Security and Cryptography 2021

III. Malicious Devices

Mirosław Kutyłowski

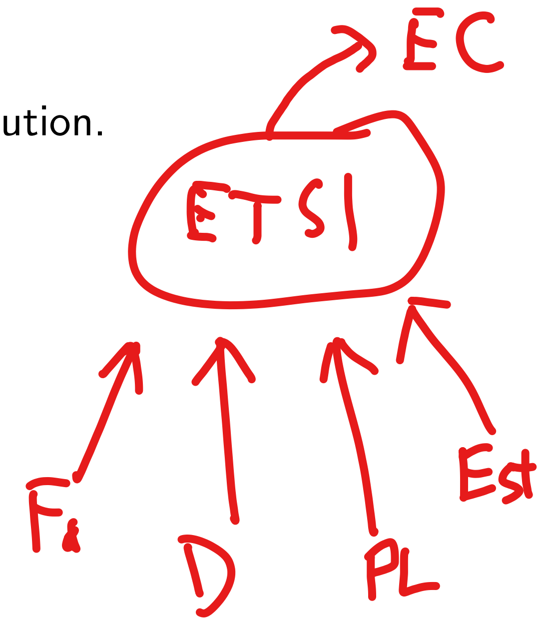
Enigma

Crypto AG

Huawei

Standards:

It is not true that a standard solution is by definition a secure solution.



Standardization process:

- representatives of countries, not necessarily specialists
- strong representation of interests of industry
- target: a unified solution
- no open evaluation as in case of e.g. NIST competitions
- long process, many standards never used in practice

Example: ANSI X9.31 PRG

- approved PRNG by FIPS and NIST between 1992 and 2016
- now deprecated by NIST
- many devices based on X9.31 have FIPS certificates, widely used

Algorithm

→ **initialization - seeding**: select initial seed $s = (K, V)$, with random V and pre-generated key K

- K used for the lifetime of the device
- V will change

→ **generate (generating bits and changing the internal state):**

1. input the current state $s_{i-1} = (K, V_{i-1})$ and the current timestamp T_i
2. intermediate value: $I_i := \mathrm{Enc}_K(T_i)$
3. output: $R_i := \mathrm{Enc}_K(I_i \oplus V_{i-1})$
4. state update: $V_i = \mathrm{Enc}_K(R_i \oplus I_i)$

V_{i-1} ?

Problems with seeding:

- NIST standard says: “This K is reserved only for the generation of pseudo-random numbers”, and explains length,
- NIST standard does not say how K is generated
- consequences:

  $\longrightarrow$ certification documentation may skip the problem of generating K

  $\longrightarrow$ in some cases the key is encoded in software or hardware and **the same** for all devices and there is no reason to reject application for a certificate

an attack is based on the key K recovered from software

1. observe R_i and R_{i+1}
2. guess timestamp T_i, T_{i+1} and check that :

$$\mathrm{Dec}_K(R_{i+1}) \oplus \mathrm{Enc}_K(T_{i+1}) = \mathrm{Enc}_K(R_i \oplus \mathrm{Enc}_K(T_i))$$

   where the sides of the equation are equal to:

$$(I_{i+1} \oplus V_i) \oplus I_{i+1} = \mathrm{Enc}_K(R_i \oplus I_i)$$

$$V_i = V_i$$

3. if the test shows equality, then the timestamps are ok and V_i appears on both sides
4. having K and V_i one can recover states forwards and backwards each time adjusting the guesses for timestamp – as long as the (portions) of the generated sequence are available. For backwards:

   → $R_t = \mathrm{Enc}_K(I_t \oplus V_{t-1})$, so $V_{t-1} = \mathrm{Dec}_K(R_t) \oplus I_t$

   → having V_{t-1} compute $R_{t-1} = \mathrm{Dec}_K(V_{t-1}) \oplus I_{t-1}$

the attack requires the key K and guessing two consecutive timestamps

→ implementations do not care about it and use consecutive outputs e.g. for DH exponent, separating them would help

→ presenting two output blocks of the PRNG is necessary for the attack – so presenting at most one block would help

→ it would help to use DH exponent as a hash of the output of PRNG and some data hard to guess by the attacker, but many protocols do not do it

→ attacking either side may help for DH, but for RSA key transport the party choosing the secret must be affected

DUAL EC -standardized backdoor

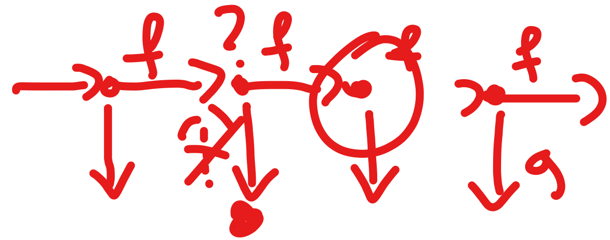
– NIST, ANSI, ISO standard for PRNG, from 2006 till 2014 when finally withdrawn

