Console Hacking 2008

Victor Muñoz
vmunoz@ingenieria-inversa.cl

¿Seguridad en Consolas de Videojuego?

- ¿Por que?
  - Evitar las copias ilegales
  - Impedir software no licenciado
  - Cheats
  - Forzar su uso exclusivamente para juegos

¿Seguridad en Consolas de Videojuego?

- Evolucion
  - La no seguridad (Atari)
  - Primer intento serio: Nintendo CIC aka 10NES aka Lockout Chip
    - no pudo prevenir el software no licenciado
    - Atari crea el Rabbit (1993)
  - Llegada del CD, protecciones anticopia
  - Llegada del juego en linea: tolerar cheats ya no es una opcion
    - Codigo firmado
    - Partidas salvadas firmadas

Gamecube

- Discos Opticos
  - no estandar, derivado del DVD
  - ejecutables en claro y no firmados
  - datos en claro y no firmados
  - proteccion anti-copia
- IPL no firmada, pero encriptada (PRNG)
- Codigo ejecutable no firmado por buses faciles de interceptar/parchar
- Partidas salvadas en claro (solo checksum)

Playstation 2

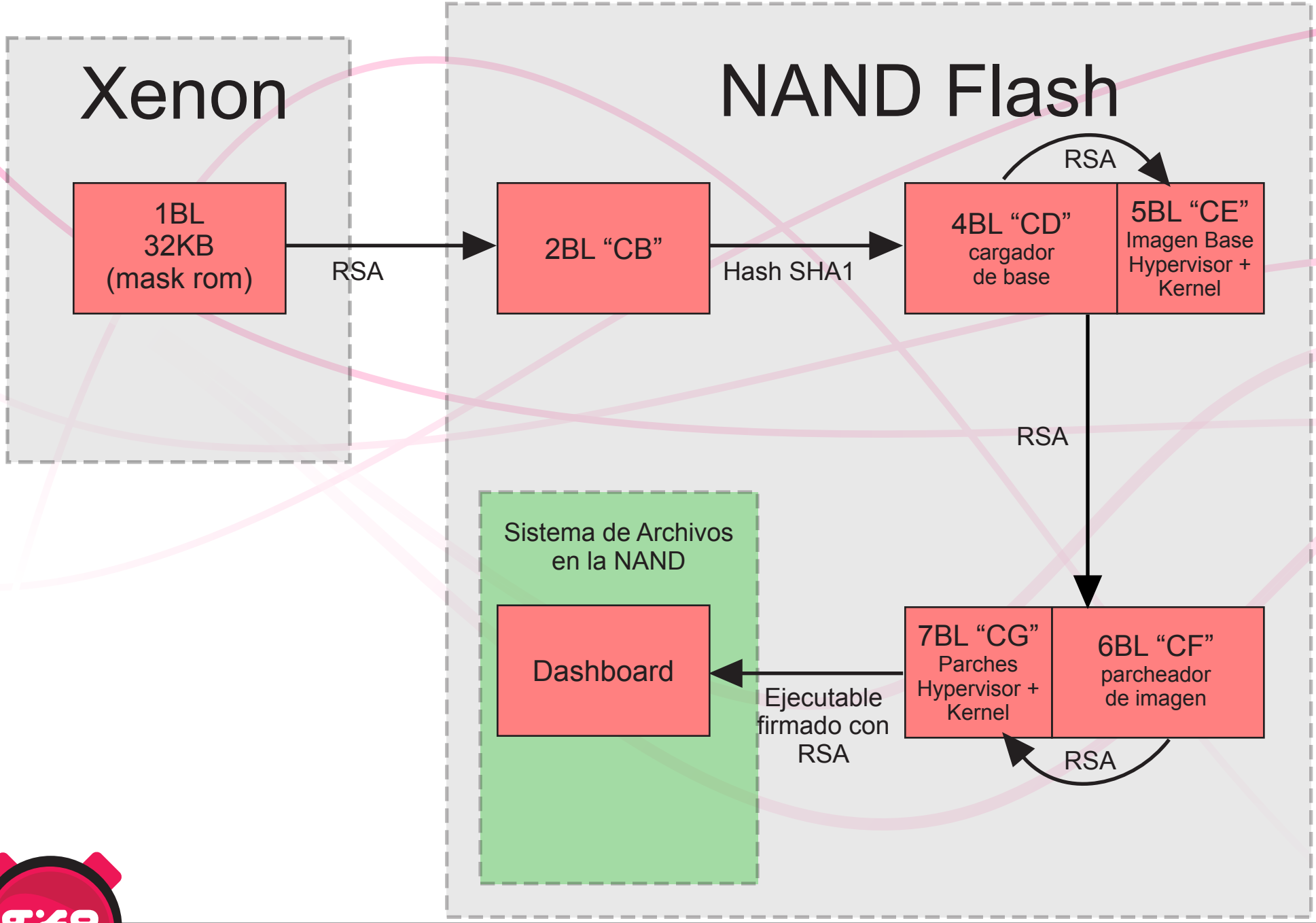
- Discos Opticos
  - CD y DVD estandar
  - ejecutables en claro y no firmados
  - datos en claro y no firmados
  - proteccion anti-copia
- BIOS en claro y no firmada
- Codigo ejecutable no firmado por buses faciles de interceptar/parchar
- Partidas salvadas en claro (solo checksum)
- Tiene infraestructura para ejecutar codigo firmado, pero su utilizacion es nula
- SCE distribuye oficialmente un kit de Linux, pero corre bajo un RTE limitado

