COSIC

KU LEUVEN

Cryptography Best Practices

Prof. Bart Preneel
COSIC, an imec lab at KU Leuven
Bart.Preneel(at)esatDOTkuleuven.be
http://homes.esat.kuleuven.be/~preneel
February 2019

1

Outline

- Architecture
- Network protocols
- Security APIs
- Key establishment: protocols, generation, storage
- Implementing digital signature schemes

2

Symmetric vs. Asymmetric Algorithms

- hardware costs: 1 K–100K gates
- performance: 10 Mbit/s – 1000 Gbit/s
- keys: 64-256 bits
- blocks: 64-128-256 bits
- power consumption: 3-30 μJ/bit

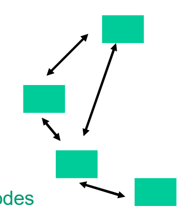
- hardware costs: 12 K-1M gates
- performance: 10 Kbit/s – 100 Mbit/s
- keys: 256-4096 bits
- blocks: 256-4096 bits
- power consumption: 1000-2000 μJ/bit
- postquantum: keys of 10-500 Kbyte

3

Architectures (1a) (sym.)

- Point to point
- Local
- Small scale

- Number of keys: 1 or n^2
- Manual keying



Example: ad hoc PAN or WLAN

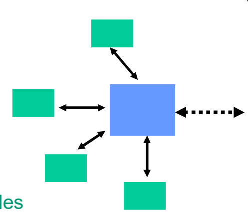
n nodes

4

Architectures (2a) (sym.)

- Centralized
- Small or large scale
- Manual keying

- Number of keys: n
- ! Central database: risk + big brother
- Non-repudiation of origin? (physical assumptions)



Example: WLAN, e-banking, GSM

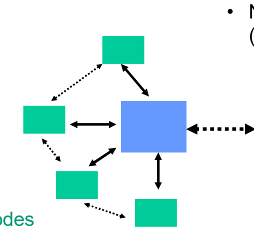
n nodes

5

Architectures (3a) (sym.)

- Centralized
- Small or large scale
- Manual keying

- Number of keys: n + 1/session
- ! Central database: risk + big brother
- Non-repudiation of origin? (physical assumptions)



Example: LAN (Kerberos)

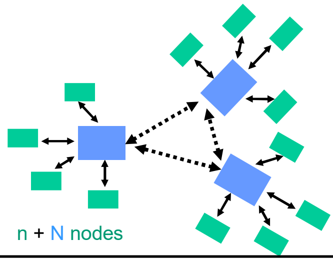
n nodes

6

Architectures (4a) (sym.)

- Decentralized
- Large scale

- Number of keys: n + N^2
- Risks?
- Trust
- Hard to manage



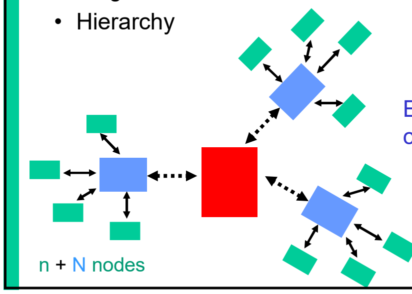
Example: network of LANs, GSM, 3G, 4G

7

Architectures (5a) (sym.)

- Centralized
- Large scale
- Hierarchy

- Number of keys: n + N



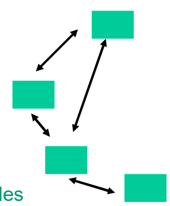
Example: credit card and ATM

8

Architectures (1b) (asym.)

- Point to point
- Worldwide
- Small networks

- No CA (e.g. PGP)



Example:
P2P, international organizations

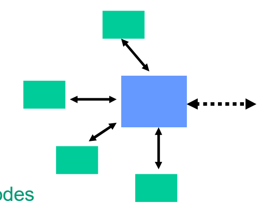
n nodes

9

Architectures (2b) (asym.)

- Centralized
- Large or small scale

- Reduced risk
- Non-repudiation of origin



Example: B2C
e-banking

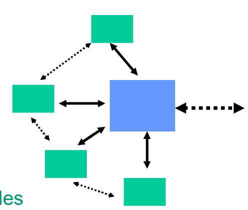
n nodes

10

Architectures (3b) (asym.)