– problems reported during standardization process: bias that would be unacceptable for constructions based on symmetric crypto, finally 2007 a paper of Dan Shumow and Niels Ferguson with an obvious attack based on kleptography (1996)

– DUAL EC dead for crypto community since 2007 but not in industry

- deal NSA -RSA company (RSA was paid to include DUAL EC)
- products with FIPS certification had to implement Dual EC, no certificate when P and Q generated by the device
- generation of own P and Q discouraged by NIST
- used in many libraries: BSAFE, OpenSSL, ...
- in 2007 an update of Dual EC that makes the backdoor more efficient
- changes in the TCP/IP to ease the attack (increasing the number of consecutive random bits sent in plaintext)

algorithm:



- basic scheme:
  - → state $s_{i+1} = f(s_i)$, where s_0 is the seed
  - → generating bits: $r_i := g(s_i)$
  - → both f anf g must be one-way functions in a cryptographic sense
- Dual EC, basic version:
  - → points P and Q “generated securely” by NSA but information classified,
  - → $s_{i+1} := x(s_i \cdot P)$ (that is, the “x” coordinate of the point on an elliptic curve)
  - → $r_i := x(s_i \cdot Q)$
  - → this option used in many libraries
- Dual EC with additional input:
  - → if additional input given then update is slightly different:
  - → $t_i := s_i \oplus H(\text{additonalinput}_i),\ s_{i+1} := x(t_i \cdot P)$

Attack: with a backdoor d, where $P = d \cdot Q$

- for basic version:
  - $\rightarrow$ from r_i reconstruct the EC point R_i (immediate, two options)
  - $\rightarrow$ compute s_{i+1} as $x(d \cdot R_i)$ (no knowledge of the internal state s_i required)
- for additional input:
  - it does not work in this way since the $\oplus$ operation is algebraically incompatible with scalar multiplication with the points of elliptic curve
  - however it does not help much: frequently more than one block r_i is needed by the consuming application and simply the next step(s) is executed without additional input – at this moment the adversary learns the internal state
  - the attacker have problems if cannot trace the additional input: gradually looses control over the state of PRNG

Dual EC 2007:

– an update to "increase security"

– an extra step after request for bits, before using additional input:

→ $s_{i+1} := x(s_i \cdot P)$

→ $t_{i+1} := s_{i+1} \oplus H(\text{additional input}_{i+1})$

→ $s_{i+2} := x(t_{i+1} \cdot P)$

→ $r_{i+2} := x(s_{i+2} \cdot Q)$

– attack:

  – reconstruct $s_{i+1} := x(d \cdot R_i)$

  – compute t_{i+1} and s_{i+2} for guessed additional input, then check against r_{i+2} (the test works also if r_{i+2} is used as an exponent for DH and only the result of exponentiation is visible for the attacker

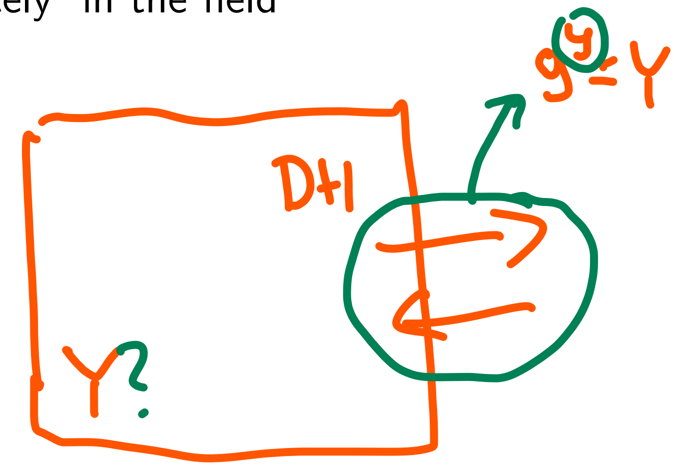
Practical attack issues:

– some products do not use entire r_i and skip some number of bits. Frequently this is 16 bits – which makes the attack 2^{16} times longer. Truncating say 100 bits would secure the design, but this is not done

– some protocols disclose the original PRNG output. Then increasing the size of such a block eases the attack, as some steps are executed without additional input and the time complexity goes down

Kleptography

- dual EC is onl one example of kleptography, unfortunately "in the field"
- idea:



  - install a trapdoor in a device
  - the trapdoor usess a "public key"
  - the attacker holds a matching private key
  - the output of the device is indistinguishable from the output of the honest machine
  - with the private key one can break security of the device, get access to secret information, etc
  - .. while with the "public key" this is impossible
- if one can find the kleptographic code in the device then the attack is evident, but what if tamper resistant?

