    - Countermeasures - DDR4
    - Countermeasures - DDR5
  - Conclusions
- Students Presentation
  - 2 - Low Cost and Precise Jitter Measurement Method for TRNG Entropy Assessment
    - Theoretical error
    - Qestions :
  - 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.

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 tough 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 b lock size to 2^{30} and disallows its usage for TLS, IPSEC, . . .

DES Feistel Structure

It is still heavly used in reality.

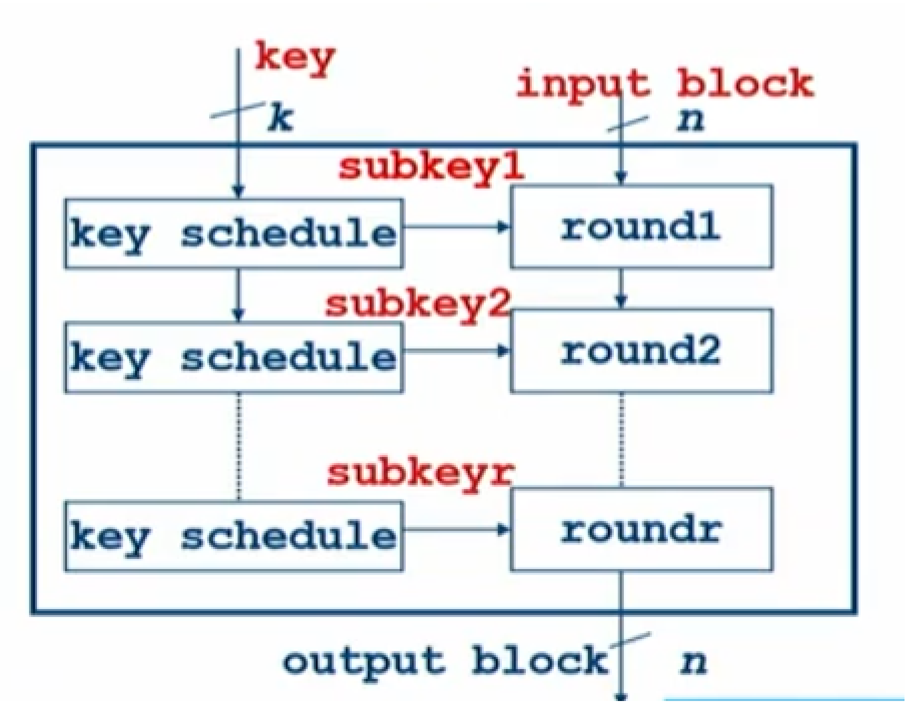


Figure 1: Block cipher example

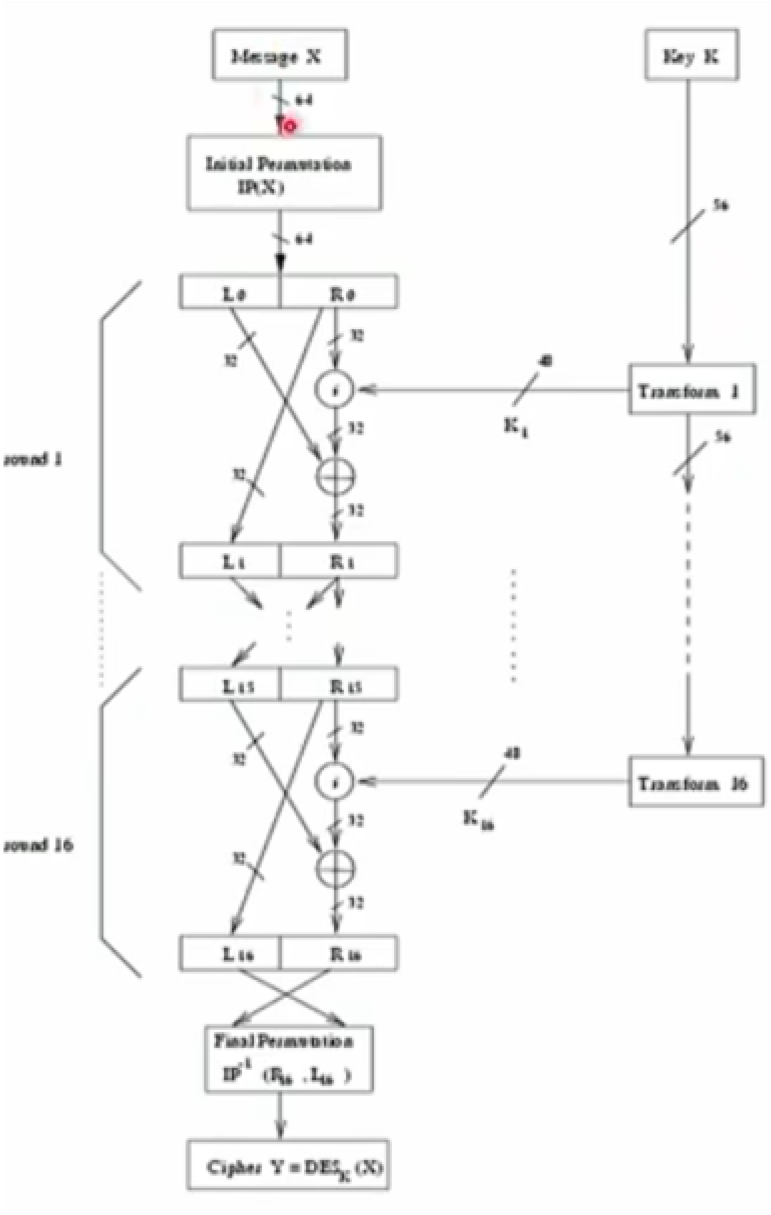