- Centralized
- Small or large scale

- Reduced risk
- Non-repudiation of origin



Example: B2B and e-ID

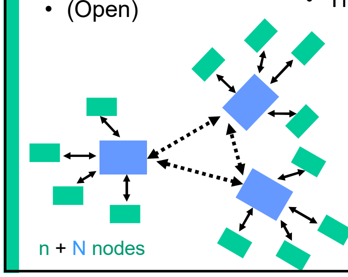
n nodes

11

Architectures (4b) (asym.)

- Decentralized
- Large scale
- (Open)

- Key management architecture?
- Trust



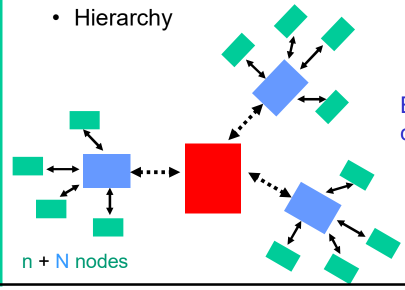
Example: B2B, GSM interoperator communication

12

Architectures (5b) (asym.)

- Centralized
- Large scale
- Hierarchy
- Open

Example: credit card EMV



13

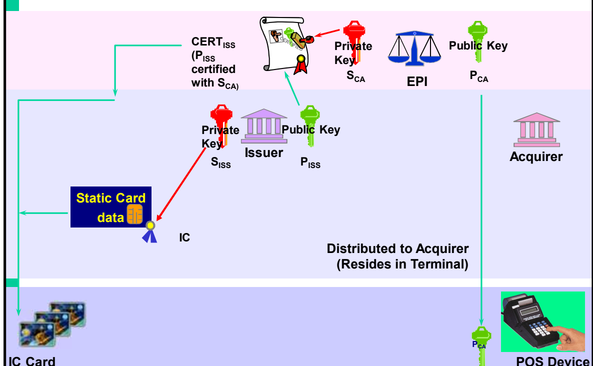
When asymmetric cryptology?

- if manual secret key installation not feasible (also in point-to-point)
- open networks (no prior customer relation or contract)
- get rid of risk of central key store
- mutually distrusting parties
  - strong non-repudiation of origin is needed
- fancy properties: e-voting

Important lesson: on-line trust relationships should reflect real-word trust relationships

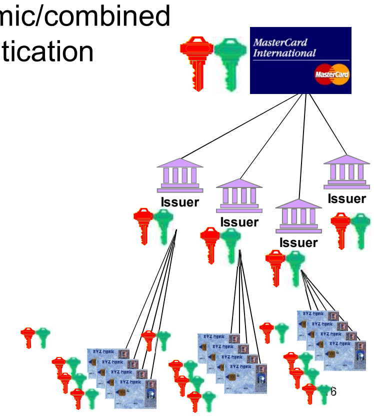
14

EMV Static Data Authentication (SDA)



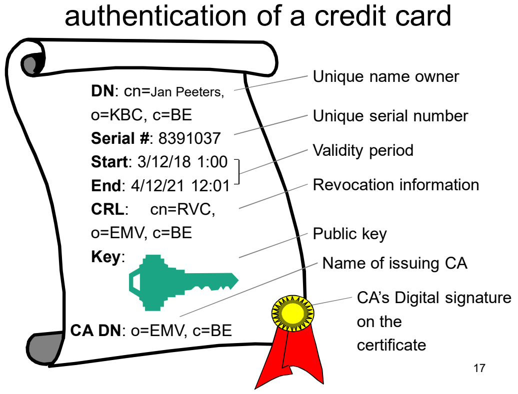
EMV: dynamic/combined data authentication