- Discos Opticos
  - DVD estandar, pero con particion escondida
  - ejecutables encritados y firmados
  - datos en claro y no firmados
  - proteccion anti-copia
- BIOS encriptada y firmada
- Partidas salvadas encriptadas y firmadas

Seguridad XBOX360

- Cadena de Confianza
- Primera instruccion desde dentro de la CPU
- Seguridad a nivel de la silicona
  - SRAM y ROM dentro de la CPU
  - memoria hasheada
  - memoria encriptada
  - eFuses
- Hipervisor
- W xor X (bye bye buffer overflows)
- Keyvault y claves unicas por consola
- Perdio la confianza en el lector de DVD

Cadena de Confianza



Memoria Hasheada

- Previene ataques DMA
- Cada 128 bytes (1 linea L2) requiere 16 bytes
- Hashes almacenados en la SRAM de la CPU (64KB)
- Solo podemos hashear 1MB, asi que lo usamos para el hipervisor

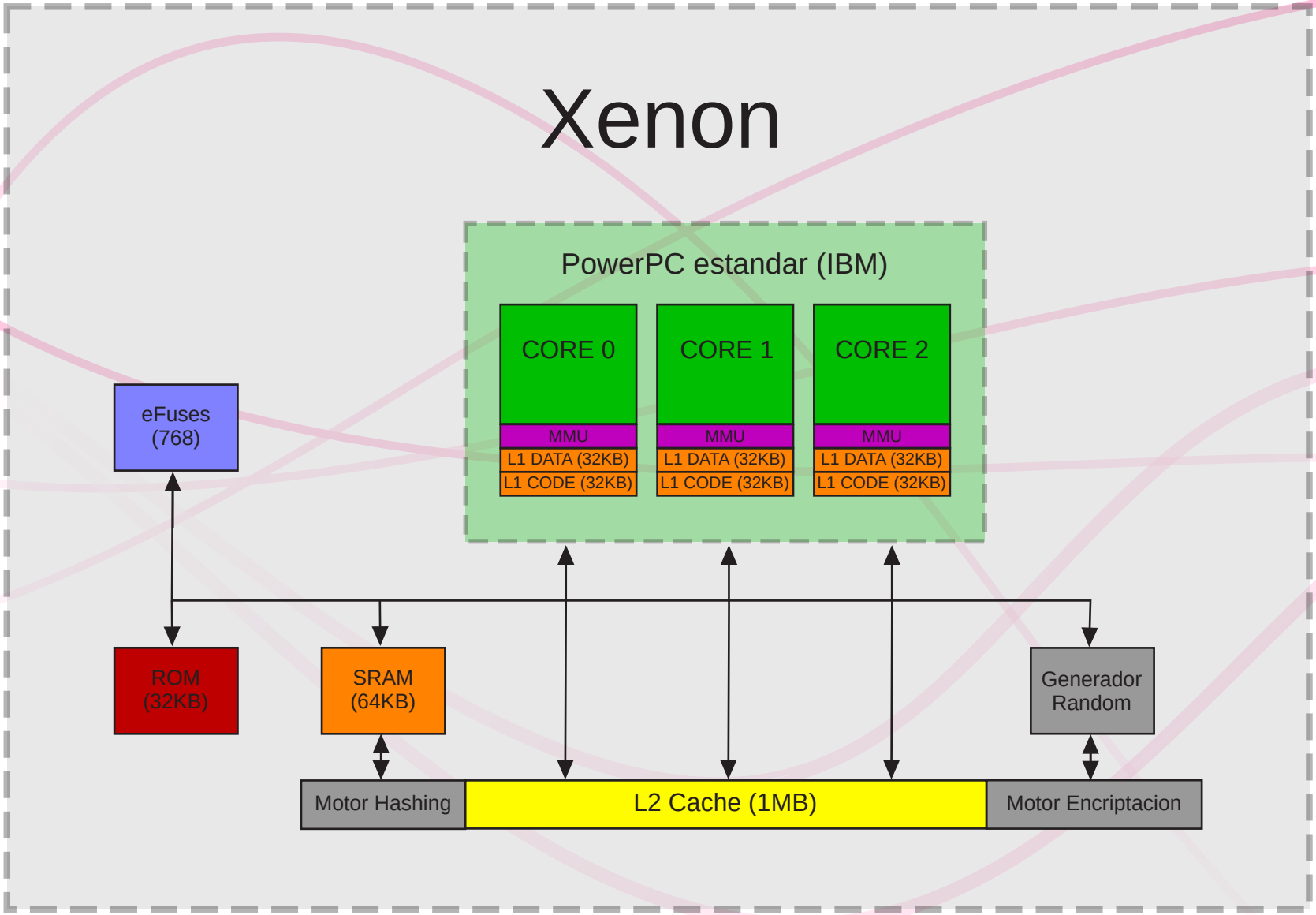
Memoria Encriptada

- Todas las paginas de codigo encriptadas
- No tan bueno como el hashing
- pero, no requiere memoria adicional
- Previene el volcado de codigo o datos sensible via DMA

eFuses

- Fusibles dentro de la CPU (aka memoria OTP)
- Creados por IBM
- Baratos!! hay 768 de ellos
- Implementan una clave unica por CPU
  - clave del Keyvault
  - amarra la consola con el lector DVD
- Deshabilitan funcionalidad de fabricacion y testeo (ej. JTAG)
- Implementan revocacion de software

Arquitectura de la CPU



Hipervisor

- Chequea autenticidad del codigo
- Implementa W xor X
- Impide exploits en el stack o heap
- El codigo de usuario (juego) no puede cargar codigo

Hipervisor y CPU

- el Hipervisor se ejecuta en modo real
- 00000100_00000000 quiere decir memoria hasheada y encriptada
- la arquitectura agrega el valor de HRMOR (00000100_00000000) cuando el MSB no esta seteado (X00000000_00000000)

Exploit Hipervisor

- Acceso a la memoria fisica usando shaders
- Los contextos (hilos) estan en memoria no hasheada/encriptada
- Entonces en un frame del stack:
  - Se pueden cambiar los registros arbitrariamente
  - Tambien la direccion de retorno

