riscure

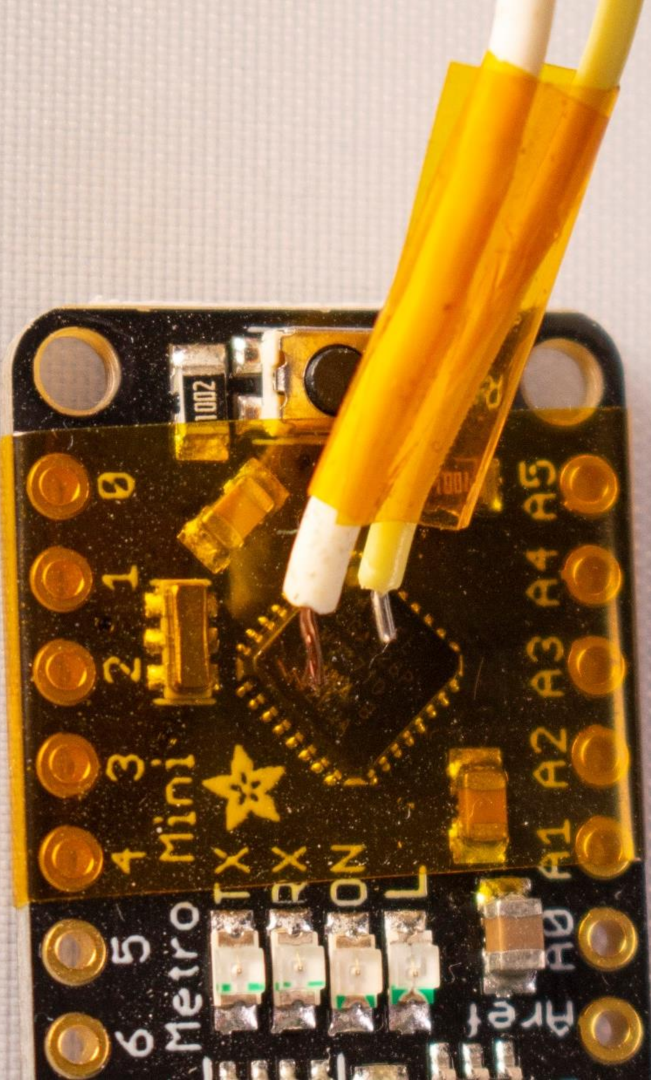
driving your security forward

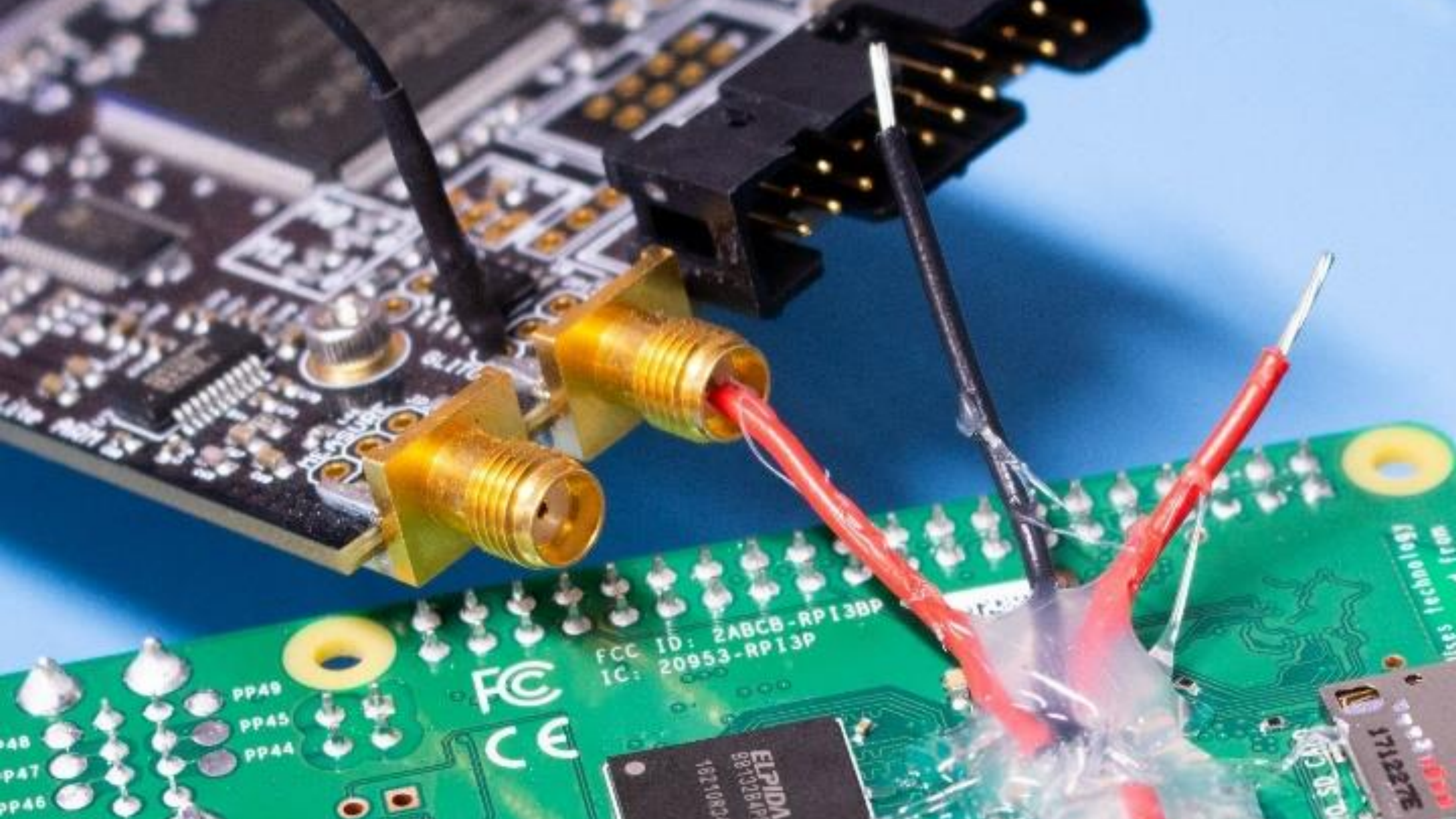
FLIPPIN' FAKE FLOPS FOR FUN, FAME AND FORTUNE

