Weak PUF vs Strong PUF

The distinction is rooted in the security properties of their challenge-response pairs

One definition of a **Strong PUF**:

Even after giving a adversary access to the PUF instance for a *prolonged period of time*, it is still possible to come up with a challenge that with high probability, the adversary **does not know the response**

This implies that

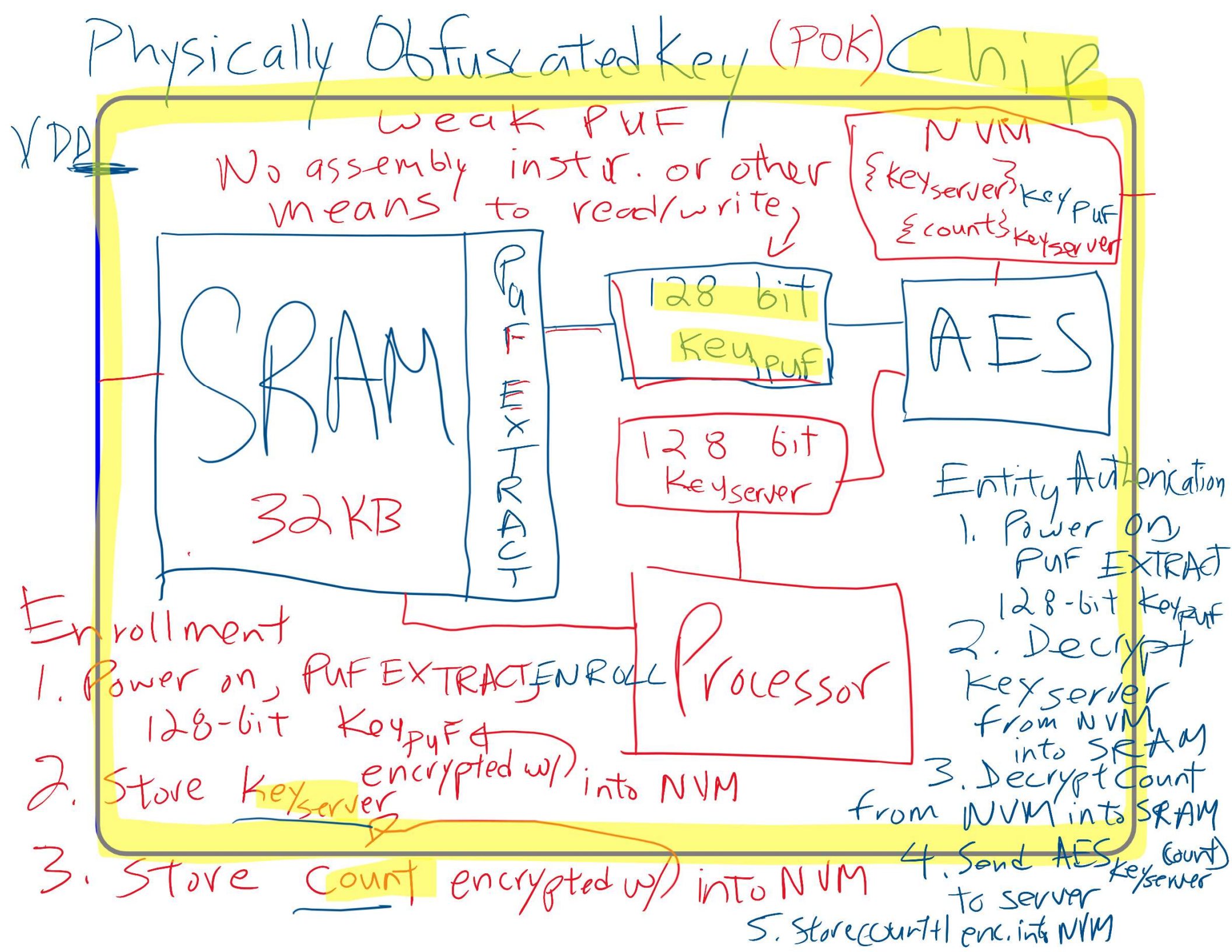
- The PUF has a **very large challenge space**, otherwise the adversary can simply query the PUF with all challenges to learn its complete CRP behavior
- It is **infeasible to build an accurate model** of the PUF using only a subset of CRPs to 'train' the model, as a means of learning its complete CRP behavior

PUFs which do not meet these requirements are called **Weak PUFs**

In the limit, some PUFs have only a single challenge and are called *physically obfuscated key* or POK

We discussed the SRAM PUF earlier that has only one challenge

Physically Obfuscated Key (POK) Chip



Enrollment

1. Power on, PUF EXTRACT ENROLL 128-bit key_{PUF}
2. Store key_{server} encrypted w/ key_{PUF} into NVM
3. Store count encrypted w/ key_{server} into NVM

Entity Authentication

1. Power on, PUF EXTRACT 128-bit key_{PUF}
2. Decrypt key_{server} from NVM into SRAM
3. Decrypt Count from NVM into SRAM
4. Send AES$_{key_{server}}$(Count) to server
5. Store Count+1 enc. into NVM

PUF Usage Scenarios

(1) • **Identification**

The PUF can be used to generate a 'serial number' to identify and/or track parts through manufacturing (the original proposed use by Keith Loftstrom in 1999!)