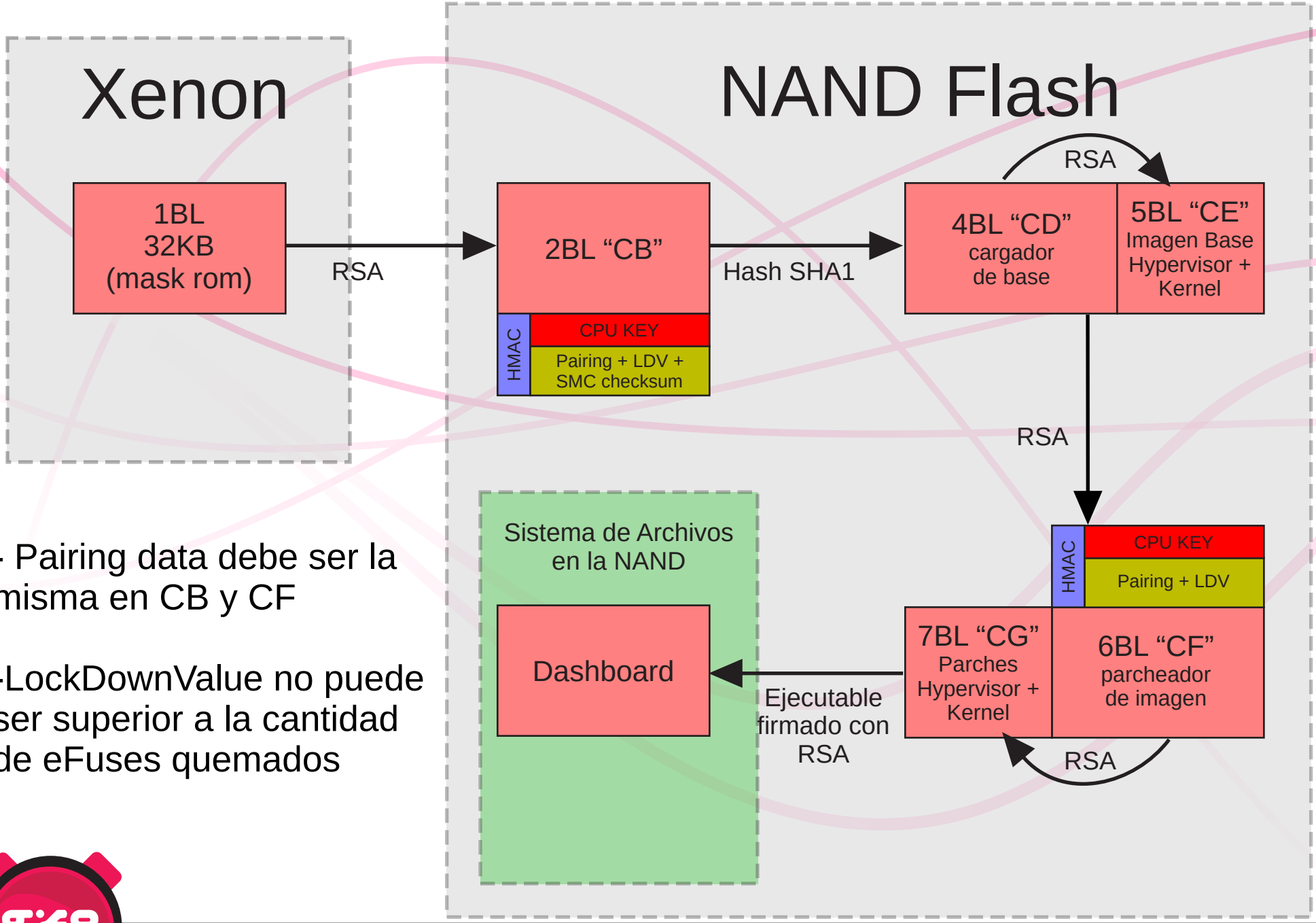
Hipervisor: manejo de syscalls

```
ROM:000013D8 # ---------------------------------------------------------------
ROM:000013D8   cmplwi  %r0, 0x61        # (word)syscall_nr > 0x61?
ROM:000013DC   bc      6, lt, loc_D7C   # Branch Conditional
ROM:000013E0   mtocrf  cr4, %r4
ROM:000013E4   mflr    %sp              # Move from link register
ROM:000013E8   std     %r4, 0x28(%r13) # Store Double Word
ROM:000013EC   std     %sp, 0x20(%r13) # Store Double Word
ROM:000013F0   rldicr  %sp, %r0, 2,61  # (dword)salto=((dword)syscall_nr*4)
ROM:000013F4   lwz     %r4, 0x1F68(%sp) # (dword)salto+=syscalls_base
ROM:000013F8   mtlr    %r4              # Move to link register
ROM:000013FC   ld      %r4, 0x48(%r13) # Load Double Word
ROM:00001400   mfspr   %sp, 0x131       # Move from Special Purpose Register
ROM:00001404   addi    %sp, %sp, 0x1F00 # Add Immediate
ROM:00001408   stdu    %rtoc, -8(%sp)   # Store Double Word with Update
ROM:0000140C   li      %rtoc, 2         # Load Immediate
ROM:00001410   rldicr  %rtoc, %rtoc, 32,31 # Rotate Left Double Word Immediate
ROM:00001414   blrl                     # goto salto;
```