Figure 2: Des Feistel Structure

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

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

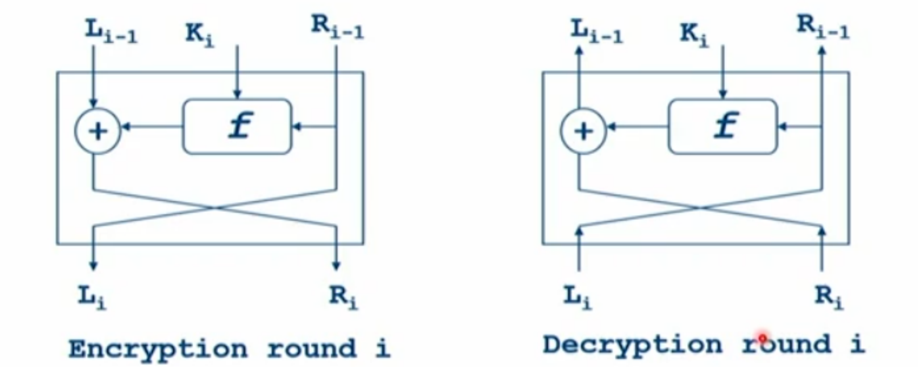


Figure 3: DES hardware

Expansion will expand and reshuffle the bit. We then have 8 S-box with 6 inputs bit and 4 bits output. S_i non-linaear subsitution.

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

We havec 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.

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 !

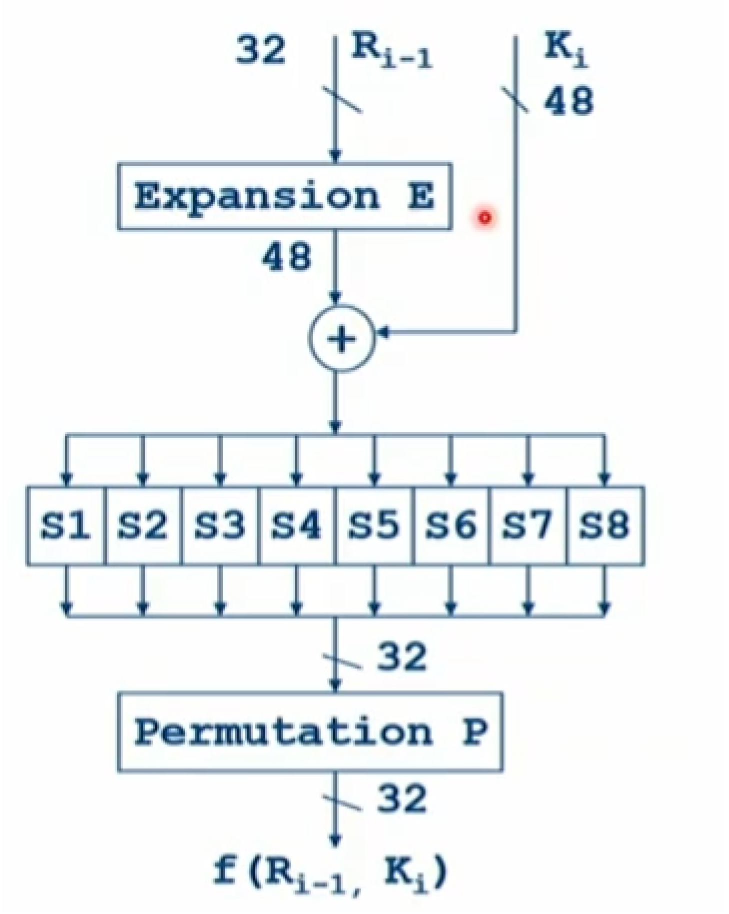
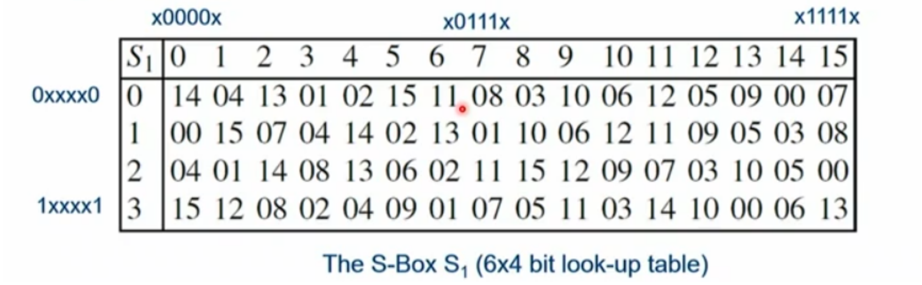