JASPER VAN WOUDENBERG
@JZVW

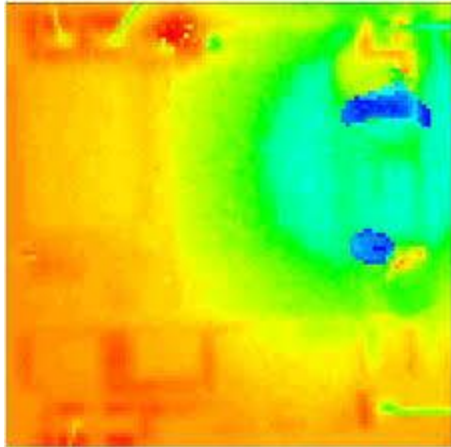
A TALE OF FAULT INJECTORS

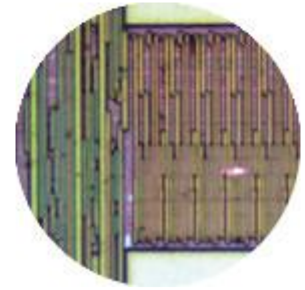
public

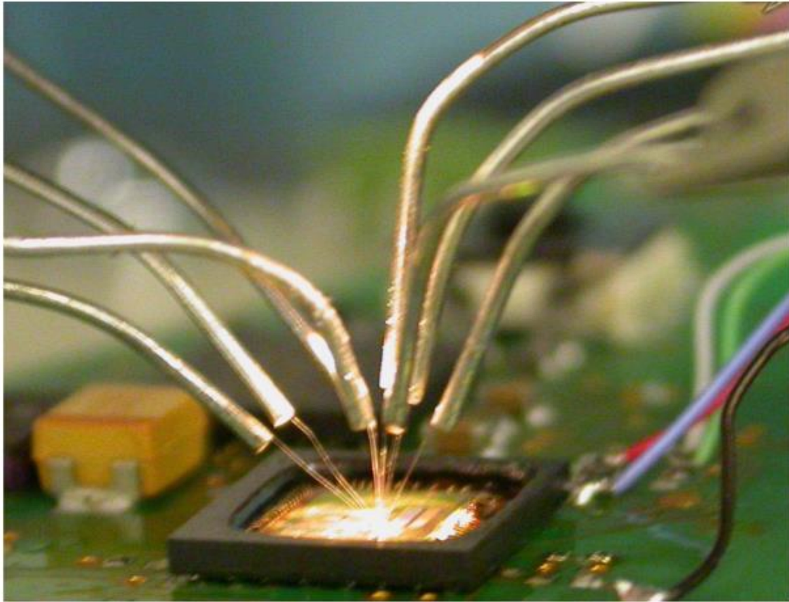

Metro Mini
TX
RX
ON
L
A5
A4
A3
A2
A1
A0


FCC ID: 2ABCB-RPI3BP