- Three layers:
  - EPI
  - Issuers
  - Cards



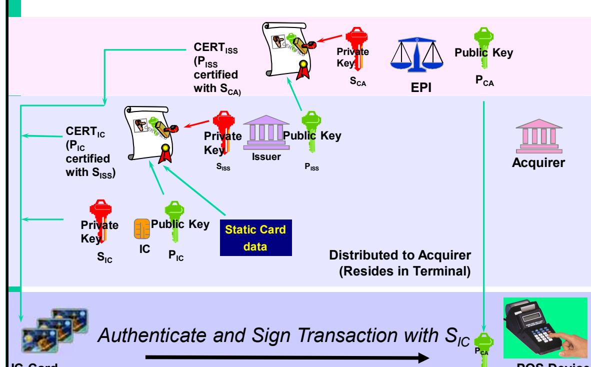
16

Certificate for dynamic data authentication of a credit card



17

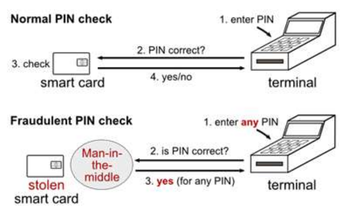
EMV Combined Data Authentication



Warning about EMV

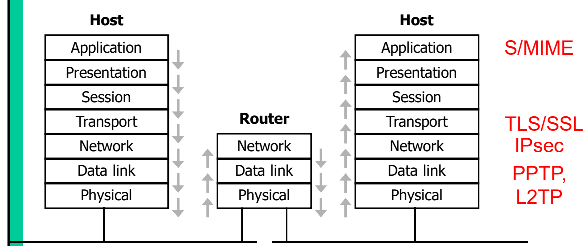
http://www.cl.cam.ac.uk/research/security/banking/nopin/oakland10chipbroken.pdf

- Pin checking and authentication are not coupled
- **EMV PIN verification "wedge" vulnerability** S.J. Murdoch, S. Drimer, R. Anderson, M. Bond, IEEE Security & Privacy 2010



19

Network protocols



20

Where to put security?

- Application layer:
  - closer to user
  - more sophisticated/granular controls
  - end-to-end
  - but what about firewalls?
- Lower layer:
  - application independent
  - hide traffic data
  - but vulnerable in middle points
- Combine?

21

Where to put security? (2)

From: Bob@crypto.com

To: Alice@digicrime.com

Subject: Re: Can you meet me on Monday at 3pm to resolve the price issue?

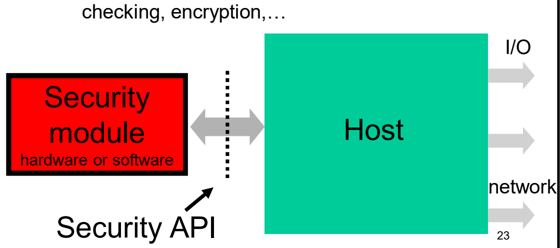
This proposal is acceptable for me.

-- Bob

22

Security APIs

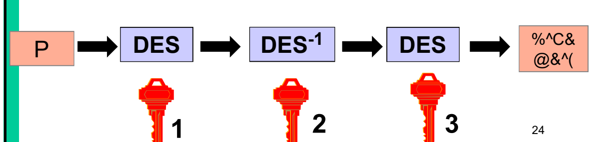
- Security module controls access to and processing of sensitive data
  - executes cryptographic commands, e.g. PIN checking, encryption,…



23

Master key/data key

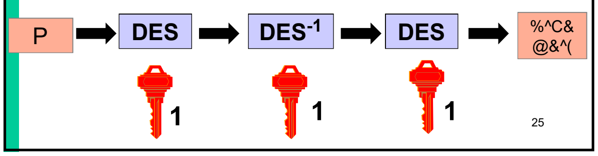
- Load master 3DES key **KM** (tightly controlled)
- Load data key:
  $3DES_{KM}(K1) || 3DES_{KM}(K2) || 3DES_{KM}(K3)$
- Send plaintext P and ask for encryption
  $DES_{K1}(DES^{-1}_{K2}(DES_{K3}(P)))$



24

Master key/data key (2)

- Load master 3DES key **KM** (tightly controlled)
- Load corrupted data key:
  $DES_{KM}(K1) || DES_{KM}(K1) || DES_{KM}(K1)$
- Send plaintext P and ask for encryption
  $DES_{K1}(DES^{-1}_{K1}(DES_{K1}(P))) = DES_{K1}(P)$



25

Control vectors in the IBM 4758 (1)

- Potted in epoxy resin
- Protective tamper-sensing membrane, chemically identical to potting compound
- Detectors for temperature & X-Rays
- "Tempest" shielding for RF emission
- Low pass filters on power supply rails
- Multi-stage "latching" boot sequence

= STATE OF THE ART PROTECTION!

