Putting pieces together

already have: symmetric channel ← encryption, MAC, key derivation function (seq.nos, master key → various keys)

Assume: everybody has his private key & everybody else's public key

A B

0. exchange nonces
1. A generates master secret M randomly
2. A encrypts M using B's pub.key → B
3. both sign steps 0–2 & exchange signatures

] or: Diffie-Hellman exchange
This gives us forward security

Practical Issues

- don't know how to design secure SW : the worst enemy is complexity
- —"— implement —"—
  - using C (Rust???)
  - not using C
  - debugging by testing wrong!
    can help: coverage analysis, fuzz testing
  - need "proof", reviews, ...
- too many dependencies (SW & HW)
- real attacker more powerful than theoretical one

① remote attacker – timing → side channel
↳ padding oracle
– interpetation depends on context
- file type → file.exe.jpg
HTTP Content-type
→ multi-type (JPG & Java bytecode)
- ambiguities: strings: chars 00 | len chars

UTF-8 "a" 1-byte
↳ 2-byte, 3-byte, 4-byte } forbidden

JSON – no fixed standard
XML <!-- .... -->

Example: body | sign
s ← MAC(key, body)
if sign ≠ s: reject
else: accept
char sign[32]
memcmp(p1, p2, size)

256 · 32 ~ 8k

Fix: (sign[0] ⊕ s[0]) ∨ (sign[1] ⊕ s[1]) ∨ ... = 0 ⇔ s = sign

<!-- --> <!-- -->

– DoS attack: flood by messages

② in the same room:
- measure power consumption
- sound
- thermal
- radio emissions
- influence computation
  - power spikes
  - elmag. spikes

$(x^e) \bmod n$
$x^2 \cdot x$

③ physical access to the computer
- install a spy bug (HW device)
- extract memory modules

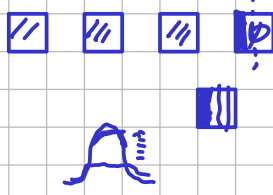
④ program on the same machine (Javascript)
- HW side effects → cache side-effects can be used to extract AES key from a different program
- CPU bugs (Meltdown, Spectre)

Storing of secrets:

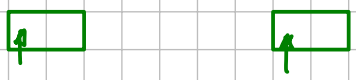
- keys in memory – can be stored to the disk ← swapping, core dump
  - can be sent over network

(uninit. memory)

↳ precautions: ① erase secrets when not needed
② mlock
③ disable core dumps



↳ split secrets



- data on disk – secure erase? → overwrite ← a file: movable parts, defragmentation
  - ↳ SSD → ???
  - ↳ "secure" erase function – internally encrypted by AES with random key
    - flash mem. with the key

    problem: ① quality of randomness
    ② manufacturers cheating
  - a disk: relocation of bad blocks, bits leave echoes

Dedicated secure HW

- chip cards
- cryptographic module

• issue of trust

Keeping state

keep nonces unique
keep state of RNGs

- reboot .... store the state
- crash → store new state during boot
- restoring from backup → handle manually!
- first boot – want to distribute diff. init state in diff. device

Solutions to practical security:

- everything can be broken
- analogy with physical security
- goals: slow down attacks → can catch attackers
  make attacks visible → log, monitors...
  cost of attack >> cost of secrets