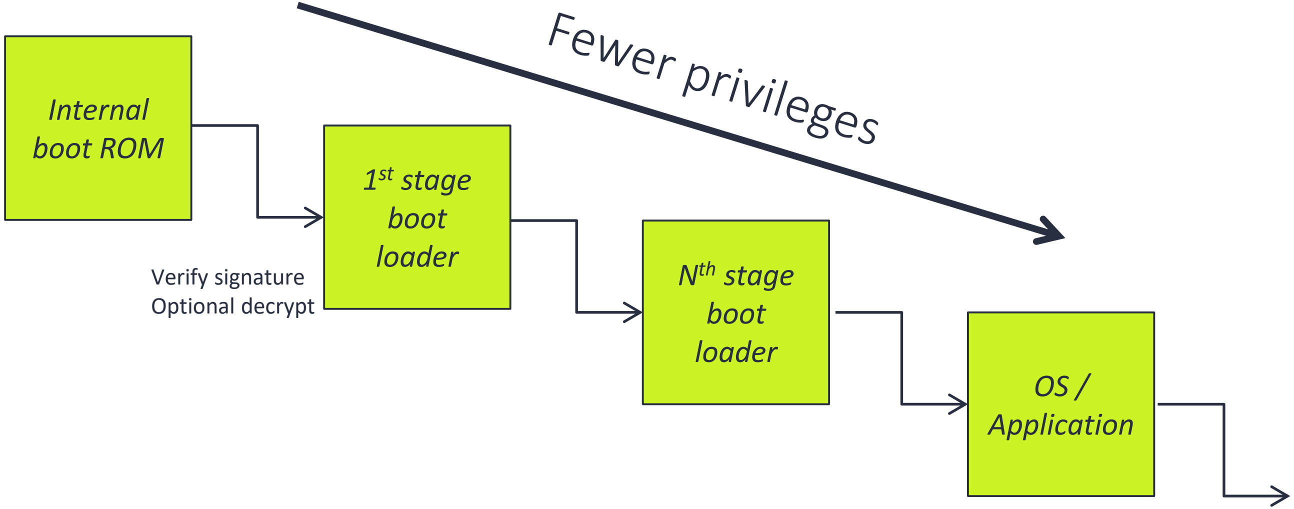
Top 10
Secure Boot mistakes

Jasper van Woudenberg
(Job de Haas)

jasper@riscure.com
@jzvw

Secure Boot Theory



Internal boot ROM

1st stage boot loader

Nth stage boot loader

OS / Application

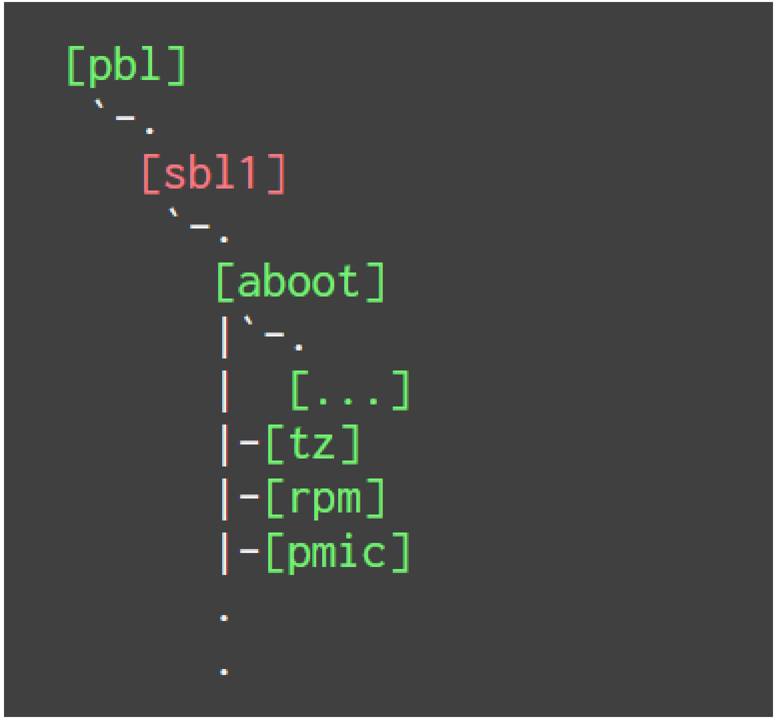
Fewer privileges

Verify signature
Optional decrypt

Secure Boot practice

Besides chain of trust…

- Memory / peripheral lockdown
- Configuration reading / parsing
- Manufacturing modes
- Debug and in-field servicing
- Power modes (resume from s3 vs cold boot)
- Firmware upgrades
- Constraints: many use cases, bootup time



```
[pbl]
 `-.
   [sbl1]
    `-.
      [aboot]
      |`-.
      |  [...]
      |-[tz]
      |-[rpm]
      |-[pmic]
      .
      .
```