26

IBM 4758

27

Features of the IBM 4758

- Control vector: type (e.g., PIN, data, MAC)
  store key of type type as $E_{Km + \text{“type”}}(k)$
  - Output of encryption with key of type "PIN" is never allowed to leave the box
  - Output of encryption with key of type data, MAC, … may leave the box
- High security master key import: 3 shares
  - Import Km as $Km_A + Km_B + Km_C$

28

Master key import

Km_A Km_B Km_C → **1**

$Km = Km_A + Km_B + Km_C$

29

Fraudulous import

Km_A Km_B $Km_C = Km_C +$ "data" – "PIN" → **2**

$Km^* = Km_A + Km_B + Km_C^* = Km$ + "data" – "PIN"

30

The attack

Transport PIN key k from box 1 to box 2

1. Encrypt on box 1, type PIN:

   $x = E_{Km + \text{"PIN"}}(k)$

2. Decrypt on box 2, type data:

   $D_{Km^* + \text{"DATA"}}(x) = D_{Km + \text{"PIN"}}(x) = k$

The system now believes that k is a key to decrypt data, which means that the result will be output (PINs are never output in the clear)

Lessons learned: security APIs

- Complex – 150 commands
- Need to resist to insider frauds
- Hard to design – can go wrong in many ways
- Need more attention

Further reading: Mike Bond, Cambridge University
http://www.cl.cam.ac.uk/users/mkb23/research.html

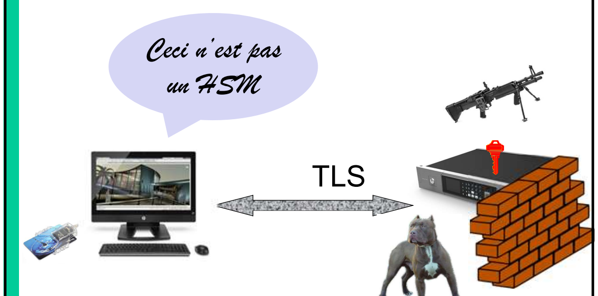
"Efficient padding oracle attacks on cryptographic hardware" (PKCS#11 devices)

[Bardou+ 12] most attacks take less than 100 milliseconds

Device	PKCS#1v1.5		CBC pad	
	token	session	token	session
Aladdin eTokenPro	X	X	X	X
Feitian ePass 2000	OK	OK	N/A	N/A
Feitian ePass 3003	OK	OK	N/A	N/A
Gemalto Cyberflex	X	N/A	N/A	N/A
RSA Securid 800	X	N/A	N/A	N/A
Safenet iKey 2032	X	X	N/A	N/A
SATA dKey	OK	OK	OK	OK
Siemens CardOS	X	X (89 secs)	N/A	N/A

The secure hardware delusion

e.g. August 2011 Diginotar: target Iranian opposition



Key management

- Key establishment protocols
- Key generation
- Key storage
- Key separation (cf. Security APIs)

Key establishment protocols: subtle flaws

- Person-in-the middle attack
  - Lack of protected identifiers
- Reflection attack
- Triangle attack

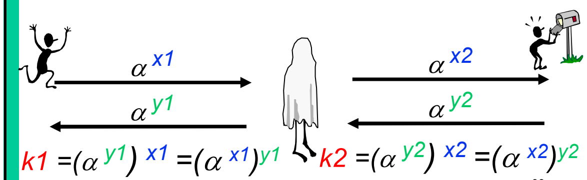
Attack model: Needham and Schroeder [1978]:

We assume that the intruder can interpose a computer in all communication paths, and thus can alter or copy parts of messages, replay messages, or emit false material. While this may seem an extreme view, it is the only safe one when designing authentication protocols.

37

Person-in-the middle attack on Diffie-Hellman

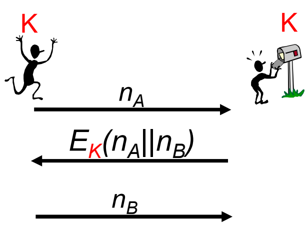
- Eve shares a key *k1* with Alice and a key *k2* with Bob
- Requires *active* attack



38

Entity authentication

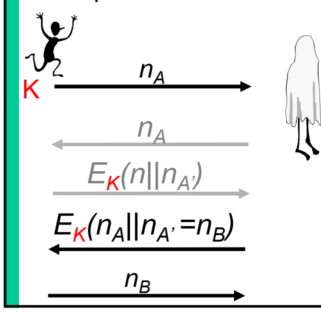
Alice and Bob share a secret K



39

Entity authentication: reflection attack

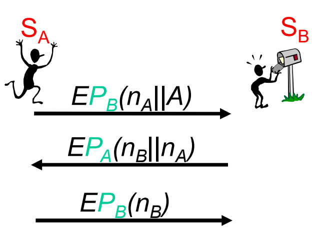
Eve does not know K and wants to impersonate Bob