IC: 20953-RPI3P
PP49
PP45
PP44
PP48
PP47
PP46
ELPIDA

riscure



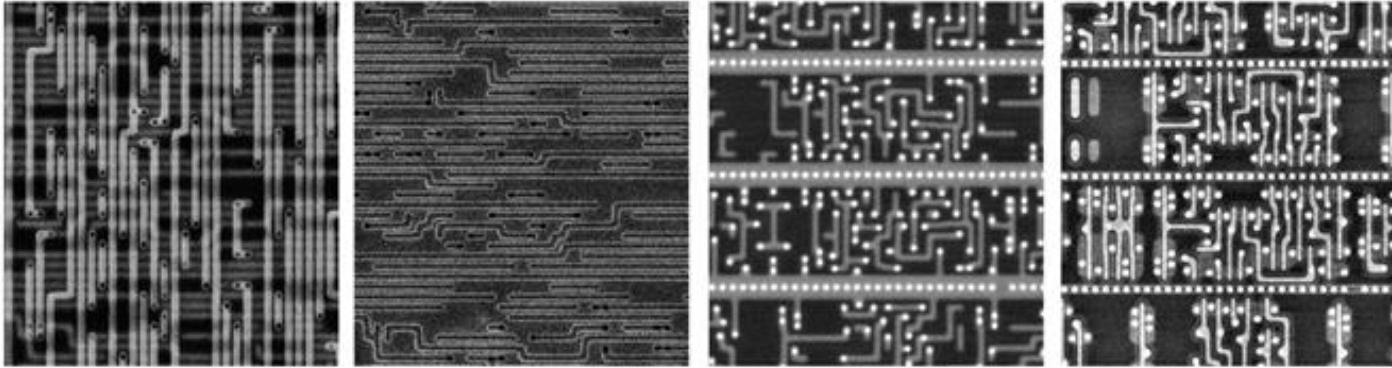




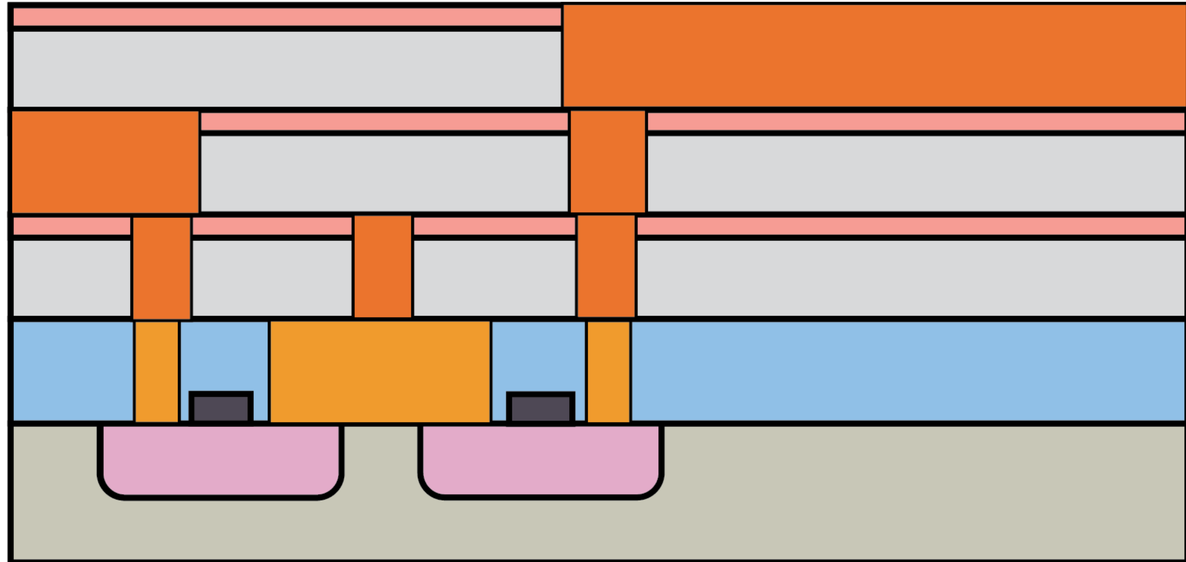
https://www.cl.cam.ac.uk/~sps32/ahsa2017_prob.pdf

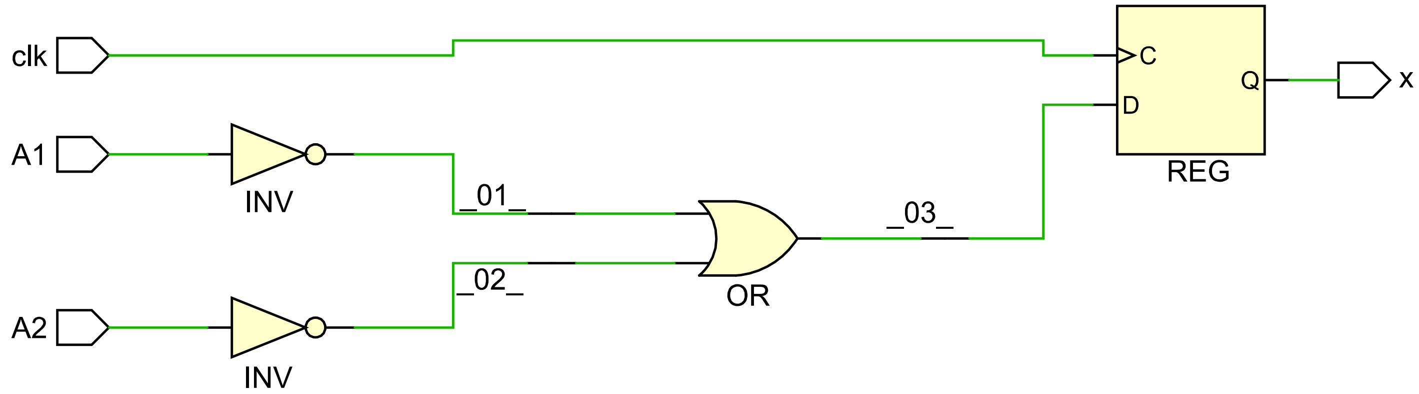
HOW DOES A CHIP WORK?