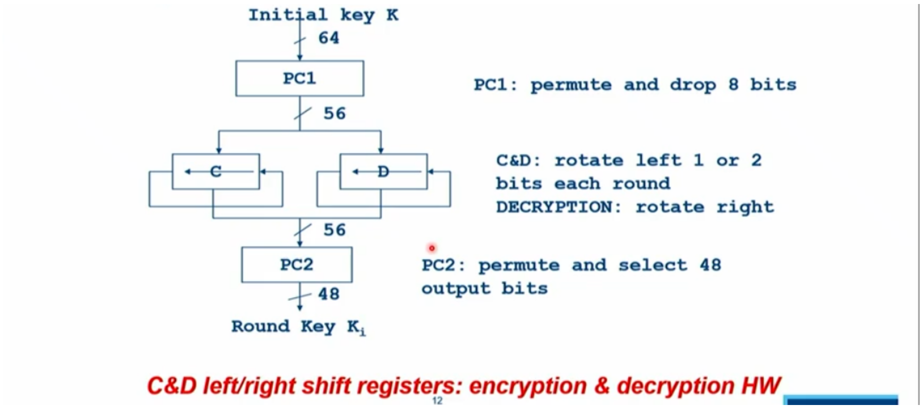
Figure 4: DEs-f function


x0000x
x0111x
x1111x
0xxxx0
1xxxx1

S_1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
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
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



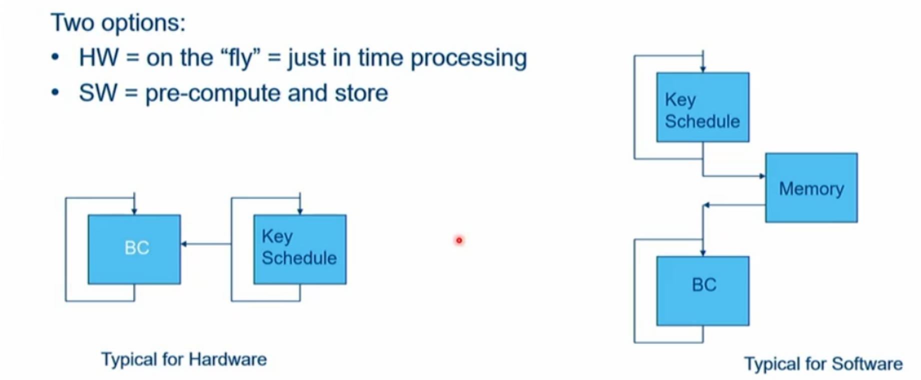
PC1: permute and drop 8 bits

C&D: rotate left 1 or 2
bits each round
DECRYPTION: rotate right

PC2: permute and select 48
output bits

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

Figure 7: On the fly in hardware

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 parellelize of n encryptions. Number of blocks in parallel n = width of register.

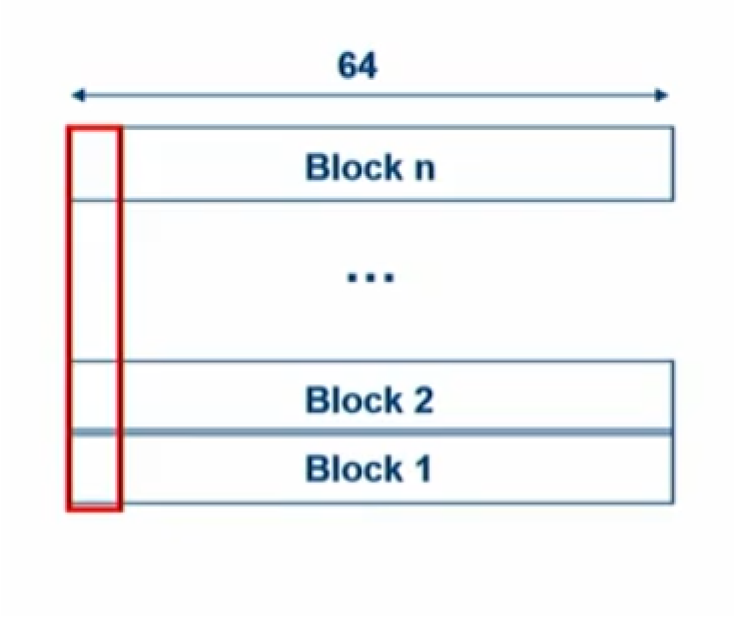


Figure 8: Operating First bits of multiple block

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

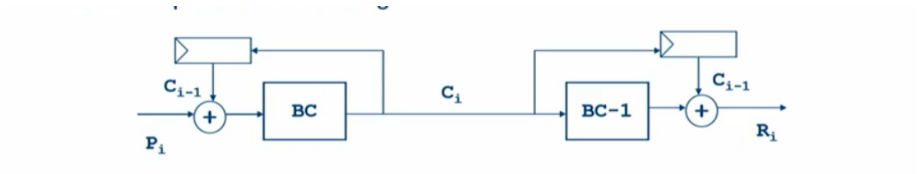


Figure 9: CBC

Error in Ci 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 Authentification Encryption (MAC) generation.

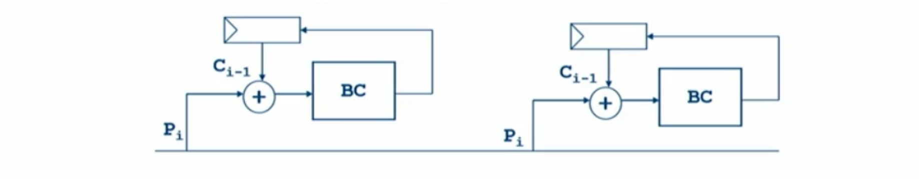


Figure 10: CBC-MAC

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.

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.