malicious crypto

Example: generating Schnorr signatures

- the malicious device contains $U = g^u$, the attacker knows u
- creating 1st signature:
  1. k chosen at random, $r := g^k$
  2. $e := \mathrm{Hash}(M, r)$
  3. $s := k - e \cdot x$
  4. output (s, e), retain k
- creating 2nd signature:
  1. $k' := \mathrm{Hash}(U^k)$, $r' := g^{k'}$
  2. $e' := \mathrm{Hash}(M', r')$
  3. $s' := k' - e' \cdot x$
- attacker getting the secret x no matter how well it has been created:
  1. $r := g^s \cdot X^e$
  2. $k' := \mathrm{Hash}(r^u)$
  3. $x := (k' - s') / e'$

$r^u = (g^k)^u = (g^u)^k = U^k$

Example: Diffie Hellman key exchange

- the malicious device contains $U = g^u$, the attacker knows u
- key exchange i :
  1. k_a chosen somehow
  2. $c_a := g^{k_a}$
  3. $K := {c_b}^{k_a}$
- key exchange $i+1$:
  1. $k'_a := \mathrm{Hash}(U^{k_a})$,
  2. $c'_a := g^{k'_a}$
  3. $K' := {c'_b}^{k'_a}$
- attacker getting session key K:
  1. $k'_a := \mathrm{Hash}(c_a^u)$
  2. $K' := {c'_b}^{k'_a}$

  warning: it suffices to have a malicious device on one side to tap the line!

Example: slow leakage via a random string

- the malicious device contains $U = g^u$, the attacker knows u, secret s to be leaked
- leaking, when PRNG secure:
  1. cryptographic boundary: k chosen at random,
  2. then $r{:=}g^k$ computed outside PRNG, $V := U^k$
  3. $a := (k$ most significant bits of $V)$
  4. test: if bit $k+1$ of V is different from ath bit of s then return to 1
  5. proceed with the original protocol, r exported as part of the output
- attacker:
  1. gets a cryptographic message with r
  2. $V := r^u$
  3. $a := (k$ most significant bits of $V)$
  4. retrieve the ath bit of s as bit $k+1$ of V

so separating generation of k is a secure perimeter helps to launch the attack: PRNG does not know what is going on outside and creates r's on demand

Furthermore: what if PRNG uses this procedure to leak own internal state? This is why we need the reseed procedure with entropy input.

Practical issues

- existence of a kleptographic code can be detected by power and time analysis,
- e.g. in case of Schnorr signatures 2 exponentiations instead of 1: total time can be hidden by speeding up, but not the statistical characteristics (average deviation of computation time for 2 exponentiations is smaller than in case of 1 (2xslower) exponentiation
- clever complicated constructions that take it into account

Further threats

- generating RSA keys so that the adversary can get the private key from the public one

Defense - reverse firewall

on top of the PRNG there is a deterministic procedure RF with a secret key installed by the user

it sanitizes the output of PRNG

Example: generating g^k for a random k:

i. PRNG outputs g^k

ii. RF computes $k' := \mathsf{Enc}_{\mathrm{SK}}(g^k)$

iii. PRNG decrypts k' to check its correctness

iv. PRNG adjusts $k := k + k' \bmod q$, and recomputes g^k

v. RF checks that the new g^k equals the old g^k times $g^{k' \mathrm{mod q}}$

PRNG outputs g^k

HARDWARE TROJANS

goal of a Trojan: change hardware so that the chip functionally seems to work as claimed, but it opens a backdoor for the attacker

attack moment:

- chip planning (easy)
- chip manufacturing (hard)
- hardware components from third parties (easy)
- outsourcing fabrication (likely to occur due to production line costs)

methods of testing:

- `functional tests` (not really helpful for trapdoors, the most dangerous are hidden faults that do not disrupt operation)
- `internal tests circuitry` (putting some values and observing results on single components along so called test path, or dedicated tests like checking CRC of memory contents)
- `distructive` - chemical-mechanical polishing and inspection under microscope etc, it can detect modifications on layout level, very costly procedure, specialized labs necessary
- `side channel information` (especially comparing with a "golden chip" behavior – the chip that is ideal and follows the specification) - delay path analysis, static current analysis, transient current analysis

classical attacks: the trojans should remain undetected during the testing phase, so the attack has to be triggered by an unlikely event. Options used:

- an attack triggered by an unlikely event known to the attacker but not to the evaluator
- an attack starts when some counter reaches a certain value
- attack occurs due to aging or via a random event (e.g. for enabling fault analysis)

some countermeasures:

- regions: design a chip so that it consists of "regions"
  - for each region there must be a test path so that the activities are concentrated in this region while the rest stays almost idle,
  - then the side channel (such as energy usage) may be attributed to that region
  - a hardware Trojan should be concentrated in some region and then substantially change the side channel of that region
- avoid rare-triggered nets
- insert configurable security monitors
- variant-based parallel execution of the same function

Analog attack: A2

goal: in a certain situation change a priviledge bit (the rest of the attack follows some scenario)

limitations:

- no change in a digital circuit, only some analog parts added
- very limited regarding area (so playground for ASICs, which are less optimized – less compressed)
- trojans preferably in layer 1 to avoid collisions with routing etc

construction idea:

- a single capacitor added,
- the capacitor is loaded each time a triggering event occurs
- if triggering events occur in a short period of time, then the capacitor loaded to a certain voltage causing a flip-flop operation to occur (changing a bit to a predefined value)
- the capacitor discharged gradually so if triggering events occur infrequently, then the flip-flop operation never executed

a more robust construction:

– choosing relative capacity of capacitors one can control the number of triggering events needed

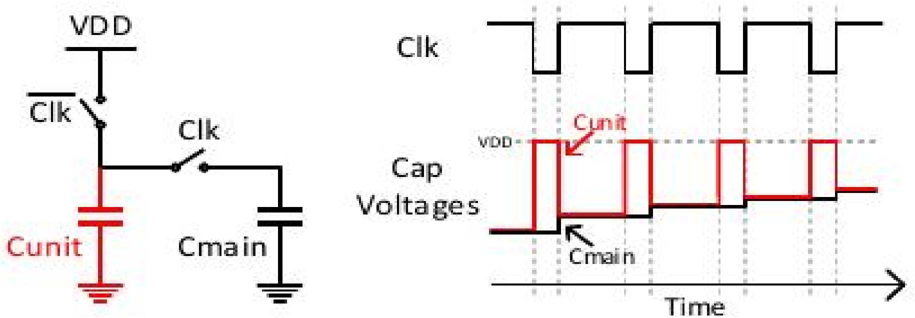


Figure 4: Design concepts of analog trigger circuit based on capacitor charge sharing.

Figure 1.

(from paper: A2: Analog Malicious Hardware, Kaiyuan Yang, Matthew Hicks, Qing Dong, Todd Austin, Dennis Sylvester)

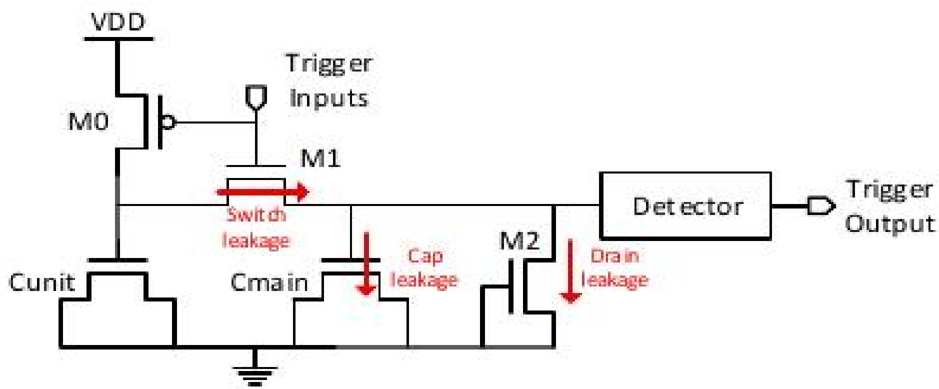


Figure 5: Transistor level schematic of analog trigger circuit.

Figure 2.

`transistor M0`: allows flow at low voltage, `transistor M1`: allows flow at high voltage

`detector`: it could be for instance an inverter – changing the output would create some malicious consequences

extensions:

- use a few such analog circuits and combine them
- e.g.: both must "fire" (AND operation), one of them suffices ("OR") – in theory any circuit possible however the attacker is limited by space available

Dopant Trojans

CMOS inverter: (image Wikipedia)

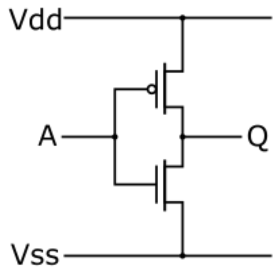


Figure 3.

where: A is the source, Vdd positive supply , Vss is ground

upper transistor: PMOS (allows current flow at low voltage)

lower transistor: NMOS (allows current flow at high voltage)

how it works:

- if voltage is low, then the lower transistor (NMOS) is in high resistance state and the current from Vdd flows to Q (high voltage)
- if voltage is high, then the upper transistor MOS) is in high resistance state and the current from Vss flows to Q while Vdd has low voltage

PMOS: in dopant area "holes" (positive) playing the role of conductor, low voltage creates depletion area, high voltage attracts them