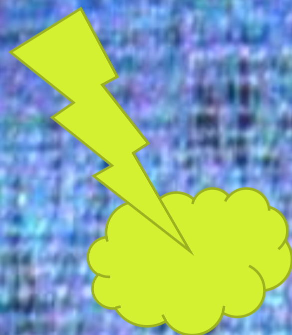
public

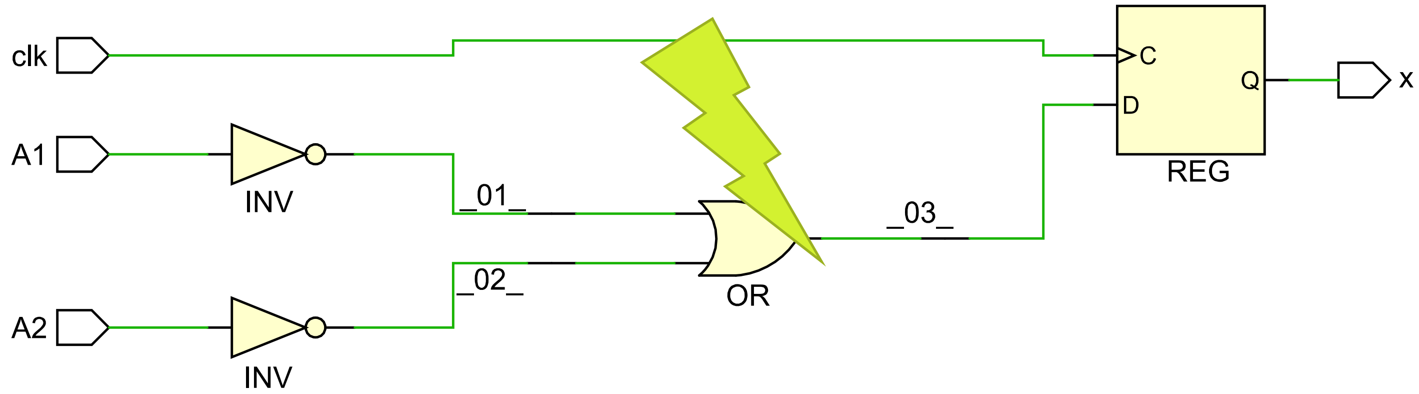


Chris Tarnovsky,
Hardware Hacking Handbook




clk
A1
A2
INV
INV
_01_
_02_
OR
_03_
C
D
Q
REG
x

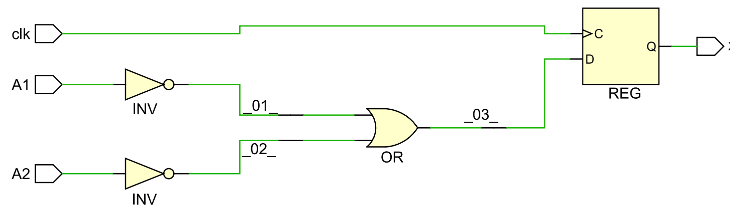



clk
A1
A2
INV
INV
_01_
_02_
OR
_03_
C
D
Q
REG
x

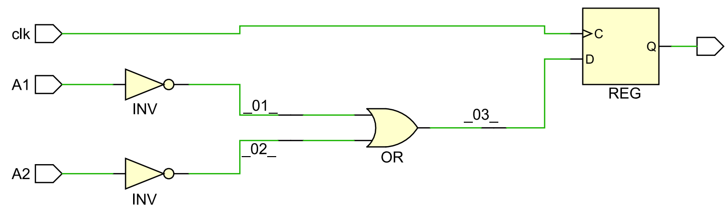
SIMULATING FAULTS

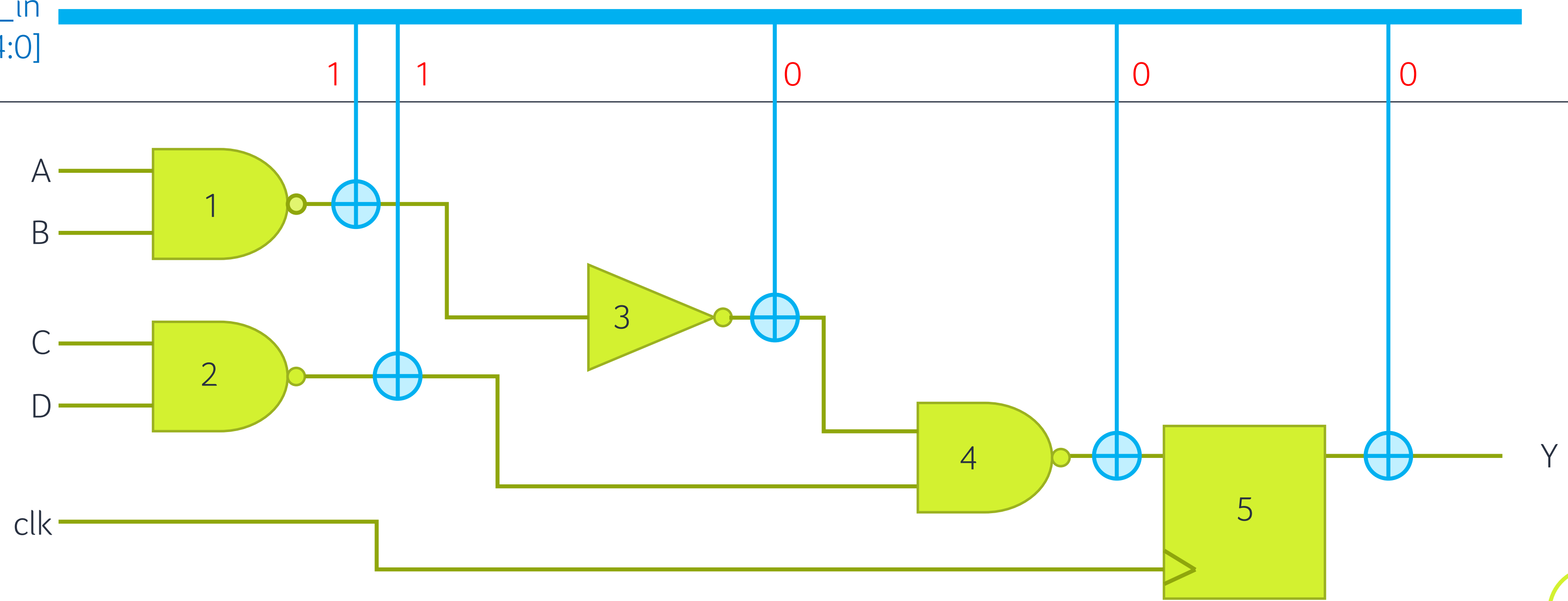
public

```
byte x, A1, A2, _01_, _02_, _03_;
_01_ = ~A1;
_02_ = ~A2;
_03_ = _01_ | _02_;
x = _03_;
```

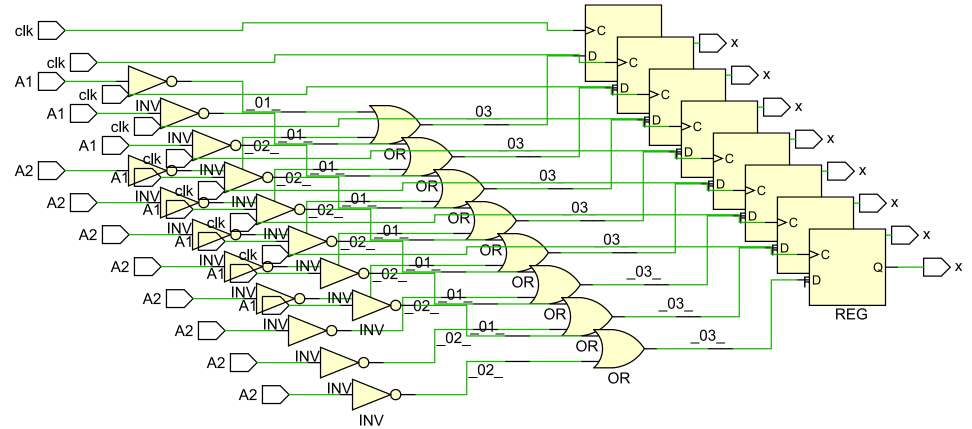


```
byte x, A1, A2, _01_, _02_, _03_;
_01_ = ~A1 & 1;
_02_ = ~A2 & 1;
_03_ = _01_ | _02_ & 1;
x = _03_ & 1;
```




glitch_in
[4:0]
1
1
0
0
0
A
B
1
C
D
2
3
4
5
clk
Y

```
byte x, A1, A2, _01_, _02_, _03_;
_01_ = ~A1;
_02_ = ~A2;
_03_ = _01_ | _02_;
x = _03_;
```



