Side-Channel Analysis (SCA) Countermeasures

Reference Mangard et. al, "Power Analysis Attacks, Revealing the Secrets of Smart Cards", Springer, 2009

Power analysis attacks are effective because the power consumption of crypto devices depends on intermediate values

The overall goal of countermeasures is to **avoid or reduce** these dependencies

For *hiding*, the goal is to break the link between power consumption and the data being processed

Here, the crypto devices execute the algo in the same way as *un*protected chips

The *hiding* countermeasure makes it difficult to data mine the power traces

Two general approaches

- Build the chip such that the power consumption **is random** (during each clk cycle)
- Build the chip such that it consumes **an equal amount of power** for all operations and data values processed

Neither of these can be accomplished 'ideally' in practice

SCA Countermeasures

Several approaches have been proposed

- Randomize power consumption by performing the operations of the crypto algorithm at different moments in time (affect the **time dimension**)
- Make the power consumption *random* or *equal* by modifying the power consumption characteristics of the operations (affect **amplitude dimension**)

Time Dimension Approaches

DPA attacks on DES, etc. depend on the attacker knowing precisely how the chip executes the algorithm

If the samples in the measured supply waveforms are not 'synchronized', then the attack will **not** succeed, i.e., there will be no 'spikes' or the spikes will be false.

Countermeasures focus on making it difficult for the attacker to 'synchronize' with the algorithm execution

This is most commonly accomplished by *random insertion of dummy operations* and *shuffling the operations*

SCA Countermeasures

A random number generator is used to determine how many **dummy** operations are inserted

The number randomly varies from execution to execution of the algorithm (for obvious reasons) but the total number remains the same

Note that inserting lots of dummies impacts throughput

For **shuffling**, the strategy is to randomly change the **sequence** of operations

For example, S-box lookup operations can be randomized during each round

Advantage is that shuffling does **not** impact throughput

Unfortunately, the sequential nature of the algorithm **limits what you can do** using randomized execution

Both *shuffling* and *dummy delay insertion* are often combined in practice

SCA Countermeasures

Amplitude Dimension Approaches

These techniques focus on reducing *information leakage* by lowering the **signal-to-noise** (SNR), ideally to 0

This is achieved by ensuring that power consumption is exactly **equal** for all operations and data values (eliminate *signal*)

Or by increasing the amplitude of the noise, ideally to *infinity*

For the latter, best approach is to carry out several independent operations in **parallel**

Not all architectures support parallelism

Another way is to use **dedicated noise engines**

Note: if the noise is random, it will cancel out in the averaging process, so it must be correlated with non-critical operations in the chip

Unfortunately, if enough examples plain/cyphertext pairs are available, noise can usually be eliminated (DPA is powerful)

Software Countermeasures

For the former technique, two approaches are possible

- **Dedicated logic styles**
- **Filter the power consumption**

More on this later

The techniques that are effective vary depending on whether the crypto algorithm is running in **software** or by dedicated **hardware** engines

For **software implementations**, the options available to alter the power traces are limited

For the *time dimension*, the most common option is to randomize the algorithm's execution

Both *shuffling* and *dummy delay insertion* are possible in software

However, these techniques require *random numbers*, that are typically generated by a crypto chip

Software Countermeasures

For the *amplitude dimension*, a choice exists with regard to which **instructions** are used to carry out the algorithm

SPA can be prevented by selecting these instructions carefully, e.g., low-leakage instructions

Unfortunately, this is usually not sufficient for DPA

Beyond detecting specific instruction executions, attackers can also detect

- Changes in **program flow**, e.g., conditional jumps and repeated basic blocks

  Conditional jumps that depend on the key (or data related to the key) should be AVOIDED!