https://alephsecurity.com/vulns/aleph-2017026

10. Verification mistakes

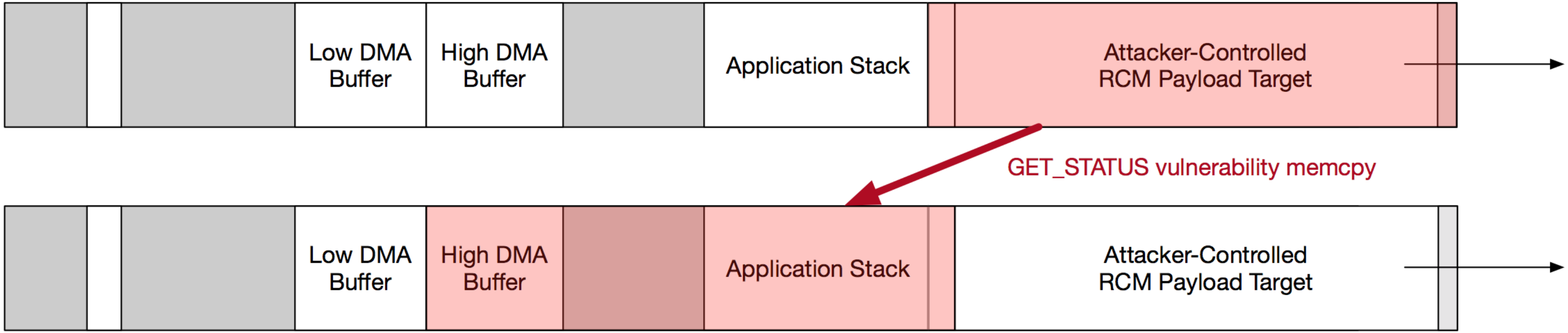
- If anything is left unsigned, what can it be used for?
- Problems start when length, loading address etc. become flexible
- Failure: Start interpreting before verification

Examples:

- iPhone 3GS, Samsung Galaxy S4, OnePlus 2
- http://theiphonewiki.com/wiki/0x24000_Segment_Overflow
- http://blog.azimuthsecurity.com/2013/05/exploiting-samsung-galaxy-s4-secure-boot.html
- https://alephsecurity.com/vulns/aleph-2017026

Mitigation:

- Sign **EVERYTHING**
- Do not use any data without/before checking authenticity (eg. headers, pointers, addresses)
- *If you really can't sign it, check very thoroughly



Low DMA Buffer

High DMA Buffer

Application Stack

Attacker-Controlled RCM Payload Target

GET_STATUS vulnerability memcpy

Low DMA Buffer

High DMA Buffer

Application Stack

Attacker-Controlled RCM Payload Target

https://github.com/Qyriad/fusee-launcher/blob/master/report/fusee_gelee.md

9. Firmware Upgrade / Recovery flaws

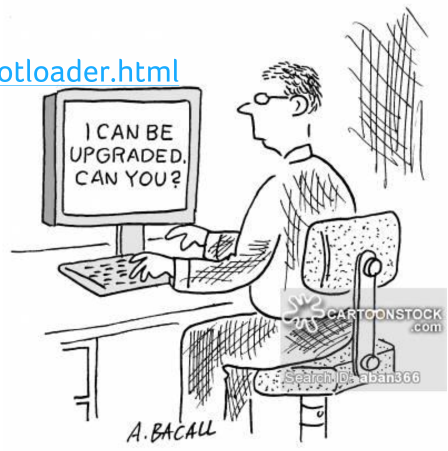
- Important feature to mitigate flaws in the field
- The mechanism itself must be hardened! Chunking difficult
- Updated firmware should follow same rules as installed firmware

Examples:

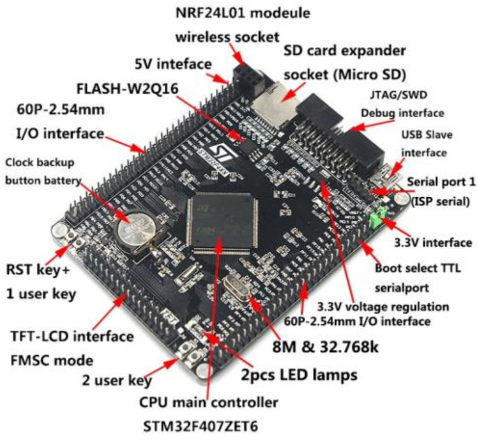
- Switch hack https://github.com/Qyriad/fusee-launcher/blob/master/report/fusee_gelee.md
- Samsung / Qualcomm ODIN overflows https://fredericb.info/2017/07/sve-2016-7930-multiple-buffer-overflows-in-samsung-galaxy-bootloader.html

Mitigation:

- Limit the functionality! Avoid partial updates, signing individual blocks
- Implement anti-rollback: can negate fixes



8. Logical bugs / Driver weaknesses



- Boot code has several functions:
  - Boot from different media including file system (USB, SD, MMC, UART, NOR, NAND, SPI)
  - Ensure fall back and restore mechanisms
  - Perform parsing of firmware image formats, certificates
- Input parsing problems can lead to overflows, integer sign problems, etc.

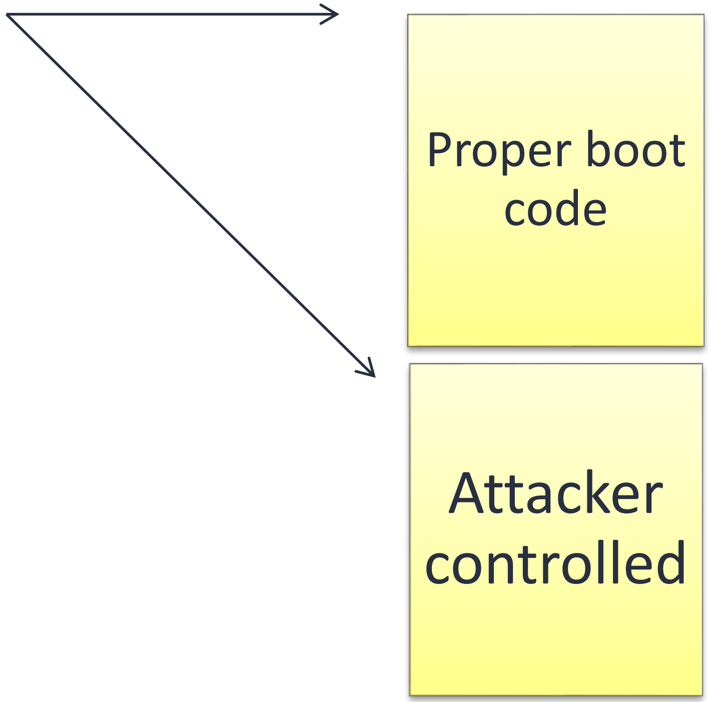
Examples:

- iPhone exploits http://theiphonewiki.com/wiki/Usb_control_msg(0xA1,_1)_Exploit, Limera1n_Exploit, SHA-1_Image_Segment_Overflow
- Nintendo 3DS: https://lab.dsst.io/slides/33c3/slides/8344.pdf
- Nintendo Switch: https://fail0verflow.com/blog/2018/shofel2/

Mitigation:

- Code review, fuzzing, etc
- Limiting functionality to bare minimum, reuse well-tested code

At the ~~push~~ glitch of a button



Proper boot code

Attacker controlled

7. TOCTOU race conditions

- Between verification and use, data can be modified
- An attacker can access data externally or multiple components have access

Examples:

- Typical case: boot from external NOR flash
  - Integrity check is performed on content in external storage
  - Code is changed and only then read or directly executed from the external storage
- Nokia BB5 unlock by Dejan Kaljevic (2007)
  http://forum.gsmhosting.com/vbb/f299/bb5-sp-unlocking-theory-443418/
- BIOS examples with SMM
  http://www.c7zero.info/stuff/AttackingAndDefendingBIOS-RECon2015.pdf

Mitigation:

- Prevent any access between check and use
- Move to internal memory, stop/block other engines

Timing attack with Infectus board

Brute forcing 16*128 = 2048 values takes **about 2 hrs**

6. Timing attacks

- Allow guessing much faster than brute-force
- Typical on compare function (HMAC, service password)

Examples:

- Hash calculated with symmetric key is stored with firmware. Boot calculates same and compares (20 bytes)
- memcmp() has different timing if byte is correct or wrong
- Xbox 360: http://beta.ivc.no/wiki/index.php/Xbox_360_Timing_Attack
- Cristofaro's talk from yesterday

Mitigation:

- Time-constant comparisons
- Side channel leakage review
  https://www.riscure.com/publication/secure-application-programming-presence-side-channel-attacks/

Slot machine EMP jamming

5. Fault injection

- FI is an effective way to subvert execution flow (even with perfect logical code!)
- Examples of faulting sensitive coding:
  - using infinite loops
  - single comparisons (signature verification)
- Seldom a persistent attack; effective as stepping stone

Examples:

- XBOX 360: reset glitch attack: http://www.free60.org/Reset_Glitch_Hack
- PS4: https://fail0verflow.com/blog/2018/ps4-syscon/
- Nintendo Switch: https://media.ccc.de/v/34c3-8941-console_security_-_switch
- https://www.blackhat.com/docs/eu-16/materials/eu-16-Timmers-Bypassing-Secure-Boot-Using-Fault-Injection.pdf
- PS Vita; https://yifan.lu/2019/02/22/attacking-hardware-aes-with-dfa/

Mitigation:

- Side channel leakage review / defensive coding

4. State errors

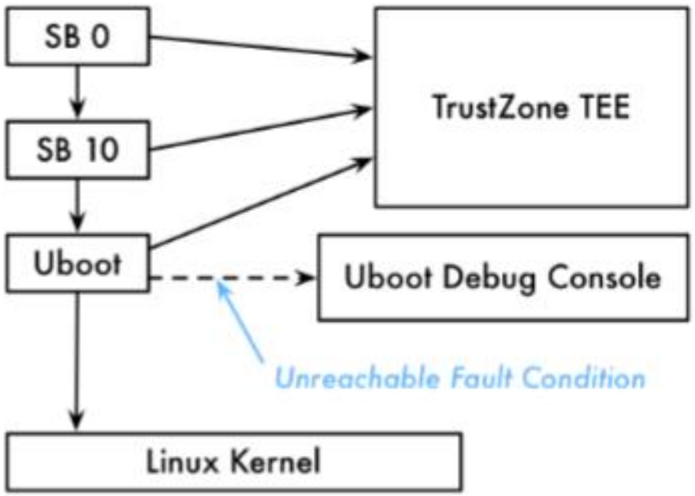
- Where is state stored? How can a state sequence be influenced?
- Suspend/resume example:
  State is stored insecurely, which allows a local exploit to subvert the boot process on resume
  → maximum privilege escalation
- Lifecycle state!

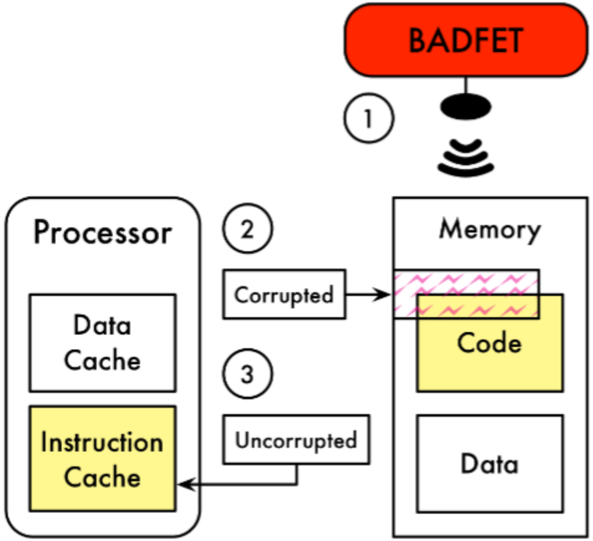
- http://i.blackhat.com/asia-19/Thu-March-28/bh-asia-Seunghun-Finally-I-Can-Sleep-Tonight-Catching-Sleep-Mode-Vulnerabilities-of-the-TPM-with-the-Napper.pdf

Mitigation:

- Cryptographically sign & verify
- Analyze all state variables in the boot sequence
  (exception handling, suspend/resume, storage, integrity)
- Consider both logical and fault injection threats

BADFET





```
⇒      help mw
mw - memory write (fill)
Usage: mw [.b, .w, .l] address value [count]
```

https://www.usenix.org/system/files/conference/woot17/woot17-paper-cui.pdf

3. Debug JTAG/service functionality

- Large topic!
  - Interfaces: JTAG, UART, Proprietary, ...
  - Force debug boot mode
  - Service backdoor / passwords