For manufacturing, ***uniqueness*** is the most important metric

A ***weak PUF*** is sufficient for this type of *low security* application

Reliability is not a concern as long as

- Bit flip errors are infrequent, i.e., HD_{intra} is relatively small, otherwise the probability of 'aliasing' gets unacceptably large
- It is possible to use a 'fuzzy match' criteria after the identifier is generated

(2) • **Authentication**

The PUF is used to securely identify the chip in which it is embedded to an authority through corroborative evidence

As we will see when we discuss authentication scenarios, a ***strong PUF*** is best, particularly when the device is resource-constrained

PUF Usage Scenarios

Also, the **challenge-response** form of authentication implemented by strong PUFs is considered **strong**, in contrast to weak forms of authentication, e.g., passwords

Note that in contrast to encryption discussed below, the PUF inputs and outputs are **exposed** (to different degrees depending on the authentication scheme)

This makes the PUF more *accessible* (and vulnerable) to adversaries, and enables **model-building** attacks

There is a rapidly growing need for hardware-based authentication, e.g., in the supply chain, in the field (electronic voting machines) and for IoT devices

For the supply chain, the PUF is an important new security primitive that can address threats related to

- IC theft
- IC reuse
- Malicious substitution (hardware Trojans)
- Reverse engineering and cloning

PUF Usage Scenarios

The same is true for 'in the field' authentication, particularly with IoT devices which are vulnerable to physical attacks and are *resource-constrained*

All three statistical metrics, i.e., uniqueness, randomness and reliability, are important for authentication

Some simple schemes relax the reliability metric as we will see

w/ temp. + volt. variations

Why use PUFs for authentication?

strong PUF

- They can **eliminate** the requirement for **NVM**, a real cost benefit for resource-con- strained devices
- They can potentially provide a **very large number of CRPs**, i.e., a much larger source of entropy when compared to an NVM
- They are **tamper-evident**, making it more difficult for adversaries to physically probe the device to steal the secrets
- They can be designed to **never reveal their secrets**, i.e., even the manufacturer does not have knowledge of the embedded secrets

hopefully

- They can be used to provide a **stronger challenge-response form of authentica- tion**

entity

PUF Usage Scenarios

- **Encryption**

The PUF is used to generate

- A key for symmetric encryption algorithms
- A random *nonce* that can be used to select a specific public-private key pair for asymmetric encryption

In typical encryption applications, the key is not revealed outside the chip and therefore, a ***weak PUF*** can be used (although a strong PUF is better here too)

The ***inaccessability*** of the PUF responses makes **model-building** impossible

However, recent work shows that power analysis attacks can be used to enable model-building, which argues in favor of using strong PUFs for encryption too

Unfortunately, in contrast to authentication schemes, **tolerance to bit flip errors is 0**

Even a difference of 1 bit in a 256-bit key completely wrecks communication between parties because of the avalanche effect

PUF Usage Scenarios

In summary

- All three applications require *uniqueness*

- Identification:

  PUF bitstrings must be large enough to suit the # of chips in the population

  HD_{intra} can be > 0 but bear in mind, this reduces the number of unique IDs that can be generated and used

- Authentication: Add *randomness* as a critical metric

  Having a very large CRP space prevents adversaries from reading them all out and building a clone, and prevents them from succeeding at model-building

- Encryption: Adds both *randomness* and *reliability* as critical metrics

  Having a large number of CRPs is **not necessary** in cases where only a single key (or small number of keys) need to be generated over lifetime of chip

  HD_{intra} must be zero, which requires **error correction** or **error avoidance**

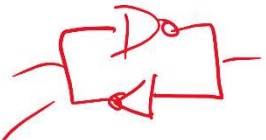
PUF Implementations

There are MANY PUF implementations that have been proposed

A rough characterization is as follows:

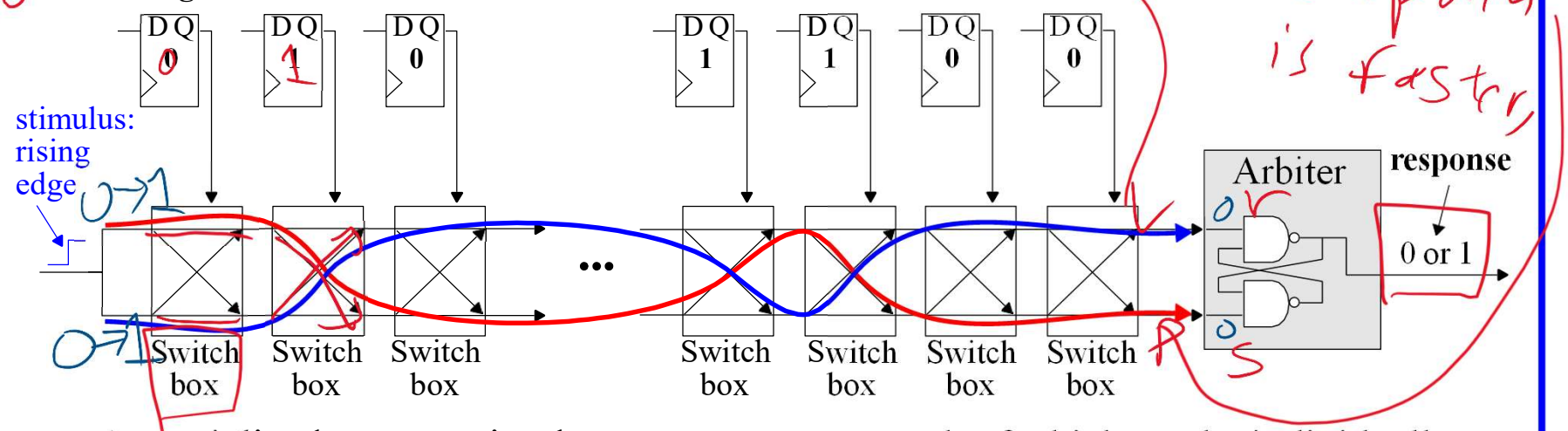
- *Delay-based PUFs*:
  - Delays along 'matched' paths (Arbiter)
  - Ring Oscillator frequencies
  - Glitches produced along paths within a functional unit
  - Delays along glitch-free paths within a functional unit (HELP)