- Memory addressing that depend on the key

  If there is a dependency, then memory addresses with the same Hamming weight should be used

Obscuring the signal can also be accomplished in software implementations by adding parallelism, e.g., using a coprocessor or communication interface

Hardware Countermeasures

For **hardware implementations**, there are significantly more options available

At the *architecture level*, in the *time dimension*,

- Dummy operations and shuffling are possible
- Operations are performed randomly (using a random number generator) using dummy data in extra registers
- Randomly **skip** clock pulses, **change** clock frequency, and use **multiple** clock sources

In any of these strategies, it is important that the attacker is not able to identify the insertions and modifications (obviously)

At the *architecture level*, in the *amplitude dimension*,

- Power consumption can be made *equal* by **filtering**

  A filter, e.g., switched capacitors, is placed between power pins of chip and the internal hardware crypto circuit
- Generate **noise** in parallel with the computation, using random number generators connected to a network of large capacitors for random charging and discharging

Hardware Countermeasures

It is important to realize that the measurement strategy adopted by the attacker impacts the effectiveness of these countermeasures, particularly noise generators

Power can be monitored by

- Small inserted resistors in series with the power supply connections
- Current probes
- EM emanation

To deal with the latter, it is important to spatially distribute the noise sources on the chip

Cell level hardware strategies

Countermeasures taken at the *cell level* were quickly adopted by industry after SCA attacks became widely known

The basic idea is to make **power consumption constant**, and independent of the data being processed in each clock cycle

This effectively means that power in each clock is set to the max value

DPA-Resistant Logic Style Countermeasures

Many approaches included here are classified under **dual-rail precharge** (DRP) logic styles

Dual-rail implies that two (matched) wires are used to implement connections, one carrying a *non-inverted* value and the other the *inverted* value

Also called *differential encoding*

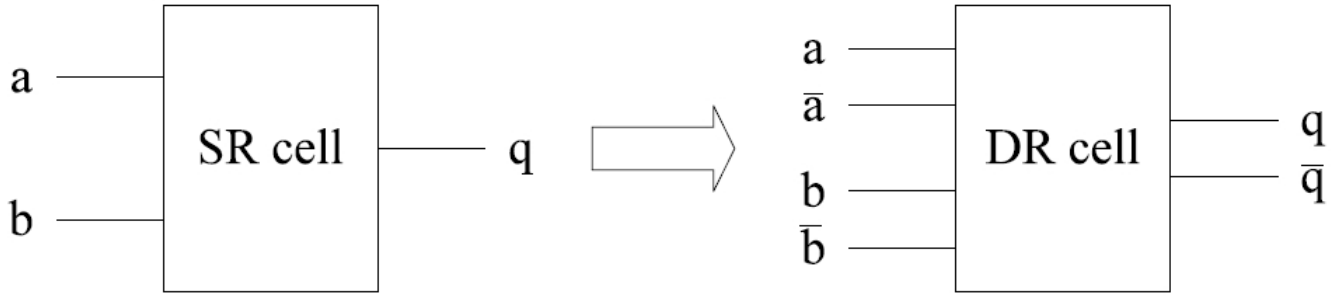


Figure 7.1. A 2-input SR cell and a corresponding 2-input DR cell.

Precharge implies the circuit operating in a *precharge-evaluate* fashion, under control of the clock

In the precharge phase, all signal wires are changed to a constant value

In evaluate, data is processed

DPA-Resistant Logic Style Countermeasures

In DRP circuits, assuming the precharge state is 0, only one of the complementary signals transitions from 0 to 1, the other one remains at 0

DRP FFs

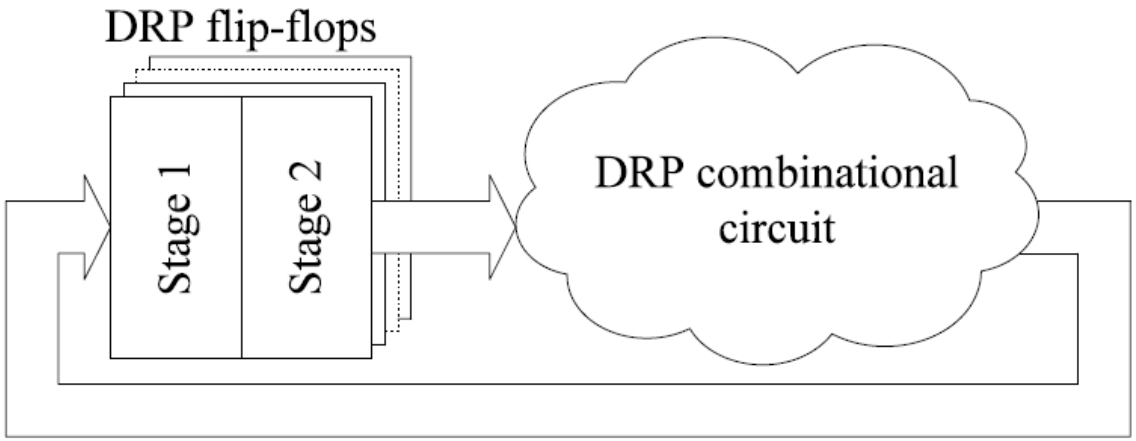


Figure 7.2. DRP flip-flops consist of two stages, which are precharged alternately.