- UART is almost as pervasive as JTAG
- Many devices leave some form of access for debug/service purposes

3. Debug JTAG/service functionality

- Everyone understands that backdoors can be bad
- More often: “It is bad, but not for my application”, then later the requirements change
- Checking a HW fuse ≠ properly hardware protected

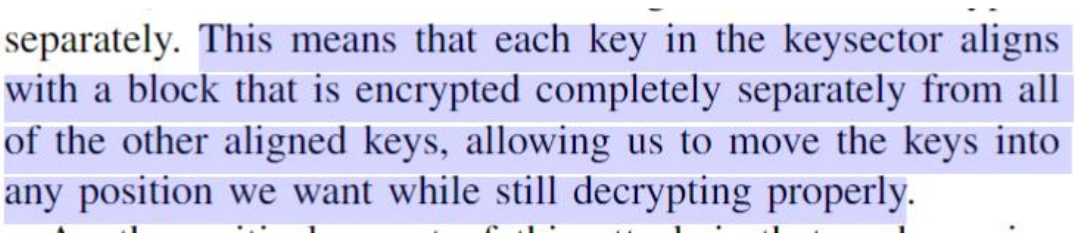
Examples:

- Nook boot lock exploit (2012)
  http://www.xda-developers.com/android/patch-this-barnes-and-noble-nook-tablet-hardware-protection-compromised/
- Many car tuning ECU cables/software, ‘Magic’ authentication allows firmware mods, changing car keys, mileage

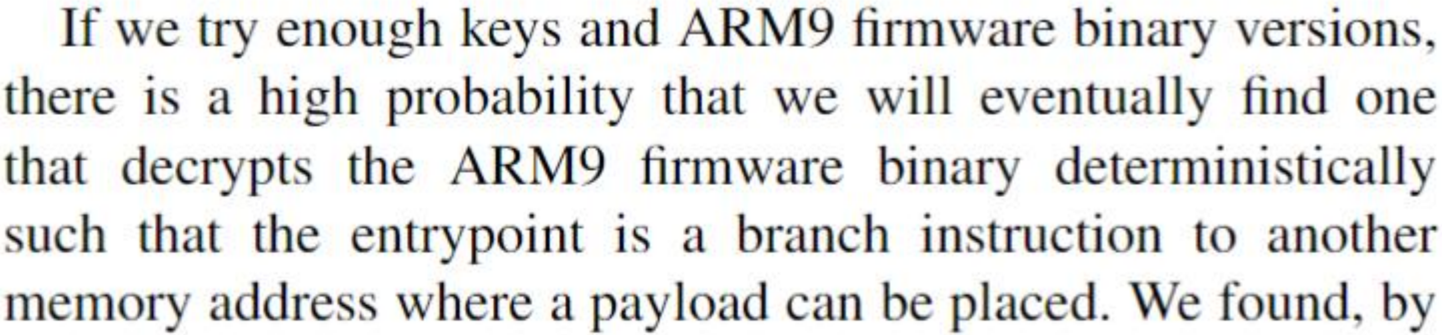
Mitigation:

- Secure chips can disable or lock JTAG
- At least use some device unique authentication
- Better to have ‘debug upgrade’ than debug built-in

3DS



separately. This means that each key in the keysector aligns with a block that is encrypted completely separately from all of the other aligned keys, allowing us to move the keys into any position we want while still decrypting properly.



If we try enough keys and ARM9 firmware binary versions, there is a high probability that we will eventually find one that decrypts the ARM9 firmware binary deterministically such that the entrypoint is a branch instruction to another memory address where a payload can be placed. We found, by

https://arxiv.org/pdf/1802.00092.pdf

2. Key management

- Checking key usage
- Signing development boot loaders with production keys
- Crypto sanitization :
  - After the boot code uses cryptographic engines they may become available for generic code
  - State can be reused, registers may be read
  - Attack: create more signatures, decrypt/encrypt more code

Examples

Samsung Galaxy S3 versus Exynos dev board boot loader
3DS clearing issue in FW 8.1.0: https://arxiv.org/pdf/1802.00092.pdf

Mitigation:

- Understand the value of all key material and signatures. Act accordingly.
- Clear registers of crypto engines and any other memory used for storing sensitive data
- https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r4.pdf

1. Wrong use of crypto

- Know and understand the weaknesses of the algorithms and protocols used
- Decryption ≠ Authentication