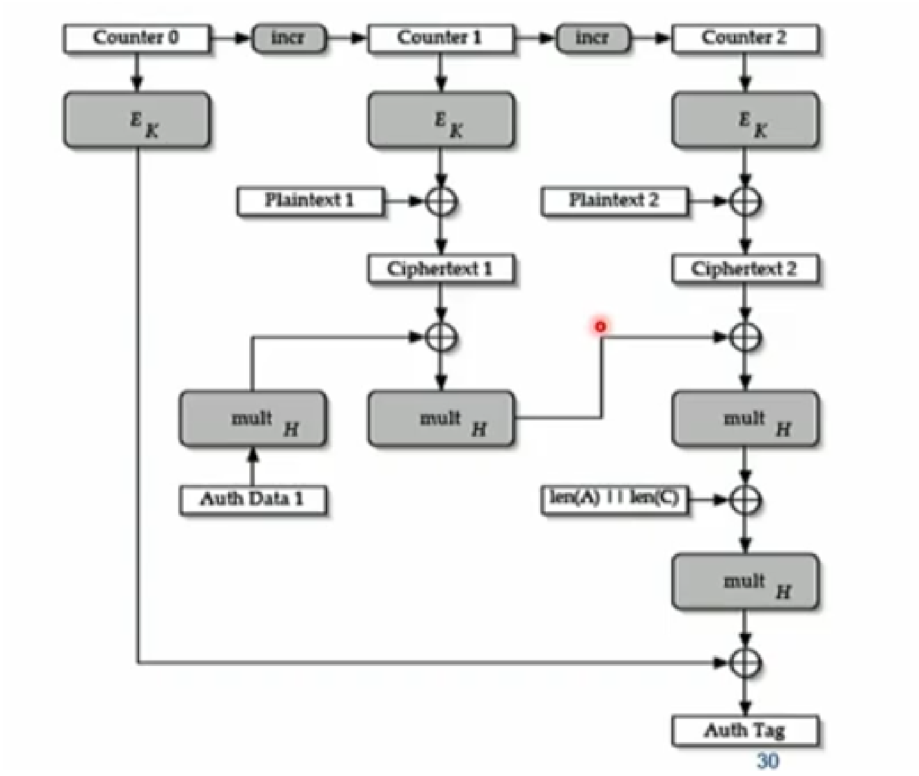
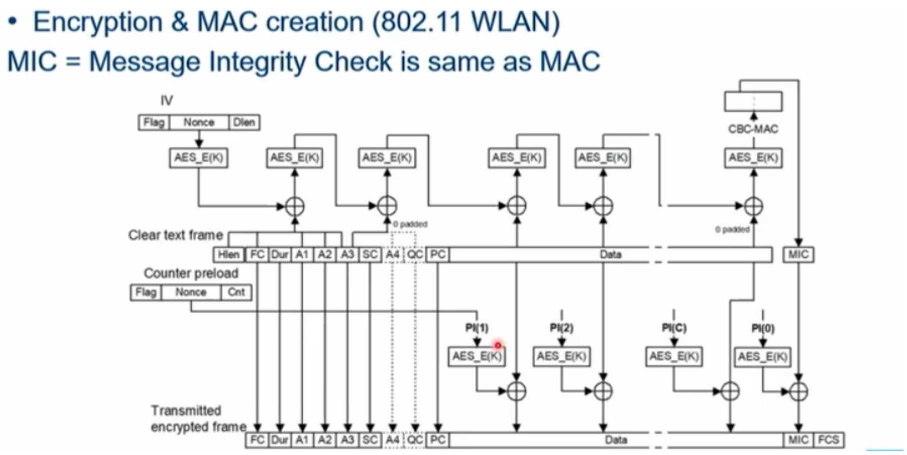


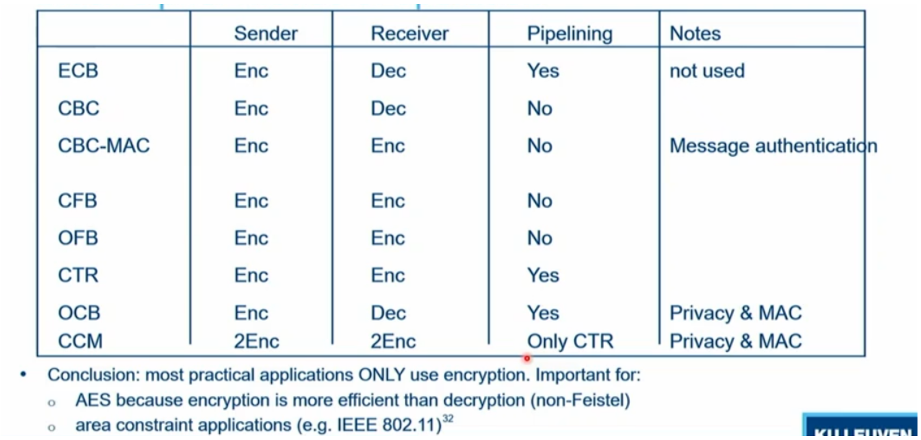
Figure 11: Galois counter mode



- Encryption & MAC creation (802.11 WLAN)

MIC = Message Integrity Check is same as MAC

Figure 12: CCM (counter + CBC MAC) mode



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

- 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)[32]

Figure 13: Block cipher modes of operation

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 circuity 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

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.