DRP FF have two stages

Stage 1 is in *precharge*, when stage 2 (and the DRP combo circuit) are in *evaluate*

During precharge of stage 2, the stage 1 FFs store the logic values

DPA-Resistant Logic Style Countermeasures

It is important to **balance the capacitive loads** on the outputs of the complementary signals (power consumption is proportional to these capacitances)

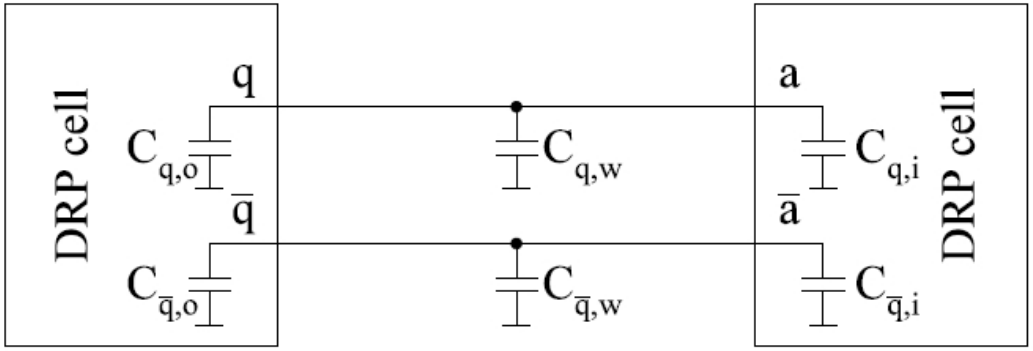


Figure 7.3. The capacitances at the complementary output of a DRP cell.

From the figure, this requires balancing the self-capacitance, wire and downstream load capacitances

Note that wire capacitances can dominate in modern technologies, so special routing is required

Careful design of the DRP cells ensures that *self* and *load* capacitances, as well as internal nodes, are balanced

DPA-Resistant Logic Style Countermeasures

Standard-cell design strategies are typically used to automate the chip generation process

Behavioral synthesis tools can **not** work with DRP -- instead, a '**single rail**' (SR) design is processed and the netlist/layout updated to dual rail afterwards

Also, typically, a single-rail external interface is usually required (plaintext inputs are usually provided in non-complimentary forms)