Exploit Hipervisor

- numero de sycall XX introducido por DMA al registro
- 20000000_000000XX
- se transforma en 80000000_ (000000XX) *4
- MSB set → HRMOR no se suma
- entonces podemos saltar del hipervisor a codigo no encriptado/hasheado

Downgrade posible?



- Pairing data debe ser la misma en CB y CF

-LockDownValue no puede ser superior a la cantidad de eFuses quemados

memcmp vulnerable en 2BL

```
RAM:000061A0 :                                       # CODE XREF: sub_4DA8+3Cp
RAM:000061A0                 cmpwi   %r5, 0           # Compare Word Immediate
RAM:000061A4                 mtctr   %r5              # Move to count register
RAM:000061A8                 beq     loc_61D0         # Branch if equal
RAM:000061AC                 lbz     %r6, 0(%r4)      # Load Byte and Zero
RAM:000061B0                 lbz     %r5, 0(%r3)      # Load Byte and Zero
RAM:000061B4                 b       loc_61C0         # Branch
RAM:000061B8 # ---------------------------------------------------------------
RAM:000061B8
RAM:000061B8 loc_61B8:                               # CODE XREF: memcmp+24j
RAM:000061B8                 lbzu    %r6, 1(%r4)      # r6=*(ptr2++);
RAM:000061BC                 lbzu    %r5, 1(%r3)      # r5=*(ptr1++);
RAM:000061C0
RAM:000061C0 loc_61C0:                               # CODE XREF: memcmp+14j
RAM:000061C0                 cmpw    %r5, %r6         # Compare Word
RAM:000061C4                 bdnzt   eq, loc_61B8     # if(r5==r6) goto loc_61B8;
RAM:000061C8                 subf    %r3, %r6, %r5    # else return r6-r5;
RAM:000061CC                 blr                      # Branch unconditionally
RAM:000061D0 # ---------------------------------------------------------------
RAM:000061D0
RAM:000061D0 loc_61D0:                               # CODE XREF: memcmp+8j
RAM:000061D0                 li      %r3, 0           # return 0;
RAM:000061D4                 blr                      # Branch unconditionally
```

