

- 5 - Cryptographic Key
  - Block cipher
  - Data Encryption Standard (DES)
    - \* Triple Data Encryption Algorithm - TDEA
    - \* DES Feistel Structure
    - \* Software
  - Mode of Operation
    - \* Design method
    - \* Cipher Block Chaining - CBC
- 6 - Countermeasures against physical attacks
  - The basis
  - Countermeasures
    - \* 1 - Re-keying
    - \* 2 - Masking
    - \* 3 - Hiding
  - Analog circuits and noise generators
    - \* Decoupling
    - \* Detach
    - \* Active flattening
  - Noise generators
  - Masking
    - \* Boolean masking
    - \* Other types
- Students Presentation
  - 2 - Low Cost and Precise Jitter Measurement Method for TRNG Entropy Assessment
    - \* Theoretical error
    - \* Questions :
  - 3 - Contactless Electromagnetic Active Attack on Ring Oscillator Based True Random Number Generator
    - \* Injection Platform
    - \* Question

## 5 - Cryptographic Key

### Block cipher

Def : “A block cipher breaks up the plaintext into strings (called blocks) of a fixed length  $t$  over an alphabet  $A$  and encrypts one bloc at a time.”

It will repeat on it we call this rounds and each rounds has subkey derived by key schedule.

Often, one cycle per round for HW architecture to ensure speed and throughput. On the other side we can make low area which is slower.

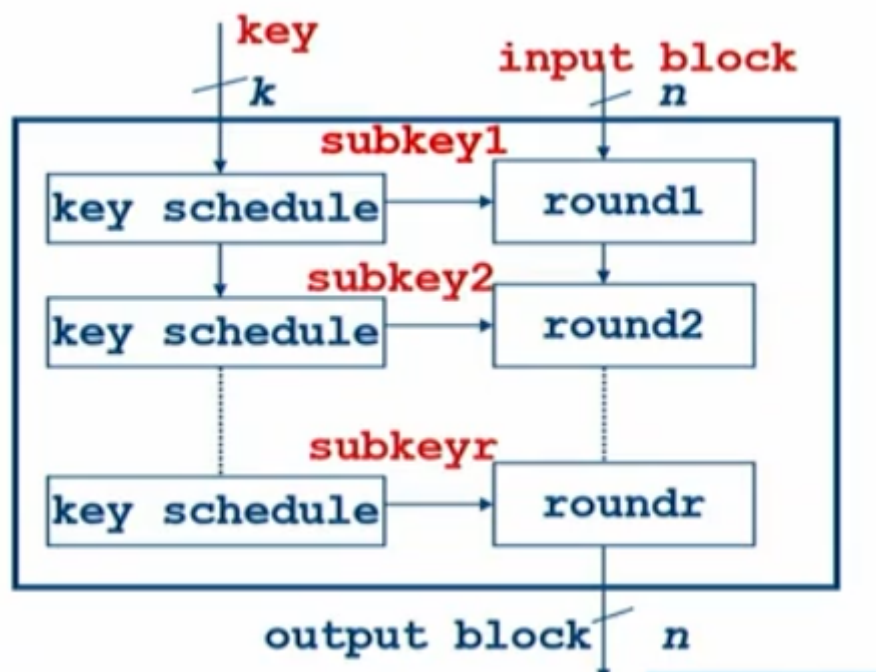


Figure 1: Block cipher example

## Data Encryption Standard (DES)

It is a *block cipher* with 64 bit I/O and 56 bit key with 8 parity bits. The idea, it is iterated cipher with 16 rounds. It has influenced modern encryption even though it is no longer considered secure as of 2004.

### Triple Data Encryption Algorithm - TDEA

We have 3 key options :

1. K1, K2, K3 different
2. K1=K3, K2 different
3. K1=K2=K3 which makes it *backward compatible with DES*

The two-key triple DES is recommended until 2009 and three-key triple DES until 2030. Still used a lot in the payment industry.

In 2017, we limit the max block size to  $2^{30}$  and disallows its usage for TLS, IPSEC, ...