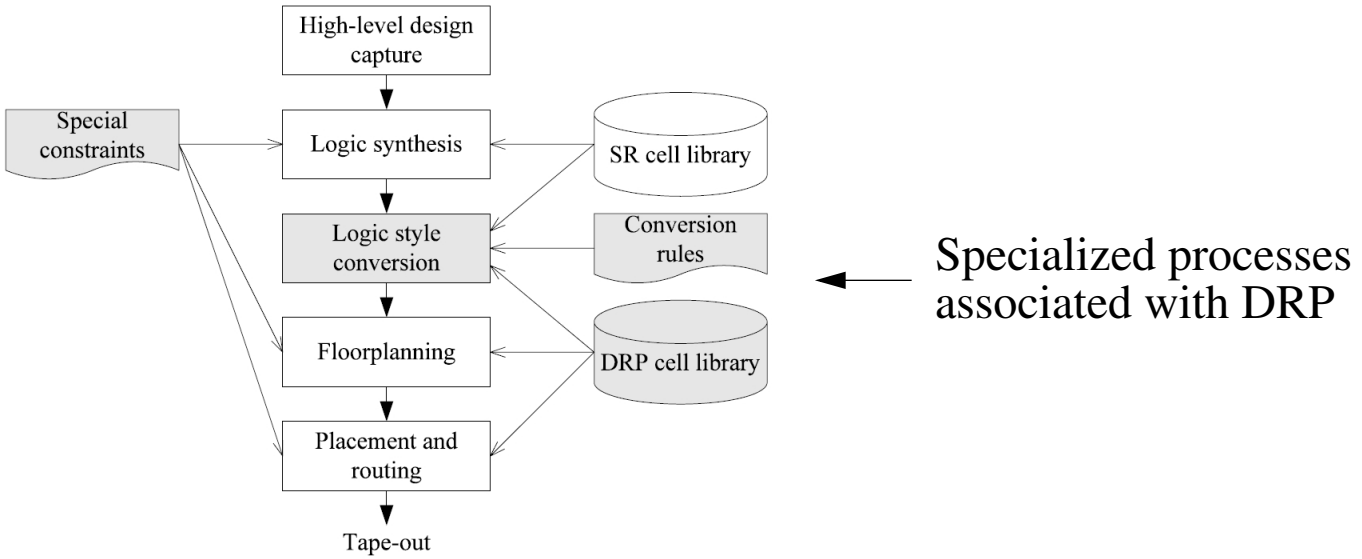


Figure 7.4. Semi-custom design flow using DRP logic styles.

DPA-Resistant Logic Style Countermeasures

High-level synthesis

Concern here is to prevent any sensitive *intermediate* results from leaving the crytpo chip (Trojans?)

Logic synthesis

Constrain SR cell library to cells available in DRP library

Logic style conversion

Perform cell substitution, addition of complementary signal nets (on all except clock and asynch reset signals) + SR interface

Note that SR inverters are removed -- not needed b/c DRP provides inversion for free

Place and Route

Must ensure capacitance (and resistance) are matched

In practice, this is **impossible** because of coupling caps and process variations

DPA-Resistant Logic Style Countermeasures

Place and Route

Two proposed methodologies for place and route are **differential routing** and **backend duplication**

For the former, the wires are routed in parallel, for the latter, two identical copies of the layout are produced, one fed with complimentary signals

Third method is route 'fat' wire and split it into true and complementary

Example DRP Logic Styles

- Sense Amplifier Based Logic (SABL)
- Wave Dynamic Differential Logic (WDDL)

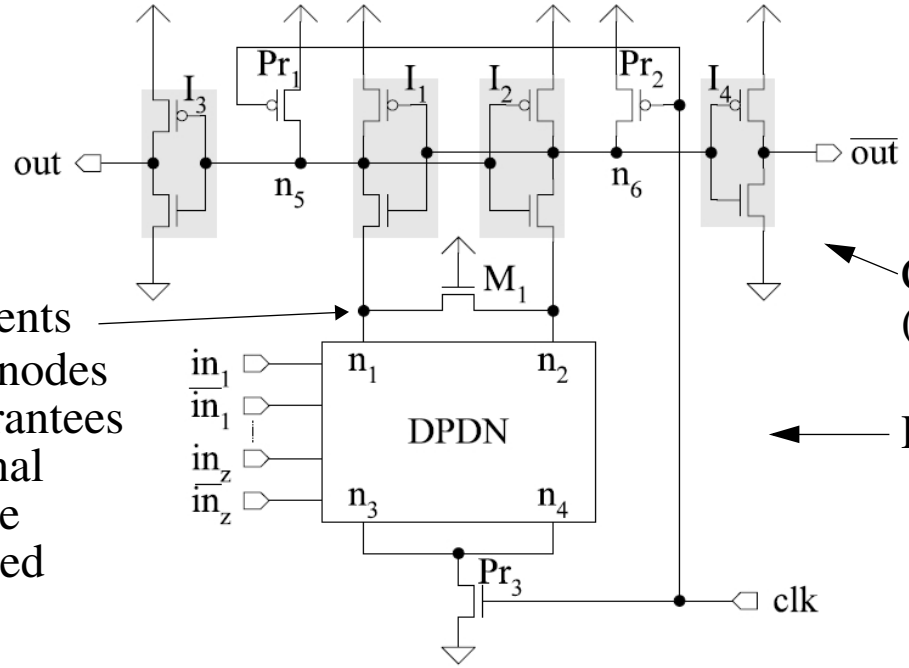
Sense Amplifier Based Logic

SABL cells are designed such that their time-of-evaluation (TOE) is data independent

That is, SABL cells evaluate only after all input signals have been set to complementary values

SABL

All SABL cells are connected to the clock signal, and all are precharged simultaneously



n-type SABL cell

M_1 prevents floating nodes and guarantees all internal nodes are discharged

Cross-coupled inverters (sense amplifier or SA)

Differential pull-down network

During precharge, all cells output 0 on both outputs, preventing downstream evaluation of other logic gates

Figure 7.5. Transistor schematic of a generic n-type SABL cell.