DEMO PICORV32

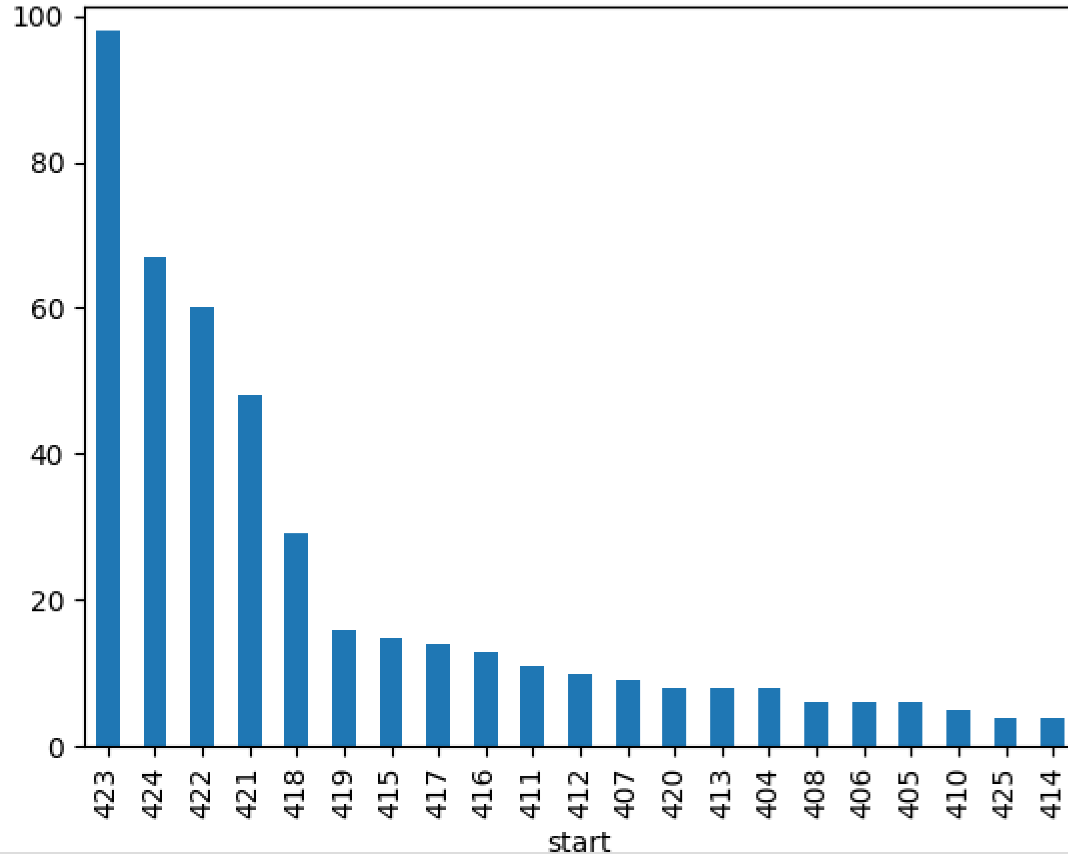
public

```c
void secureboot(void) {
    set_test_status(TEST_START);
    debug("branch test\n");

    set_test_status(TEST_TRIGGER_UP);
    char success = *(volatile unsigned char *)(UART0_BASE_ADDR
    if(success) {
        set_test_status(TEST_TRIGGER_DOWN);
        debug("success\n");
        set_test_status(TEST_FI_SUCCESS);
    } else {
        set_test_status(TEST_TRIGGER_DOWN);
        debug("failure\n");
        set_test_status(TEST_FI_FAIL);
    }
}
```

```
[*] Jobs total: 995329, No fault: 949921, Exploitable: 514, Unexploitable: 44894, Unknown fault: 0, Jobs/s: 16579.85276926696З, Fin
[*] Jobs total: 1435392, No fault: 1376828, Exploitable: 519, Unexploitable: 58045, Unknown fault: 0, Jobs/s: 17316.184422914328, F
[*] finished. Time: 85.72568535804749
/nlsim/tools/fi_grep.py -v -n 10 -g /nlsim/output/picorv32/ffonly/instrumented.v.map -r /nlsim/output/picorv32/ffonly/repro -o /nls
ffonly/figrep_bruteforce.res | tee /nlsim/output/picorv32/ffonly/repro.repro.log
[*] Loading glitchmap
Version: 3, date: 2021-10-28 04:23:55.017460, machines: 256, numpints: 1596, glitchstart: 0, glitchend: 800, glitchwidth: 1, clocks
[*] Converting to numpy arrays
Data points: 519
Start: 423, num: 98/18%, pins: _14823_(_00152_->pcpi_rs1,None)@209, _15465_(_00752_->mem_la_wdata,None)@851, _15486_(_00773_->pcpi_
_(_00775_->pcpi_rs2,None)@874, _15489_(_00776_->pcpi_rs2,None)@875, _15490_(_00777_->pcpi_rs2,None)@876, _15491_(_00778_->pcpi_rs2,
0780_->pcpi_rs2,None)@879
Start: 424, num: 67/12%, pins: _14823_(_00152_->pcpi_rs1,None)@209, _15467_(_00754_->mem_la_wdata,None)@853, _15465_(_00752_->mem_la
5492_(_00779_->pcpi_rs2,None)@878, _15493_(_00780_->pcpi_rs2,None)@879, _15494_(_00781_->pcpi_rs2,None)@880, _15495_(_00782_->pcpi_
_(_00848_->pcpi_rs1,None)@947
Start: 422, num: 60/11%, pins: _15687_(_00974_->\cpuregs[15][31],None)@1073, _15000_(_00017_->\decoded_imm_j[15],None)@386, _16177_
_imm[5],None)@26, _16175_(_01462_->instr_blt,None)@1561, _15670_(_00957_->\cpuregs[15][14],None)@1056, _15669_(_00956_->\cpuregs[15
)@382, _15668_(_00955_->\cpuregs[15][12],None)@1054, _15667_(_00954_->\cpuregs[15][11],None)@1053
Start: 421, num: 48/9%, pins: _15662_(_00949_->\cpuregs[15][6],None)@1048, _14702_(\alu_out[16]->\alu_out_q[16],None)@88, _14696_(\a
->\alu_out_q[11],None)@83, _14698_(\alu_out[12]->\alu_out_q[12],None)@84, _14699_(\alu_out[13]->\alu_out_q[13],None)@85, _14700_(\al
>\alu_out_q[15],None)@87, _14703_(\alu_out[17]->\alu_out_q[17],None)@89, _14694_(\alu_out[8]->\alu_out_q[8],None)@80
Start: 418, num: 29/5%, pins: _16140_(_01427_->\reg_next_pc[2],None)@1526, _16180_(_01467_->instr_slti,None)@1566, _15660_(_00947_->
None)@43, _14656_(_07186_->\reg_out[6],None)@42, _14655_(_07185_->\reg_out[5],None)@41, _16183_(_01470_->instr_ori,None)@1569, _1618
_sltiu,None)@1567, _14654_(_07184_->\reg_out[4],None)@40
Start: 419, num: 16/3%, pins: _14718_(_00001_->is_lui_auipc_jal,None)@104, _14807_(_00136_->pcpi_rs1,None)@193, _14685_(_00000_->de
1568, _16183_(_01470_->instr_ori,None)@1569, _16184_(_01471_->instr_andi,None)@1570, _14730_(_00011_->\cpu_state[4],None)@116, _1619
s1,None)@190, _16193_(_01480_->instr_or,None)@1579
```

```
VM000 has 1 glitches at clock 67 for 1 cycles on pins 1531
prerun | machines: 1 start: 0 end: 67
VM000 pc:    00000000
VM000 pc:    00000044
Allocating 0x10000000 bytes of memory for block 0x2. Addr = 0x002ffffc
VM000: trigger for test status to 80 is set
VM000: setting test status to 80
VM000 pc:    00000020
glitch | machines: 1 start: 67, clocks: 800 end_check: -1
VM000: Inserting glitches at cycle 67
VM000 pc:    00000024 (at fault)
VM000 pc:    000000a8
VM000: trigger for test status to 83 is set
VM000: setting test status to 83
VM000 pc:    00000020
VM000 pc:    00000030
VM000 pc:    00000030
VM000 pc:    00000030
VM000 pc:    00000030
VM000 pc:    00000030
VM000 pc:    00000030
VM000 pc:    00000030
VM000 console: success
VM000 pc:    000000c4
VM000: trigger for test status to 40 is set
VM000: setting test status to 40
VM000 Handle tick done, test done from sw; newstatus 40
Test done at cycle 352
```


