank verifies the text integrity and authenticity
- Similar to signature, though does not provide non-repudiation

Hash Function and **HMAC**

- SHA-1 with **RSA** in EMV 4.3: 20-byte output
- SHA-256, SHA-512 with **ECC** according to the draft EMV 4.1z ECC
- **HMAC** a hash-function based **MAC**
- $\mathbf{HMAC}(K, \mathrm{MSG}) = \mathbf{Hash}[K \oplus C_1 || \mathbf{Hash}[K \oplus C_2 || \mathrm{MSG}]]$

RSA

- **RSA** functions
- Signature Generation/Verification
- Secure **RSA** Key Generation

RSA functions

- **RSA** Public key P_k: $n = pq$ and $e = 3$ or $2^{16} + 1$
- **RSA** Private key S_K: $n = pq$ and d, where $de = 1 \mod (p-1)(q-1)$

- S = $\textbf{Sign}(S_K)[X] = X^d \mod n$ (also Decryption)
- X = $\textbf{Recover}(P_K)[S] = S^e \mod n$ (also Encryption)

Signature Generation/Verification

- Compute 20-byte $\mathrm{H} = \textbf{Hash}(\mathrm{MSG})$
- Split $\mathrm{MSG} = \mathrm{MSG}_1 || \mathrm{MSG}_2$
- N-byte $X = \mathrm{MSG}_1 || \mathrm{H} || \mathrm{Constant}$
- To Sign: $\mathrm{S} = \textbf{Sign}(S_K)[X]$,
- Signed message: $\mathrm{MSG} || \mathrm{S}$

- To verify: $X = \textbf{Recover}(P_K)[\mathrm{S}]$
- So recover MSG_1, and H, and Constant
- compute **Hash**(MSG)
- Compare

RSA Key Generation

- Fixed e, each card has its own $n = pq$ and d
- Massive production p, q is required
- Security requirements:
  1. $p - q$ large
  2. $p - 1$ has a large prime factor
  3. prime $r | p - 1$, then $r - 1$ has a large prime factor
- Probable prime: likely prime, small error probability
- Provable prime: there is a proof

Probable Primes with Rabin Test

- N odd natural number to test
- $N - 1 = a2^h$, a is odd
- Randomly choose b,
- "Pass" if $b^a \equiv \pm 1 \mod N$
- or $b^{a2^i} \equiv -1 \mod N$, where $0 < i < h$
- "Fail" otherwise

- $\mathbf{Pr}(\text{"Pass"} \,|\, N \text{ composite}) \leq \frac{1}{4}$

Provable Primes with Poklington Theorem

- N odd natural number to test
- $N - 1 = F \times R$, and $q_1, \ldots, q_t$ distinct prime factors in F
- Let for some a
    1. $a^{N-1} \equiv 1 \mod N$
    2. $\gcd(a^{(N-1)/q_i} - 1, N) = 1$ for all $i = 1, .., t$
- and $F > \sqrt{N}$
- Then N is prime

- Improvement by Brillhart, Lehmer, Selfridge: $F > \sqrt[3]{N}$ and some additional condition
- Combination: first Rabin, then Poklington

Elliptic Curve Crypto

- Why **ECC**?
- Brief definitions
- Recommended Curves
- **ECC** Signature Algorithm
- **ECC** Verification Algorithm
- **ECC** for Encrypting

Why **ECC**?

- EMV: **RSA** public key is up to 248 bytes=1948 bits
- Factoring progress: December 2009, 768-bit **RSA** modulus was factored
- **RSA** vulnerability$\Rightarrow$ modulus increases
- No progress(until very recently in binary case) in discrete log on elliptic curves
- Move to elliptic curve crypto?
- Same security with lower parameters
- EMV proposes **ECC** to start 2015

Elliptic Curve

- Modulus: prime number p
- Set $E = (x, y) \mod p : y^2 = x^3 + ax + b$ and P_∞. FIPS 186-2: $a = -3$
- Group operation:

$$(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$$

- x_3, y_3 are rational functions in x_1, y_1, x_2, y_2
- Relatively easy to compute, a few multiplications $\mod p$
- Cyclic subgroup generator $G \in E$ of order n
- **Private key** $d \mod n$, **Public key** $P = d \times G$
- Given G: $P \Rightarrow d$ is hard(Elliptic Curve discrete log problem)