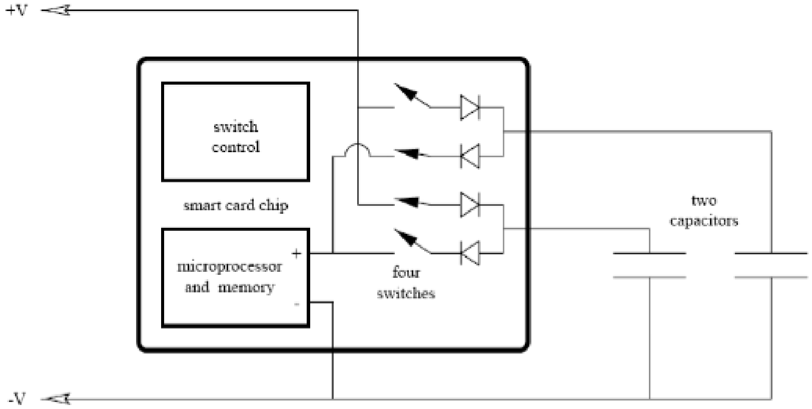


Figure 14: Detached Power supply

Active flattening

We add a sensor that measures the supply current and we have a sink that mhlps 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:

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

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

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

Here, "$\cdot$" represents a type of masking. The most basic one is the **boolean masking**

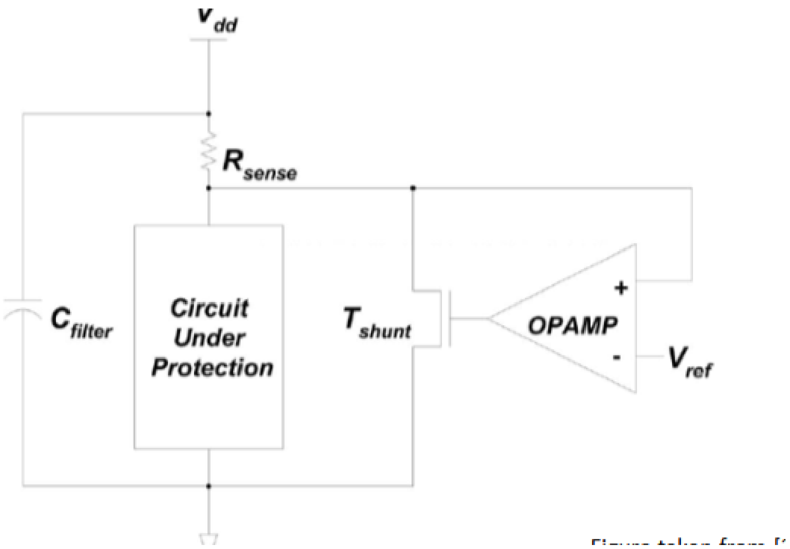


Figure 15: Active flattening

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 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.

7 - Trusted Computing

It is the **assurance on running code**, we need to prove its identity, authenticity and secrecy. Making sure to keep sensitive data private. This is enable through **hardware support**.

Trusted Computing all rely on root of trust and chain of trust. TC solutions need to be tailored for every application.

Theory

> *Trusted system or component is one whose failure can break the security policy, while a trustworthy system or component is one that won't fail*

So a trusted device is a *dangerous* attack point. It is part of a potential attack surface. And as a hardware designer it is our task to prove security since we can't rely on further lower level supposition since we are the lowest level. That's why we are creating TPM or special operating mode TEE.

TCB and RoT

- Trusted Computing Base: collection of everything in a system that is trusted
  - Often implemented in a hierarchical manner where A verifies B, ...
  - Root of Trust: the anchor of this chain

eg: when booting in windows 11, the boot ROM starts (RoT) and verifies the BIOS, then the bootloader gets verified and finally the OS. Then any untrusted applications can execute.

By using this kind of chain, we can verify authenticity and make sure no malicious actor tampered the software, ...

But also good for Digital Right Management and make sure there is no fake copy or pirated copy that are being used.

Technique	Goal	Examples
Secure boot	Ensure only signed software is running, encrypt software binaries	every game console and phone
Measured boot	Unforgeable identity of running code	DICE
Trusted execution	Shield away code and data from untrusted OS	SGX, TrustZone
Remote attestation	Prove to other party which software stack you are running	Apple App Attest

Techniques

Secure boot goals

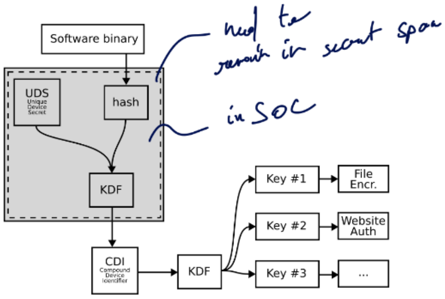
We don't want to leak the code or secrets running on device and make sure we can only execute vendor-approved code. Good for **confidentiality** and **integrity & authenticity**.

note: Secure boot is the general term, Secure Boot is the windows implementation.

Here we have on the hardware a key written with fuses and it sends its public key to the TPM unit that make sure the BIOS is not corrupted. Here the RoT is the boot ROM and the engraved key.