40

Needham-Schroeder (1978)

Alice and Bob know each other's public key P_A and P_B

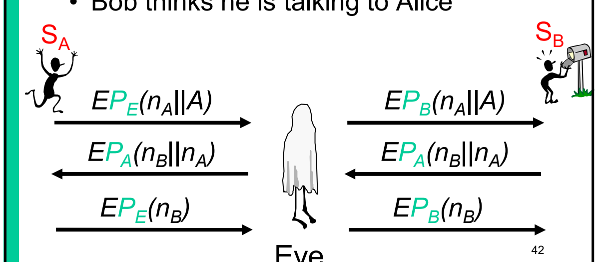


Derive a session key k from $n_A||n_B$

41

Lowe's attack on Needham-Schroeder (1995)

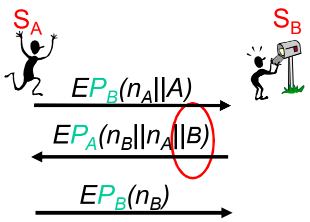
- Alice thinks she is talking to Eve
- Bob thinks he is talking to Alice



42

Lowe's attack on Needham-Schroeder (1995)

- Eve is a legitimate user = insider attack
- Fix the problem by inserting B in message 2



43

Lessons from Needham-Schroeder (1995)

- Prudent engineering practice (Abadi & Needham): include names of principals in all messages
- IKE v2 – plausible deniability: don't include name of correspondent in signed messages: http://www.ietf.org/proceedings/02nov/I-D/draft-ietf-ipsec-soi-features-01.txt

44

Rule #1 of protocol design

Don't!

45

Why is protocol design so hard?

- Understand the security properties offered by existing protocols
- Understand security requirements of novel applications
- Understanding implicit assumptions about the environment underpinning established properties and established security mechanisms

46

And who are Alice and Bob anyway?

- Users?
- Smart cards/USB tokens of the users?
- Computers?
- Programs on a computer?

If Alice and Bob are humans, they are vulnerable to social engineering

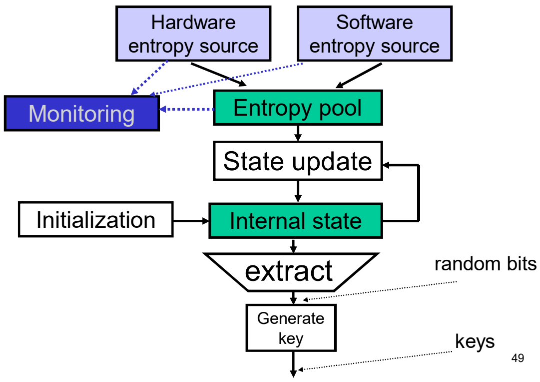
47

Random number generation

- "The generation of random numbers is too important to be left to chance"
- John Von Neumann, 1951: "Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin"
- Used for
  - key generation
  - encryption and digital signatures (randomization)
  - protocols (nonce)

48

Key generation: overview



49

Key generation: hardware entropy sources

- radioactive decay
- reverse biased diode
- free running oscillators
- radio
- audio, video
- hard disk access time (air turbulence)
- manually (dice)
- lava lamps

Risk: physical attacks, failure

50

Key generation: software entropy sources

- system clock
- elapsed time between keystrokes or mouse movements
- content of input/output buffers
- user input
- operating system values (system load, network statistics)
- interrupt timings

Risk: monitoring, predictable

51

Key generation: monitoring

- Statistical tests (NIST FIPS 140)
- typical tests: frequency test, poker test, run's test
- necessary but not sufficient
- 5 lightweight tests to verify correct operation continuously
- stronger statistical testing necessary during design phase, after production and before installation

52

State update

- Keep updating entropy pool and extracting inputs from entropy pool to survive a state compromise
- Combine both entropy pool and existing state with a non-invertible function (e.g., SHA-512, x^2 mod n,…)

53

Output function

- One-way function of the state since for some applications the random numbers become public
- A random string is not the same as a random integer mod p
- A random integer/string is not the same as a random prime