Recommended Curves

- curve $P - 256$
- Modulus $p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$, explicit b, n, G
- curve $P - 512$
- Modulus $p = 2^{512} - 1$, explicit b, n, G

ECC Signature Algorithm

- To sign message MSG
- Take random $k \mod n$
- Compute $kG = (x_1, y_1)$ and $r \equiv x_1 \mod n$
- Compute $s \equiv k^{-1}(\textbf{Hash}[\mathrm{MSG}] + d\, r) \mod n$
- Signature s, r

ECC verification algorithm

- To verify MSG signature s, r
- Compute $u_1 \equiv \textbf{Hash}[\text{MSG}]\, s^{-1} \mod n$
- Compute $u_2 = r\, s^{-1} \mod n$
- Compute $(x_0, y_0) = u_1 G + u_2 P$
- Verify the congruence $r \equiv x_0 \mod n$

ECC for Encrypting

- Used for PIN encipherment
- Combination of ECC and Hash algorithm
- approved Hash algorithms:
- for P-256 SHA-256
- for P-512 SHA-512
- Generates HMAC on the cipher-text

ECC encryption algorithm

- PIN $\Rightarrow$ 17-byte MSG
- Random k, compute kP and kG
- Key-stream: $K_1||K_2||.. = \textbf{Hash}[kP||0..01]||\textbf{Hash}[kP||0..02]$
- Encrypting $X = K_1 \oplus \text{MSG}$
- $T = \textbf{HMAC}(K_2||X)$
- Encrypted message $kG||X||T$
- To decrypt X and verify T:
- Compute $kP = d(kG)$, generate key-stream,..

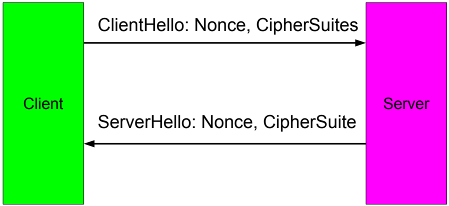
SSL/TSL protocol

- Secure Sockets Layer (SSL) since 1994
- Transport Security Layer (TSL) since 1999, last version 2011
- Provides communication security over the Internet: on-line shopping
- The transaction was protected with 128-bit SSL?

SSL/TSL protocol Goals

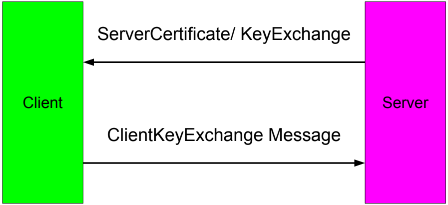
- Negotiate Crypto-tools:
  - Asymmetric crypto for key exchange
  - Symmetric encryption for privacy
  - Message authentication codes (MAC) for message integrity
- Negotiate key material for Symmetric Encryption and MAC

Phase 1: Handshake Protocol



- CipherSuites:
  TLS_ **RSA**_WITH_**AES**_128_CBC_SHA256
  ...
  TLS_**DH**_**DSS**_WITH_**3DES**_EDE_CBC_SHA
- The Server chooses a CipherSuite to use

Phase 2: share 48-byte PremasterSecret



- **Server Message**: if RSA, contains $k_{CA}^{-1}(k_S)$
- if DH_DSS, certified DSA public key, certified $p, g, g^x \mod p$
- **Client Message**: if RSA, contains k_S(PremasterSecter)
- if DH, contains(certified?) $g^y \mod p$

Phase 3: PremasterSecret $\rightarrow$ MasterSecret

- if **RSA**, then Client, Server share PremasterSecret
- if **DH**, PremasterSecret = truncated $g^{xy}\, mod p$
- MasterSecret=**PRF**(PremasterSecret,"master secret",ClientNonce+ServerNonce) truncated to 48 bytes
- **PRF** PseudoRandomFunction constructed with SHA256 based **HMAC**

Phase 4: Key Calculation

- KeyBlock=**PRF**(MasterSecret,"key expansion",ClientNonce+ServerNonce)
- of length enough for the CipherSuite
- KeyBlock is partitioned to
  1. Client **MAC** key
  2. Server **MAC** key
  3. Client symmetric key
  4. Server symmetric key
  5. Client IV(if necessary)
  6. Server IV(if necessary)
- **AES**_256_CBC_SHA256 requires maximum of 128 bytes of key material