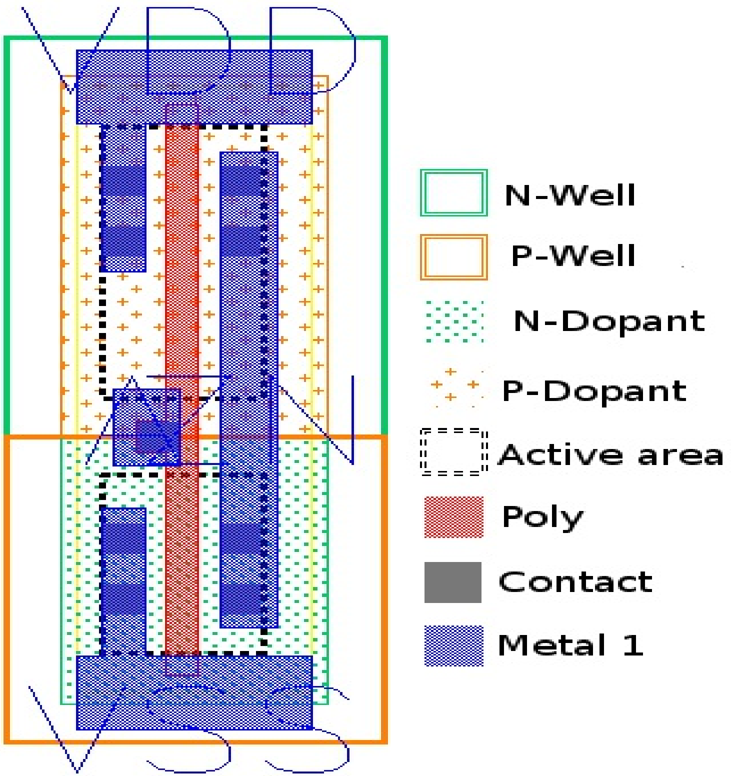
NMOS: in dopant area electrons (negative) playing the role of conductor, high voltage pushes the electrons out

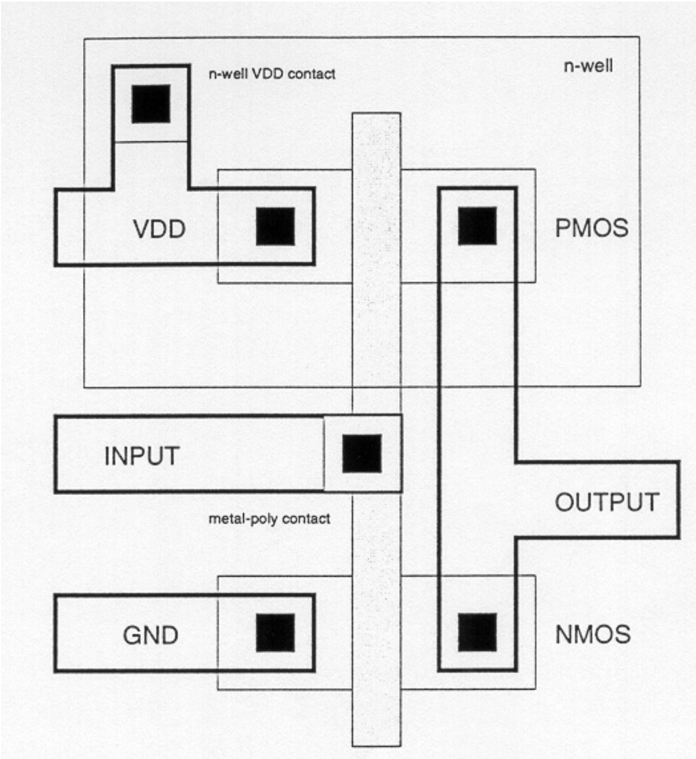
For physical realization of a transistor see excellent videos from