- *Bi-stable PUFs*:
  - SRAM
  - Butterfly Buskeepers
  - FFs and Latches



- *Mixed-Signal PUFs*: (These require a specialized analog-to-digital converter: ADC)
  - Transistor threshold voltage/transconductance
  - Dynamic/leakage current
  - Resistance/Capacitance

Arbiter PUF



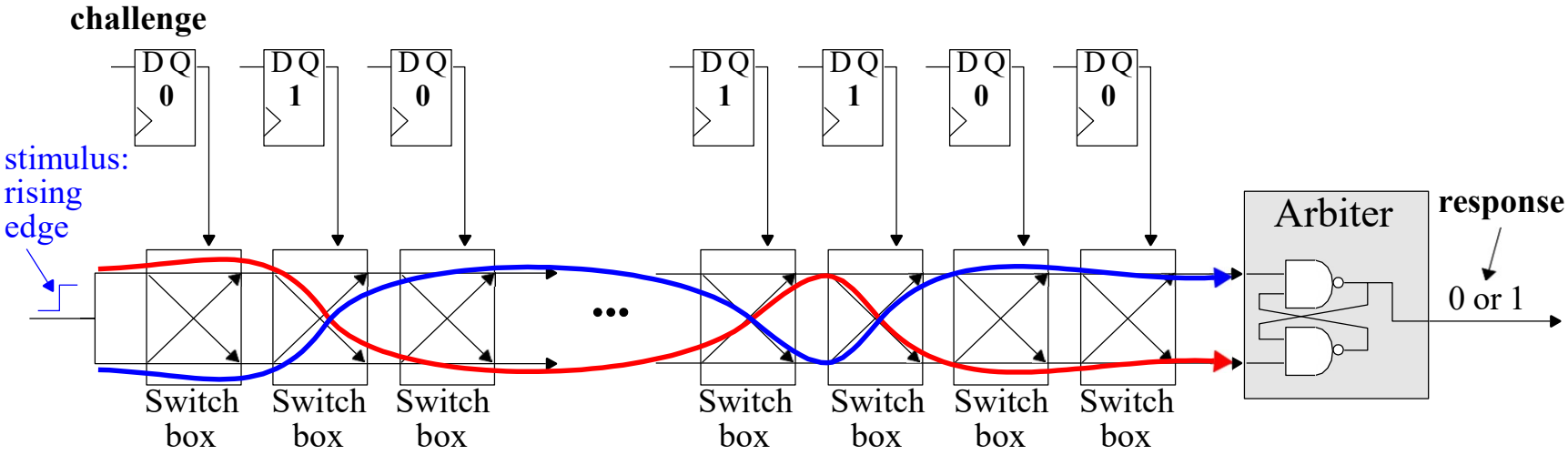
A specialized structure implements **two paths**, each of which can be individually configured using a set of *challenge bits*

Each of the challenge bits controls a 'Switch box', that can be configured in either **pass mode** and **switch mode**

Pass mode connects the upper and lower path inputs to the corresponding upper and lower path outputs, while *switch mode* flips the connections

A stimulus, represented as a rising edge, *cause two edges to propagate* along the two paths configured by the challenge bits

Arbiter PUF



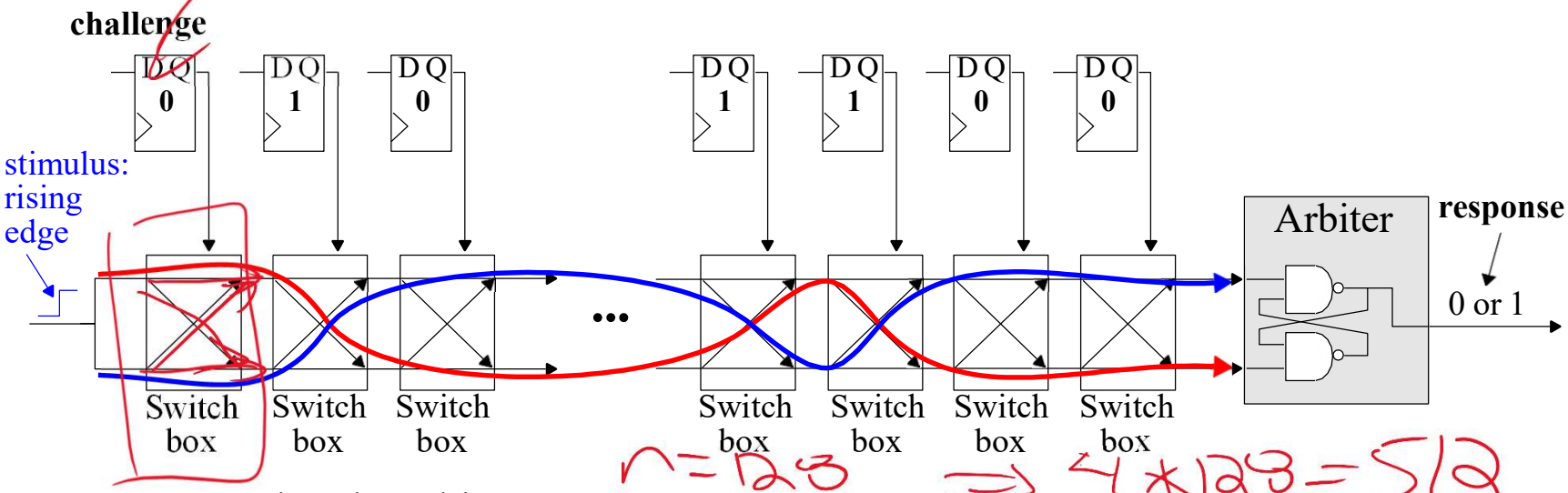
The faster path *controls the value stored* in the **Arbiter** located on the right side of the figure