top 100 number of possible faults per start time
100
80
60
40
20
0
423
424
422
421
418
419
415
417
416
411
412
407
420
413
404
408
406
405
410
425
414
start

```
VM255: Handle tick, status 82
clock 423
VM255: Handle tick, status 82
VM255 pc:    00000088
```

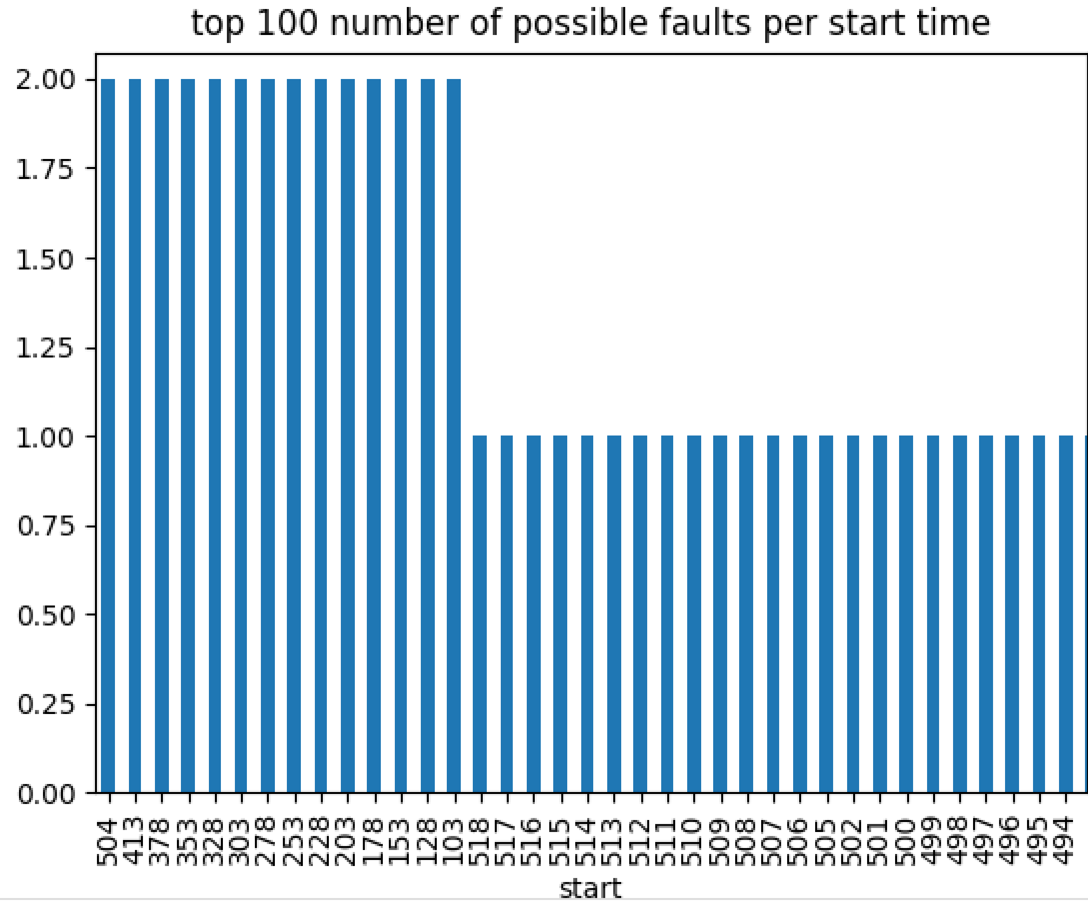
```
  char success = *(volatile unsigned char *)(UART0_BASE_ADDRESS + UART0_RX_DATA);
7c:   a00007b7                lui     a5,0xa0000
80:   0087c783                lbu     a5,8(a5) # a0000008 <secureboot+0x9ffffc4>
84:   0ff7f793                andi    a5,a5,255
  if(success) {
88:   02078e63                beqz    a5,c4 <secureboot+0x80>
```