### DES Feistel Structure

It is still heavily used in reality.

The encryption and decryption is the same function so it is super hardware efficient !

In the 16 rounds we have an initial and final permutation.

Expansion will expand and reshuffle the bit. We then have 8 S-box with 6 inputs bit and 4 bits output. S<sub>i</sub> non-linear substitution.

We have some needed linearity to have good and robust encryption. But too much non-linearity is costly on hardware.

We have then the key schedule, with a 64 bits key input but we will only have 56 used

We rotate in those C and D register and it depends on the round we are in that decide how much we shift.

The key never really change so we prefer in SW to first compute them and store in memory to easily access it.

If we want to store the key in memory we have one 128 bit key which is 1208 bits round keys so 10 rounds and initial key. But half of internet packets are only 64 bytes in length (512 bits) it doesn't make any sense anymore. It had too much overhead to load and forget keys of every users.

DES was originally developed for efficiency in HW and is quite unefficient in SW.

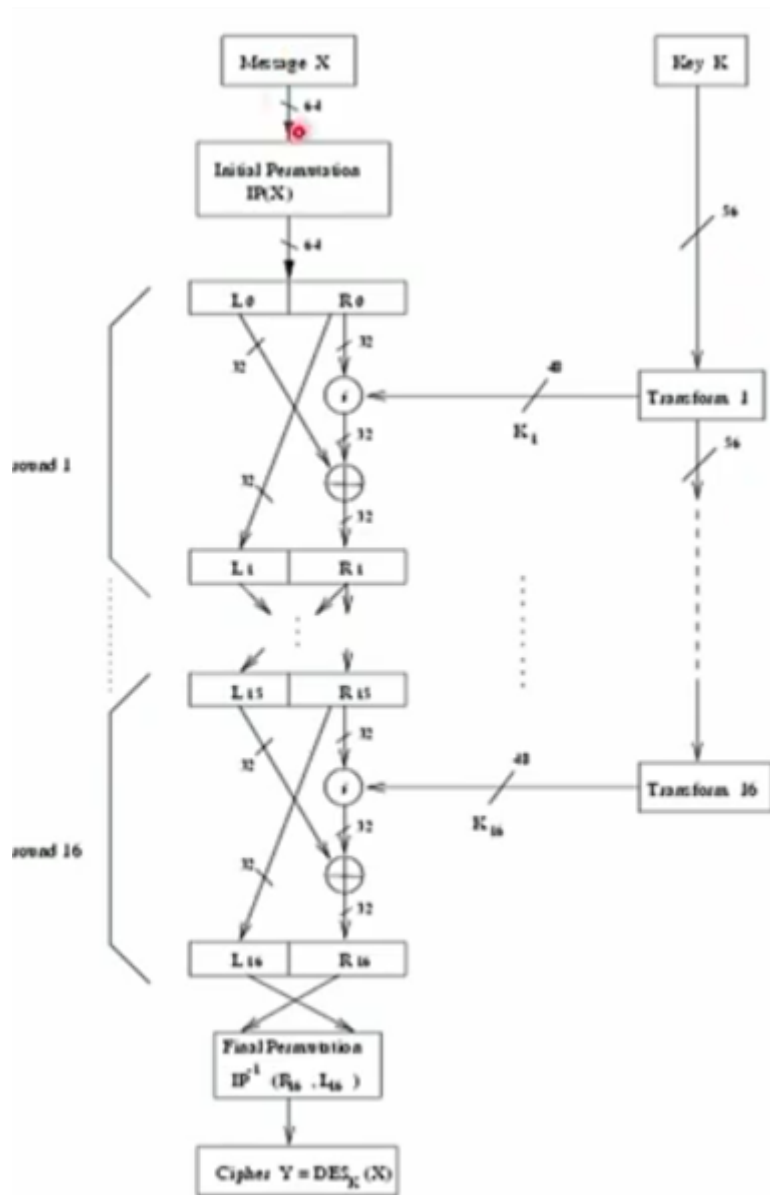


Figure 2: Des Feistel Structure

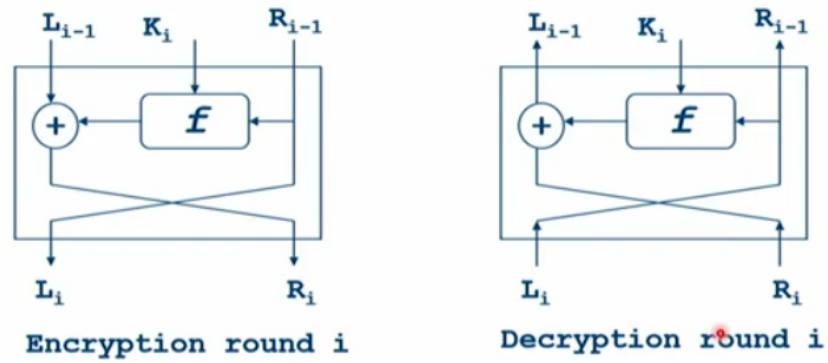


Figure 3: DES hardware

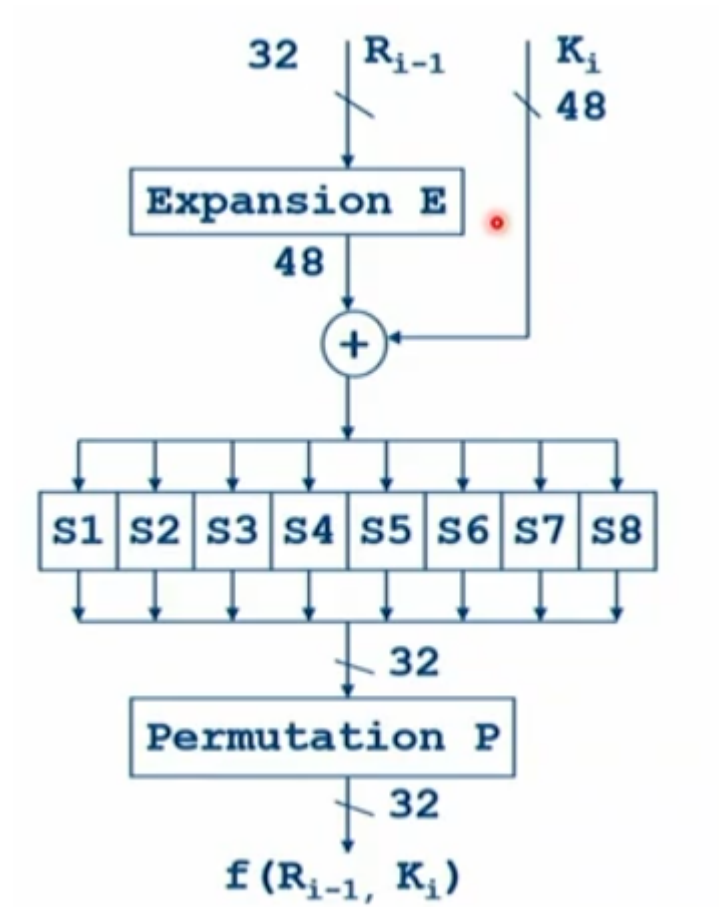
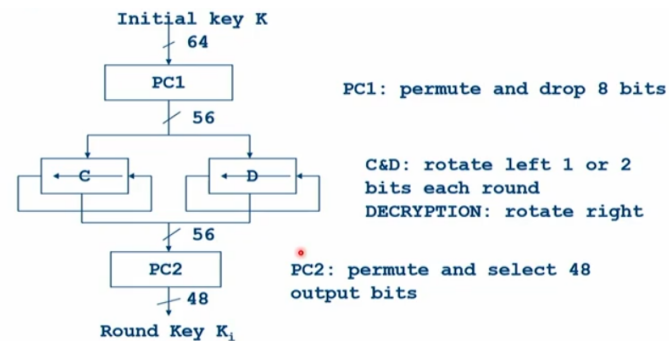


