COMP7370 Advanced Computer and Network Security

Cold Boot Attacks on Encryption Keys (2)

Topics: Imaging Residual Memory
Key reconstruction
Identifying keys in memory

Topic 1: Imaging Residual Memory

1. Memory Wiping
   - Can wiping memory at POST (Power-On Self Test) solve the memory attacking problem?
   - Attackers can transfer memory data to a machine that does not wipe its memory on boot.

2. Imaging tools
   - Challenging: booting a system can overwrite some portions of memory
   - Load a full OS? Why not? Very destructive
   - Solution: tiny special purpose program (trivial amounts of RAM)
   - PXE Network boot

   Laptop ----------> (PXE) target PC
   UDP(enthernet)
   <---- memory image ---- (30 sec. for 1GB RAM data)

   - Boot from USB drives

   Flash device(10KB SYSLINUX bootloader) ----------> (USB) target PC
   <----- memory image ----

   - EFI boot – Extensible firmware interface instead of BIOS
   - iPods <---- imaging tools

3. Imaging attacks
   - Approach 1: Simple reboot: change BIOS; boot the imaging tool
     - Warm boot? OS restart: software will have an opportunity to wipe sensitive data prior to shutdown
     - Cold boot: briefly remove and restore power
   - Approach 2: Transferring DRAM modules.
     - Physical access to target machines
     - Cool DRAM using “canned air” dusters

Topic 2: Key reconstruction

1. Motivation:
   - Why recover keys? Bit errors (rate 5% to 50%)
   - Symmetric and private keys.

- Brute-force search: Time consuming
- Observation: many encryption systems precompute key schedules and keep them in memory.

Key -------> key schedules; subkey for each round (for block ciphers)
<---------- how to find code word (i.e., key schedules)

- Modeling decay:
  - all bits decay to the same ground state
  - probability /delta_0 flipping from 1 to 0 Note: very small 0.1%
  - probability /delta_1 flipping from 0 to 1

2. Reconstruct DES keys (error-correction; consider a specific type of keys)
   - 56-bit key -- > 16 48-bit subkeys
   - Voting:

   Original Key 1st, 2nd 3rd …, ith,…. 56th

   Each bit (i.e., ith bit) from the original key is repeated in about 14 of the 16 subkeys.

   Subkey 1 1-1, 1-2 (ith), 1-3, … 1-48
   2 2-1(ith), 2-2, 3-3, … 3-48
   …
   16 16-1, 16-2, 16-3 (ith), … 16-48

   For ith bit in the original key: 1-2, 2-1, …..16-3
   1 0 1

   If /delta_0 = /delta_1, then let's vote: ith bit is 0 if more than n/2 of the recovered bits are 0, and 1 otherwise.

   If /delta_0 != /delta_1, then ith bit is 0 if more than nr of the recovered bits are 0

   (see slides)

$$r = \frac{\log(1-\delta_0) - \log\delta_1}{\log(1-\delta_0) + \log(1-\delta_1) - \log\delta_1 - \log\delta_0}.$$

e.g.,
/delta_0 = 0.1
/delta_1 = 0.001 r = 0.75

- Result: error rate = 50%; 98% probability to reconstruct DES key
- Suggestions for key reconstructions:
  - What cryptosystem?
  - How does the cryptosystem work?

3. Reconstruct AES keys
   - Original 128-bit key -> 11 (4-word) round keys

(see slides)

1 bit flip in original key -> many flips in round keys

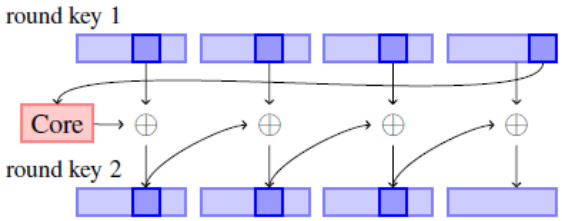


Figure 6: In the 128-bit AES key schedule, three bytes of each round key are entirely determined by four bytes of the preceding round key.

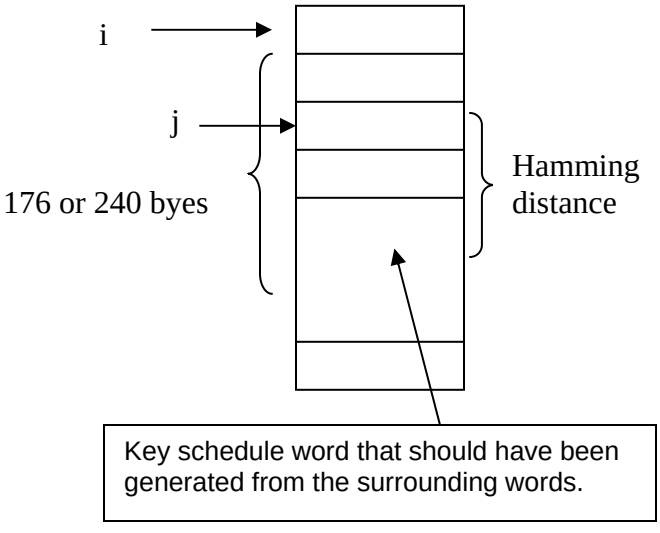
- Step 1: A guess - value for each slice of 7 bytes. Why guess? Decrease total likelihood
- Step 2: calculate expanded key schedules

Expended key schedules (from guessed key) vs. recovered key schedules (in target RAM)

- Step 3: Is likelihood of expanded key schedules (from the guessed key) decaying to recovered key schedule high?

Topic 3: Identifying keys in memory

- **Question:** How will you identify keys in RAM?
  - Statistical tests
  - Locate program data structures
- Identify AES keys (see slide)
  - Input: a memory image
  - Output: a list of likely keys
  - Basic idea: (1) key schedules rather than original keys
    (2) recover keys from their key schedules



1. Iterate through each byte of memory. Treat the following block of 176 or 240 bytes of memory as an AES key schedule.
2. For each word in the potential key schedule, calculate the Hamming distance from that word to the key schedule word that should have been generated from the surrounding words.
3. If the total number of bits violating the constraints on a correct AES key schedule is sufficiently small, output the key.