public

```c
void secureboot(void) {
    set_test_status(TEST_START);
    debug("branch test\n");

    set_test_status(TEST_TRIGGER_UP);
    char success = *(volatile unsigned char *)(UART0_BASE_ADDRESS + UART0_
    if(success) {
        success = *(volatile unsigned char *)(UART0_BASE_ADDRESS + UART0_RX
        if(success) {
            set_test_status(TEST_TRIGGER_DOWN);
            debug("success\n");
            set_test_status(TEST_FI_SUCCESS);
        }
    } else {
        set_test_status(TEST_TRIGGER_DOWN);
        debug("failure\n");
        set_test_status(TEST_FI_FAIL);
    }
}
```

519 -> 393 faults



OPENTITAN
AES ROUND TRUNCATE

public

```
\aes.aes_core.aes_control.key_init_new_q[2]
\aes.aes_core.aes_control.key_init_new_q[3]
\aes.aes_core.aes_control.key_init_new_q[4]
\aes.aes_core.aes_control.key_init_new_q[5]
\aes.aes_core.aes_control.key_init_new_q[6]
\aes.aes_core.aes_control.key_init_new_q[7]
\aes.aes_core.aes_control.num_rounds_q[1]
\aes.aes_core.aes_control.num_rounds_q[2]
\aes.aes_core.aes_control.num_rounds_q[3]
\aes.aes_core.aes_control.round_q[0]
\aes.aes_core.aes_control.round_q[1]
\aes.aes_core.aes_control.round_q[2]
\aes.aes_core.aes_control.round_q[3]
\aes.aes_core.key_init[0]
\aes.aes_core.key_init[100]
\aes.aes_core.key_init[101]
\aes.aes_core.key_init[102]
\aes.aes_core.key_init[103]
\aes.aes_core.key_init[104]
\aes.aes_core.key_init[105]
\aes.aes_core.key_init[106]
\aes.aes_core.key_init[107]
\aes.aes_core.key_init[129]
\aes.aes_core.key_init[130]
\aes.aes_core.key_init[131]
```

Commits:
B50dc9d7c6148ad94cd77ab24d59da027567f1d2: before
Fd3708aa4d674ce7ed871e4f41ad7fc7abfa2f69: after countermeasure

https://github.com/lowRISC/opentitan

```
/////////////////////////////
// Round Counter Protection //
/////////////////////////////
// To protect the round counter against fault injection, we use two counters:
// - rnd_ctr_d/q counts the executed rounds. It is initialized to 0 and counts up.
// - rnd_ctr_rem_d/q counts the remaining rounds. It is initialized to num_rounds_q and counts
//   down.
// In addition, we use one parity bit for the rnd_ctr_d/q counter.
//
// An alert is signaled and the FSM goes into the terminal error state if
// i) the sum of the counters doesn't add up, i.e. if rnd_ctr_q + rnd_ctr_rem_q != num_rounds_q, or
// ii) the parity information is incorrect.

// The following primitives are used to place size-only constraints on the
// flops in order to prevent optimizations on the protected round counter.
prim_flop #(
  .Width(4),
  .ResetValue('0)
) u_rnd_ctr_regs (
  .clk_i,
  .rst_ni,
  .d_i ( rnd_ctr_d ),
  .q_o ( rnd_ctr_q )
);
```

```
// Generate parity bits.
assign rnd_ctr_parity_d = ^rnd_ctr_d;
assign rnd_ctr_parity   = ^rnd_ctr_q;

// Detect faults.
assign rnd_ctr_err_sum    = (rnd_ctr_q + rnd_ctr_rem_q != num_rounds_q) ? 1'b1 : 1'b0;
assign rnd_ctr_err_parity = (rnd_ctr_parity != rnd_ctr_parity_q) ? 1'b1 : 1'b0;

assign rnd_ctr_err = rnd_ctr_err_sum | rnd_ctr_err_parity;
```

```
+    // invalid or in case we have detected a fault in the round counter.
+    if (mux_sel_err_i || rnd_ctr_err) begin
       aes_cipher_ctrl_ns = ERROR;
     end
```

RESULT OVERVIEW

	Before	After
Total	27846112	27807584
No fault	27564705	27527206
Round truncate	16	17
Other faults	281391	280361

FINAL THOUGHTS

public

Riscure FiSim (training edition)