If the propagating rising edge on the upper input to the Arbiter arrives first, the response bit output becomes a '0', otherwise a '1'

The switch boxes are designed **identically** as a means of avoiding any type of *systematic bias* in the delays of the two paths

Within-die process variations change the delay through the switch boxes, which makes **each instance** of the Arbiter PUF **unique**

Arbiter PUF



It is clear that the arbiter PUF has an **exponential number of input challenges**

In particular, 2^n with *n* representing the number of switch boxes

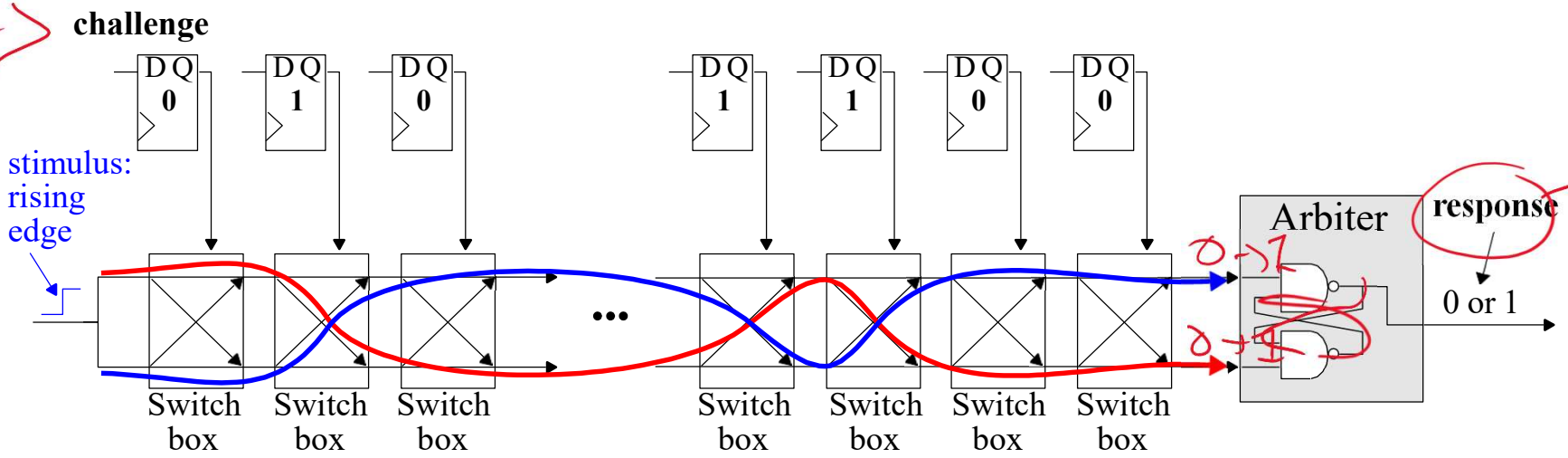
However, the **total amount of entropy** is relatively small

For *n* equal to 128, the total number of path segments that can vary individually from one instance to another is 4*128 = 512

The exponential number of challenges simply *combine the entropy in different ways*

Although the Arbiter PUF is considered a **strong PUF**, researchers have 'broken' it using **model building** many times

Arbiter PUF



Another important issue is *meta-stability*

What happens with the two edges *arrive simultaneously* at the inputs to the arbiter?

The metastable condition eventually resolves, but the response bit in this case is **not stable**

In other words, repeating the challenge will produce different responses

The number of challenges that produce *metastable* (noisy) bits increases when temperature and supply voltage are varied

Model Building

The number of individual sources of entropy in the Arbiter is only linear with n

Therefore, **dependencies must exist** among the 2^n challenges and response bits

For example, if it were possible for the adversary to learn the *individual path segment delays*, then the PUF is no longer needed to predict the responses

Modeling attacks leverage a simple **additive delay model** where the delay of the entire path is equal to the sum of the individual segment delays

By strategically selecting CRPs, *machine-learning* techniques can quickly determine the **relative delays** through each switch box

Machine-learning techniques include *artificial neural networks* (ANNs), *support-vector machines* (SVMs), *genetic algorithms* and *decision trees*

Goal is deduce the relationship of segment delays using as few CRPs as possible

A PUF is **(p_{model}, q_{train})-modelable** if known modeling attacks exist which have a successful prediction rate of p_{model} after training with q_{train} CRPs

Arbiter PUF Evolution

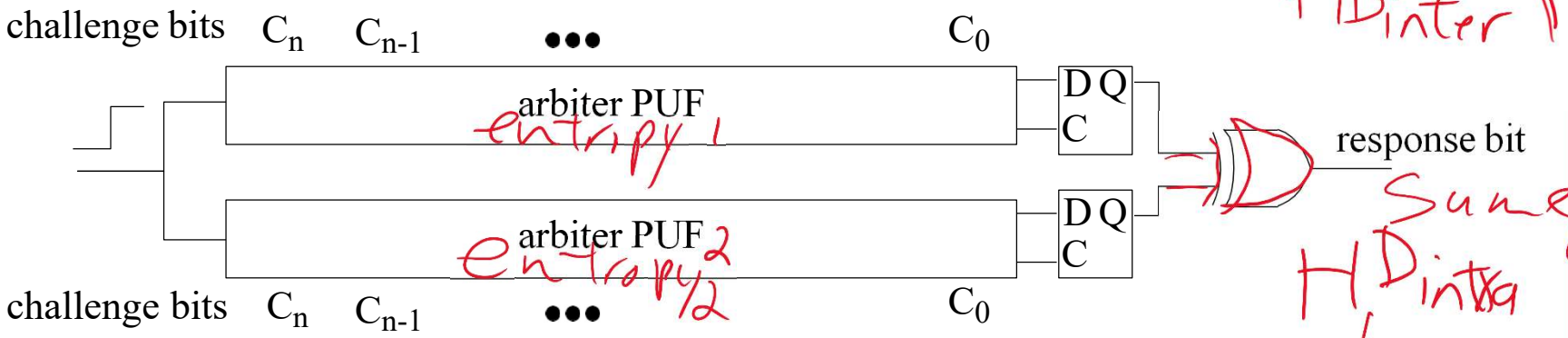
Early examples in the literature on ASIC implementations show