Evaluate enables Pr_3, creating a pull-down path on n_1 or n_2, setting the state of SA

Note that all internal nodes that are **discharged** during evaluate are **precharged** during the precharge stage b/c the inputs do NOT instantaneously change

SABL

Other requirements for SABL

- All internal nodes of DPDN are connected to one of the four output nodes, n_1 through n_4

  This ensures all internal nodes of the DPDN are discharged to 0 (except either n_5 or n_6) during the evaluate phase and charged to 1 during precharge
- All conducting paths have the same resistance
- Both wires on every complementary input wire must be connected to the same number of gate terminals
- A conducting path through the DPDN may only occur after ALL input signals have settled

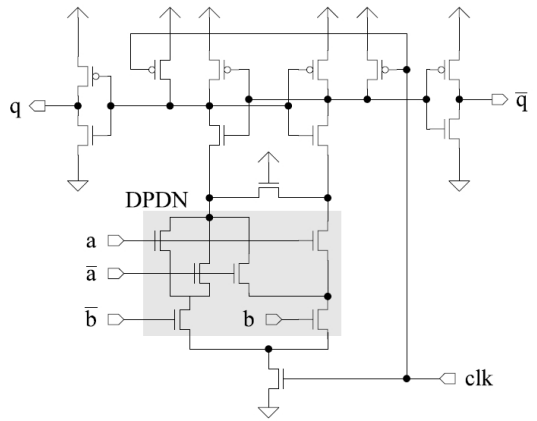


Figure 7.6. Transistor schematic of an n-type SABL NAND cell.

NOR gate implemented by inverting input signals *a* and *b* and the output signal *q*