Figure 4: DES-f function

		x0000x					x0111x					x1111x					
	$S_1$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0xxxx0	0	14	04	13	01	02	15	11	08	03	10	06	12	05	09	00	07
	1	00	15	07	04	14	02	13	01	10	06	12	11	09	05	03	08
	2	04	01	14	08	13	06	02	11	15	12	09	07	03	10	05	00
1xxxx1	3	15	12	08	02	04	09	01	07	05	11	03	14	10	00	06	13

The S-Box  $S_1$  (6x4 bit look-up table)

Figure 5: S-box

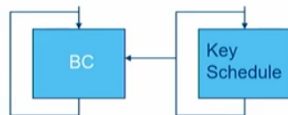


**C&D left/right shift registers: encryption & decryption HW**

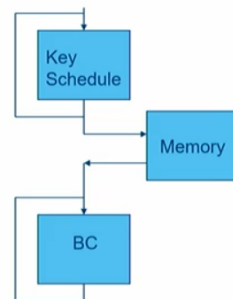
Figure 6: DES Key schedule

Two options:

- HW = on the "fly" = just in time processing
- SW = pre-compute and store



Typical for Hardware



Typical for Software

Figure 7: On the fly in hardware

## Software

We need to do many permutation of the plain text input. The issue is that it is **bit oriented** so it is annoying to use mask and move it. In ASM we don't have bit-wise operation. Cheap in HW (simple wire) expensive in SW.

The simple way takes around 300 instructions per block !

**Bit slicing : alternative data representation** Each register contains 1 bit of eg 32 blocks. Block size is defined by algorithm for DES block is 64 bit. We are going to parallelize of n encryptions. Number of blocks in parallel  $n = \text{width of register}$ .

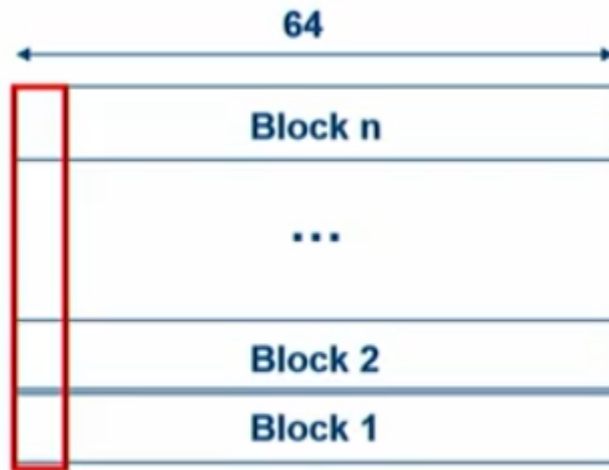


Figure 8: Operating First bits of multiple block

So in a register we have all the first or n bits of all the blocks etc. So now the CPU can be viewed as 16/32/54 one-bit parallel processors (depending of the size of the inputs). CPU is like a Single-Instruction Multiple-Data SIMD processor.

And now, for permutation we can easily copy the content of a register to another, it is no longer bit-wise operation. Easy !

## Mode of Operation

The mode of operation tells what to do when we have **multiple block of data**. If we are simply following the block in the normal way, we could actually reveal

the data and not encrypt it correctly. The issue is if we encrypt “A” it will have the same encrypted version for any encrypted “A”.

### Design method

Inclue modes of operation into hardware IP module or co-processor. It gives more hardware but more clean security partitioning, reduces communication overhead and traffic.

### Cipher Block Chaining - CBC

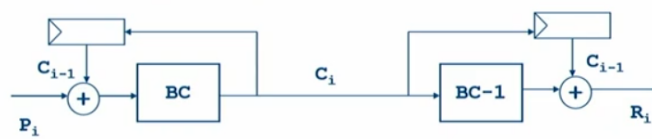


Figure 9: CBC

Error in  $C_i$  propagation over 2 blocks ! If we have a loss of block synchronization it is fatal. If we have an error in  $P_i$  we will propagate it to the other blocks. It is mostly used with encryption only for Message Authentication Encryption (MAC) generation.