- HD_{intra} of 4.82% with a temperature range of 25°C to 67°C
- HD_{inter} of 23%
- SVM-based machine learning attack produced (p_{model} = 96.45%, q_{train} = 5000), which indicates the implementation is not secure

All subsequent work attempt to make model-building attacks more difficult by:

- Introducing **non-linearities**, i.e., *feed-forward* and *XOR-mixed* versions
- Obfuscating the challenges to the PUF and the responses from the PUF

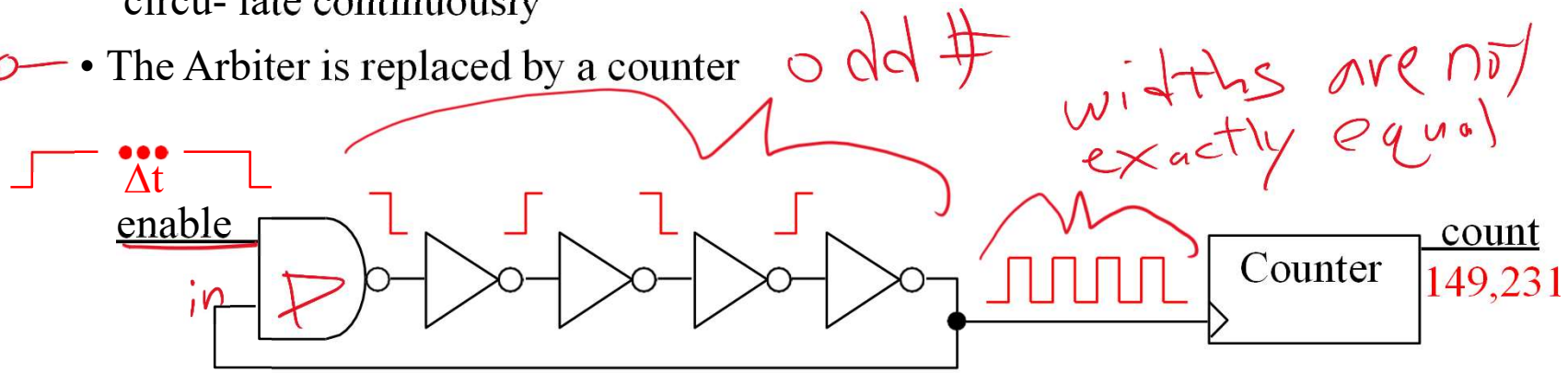
XOR-mixed version



Ring Oscillator PUF

The **RO PUF** is also a *delay-based PUF* but the configuration and measurement technique are different from the Arbiter PUF

- An odd number of inverters are connected in a ring, which causes an edge to circu- late continuously
- The Arbiter is replaced by a counter

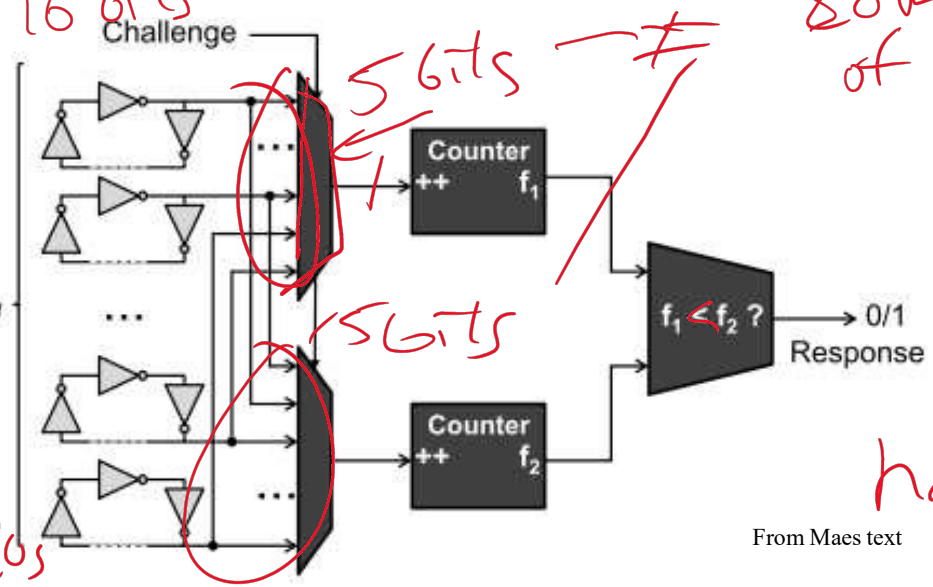


By enabling the RO for a fixed Δt, the frequency of the RO is reflected in the count, and is given by count/Δt

But since Δt is constant for all RO testing, the digital count value can be used instead

Similar to the Arbiter PUF, a *differential frequency post-processing* scheme is typically used to compensate for temperature/supply voltage variations