Measured boot goals

Know which code is being executed and can attest it. Good for **integrity**. Here, even if the code is not the right one, it is still allowed to run but we can check that it is not the correct one.



need to remain in secret space

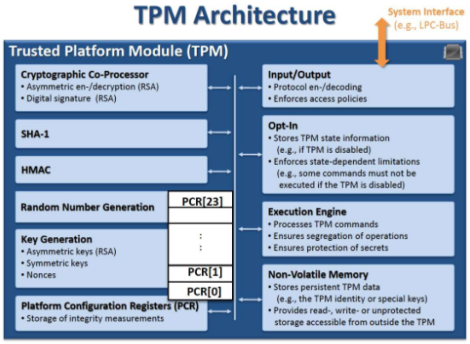
in SoC

Figure 16: Device Identity Composition Engine

They grey box is done by a secured piece of code that is trusted by the boot ROM. The CDI will change at any data change in the code which creates new key making previous encrypted data unreadable.

Trusted Platform Module

It uses cryptography and is a form of data storage that can realize attestation. Heavily used and now standardized by the TCP.



TPM Architecture

System Interface (e.g., LPC-Bus)

Trusted Platform Module (TPM)

Cryptographic Co-Processor
- Asymmetric en-/decryption (RSA)
- Digital signature (RSA)

SHA-1

HMAC

Random Number Generation

Key Generation
- Asymmetric keys (RSA)
- Symmetric keys
- Nonces

Platform Configuration Registers (PCR)
- Storage of integrity measurements

PCR[23]
:
:
PCR[1]
PCR[0]

Input/Output
- Protocol en-/decoding
- Enforces access policies

Opt-In
- Stores TPM state information (e.g., if TPM is disabled)
- Enforces state-dependent limitations (e.g., some commands must not be executed if the TPM is disabled)

Execution Engine
- Processes TPM commands
- Ensures segregation of operations
- Ensures protection of secrets

Non-Volatile Memory
- Stores persistent TPM data (e.g., the TPM identity or special keys)
- Provides read-, write- or unprotected storage accessible from outside the TPM

Figure 17: A. Sadeghi, used with permission

Trusted Execution Environment goals

It hides secrets from other system parts **confidentiality** and isolate execution from untrusted parts (like OS too) **availability**. Can be broken through leakage and side channel at the microarchitectural level.

This can be implemented like an ARM TrustZone where on top of the privileged, unprivileged mode of execution, we have a secure and non-secure mode:

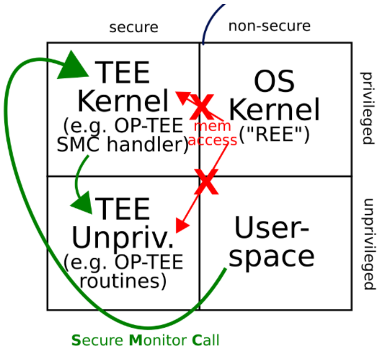


Figure 18: Arm TrustZone

Also relies on code supplied by the boot firmware.

SGX

It creates some *unprivileged secure* islands. In other words it creates enclave shielded from the OS and uses memory encryption. It also uses attestation to prove authenticity and is implemented using some CPU microcode.

Remote Attestation

This is a mix of TEE and measured boot and is called **remote attestation**. We ask a server to verify some code that it is running. Then it will measure the code and will sign hash and challenge with a protected signing key.

But we need to protect some nonvolatile key storage and need a TEE implementation. We also need to be protected against some classic attack. How to anonymize prover, give keys, ...

Conclusion

We will try to minimize the Trusted Computing Base and will move towards RoT with one single pont of failure. There isn't a one-fit-all solution, we need to create one for each threat we want to protect against.

8 - Micro-Architectural attacks and defense

Micro-architecture is something pretty subtle as it is not software nor really hardware. I sits right between the Software and ISA and the Hardware. It is something that differentiates each generation of processor between each other.

We have the Instruction Set Architecture that decodes and executes. The micro-architecture is like the secret sauce that makes everything works and that allows Hardware engineer to create and extend some functionality without redesigning ISA from the ground-up.

Specific implementation of the ISA is called the micro-architecture.

Caches

In hardware, everything is a mirage and we tend to idealize everything. We want fast and wide memory. But this is a pure dream, so we have to create *cache-levels*. Each level is a bit bigger than the previous one but also slower.

Data that was recently accessed are often physically located close to a CPU core for quick access in the future.

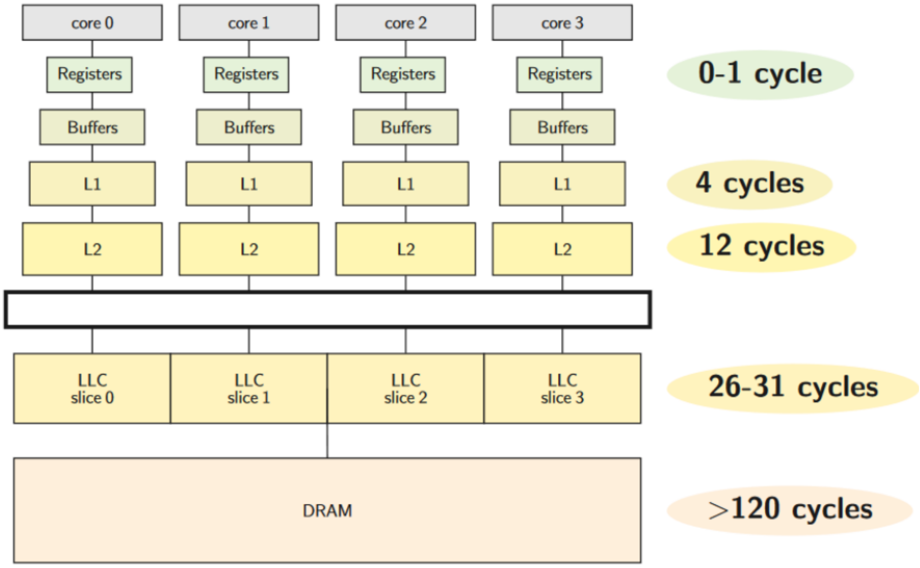


Figure 19: Cache levels

Set-associative cache

If we understand the architecture and the algorithm behind caches, it can potentially indicates sensitive informations to an attacker.