Clean Build Verify Simulate

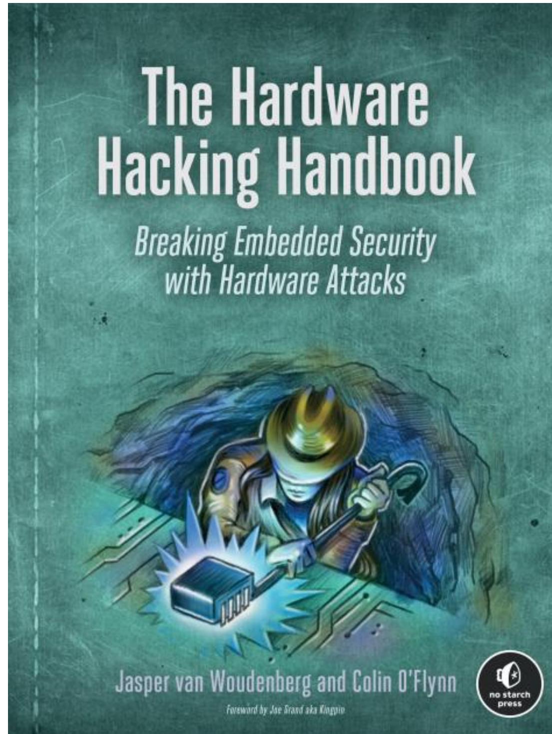
- bin
- fip
- include
- mbedtls
- src
  - config.c
  - console.c
  - crypto.c
  - firmware.c
  - flash.c
  - gpio.c
  - hash.c
  - main.c
  - mem.c
  - mmc.c
  - otp.c
  - sdmmc.c
  - serial.c
  - string.c
  - timer.c
- build_fip.py
- generate_key.py
- Makefile.AArch32

main.c

```
29	if (!is_sec_boot_en) {
30		printf("Enable secure boot\n");
31		sec_boot_enable();
32	
33		is_sec_boot_enabled(&is_sec_boot_en);
34	}
35	
36	if ((uintptr_t)(fip_head = load_fip_from_flash(1)) == -1) {
37		printf("Loading FIP failed\n");
38		__SET_SIM_FAILED();
39	}
40	
41	void* fip_base = fip_head->boot_head.base_addr;
42	size_t fip_size = fip_head->boot_head.length - RSA_SIGN_SIZE;
43	void* sign_base  = fip_base + fip_size;
44	
45	hash_sha1(fip_base, fip_size, fip_hash);
46	
47	crypto_rsa_init();
48	
49	if(is_sec_boot_en && !crypto_rsa_verify_sha1(fip_hash, sign_base ) ) {
50		printf("Auth failed!\n");
51		__SET_SIM_FAILED();
52	}
53	
54	if (unpack_images_from_fip()) {
55		printf("Unpacking FIP failed\n");
56		__SET_SIM_FAILED();
57	}
58	
59	pmc_release();
60	
61	config_uboot_init(dram_size);
62	
63	printf("Boot next stage\n");
64	
65	boot_next_stage();
66 }
```

```
8000531C(1/1): MOV r2, #1 -> MOV r2, #3
80005330(1/1): LDR r3, [fp, #-8] -> LDR r3, [fp, #-0]
80005334(1/1): STR r2, [r3, #4] -> ORR r2, r3, r4 (8x)
800052F8(1/1): MOV r0, #0x10 -> MOV r0, #0x10000000 (11x)
8000522C(1/1): LDR r3, [pc, #0x70] -> STR r3, [pc, #0x70]
80005230(1/1): PUSH {fp, lr} -> STMDB pc!, {fp, lr}
8000525C(1/1): LDR r3, [fp, #-8] -> LDR r3, [fp, #-0]
80005264(1/1): STR r0, [r3, #8] -> STR r1, [r3, #8] (3x)
80008580(1/1): MOV r2, #0xac -> ORR r2, r0, #0xac (6x)
8000B6E8(1/1): BNE #0x8000b708 -> BNE #0x8000b748
80008584(1/1): LDR r4, [pc, #0x60] -> LDR r4, [pc, #-0x60]
80008594(1/1): ADD r0, r4, #0x690 -> ADD r0, r4, #0x6b0 (2x)
8000B728(1/1): LDR r3, [pc, #0x94] -> LDR r3, [pc, #0x9c] (9x)
8000B730(1/1): LDRB r3, [r3, #0x73c] -> LDRB r3, [r3], #0x73c (14x)
8000B738(1/1): BEQ #0x8000b76c -> BNE #0x8000b76c (3x)
8000B734(1/1): CMP r3, #0 -> CMP r3, #1 (7x)
80007C0C(1/1): ADD fp, sp, #4 -> STR fp, [sp], r4 (5x)
80008550(1/1): LDR r0, [pc, #0x20] -> LDR r0, [pc, #0x20]! (2x)
80008564(1/1): BL #0x80007c08 -> B #0x80007c08 (5x)
80007C08(1/1): PUSH {fp, lr} -> STMDB pc!, {fp, lr}
80007818(1/1): PUSH {r4, r5, r6, r7, r8, sb, sl, fp, lr} -> STMDB pc!, {r4, r5, r6, r7
8000B72C(1/1): ADD r3, pc, r3 -> ADD r3, pc, fp (5x)
80007C3C(1/1): B #0x80007814 -> B #0x80007804 (2x)
80007C1C(1/1): BEQ #0x80007c30 -> BEQ #0x80007c70
8000B744(1/1): BL #0x80008534 -> BL #0x800084f4 (2x)
80007854(1/1): BL #0x800055e8 -> B #0x800055e8
8000781C(1/1): ADD fp, sp, #0x20 -> ADD sl, sp, #0x20 (7x)
800030A8(1/1): ADD fp, sp, #4 -> STR fp, [sp], r4
80003188(1/1): ADD fp, sp, #4 -> STR fp, [sp], r4
80007898(1/1): BNE #0x800078e4 -> BEQ #0x800078e4 (3x)
800078BC(1/1): MOV r8, r0 -> MOV r8, r8 (12x)
80007904(1/1): MOV r0, r8 -> MOV r0, sb (4x)
80008568(1/1): CLZ r0, r0 -> CMN pc, r0, lsl pc (12x)
8000B748(1/1): CMP r0, #0 -> CMP r8, #0 (11x)
80007908(1/1): SUB sp, fp, #0x20 -> SUB sp, fp, #8
8000856C(1/1): LSR r0, r0, #5 -> LSR r0, r2, #5 (6x)
8000B74C(1/1): BNE #0x8000b76c -> BEQ #0x8000b76c (3x)
80008574(1/1): POP {fp, pc} -> ADCS r8, sp, r0, lsl #16 (3x)
```

<PLUG>



</PLUG>

hardwarehacking.io

REALLY FINAL THOUGHTS

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

driving your security forward