Ring Oscillator PUF



From Maes text

Here, a pair of ROs are selected to drive 2 separate counters

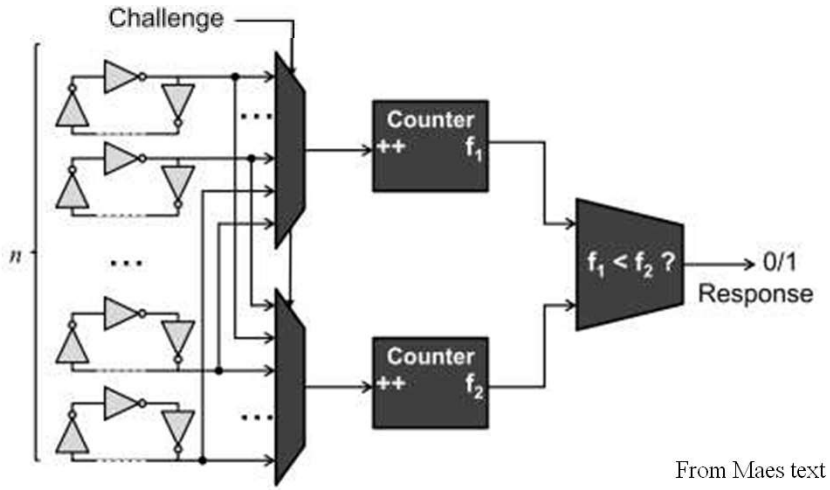
TV variations change the frequencies of both ROs in a similar fashion, significantly improving the *reliability* of the RO PUF

The RO PUF is a **weak PUF**

Assuming any RO can be paired with any other, we have $n(n-1)/2$ pairings

Remember, model-building is not ~~applicable to~~ weak PUFs because it is possible to read out all possible bitstrings when the number is limited to n^2

Ring Oscillator PUF



From Maes text

The slowest RO freq has poss. comp. w/ n-1 other freq. The next slowest poss. compr. w/ n-2 ...

However, not all these pairing produce **independent** evaluations

If RO A is faster than RO B, and B is faster than C, than A is faster than C

$RO_A > RO_B$, $RO_B > RO_C \Rightarrow RO_A > RO_C$

Therefore, the third response bit is dependent on the previous 2 bits

The true amount of entropy is a function of the number of **possible ordering** of n frequencies, which is $n!$

Independant Identically Distributed

Assuming each ordering is IID, the max. number of **independent comparisons** is $\log(n!) = \sum_{i=2}^{n} \log_2(i)$

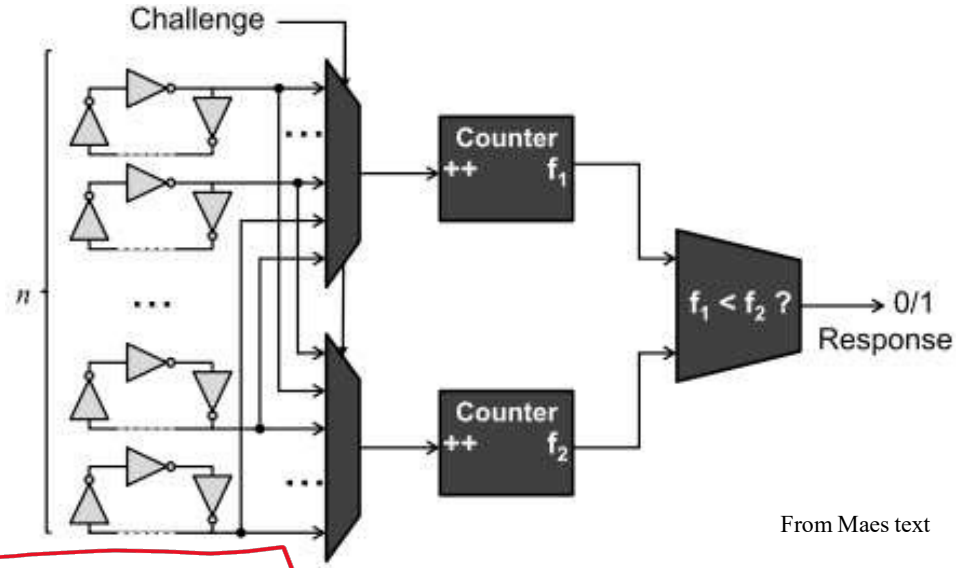
e.g., n=10 ring oscillators $\Rightarrow \frac{10(9)}{2} = 45$

But, is ≈ 21

for n=10, $\sum_{i=2}^{10} \log_2(i) = \log_2 2 + \log_2(3) + \log_2 4 + \cdots + \log_2 10 = 21.791\ldots$

$= 1 + 1.58496\ldots + 2 + \cdots + 3.3219 = 21$

Ring Oscillator PUF



From Maes text

Lehmer-Gray encoding has been proposed to optimize entropy and nearly achieves the maximum $\log_2(n!)$ number of independent response bits