Typically in set-associative caches, We use the something like:

```
[-----tag-----:---index---:offset]
```

Where the index is the row in our cache table, the tag is there to be sure that we are accessing the right data. The offset is usually 2 bits since we often work in 32 bits system so $4 \cdot 8$ bits (we rarely access data or instructions in the middle).

So if we fill up a cache line with garbage data, we can time how long a program will run and compare various executions. This is the principle of cache attack and such attacks can be extended to page-fault attacks.

Cache side channels

To make those attacks feasible and plausible, we have to rule a set of constraint and access to make an attack count as is.

Flush + Reload

- Victim and attacker run on the same CPU and shares micro-architectural elements
- Attacker cannot read victim secrets directly, they do share some memory (shared library)
- Attacker can directly manipulate cache state, eg flush a cache line from the cache

The idea is to flush all the data from a cache line, especially the victim's address. Then let the victim run his codes, finally we measure the time access. If it is slow, the victim didn't access this line, if it is fast then the victim did in fact access this.

We can apply this attack model on RSA where a clear bit will simply square as a set bit will square and multiply. The idea is to wipe the access address of the square and multiply functions. Then we let a loop run and check the time access for both functions. We can then determine if the victim accessed one or both functions.

There exists some countermeasures for this as:

- Constant-time programming
- Not allowing cache state manipulation
- ...

Evict + Reload

- Victim and attacker run on the same CPU and shares micro-architectural elements
- Attacker cannot read victim secrets directly, they do share some memory (shared library)

- Attacker can directly manipulate cache state, eg evict so remove cache line by accessing others

We first access some garbage addresses to evict a potential victim's address. The we let the victim run his codes and finally we measure the speed.

Prime + probe

- Victim and attacker run on the same CPU and shares micro-architectural elements
- Attacker cannot read victim secrets directly, they do share some memory (shared library). Prime cache with attacker-controlled addresses.
- Attacker can directly manipulate cache state, eg flush a cache line from the cache

Here, we will evict a cache line by using a set of addresses that map to the same cache set as the victim address. After, we will measure the speed of the cache set, if it has one or more slow access, we know the victim used this cache set.

Some counter-measures can involve:

- constant time programming
- Restricting access to timers
- Disabling cache
- Partitioning cache
- Cryptographically secure indexing function

Cache covert channels

The idea is to communicate between a sender a receiver without using an official channel. But we use the cache to send those data since the two are running on the same cpu.

For example, we have a large buffer, we can flush all of the data from the cache and then the transmitter can do a dummy access to a specific offset in this array. Then the receiver scan the access time to every a_i.

Transient Execution Attacks

Most modern CPUs implement some branch prediction and other type of algorithm to avoid stalls due to data dependency between different instructions. Most of those CPU's are what we call **out of order** meaning they can execute one or another instructions before another one. They will be called **transient instructions** where we have the result but we do not commit yet (no write back).

Those transient instructions will leave some micro-architectural side effects, it will change the cache state.

Spectre

```
buf_size = len(buf)
if (x < buf_size){
  s = buf[x]
  a = array[s*4096]
}
```

Typically we will iterate over the buffer, most of the time `x < buf_size` so it will avoid checking all the time and will execute the two instructions before. So we have some transient instructions and as seen before, the last line is a way to use covert channels.

We can access extra values without triggering a buffer overflow or we could also leak some data about the size of the buffer.

Some countermeasures can be the insertions of `lfence()` to stop speculative execution up until a certain point, making hte program slower. On the *hardware* level we can disable all speculation, prevent forwarding of transiently loaded data, roll back micro-architectural state, close cache covert channel, . . .

Meltdown

```
s = load(kernel_address)
a = array[s*4096]
```

Result from faulting instruction may have already been forwarded. Executing the array access affects the cache and an attacker can infer value of s from cache state.

There exists some software and hardware countermeasures for it such as:

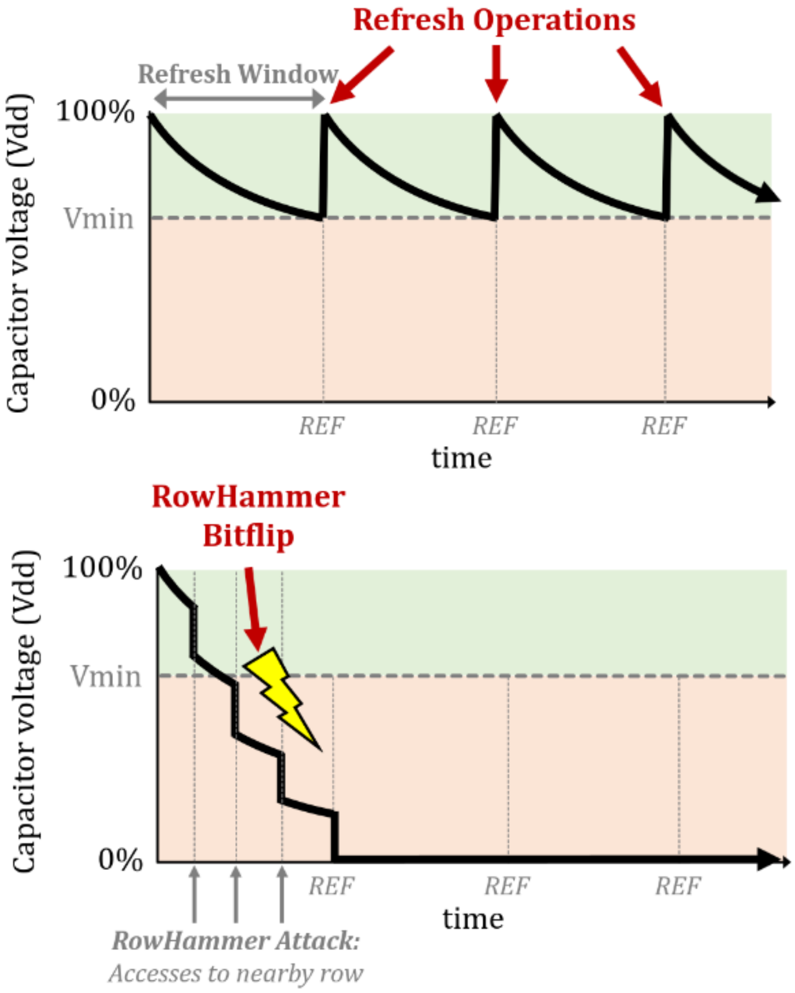
- Software: Kernel Page Table Isolation
  - Don't map kernel pages in userspace
- Hardware: Don't forward values from faulting instructions

Rowhammer

This type of attacks is even more low-level as it is linked with the physical way that data is stored in RAM. In a RAM stick, each data is stored on a capacitance that is accessed through the word- and bitline. This cap needs to be charged and discharged quite often.

Reading in DRAM is destructive and requires to refresh the value. We use a *row buffer* to store the row that is being currently read.

BUT, reading all the time can disturb the neighboring rows. We can witness some bitflip.



Neighboring rows loose a tiny amount of charge which causes to flip.

Countermeasures - DDR4

Target Row Refresh (TRR), we keep a counter for each row and if it exceeds a certain value we also refresh neighboring rows. But it is quite not feasible to implement such counter. So we have a limited number of counters per bank. So that makes DDR4 still vulnerable to rowhammer.

Countermeasures - DDR5

We have on die ECC, less refresh intervals so less time to hammer the rows. But it as been proved on AMD processor that we can do some rowhammer.

Conclusions

Correct execution doesn't mean it is safe. We have to often reduce performance to enhance security since performance can introduce vulnerabilities.

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:

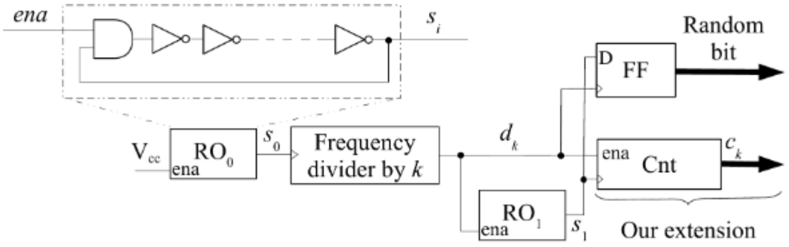


Figure 20: Counter to check jitter

New method proposed:

We most likely observes the k clock cycles as expected

Theoretical error

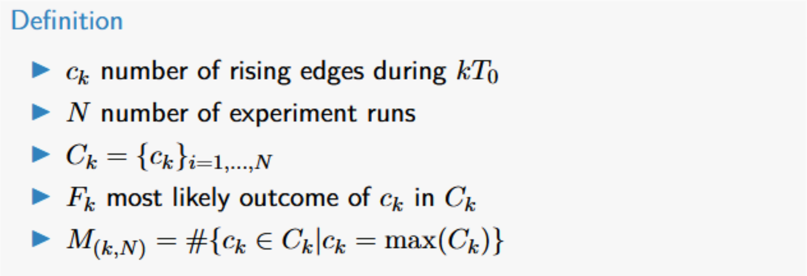
F_k is the most probable outcome.

Qestions :

"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.



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
- $M_{(k,N)} = \#\{c_k \in C_k | c_k = \max(C_k)\}$

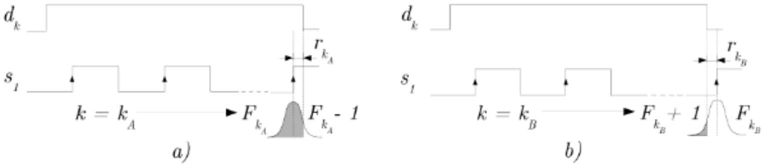


Figure 21: New method

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

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

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

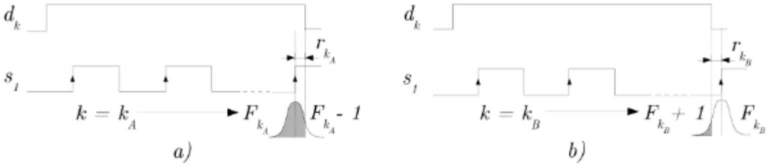


Figure 22: 2 cases

Inject EM harmonic signal to bias that source of entropy. They us a micro-probe for it. The TRNG is implement 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 ?”