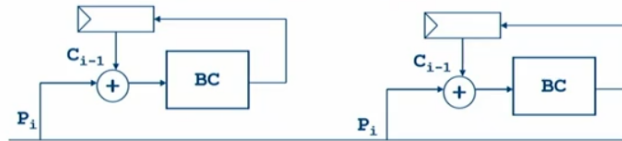


Figure 10: CBC-MAC

**CBC-MAC** Feedback inhibits pipeline. But due to feedback we can’t easily pipeline it. It gets even worse for triple DES. Worse for bit slicing and certain masking schemes.

**Modes of operation counter** Add confidentiality and encryption.

The third column indicates how we encrypt or decrypt.

The counter mode, we don’t have any pipelining anymore.





	Sender	Receiver	Pipelining	Notes
ECB	Enc	Dec	Yes	not used
CBC	Enc	Dec	No	
CBC-MAC	Enc	Enc	No	
CFB	Enc	Enc	No	
OFB	Enc	Enc	No	Message authentication
CTR	Enc	Enc	Yes	
OCB	Enc	Dec	Yes	
CCM	2Enc	2Enc	Only CTR	

- Conclusion: most practical applications ONLY use encryption. Important for:
  - AES because encryption is more efficient than decryption (non-Feistel)
  - area constraint applications (e.g. IEEE 802.11)<sup>32</sup>

Figure 13: Block cipher modes of operation

## 6 - Countermeasures against physical attacks

To develop good countermeasure, we can't just focus on one area and then think it is good enough. No we need to add many layers of protection at all levels to be more resilient against attacks, we are applying it to **all level of abstraction**.

There is 2 dominant philosophies:

1. **Bottom up**: if we protect a low level component then all the higher one are resilient and safe
2. **Top down**: if one component can tolerate side-channel leakage of lower components, then the lower components need no protection.

We are often limited by the “what” since we operate on input and output and also by the “how” aka the level we are working at. We can't simply change protocol and primitives as they are fixed.

### The basis

To know if something is vulnerable, we need to have:

1. Sensitive data
2. Processed values that uses the sensitive data
3. Physically observable operation (power, sidechannel, ...)

For this we have multiple ways to prevent ourselves:

- Masking: decorrelate the sensitive data and the data processed
- Hiding: we are using lower SNR to make the recovery harder for an attacker
- Provable secure countermeasure: usually rely on masking but hard to validate and easy to invalidate

To know if we are at risk, we need to see if some intermediate data depends on a secret and get be guessed with some secret found by the attacker. We have a leak of information.

## Countermeasures

We have 3 main CM

### 1 - Re-keying

We change the key every few often so the attacker doesn't have enough sample to find the key. We also need to make sure that the re-keying is protected and some protocols don't support this.

But it is not always enough as some protocols just need one sample to be broken.

### 2 - Masking

We make intermediate values independent of the secret. We are protecting ourselves to low-order attacks but it gets harder and harder for high-level one.

We only need to store masked sensitive data ! Masking make the data unpredictable. But can be quite challenging to implement it and not accidentally leak or have some glitches

### 3 - Hiding

We try to reduce the SNR. We can either reduce the signal using some encoding or decoupling technique or increase the noise using some additional circuitry and doing some timing jitter that is controlled in software. Those effects shouldnt be undoable.

We can also make the operations always look the same or add dummy operation to not reveal the branch we are currently on.

We need to be prepared to fail as no CM is perfect.

## Analog circuits and noise generators

We want to hide the power consumption, the ideal scenario would be a power supply that is isolated and cannot be seen or measured by the attacker. But we can't ofc.

### Decoupling

We had a capacitor between  $V_{DD}$  and ground to smooth out peaks in the power consumption. But we can't perfectly smooth things out and the cap cannot be infinitely large.

