Key-hiding on the ARM platform

Alexander Nilsson, Marcus Andersson,
Stefan Axelsson

Blekinge Institute of Technology
Sweden

Background

- Disk encryption helps (or hinders depending...)
  - Yes OK let's call it bad here
- But cold boot attacks, memory dumping and the like reveal the key
- Idea: Don't store the key in memory!

Related work

- FrozenCache
  - Make the key stick in cache
    - Didn't work – cache semantics are hard and not well defined
- AESSE
  - Stick keys in SSE registers
    - Which you then can't use
- *TRESOR* by Johannes Gotzfried and Tilo Müller
  - Keys in debug registers!
    - With AES NI insn on x86 it's even ***faster*** than key expansion in main memory
  - also LoopAmnesia – using perfcount registers
  - TreVisor putting it in the virtual machine unavailable to the OS)

Related work

- Android important
  - Frost – tool that helps with cold boot attacks
- However TRESOR tech doesn't work on Android
  - ARM not x86 – no AES NI...
  - Enter *ARMORED*, by the TRESOR team implemented same idea on Android/ARM using NEON
    - Own AES impl. to not touch memory (key expansion)
    - About half speed compared to original implementation of AES with memory key schedule expansion

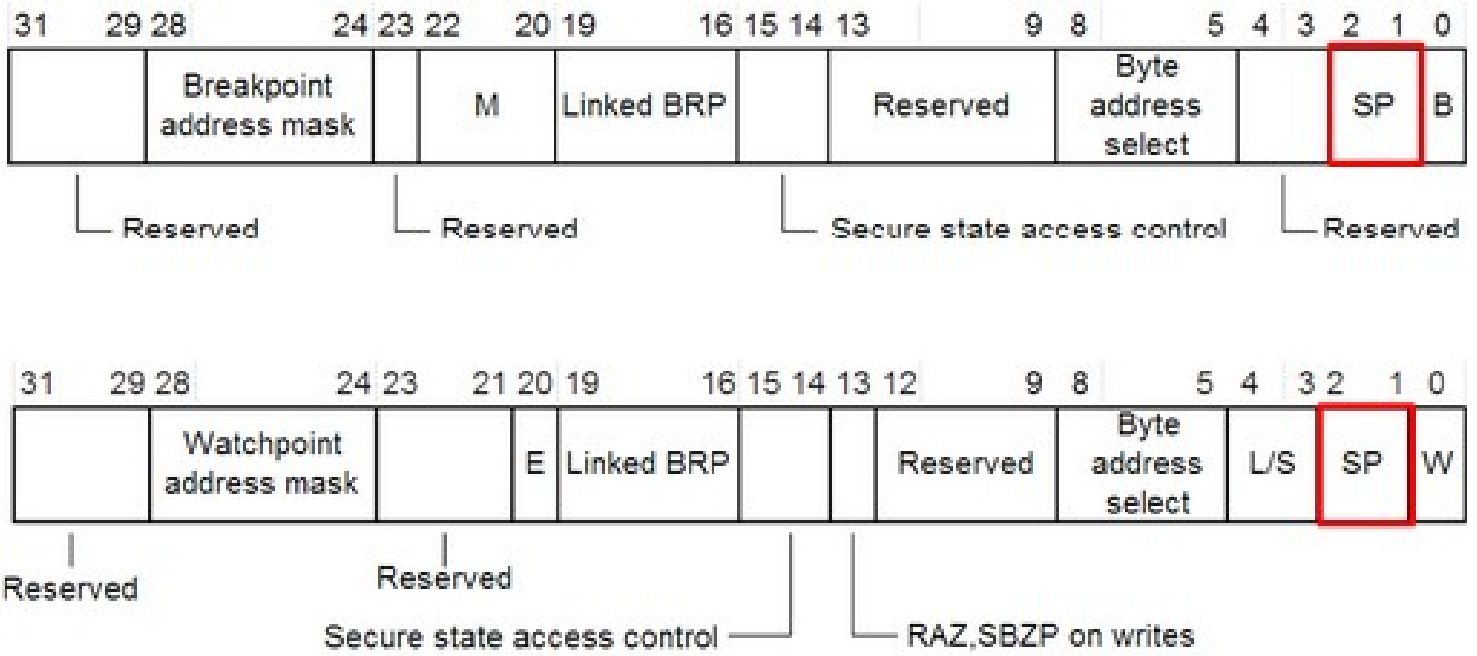
"[Regarding ARM breakpoint registers] But unlike the debug registers in 64-bit x86 CPUs, they are too small to hold AES-256 keys. (Note that debug registers can be written into RAM due to context switching as well, but we specifically prohibit that by patching respective kernel routines.)

On ARM, the least significant two bits of each 32-bit break- and watchpoint register are necessarily zero due to the memory alignment in ARM. Since instructions are consistently 32-bit wide, they are always located at 4-byte aligned addresses. Hence, the least significant two bits are omitted for setting break- and watchpoints because they must be zero anyway. *As a consequence, these bits are not available as key storage* [Our emphasis]. For the sake of convenience, we divided the key-sequence into 16-bit chunks; specifically, we use four breakpoint and four watchpoint registers, giving us a total of $8.16 = 128$ bits as key storage. This is enough to accommodate AES-128, but not enough to accommodate AES-256. However, since Android's encryption feature is based on AES-128, this does not pose a problem.

In future releases we could store more than 16 bits per register, and if we find additional break- and watchpoint registers, we could accommodate AES-256."

Two bits short

- This piqued our curiosity
  - Is there another way to find space for two bits (per register)?
- Yes!



Results

- Implemented a 256 bit key storage extension test syscall
  - And a test application that used it
- We also have to make sure that on-one else uses the debug registers
  - i.e. adb/gdb – That works since they can use more break/watch-points than we have registers
- Also checked that the values didn't end up in memory and that no other part of the kernel than *ptrace* uses these
- Didn't test performance, BUT only a couple of insn per 32 bit key at key setup so it *shouldn't* have any noticeable effect
  - Famous last words...

Conclusions

- So if you need/want 256-bit AES on Android/ARM it can be done with debug registers despite the last two being unavailable
- If you're doing memory forensics then there's other stuff you ***might*** need to dump as well i.e. CPU debug registers (and others?)
  - And cold boot variations won't work since that clears CPU registers

Conclusions

- That's our ***two bit*** contribution to forensics science...