https://www.youtube.com/watch?v=7ukDKVHnac4&t=116s

https://www.youtube.com/watch?v=stM8dgcY1CA

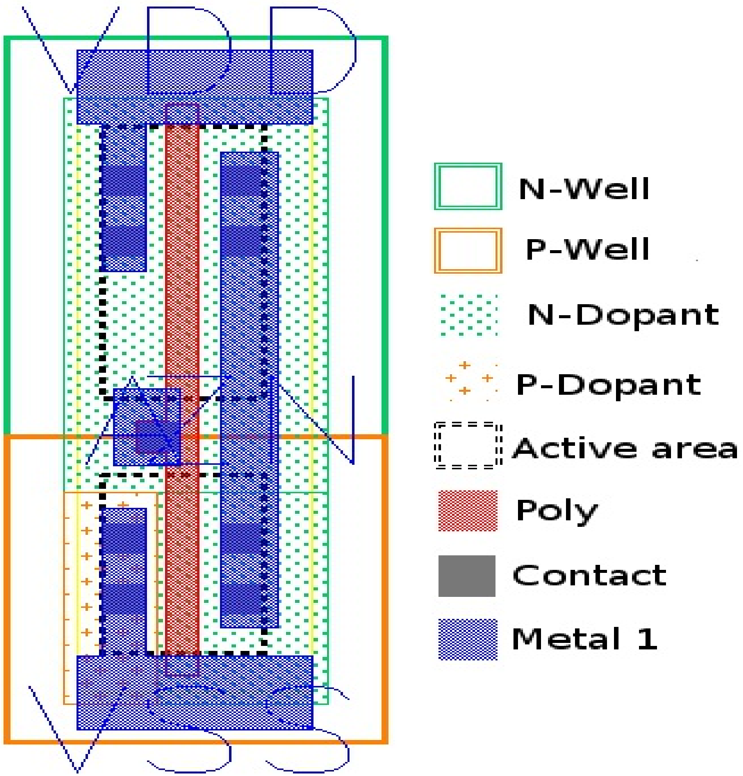
CMOS inverter in the “bird eye perspective”:





(nice diagram from EPFL, “Design of VLSI Systems”)

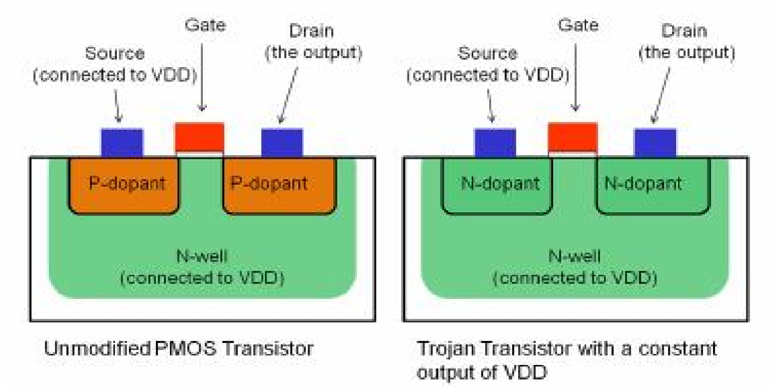
Trojan design:

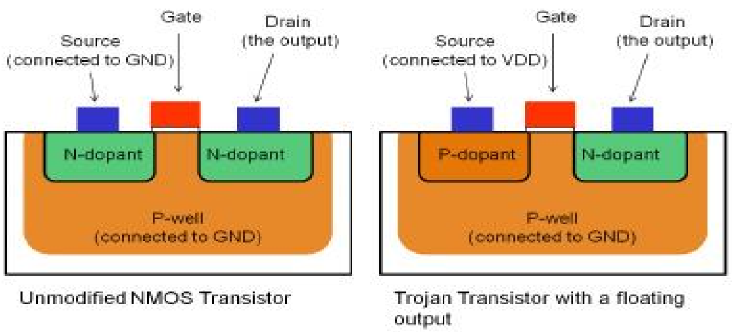


The idea is to inject wrong dopant and thereby disable or enable connection regardless of the voltage

- whatever happens the VDD is connected to the output
- whatever happens the VSS is disconnected with the output

Detailed pictures from the original paper:





(b) n-MOS Transistor

Trojan True Random Number Generator consists of

- entropy source (physical)
- self test circuit (OHT - online health test)
- deterministic RNG, Intel version:

  generate 128-bit numbers when the internal state is (K, c) (by “rate matcher”):

  1 $c := c + 1$, $r := \mathrm{AES}_K(c)$, output r

  2 $c := c + 1$, $x := \mathrm{AES}_K(c)$

  3 $c := c + 1$, $y := \mathrm{AES}_K(c)$

  4 $K := K \oplus x$

  5 $c := c \oplus y$

  - **reseeding** (by “conditioner”)

  1 $c := c + 1$, $x := \mathrm{AES}_K(c)$

  2 $c := c + 1$, $y := \mathrm{AES}_K(c)$

  3 $K := K \oplus x \oplus s$

  4 $c := c \oplus y \oplus t$

attack option 1: fix K by applying Trojan transistors, if K is known, then it is easy to find internal state c from r and then the consecutive random numbers r

attack option 2: fix all but n bits of c then only n bits of entropy and the output r has only n entropy bits - to the attack does not need to see anything, just prediction possible (helpful e.g. against randomized signature schemes)

problem with Built-In-Self-Test: implemented according to FIPS: after power-up the RNG is tested against aging:

- known LFSR creates bits strings for conditioner and rate matcher, entropy source disabled, a 32-bit CRC from the result computed and checked against a known value,
- knowing the test one can find how to manipulate K and c without detection, simple exaustive search can be applied

Side channel Trojan:

- side channel resistant logic: Masked Dual Rail Logic

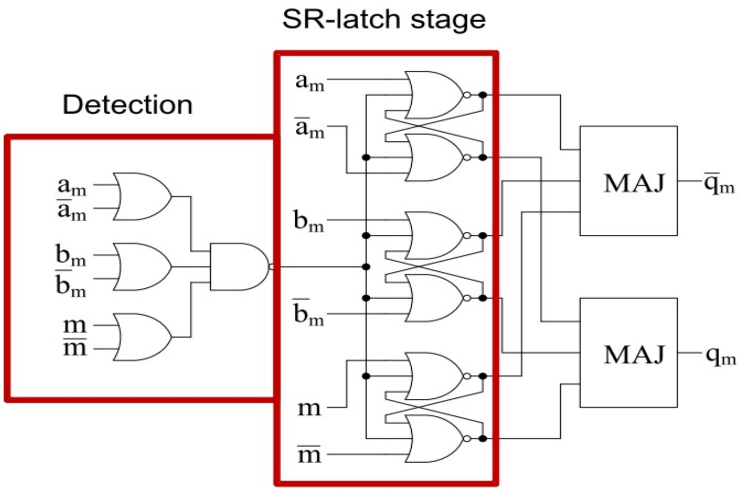
  i for each a both a and negation of a computed

  ii precharge: each phase preceded by charging all gates

  iii **masking operations** by random numbers

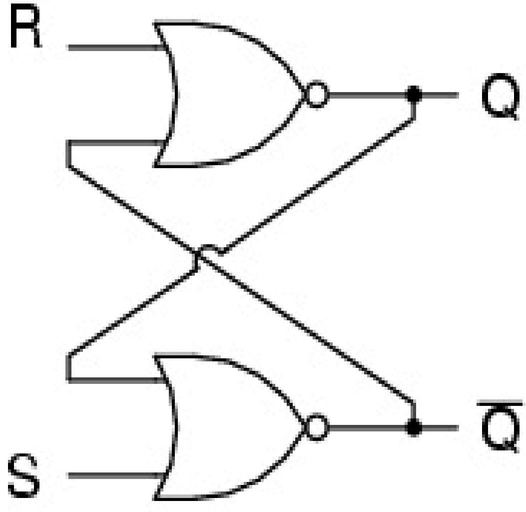
computing $a \wedge b$:

- input $a \oplus m$, $a \oplus \neg m$, $b \oplus m$, $b \oplus \neg m$, m, $\neg m$
- detection, SR-latch stage and majority gate



gates on the picture: OR – 3 gates in the detection , AND - the right gate in the Detection, NOR (output 1 if all inputs 0)- the OR gate with a dot

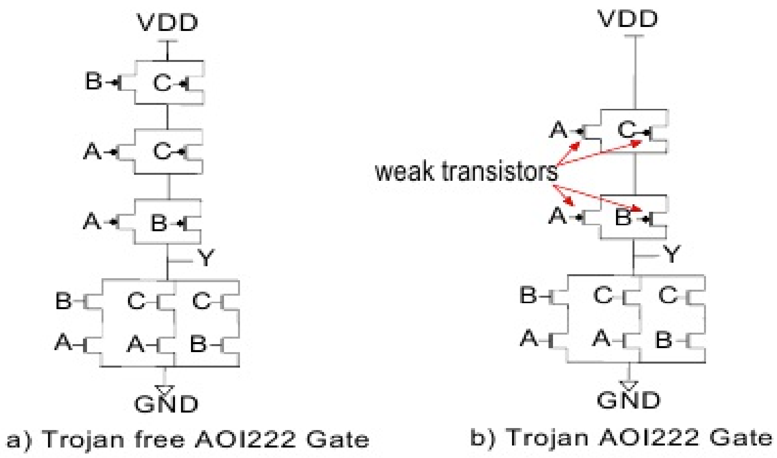
SR-latch is a bi-stable circuit. It remains stable in the state (0,1) and in (1,0). These values encode two bitvalues



S	R	Q	$\overline{Q}$
0	0	latch	latch
0	1	0	1
1	0	1	0
1	1	0	0

see https://www.allaboutcircuits.com/textbook/digital/chpt-10/s-r-latch/

attacking not-majority gate (original picture):