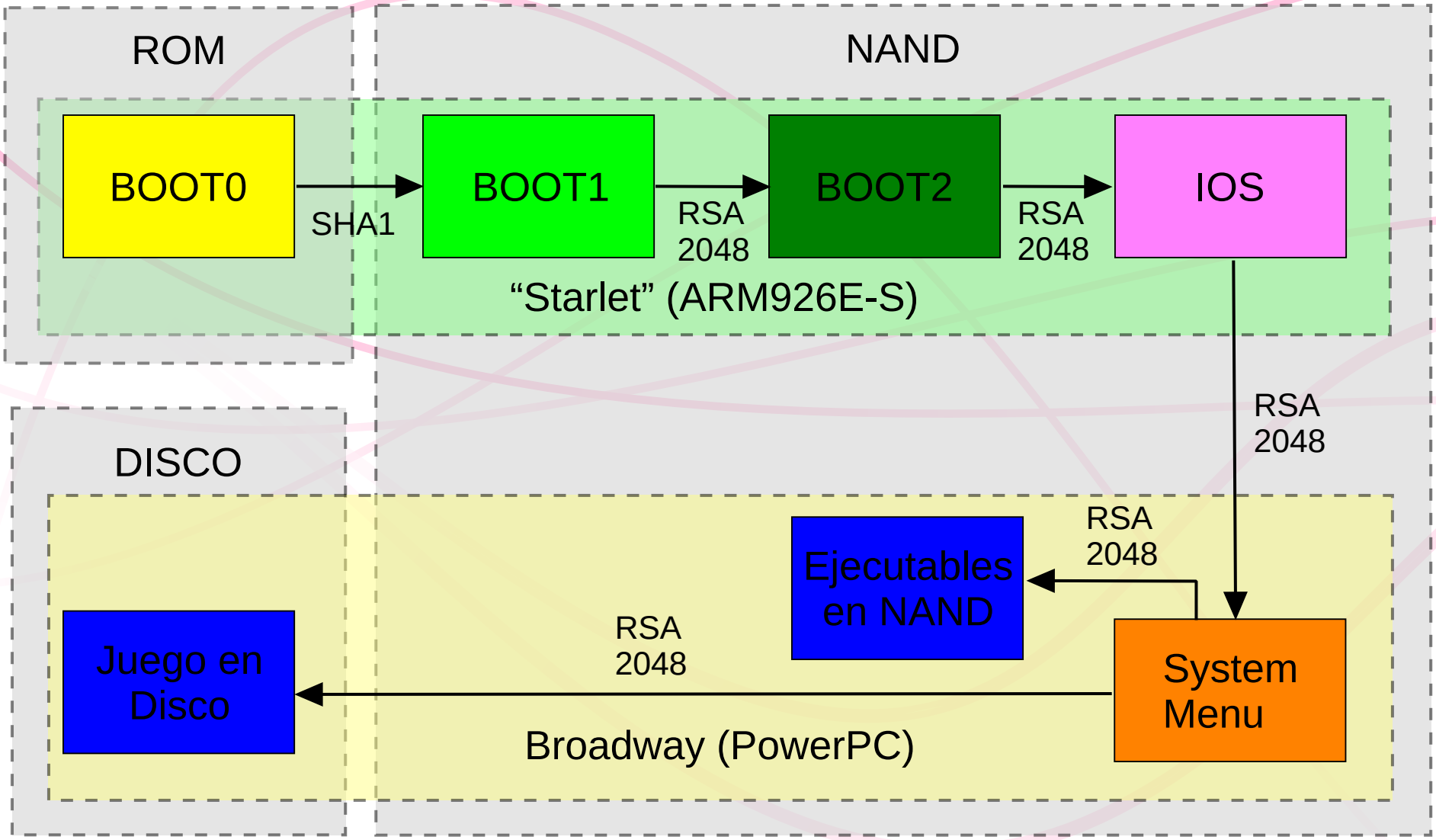
Timing Attack al bloque CB

- Permite hacer downgrade
- Prueba desde 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  hasta FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
- pero no es 2^{128}
- es solo $2^8 \times 16$

Seguridad Wii

- Cadena de Confianza
- Primera instruccion desde dentro de la CPU
- Procesador adicional encargado de la seguridad y E/S
- Seguridad a nivel de la silicona
  - SHA1 y AES por hardware
  - SRAM y ROM dentro de la CPU
  - Memoria OTP dentro de la CPU
- Retrocompatibilidad directamente por hardware, segura
- Keystore y claves unicas por consola
- Partidas salvadas encriptadas y firmadas (ECC)
- Contenidos de los discos opticos totalmente encriptado y firmado

Wii: Cadena de Confianza



Chequeo de firmas: OK?

```
      ADDS    R5, R5, R6          ; Rd = Op1 + Op2
      ADDS    R0, R5, #0          ; Rd = Op1 + Op2
      SUBS    R0, #20             ; Rd = Op1 - Op2
      LDR     R1, [SP,#0xA44+SHA1_signature] ; Load from Memory
      MOVS    R2, #20             ; Rd = Op2
      LDR     R3, =(+1) ; Load from Memory
      BLX     R3                  ; Branch with Link and Exchange (register indirect
      CMP     R0, #0              ; Set cond. codes on Op1 - Op2
      BEQ     firma_OK           ; if(!memcmp(SHA1_cert, SHA1_signature, 20))
goto firma_OK;
      MOVS    R0, #7              ; Rd = Op2
```