SABL

N-Type SABL D-FF

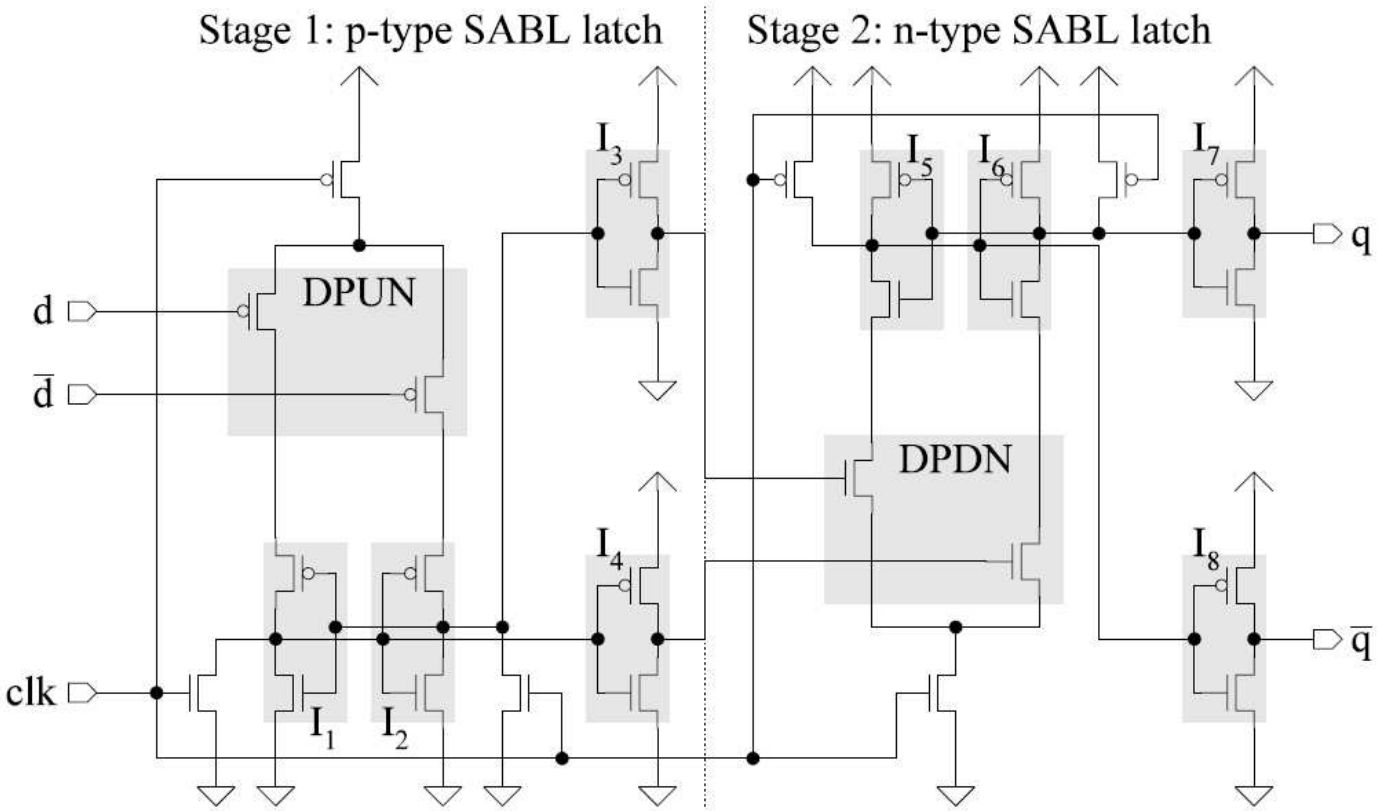


Figure 7.7. Transistor schematic of an n-type SABL D-flip-flop.

For SABL, area and clock cycle time are at least doubled, and power requirements is significantly increased

WDDL

Wave Dynamic Differential Logic

Unlike SABL, WDDL cells are based on single rail cells that are available in standard cell libraries, and are much simpler

Cost is that WDDL is less DPA-resistant than SABL

Their internal power consumption and their TOE are data dependent

Structure of a combinational WDDL cell

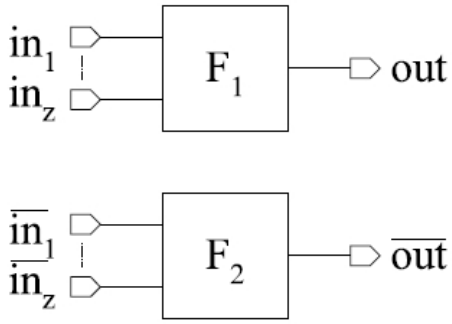


Figure 7.8. Generic structure of a combinational WDDL cell.

The two circuits that must satisfy the following requirement

$$F_1(in_1, \ldots, in_z) = \overline{F_2(\overline{in_1}, \ldots, \overline{in_2})}$$

WDDL

F_1 and F_2 must be **positive monotonic** Boolean functions

Positive monotonic Boolean functions exhibit a **single output transition**, and both the inputs and output change **in the same direction**

During **evaluate**, only one output of the complementary pair will have a transition, producing a *wave* that propagates through the combinational logic

A similar behavior occurs during precharge

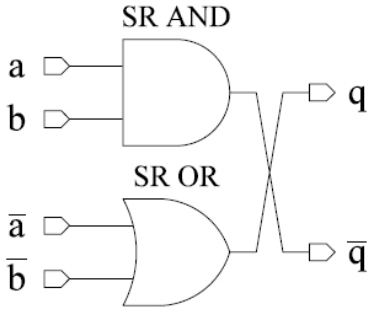


Figure 7.9. Cell schematic of a WDDL NAND cell.

Table 7.3. Truth table of a WDDL NAND cell for complementary input values.

a	b	$\overline{a}$	$\overline{b}$	$q = OR(\overline{a}, \overline{b})$	$\overline{q} = AND(a, b)$
0	0	1	1	1	0
0	1	1	0	1	0
1	0	0	1	1	0
1	1	0	0	0	1

During precharge, ALL inputs, e.g., a and $\overline{a}$, are set to 0, and so both outputs are 0

During evaluate, inputs are set to complementary values and 1 output transition occurs