## Detach

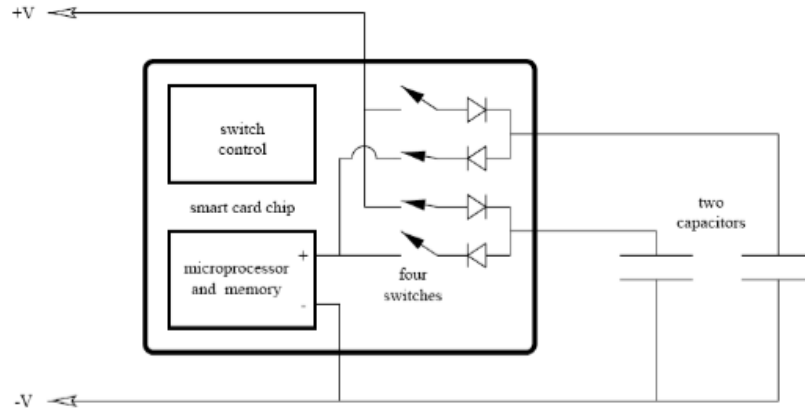


Figure 14: Detached Power supply

The idea is to have one capacitor that charges while the other is supplying the current. Not perfect because we will see peaks in the toggling of the cap which also leak information.

So the best thing is to supply the current through a cap, discharge it completely and then charge it again so we have a constant power consumption.

## Active flattening

We add a sensor that measures the supply current and we have a sink that helps having a constant power and current consumption using this sink. But this means we will have maximum current consumption all the time.

It is never perfect and even GPIO pins leak information. We are not protected against EM side channel attacks with those techniques.

## Noise generators

The noise must come from an independent noise source. We gotta randomly charge and discharge a capacitor. We can also activate some unused co-processors. Both methods rely on using a source of randomness.

## Masking

We know that intermediate data will be a function of a key and a sensitive data:

$$x = f(p, k)$$

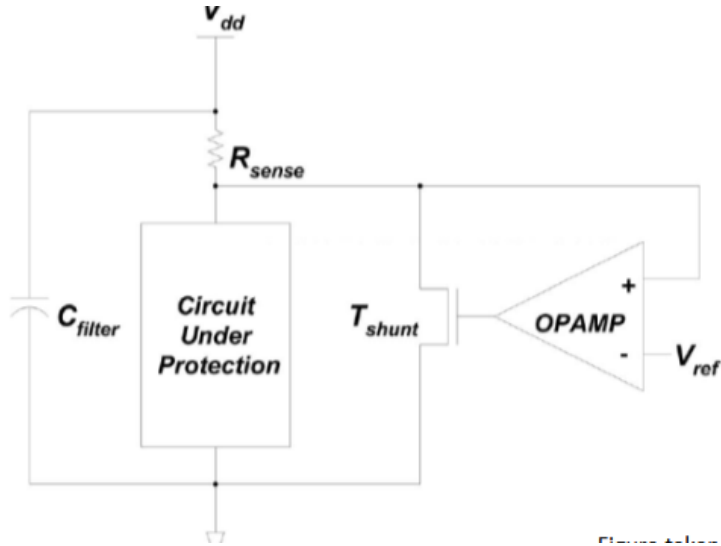


Figure 15: Active flattening

But with masking, we are creating  $d$  shares which  $d \geq 1$ .

$$v = m_1 \cdot m_2 \cdot \dots \cdot m_d \cdot x$$

Here, “.” represents a type of masking. The most basic one is the **boolean masking**

### Boolean masking

It uses a xor operation  $\oplus$ , it is linear which makes linear operations easy to mask. Non-linear operations are harder to mask.

### Other types

We can also have the multiplicative one where  $\otimes$ (mult in  $\text{GF}(2^8)$ ). It is really used for S-box in AES, but we need to be careful cause we can't mask a 0 with this type of masking.

We can also use modulo addition where  $+$ (addition mod  $n$ ). Used for the first versions of SHA and now in Post Quantum Cryptography.

With the  $\oplus$  we can recover the secret after reapplying the mask. In general, we have a vector and mask pair.

We can't do masking before going into a S-box since they are non-linear. We have to create a new S-box  $S'$ . We have to recompute based on our S-box the new  $S'$ -box based on the new mask.