Chequeo de firmas: memcmp?

```
memcmp                                  ; CODE XREF: compara_20_bytes+8Ej
                                        ; sub_FFFF2600+98p ...
                PUSH    {R4,R5,LR}      ; memcmp(u8 *ptr1, u8 *ptr2, int cnt)
                ADDS    R4, R0, #0      ; Rd = Op1 + Op2
                .....
loop_compare                            ; CODE XREF: memcmp+32j
                CMP     R1, #0          ; Set cond. codes on Op1 - Op2
                BEQ     ret_diferencia  ; if(cnt==0) goto ret_diferencia;
                CMP     R0, #0          ; Set cond. codes on Op1 - Op2
                BEQ     ret_diferencia  ; if(*ptr1==0) goto ret_diferencia;
                ADDS    R4, #1          ; ptr1++;
                ADDS    R5, #1          ; ptr2++;
                SUBS    R1, #1          ; cnt--;
                BCC     ret_diferencia  ; Branch
                LDRB    R0, [R4]        ; Load from Memory
                LDRB    R3, [R5]        ; Load from Memory
                CMP     R0, R3          ; Set cond. codes on Op1 - Op2
                BEQ     loop_compare    ; if(*ptr1==*ptr2) goto loop_compare;
ret_diferencia                          ; CODE XREF: memcmp+10j memcmp+1Aj ...
                LDRB    R2, [R4]        ; Load from Memory
                LDRB    R3, [R5]        ; Load from Memory
                SUBS    R0, R2, R3      ; Rd = Op1 - Op2
loc_FFFF698E                            ; CODE XREF: memcmp+Cj
                POP     {R4,R5}         ; Pop registers
```

No es memcmp... es strcmp... WTF!

```
strcmp                                  ; CODE XREF: compara_20_bytes+8Ej
                                        ; sub_FFFF2600+98p ...
                PUSH    {R4,R5,LR}      ; strcmp(u8 *ptr1, u8 *ptr2, int cnt)
                ADDS    R4, R0, #0      ; Rd = Op1 + Op2
                .....
loop_compare                            ; CODE XREF: strcmp+32j
                CMP     R1, #0          ; Set cond. codes on Op1 - Op2
                BEQ     ret_diferencia  ; if(cnt==0) goto ret_diferencia;
                CMP     R0, #0          ; Set cond. codes on Op1 - Op2
                BEQ     ret_diferencia  ; if(*ptr1==0) goto ret_diferencia;
                ADDS    R4, #1          ; ptr1++;
                ADDS    R5, #1          ; ptr2++;
                SUBS    R1, #1          ; cnt--;
                BCC     ret_diferencia  ; Branch
                LDRB    R0, [R4]        ; Load from Memory
                LDRB    R3, [R5]        ; Load from Memory
                CMP     R0, R3          ; Set cond. codes on Op1 - Op2
                BEQ     loop_compare    ; if(*ptr1==*ptr2) goto loop_compare;
ret_diferencia                          ; CODE XREF: strcmp+10j strcmp+1Aj ...
                LDRB    R2, [R4]        ; Load from Memory
                LDRB    R3, [R5]        ; Load from Memory
                SUBS    R0, R2, R3      ; Rd = Op1 - Op2
loc_FFFF698E                            ; CODE XREF: strcmp+Cj
                POP     {R4,R5}         ; Pop registers
```