54

What **not** to do

- use rand() provided by programming language or O/S
- restore entropy pool (seed file) from a backup and start right away
- use the list of random numbers from the RAND Corporation
- use numbers from http://www.random.org/
  - **66198 million random bits served since October 1998**
- use digits from π, e, π/e,…
- use linear congruential generators [Knuth]
  - $x_{n+1} = a\, x_n + b \bmod m$

55

RSA moduli

- Generate a 1024-bit RSA key

  Use random bit generation to pick random a integer r in the interval $[2^{512}, 2^{513}-1]$

  If r is even r:=r+1

  Do r:=r+2 until r is prime; output p

  Do r:=r+2 until r is prime; output q

What is the problem?

56

The Infineon Library: RSAlib

[Nemec,Sýs,Švenda,Klinec,Matyáš '17]

- RSA keys: product of two large primes: N = p.q
- How do I generate p and q?
- Pick a random number x and test for primality
- Improvement 1: pick a random odd number x and test
  - Note x = 1 mod 2
- Improvement 2: pick a random odd number x not divisible by 3 and test for primality
  - Note: x = 1 mod 6 or x = 5 mod 6
- Improvement 3: pick a random odd number x not divisible by 3 and 5 and test for primality
  - Note: x = 1,7,11,13 mod 15
- **Idea: control the value of candidates x modulo the product of the first n primes**

The Infineon Library: RSAlib

- RSAlib: generate prime candidates x as follows
  - M_n = product of first n primes
  - $x = k \cdot M_n + (65537^a \bmod M_n)$
- Unfortunately this can be detected easily:

  $N = 65537^a \bmod M_n$
- And M_n was chosen too large so k and a are small and can be recovered easily leading to factorization:
  - 1024-bit keys: < 3 CPU months on a single core
  - 2048-bit keys: 100 CPU-years
- Improvements by 25%: [Bernstein-Lange]

The Infineon Library: RSAlib

- https://crocs.fi.muni.cz/public/papers/rsa%1Fccs17
- Aug. 2016: non-randomness of Infineon keys detected
- Jan. 2017: vulnerability found
- Feb. 2017: Infineon warned
- 16 Oct. 2017: results announced (without details)
- 31 Oct. 2017: paper released
- 3 Nov. 2017: Estonia blocks Infineon keys (more than 750,000 ID cards)
- Other problems: Yubikey, TPMs, TLS, Github,…

RSAlib was certified by BSI based on tests by TÜV Informationstechnik GmbH

What to consider/look at

- **Standardized random number generators: NIST SP800- 90C (but do not use Dual_EC_DRBG)**
- **Modern Intel processors have a built-in RNG (since 2010)**
- Learn from open source examples: ssh, openpgp, linux kernel source (e.g. /dev/random – but slow)
- Yarrow/Fortuna
- ANSI X9.17 (but parameters are marginal)
- Other references:
  - D. Wagner's web resource: http://www.cs.berkeley.edu/~daw/rnd/
  - P. Gutmann, http://researchspace.auckland.ac.nz/handle/2292/2310
  - L. Dorrendorf, Z. Gutterman, Benny Pinkas, Cryptanalysis of the Windows random number generator. ACM CCS 2007, pp. 476-485
  - Z. Gutterman, Benny Pinkas, T. Reinman, Analysis of the Linux random number generator. IEEE Symp. Security and Privacy 2006, pp. 371-385
  - Mario Cornejo, Sylvain Ruhault. Characterization of Real-Life PRNGs under Partial State Corruption. ACM CCS 2014, pp. 1004-1015

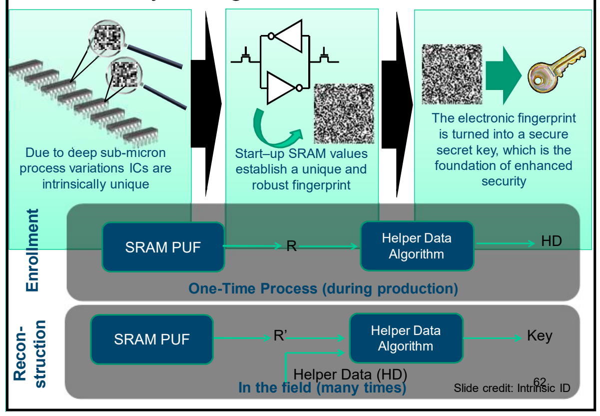
60

How to store keys

- Disk: only if encrypted under another key
  - But where to store this other key?
- Human memory: passwords limited to 48-64 bits and passphrases limited to 64-80 bits
- Removable storage: Floppy, USB token, iButton, PCMCIA card
- Cryptographic co-processor: smart card USB token
- Cryptographic co-processor with secure display and keypad
- Hardware security module
- PUFs: Physical Uncloneable Functions