Examples

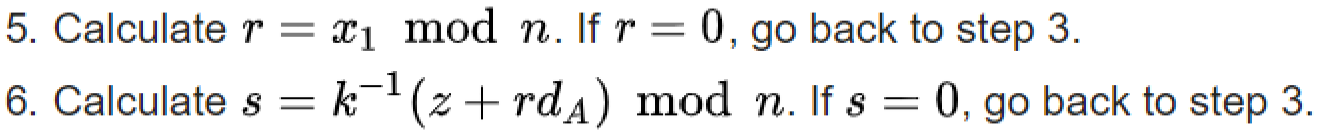
- Nokia DCT4 2nd stage loader u_2nd.fia could be patched to load unencrypted 3rd stage http://www.dejankaljevic.org/download/dct4_rd.zip 2002/2005
- RSA small exponent signature verification
- Amlogic forgot HMAC: https://fredericb.info/2016/10/amlogic-s905-soc-bypassing-not-so.html
- 3DS Key shuffling attack

Mitigation:

- Cryptographic review

1. Wrong use of crypto

- **ECDSA** is a signature scheme
- **Input**: curve parameters, private key (**dA**), message (**m**), nonce (**k**)
- **Output:** signature **r,s**

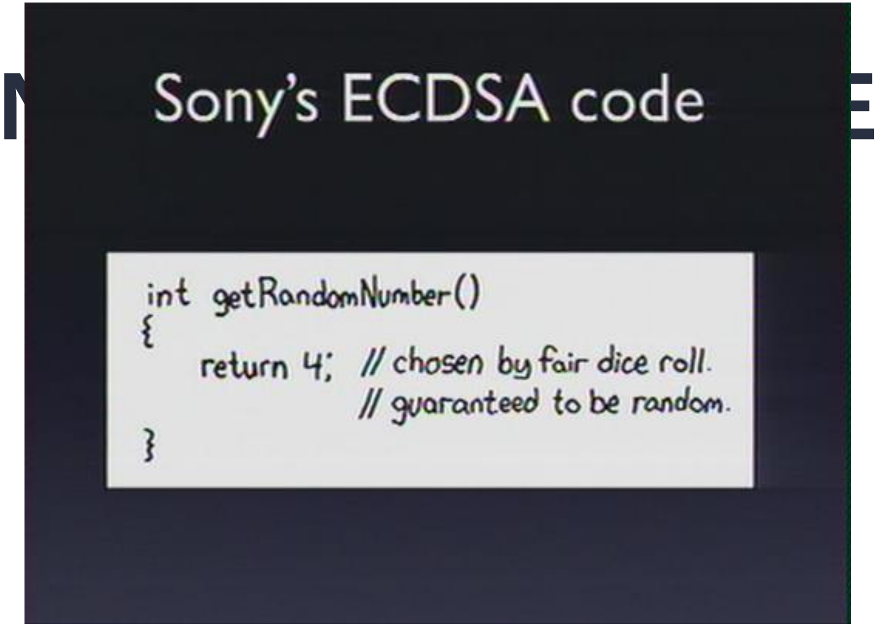


5. Calculate $r = x_1 \mod n$. If $r = 0$, go back to step 3.
6. Calculate $s = k^{-1}(z + rd_A) \mod n$. If $s = 0$, go back to step 3.

- Nonce reuse: dA = (m1*s2 - m2*s1) / (r*(s1 - s2)) mod n

1. Wrong use of crypto

PS3 Epic Fail



Sony's ECDSA code

```
int getRandomNumber()
{
    return 4;  // chosen by fair dice roll.
               // guaranteed to be random.
}
```

Source: http://events.ccc.de/congress/2010
Console Hacking 2010

Now what?

- (Securely) booting a system is a complex operation
- In the field patching of boot components is practically very hard – impossible

- **Security researchers:** learn hw attacks, explore attack surface
- **Developers**: secure dev practices, limit attack surface, test!
- **Integrators:** ask developers/third party to provide assurance on security

Riscure B.V.
Frontier Building, Delftechpark 49
2628 XJ Delft
The Netherlands
Phone: +31 15 251 40 90

www.riscure.com

Riscure North America
550 Kearny St., Suite 330
San Francisco, CA 94108 USA
Phone: +1 650 646 99 79

inforequest@riscure.com

Riscure China
Room 2030-31, No. 989, Changle Road, Shanghai 200031
China
Phone: +86 21 5117 5435

inforcn@riscure.com

riscure

Challenge your security