## Students Presentation

Here is the compilation of all students questions and prepared answer

### 2 - Low Cost and Precise Jitter Measurement Method for TRNG Entropy Assessment

We have issues of drift due to jitter, duty cycle can slightly change. It is pretty volatile cause the source of noise varies. To measure the jitter, we will avoid using probes etc since it can be worse. Use a counter:

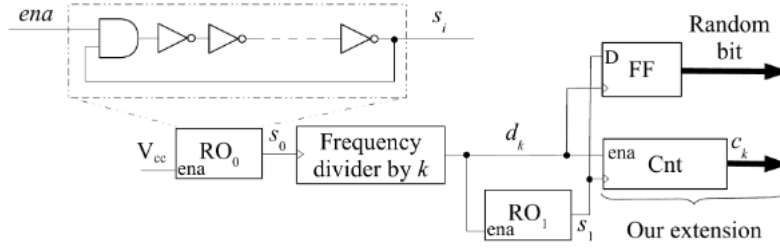


Figure 16: Counter to check jitter

New method proposed:

#### Definition

- ▶  $c_k$  number of rising edges during  $kT_0$
- ▶  $N$  number of experiment runs
- ▶  $C_k = \{c_k\}_{i=1, \dots, N}$
- ▶  $F_k$  most likely outcome of  $c_k$  in  $C_k$
- ▶  $M_{(k,N)} = \#\{c_k \in C_k | c_k = \max(C_k)\}$

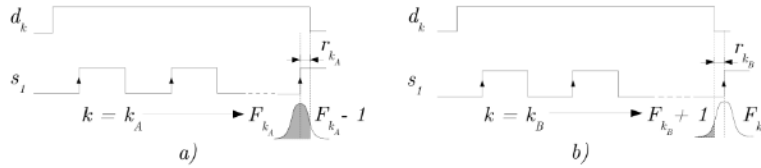


Figure 17: New method

We most likely observes the  $k$  clock cycles as expected

$$\varphi_0 + (F_{k_A} - 1)T_1 + r_{k_A} = k_A T_0.$$

$$c_k = F_{k_A} \Leftrightarrow q_{F_{k_A}} \leq r_{k_A},$$

$$Pr(c_k = F_{k_A}) = Pr(q_{F_{k_A}} \leq r_{k_A}).$$

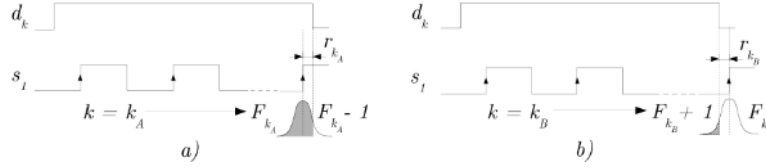


Figure 18: 2 cases

### Theoretical error

$F_k$  is the most probable outcome.

### Questions :

“Why is a short accumulation time a desirable property of a jitter measurement technique?”

## 3 - Contactless Electromagnetic Active Attack on Ring Oscillator Based True Random Number Generator

It is about TRNG that are based on ring-oscillator. It generates a jittery clock. The paper will focus on attacking the source of attack, it is an active attack.

Inject EM harmonic signal to bias that source of entropy. They use a micro-probe for it. The TRNG is implemented on a FPGA board.

### Injection Platform

- Power injection chain
  - Inject wave of different power, with a frequency close to the RO (300-325 MHz)
- Control chain
  - Check when shielded from EM. Try with and without EM injection. Store and compare TRNG output bitstream.
- data acquisition chain

They use some powermeter and oscilloscope.

They used DFTRi (discrete fourier transform ratio). They check the difference between the power at the injected frequency and the power of the output RO frequency. Higher DFTRi means higher effective attack.

OFC, the higher power we inject the more effective it is.

We can use some circuitry to make it dynamic attack.

### **Question**

“The attack demonstrates that ROs can be locked onto an injected frequency, leading to a biased TRNG output. How can mutual information be used to assess the effectiveness of this attack ?”