61

Secure key storage with non-initialized SRAM



62

How not to store keys

[Shamir-van Someren'99] Playing hide and seek with stored keys, Financial Cryptography

63

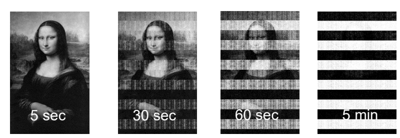
Implementation attacks cold boot attack

Why break cryptography? Go for the key!

Data remanence in DRAMs

Lest We Remember: Cold Boot Attacks on Encryption Keys [Halderman-Schoen-Heninger-Clarkson-Paul-Calandrino-Feldman-Appelbaum-Felten'08]

- Works for AES, RSA,…
- Products: BitLocker, FileVault, TrueCrypt, dm-crypt, loop-AES



Cold boot attacks on keys in memory

(Feb. 2008)

- Key is stored in DRAM when machine is in sleep or hibernation
- Option 1: Reboot from a USB flash drive with O/S and forensic tools (retaining the memory image in DRAM), scan for the encryption keys and extract them.
- Option 2: physically remove the DRAM
  - Cool DRAM using compressed-air canister (-50 C) or liquid nitrogen (-196 C)
- Solution: hardware encryption or 2-factor authentication

65

How to back-up keys

- Backup is essential for decryption keys
- Security of backup is crucial
- Secret sharing: divide a secret over n users so that any subset of t users can reconstruct it

$ 11,000

Destroying keys securely is harder than you think

66

Implementing crypto libraries is hard

Check out this 2017 talk by Quan Nguyen (quannguyen@google.com)

Practical Cryptanalysis of Json Web Token and Galois Counter Mode's Implementations

https://rwc.iacr.org/2017/Slides/nguyen.quan.pdf

67

Implementing digital signatures is hard

- ElGamal
- RSA

68

The risks of ElGamal (1/3)

- ElGamal-type signatures (including DSA, ECDSA)
- public parameters: prime number p, generator g (modulo p operation omitted below)
- private key x, public key $y = g^x$
- signature (r,s)
  - generate temporary private key k and public key $r = g^k$
  - solve s from $h(m) \equiv x\,r + k\,s \bmod (p-1)$
- verification:
  - Signature verification: $1 < r < p$ and $h(m) \equiv y^r\ r^s \bmod p$

The risks of ElGamal (2/3)

- long term keys: $y = g^x$
- short term keys: $r = g^k$
- the value k has to be protected as strongly as the value x
  - Ex. 1: NIST had to redesign the DSA FIPS standard because of a subtle flaw in the way k was generated [Bleichenbacher'01]
  - Ex 2: attack on ElGamal as implemented in GPG [Nguyen'03]

The risks of ElGamal (3/3)

- $y = g^x$
- signature:
  - $r = g^k$
  - $h(m) \equiv x\,r + k\,s \bmod (p-1)$
- what if k would be the same every time?
  - $h(m_1) \equiv x\,r + k\,s \bmod (p-1)$
  - $h(m_2) \equiv x\,r + k\,s \bmod (p-1)$
- 2 linear equations in 2 unknowns: easy to solve: yields the signing key x
- one solution: choose $k = h(m \,||\, x)$

Problematic public keys (1/3)

[Lenstra-Hughes+ Crypto 12]

- 11.7 million openly accessible public keys (TLS/PGP)
- 6.4 million distinct RSA moduli
- rest: ElGamal/DSA (50/50) and 1 ECDSA

- 1.1% of RSA keys occur in >1 certificate

- easy to factor: 0.2% of RSA keys
  - 12,000 keys!
  - 40% have valid certs

[Heninger+ Usenix Sec. 12]

12 million openly accessible public keys (5.8 TLS/6.2 SSH)
23 million hosts (12.8/10.2)