The cost is increased processing complexity

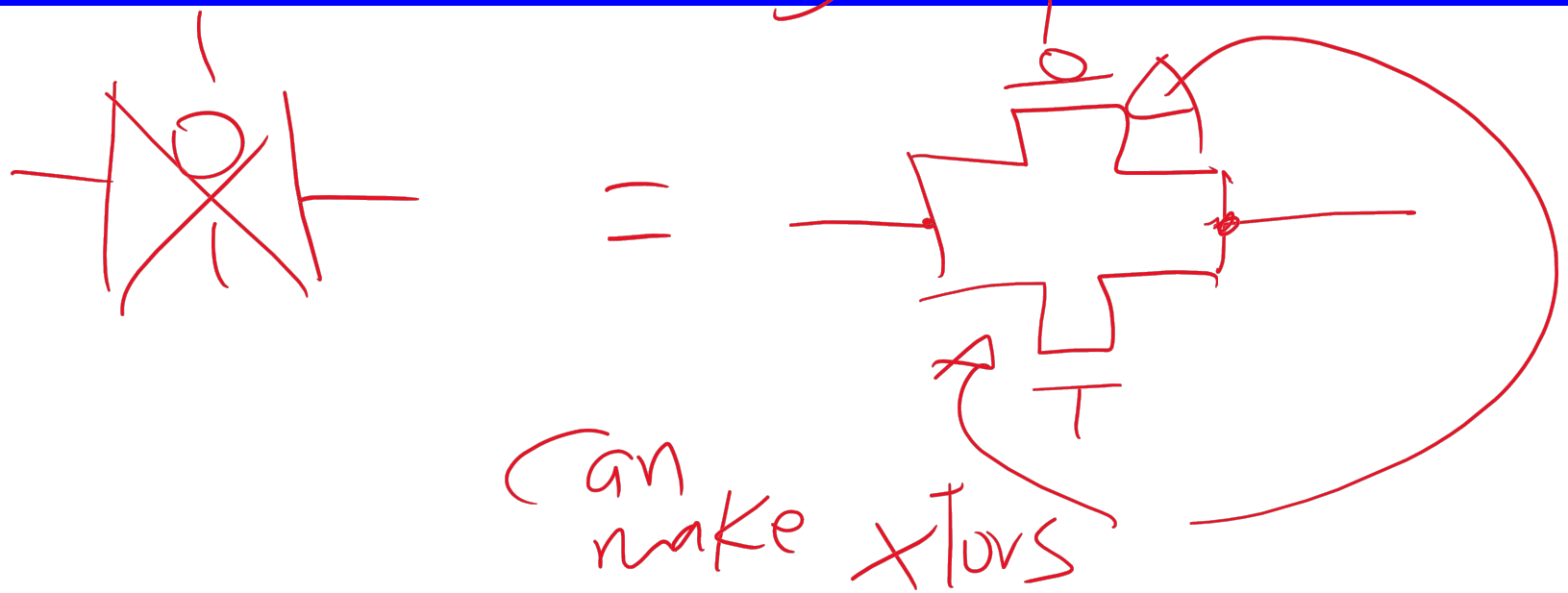
i.e., no post processing

A low-overhead strategy for dealing with dependencies is to use each RO **in only one comparison**

This strategy is not optimal, however, in utilizing the available entropy, reducing the number of generated response bits to *n/2*

n-to ROs: RO_0 vs. RO_1, RO_2 vs. RO_3, RO_4 vs. RO_5, RO_6 vs. RO_7, RO_8 vs RO_9 } 5 comp. = n/2

"transmission gate"

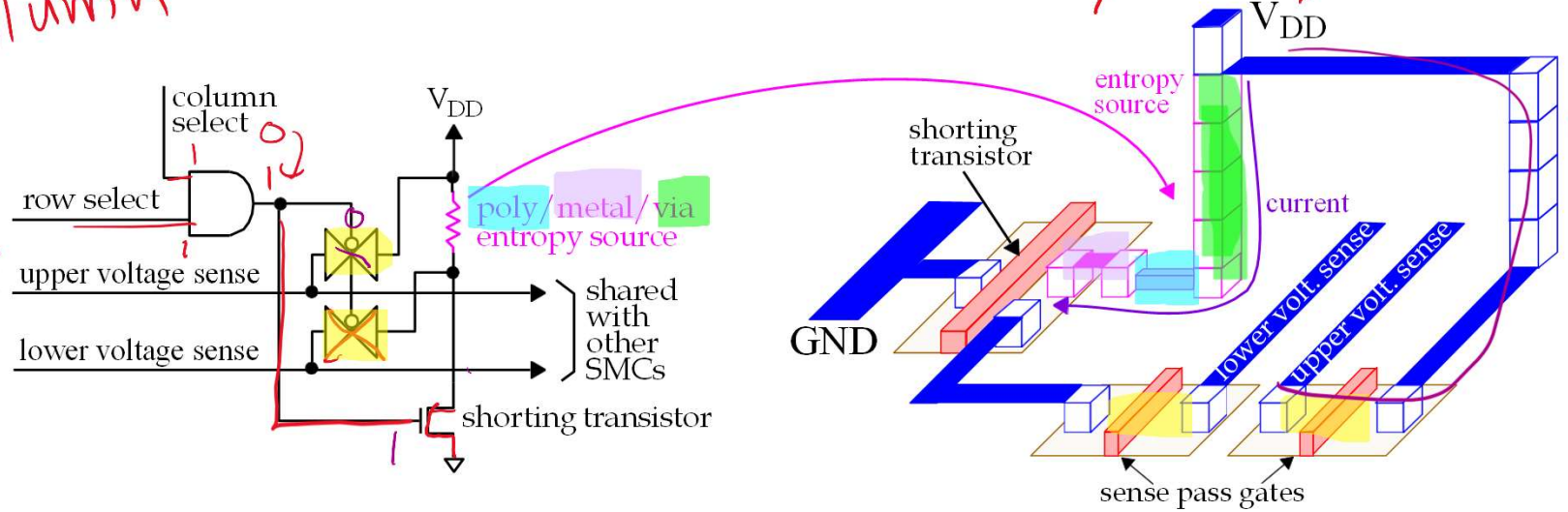


can make xtors very wide => low resistance

"pass gate"

Metal Resistance PUF

The metal PUF measures voltage drops across polysilicon wires, metal wires and vias as the source of entropy



Stimulus-Measure-Circuit (SMC)

An SMC cell from a larger array is selected using *column* and *row* select signals