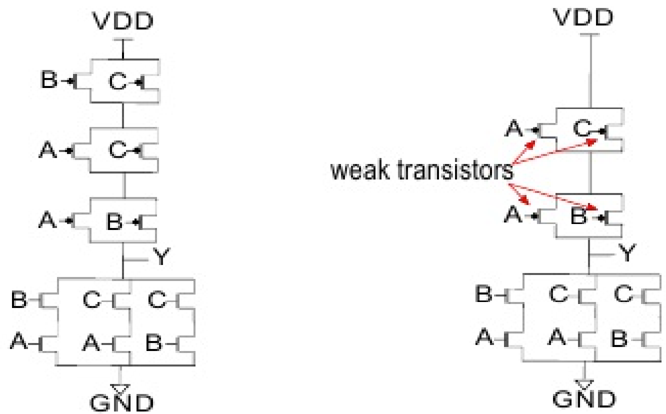


Idea: instead of cutting output there is low voltage in a certain situation

- the same behavior except for $A = 0$ and $B, C = 1$, where good output but high power consumption due to connection between VDD and VSS
- the upper pair of transistors do not disappear from the layout but are changed so that in fact constant connections are created.
- weakness of the transistors is created via reducing dopant areas (dopant creates free electrons or hole that may "jump". Alone reducing the size of active area makes a transistor weak.
- computing majority works as normal except for the case that $a_m = 0, b_m = 1, m = 1$ or

$\bar{a}_m = 0, \bar{b}_m = 1, \bar{m} = 1$. In both cases we have $a = 1, b = 0$

– **high power consumption can be detected, in this way we learn the internal state**



a) Trojan free AOI222 Gate b) Trojan AOI222 Gate

Artificial aging

make some transistors disfuctional (as ithe case of PRNG)

method:

- apply too high voltage at certain areas
- the electrons accelerate and break barrier - damages
- effect the same as of aging a chip
- the transistor changes its operational characteristic

Problems:

– Trojan may be triggered by some particular event, detection becomes harder

– Trojan may work in very particular physical conditions, e.g. temperature, voltage

Defense methods:

- on-chip checks: detection of unexpected behavior, e.g. delay characteristics: workload path and a shadow path that provides result after fixed time, + comparison
- ring osscilators on the chip detecting nonstandard behavior
- methods to enable activation in certain areas only
- inserting PUFs, (either randomize as much as possible - noise over trojan information)
- keep algorithms deterministic
- secure coding: take into account the situation that certain components are not working properly
- **external watchdog** techniques

Sophisticated design problems

motto: high level description might be perfect, but some advanced mechanisms in hardware that are invisible to users may create trapdoors

situation: low level hardware details are frequently proprietary information

Meltdown – attack on modern processors

- standard acceleration technique: **out-of-order execution** of commands:
  - instead of executing just the current operation i, the processor executes operations i, $i+1$, ..., $i+k$
  - apart from the current operation, the next ones are executed conditionally: if the execution of operations $i, i+1, ..., i+j-1$ have influence on the input of operation $i+j$ then the result for operation $i+j$ executed in advance is discarded
  - ... the way to speed up when the hardware has reached its limits
- **kernel and checking access rights:**
  - the system is organized as "secure operating system" - (recall FIPS)

- logically the rules are strict: access rights checked so a user cannot access restricted data in the protected kernel area
- it takes care of read/write access in the sense of the operating system
- ... but there are **indiect ways to learn** the data

Core idea

goal: read arbitrary memory address by an unpriviledged user

instruction sequence

1; `rcx = kernel address`

2; `rbx = probe array`

3 `retry:`

4 `mov al, byte [rcx]` *reading a byte from protected address* rcx *to* al

5 shl rax, 0xc *multiplying* rax *by 4096, so the byte from* al *is shifted*

6 jz retry *jump due to some bias to 0 in* al

7 mov rbx, qword [rbx + rax] *reading from location* rbx+rax

How it is executed:

- the instruction 4 leads to violation of access rights and consequently it will be interrupted, with temporary values erased
- in the meantime instructions 5-7 might be executed in advance, all results retired after interrupt – except for the effects of accessing the cache

Cache

- cache is necessary: gap between CPU speed and latency of memory access, innermost cache access ≈0.3ns, main memory access $\approx 50\,\mathrm{ns}$ to $150\,\mathrm{ns}$
- set-associative memory cache:
  - cache line (cache block) of B bytes
  - a row consisting of W cache lines
  - S cache sets, each consisting of a row
  - when a cache miss occurs, then a memory block is copied into one of cache lines evicting its previous contents
  - a memory block with address a can be cached only into the cache set with the index i such that $i = \lfloor a/B \rfloor \bmod S$ — **this is crucial for the attack**
- cache levels: slight complication to the attacks but differences of timing enable to recognize the situation

Attack

array rbx has size $256 \cdot 4096$ (256 pages)

mechanism:

- before we execute the code we make sure that the whole array rbx is evicted from the cache
  - by overwriting all line of the cache by different read operations
- during the code execution only one address is fetched to the cache because of cache miss
  - provided that instruction 7 is executed before the sequence is retired due to interrupt
- afterwards the attacker reads the whole array rbx page by page:
  - all time the cache misses (long execution time) ...
  - except for the page with the number stored in rcx