1%: 512-bit RSA keys

- 5.6% of TLS hosts share public keys
- 5.2% default manufacturer keys
- 0.34% have by accident the same key

- easy to factor: 0.5% of TLS hosts and 0.03% of SSH hosts
- DSA key recovery: 1.6% of DSA hosts

Problematic public keys (2/3)

- low entropy during key generation
- RSA keys easy to factor, because they form pairs like: n = p.q and n' = p'.q so gcd(n,n')=q
- DSA keys: reuse of randomness during signing or weak key generation

- why ???
  - embedded systems
    - routers, server management cards, network security devices
  - key generation at first boot

RSA versus DSA
Ron was wrong, Whit is right or vice versa?

Problematic public keys (3/3)

ethical problem: how to report this?

details:

Lenstra, Hughes, Augier, Bos, Kleinjung, Wachter, "Ron was wrong, Whit is right" http://print.iacr.org/2012/064.pdf, *or with as title* "Public keys," Crypto 2012.

Heninger, Durumeric, Wustrow, Halderman, "Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices," Usenix Security 2012, https://www.usenix.org/conference/usenixsecurity12/tech-schedule/technical-sessions

More PRNG flaws

- 1996: Netscape SSL [Goldberg-Wagner]
- 2008: Debian SSL [Bello]
- 15 Aug. 2013: Android Java and OpenSSL PRNG flaw led to theft of Bitcoins
- Sept. 2013: Bullrun and DUAL_EC_DRBG

16 Sept. 2013 Factoring RSA keys from certified smart cards: Coppersmith in the wild
[Bernstein-Chang-Cheng-Chou-Heninger-Lange-van Someren'13] IACR Cryptology ePrint Archive 2013: 599

184 keys from Taiwan Citizen Digital Certificate cards
card + OS: EAL 4+; FIPS 140-2 Level 2

How to sign with RSA?

- public key: (n,e)
- private key: d
- s = t^d mod n = $t^{1/e}$ mod n

- But
  - message M is often larger than modulus n
  - RSA(x*y) = RSA(x)*RSA(y)
  - RSA(0) = 0, RSA(1) = 1,…

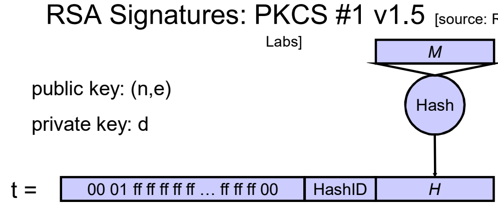
- Solution: hash and add redundancy
  - PKCS #1
  - RSA-PSS

76

RSA Signatures: PKCS #1 v1.5 [source: RSA Labs]

public key: (n,e)

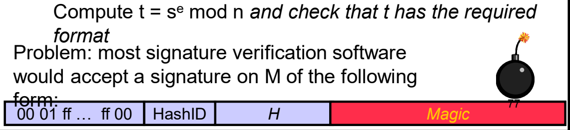
private key: d



Generation of RSA signature on M: s = t^d mod n = $t^{1/e}$ mod n

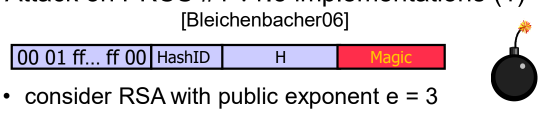
Verification of RSA signature s on M
Compute t = s^e mod n *and check that t has the required format*

Problem: most signature verification software would accept a signature on M of the following form:



Attack on PKCS #1 v1.5 implementations (1)

[Bleichenbacher06]



- consider RSA with public exponent e = 3
- for any hash value H, it is easy to compute a string "Magic" such that the above string is a perfect cube of 3072 bits
  - example of a perfect cube 1728 = 12^3
- consequence:
  - one can sign any message (H) without knowing the private key
  - this signature works for any public key that is longer than 3072 bits
- vulnerable: OpenSSL, Mozilla NSS, GnuTLS

78

Fix of Bleichenbacher's attack

- Write proper verification code (but the signer cannot know which code the verifier will use)
- Use a public exponent that is at least 32 bits
- Upgrade – finally – to RSA-PSS

79

Conclusion

- Implementing cryptography requires a high level of cryptographic expertise
- Application developers should become specialists
  - "A specialist is someone who knows when to call an expert" [Peter Landrock]

80