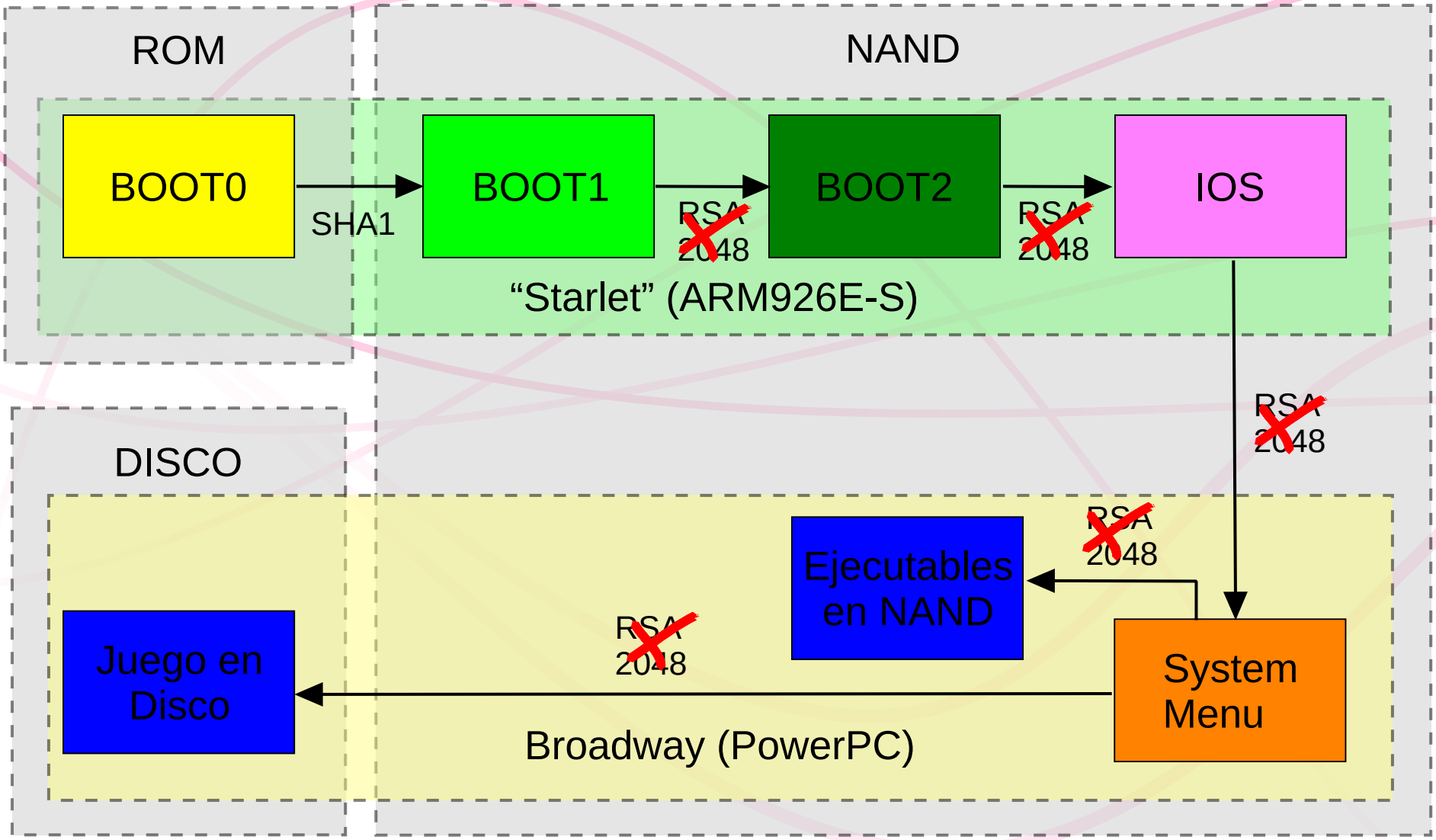
Firmas Truchas

```
      ADDS    R5, R5, R6          ; Rd = Op1 + Op2
      ADDS    R0, R5, #0          ; Rd = Op1 + Op2
      SUBS    R0, #20             ; Rd = Op1 - Op2
      LDR     R1, [SP,#0xA44+SHA1_signature] ; Load from Memory
      MOVS    R2, #20             ; Rd = Op2
      LDR     R3, =(+1) ; Load from Memory
      BLX     R3                  ; Branch with Link and Exchange (register indirect
      CMP     R0, #0              ; Set cond. codes on Op1 - Op2
      BEQ     firma_OK           ; if(!strcmp(SHA1_cert, SHA1_signature, 20))
goto firma_OK;
      MOVS    R0, #7              ; Rd = Op2
```

Firmas Truchas

- `00 43 3F 44 FC 43 77 BA 73 A0 32 43 73 32 C3 D4 56 33 DC 56`
- `00 67 FC DD 44 66 FC 45 21 AA 43 23 43 66 AA 66 44 DC 43 66`
- coinciden bajo este bug!

Firmas Truchas



¿Preguntas?

vmunoz@ingenieria-inversa.cl