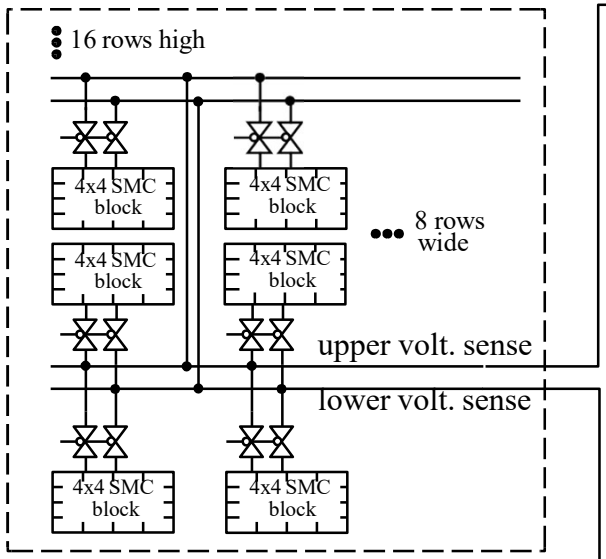
Once selected, a Stimulus-Measure-Circuit (SMC) enables a *shorting transistor* (stimulus) which creates a voltage drop across the poly-metal-via stack

Two 'pass gates' are also enabled that allow voltages to be sensed and measured

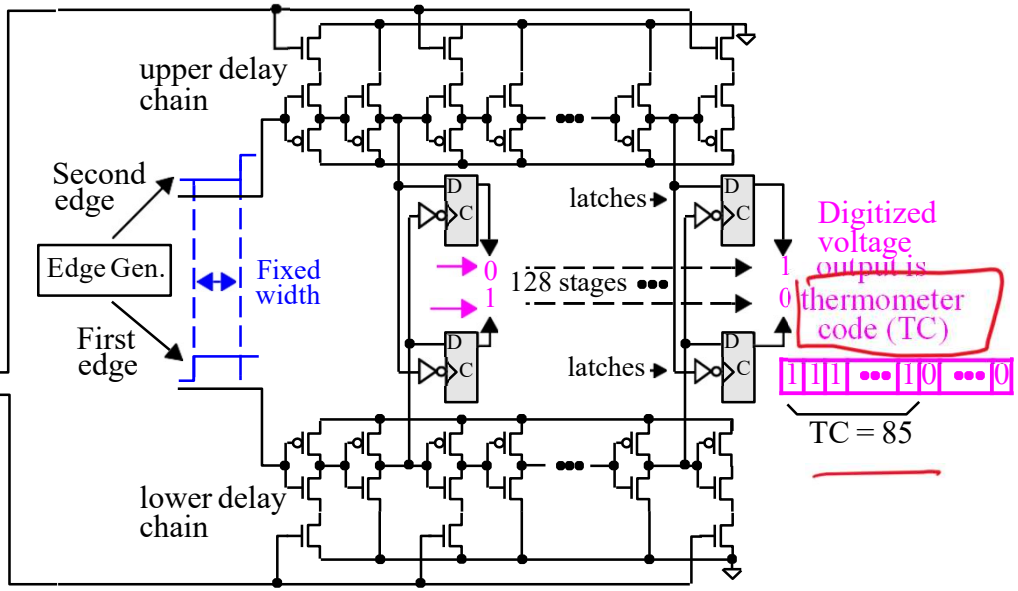
Metal Resistance PUF

Voltages generated by an element in the SMC are digitized by a VDC

SMC array of 2048 elements

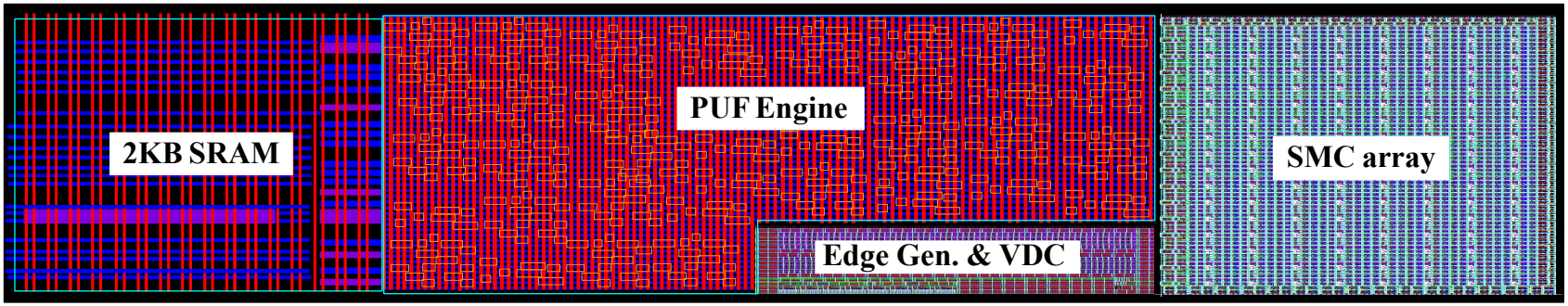


Voltage-to-digital-converter (VDC)



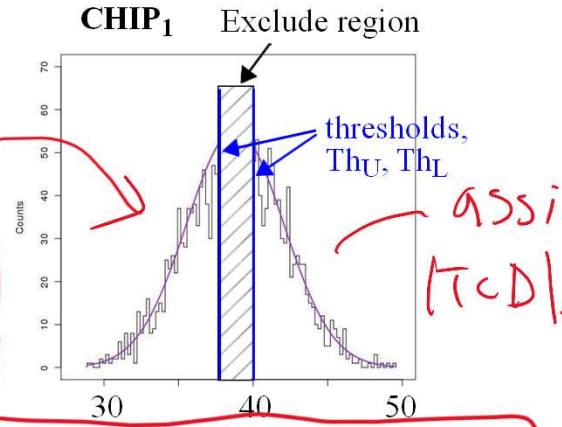
D. Ismari and J. Plusquellic, "IP-Level Implementation of a Resistance-Based Physical Unclonable Function", HOST, 2014.

Layout of the PUF Engine, VDC and SMC array IP block

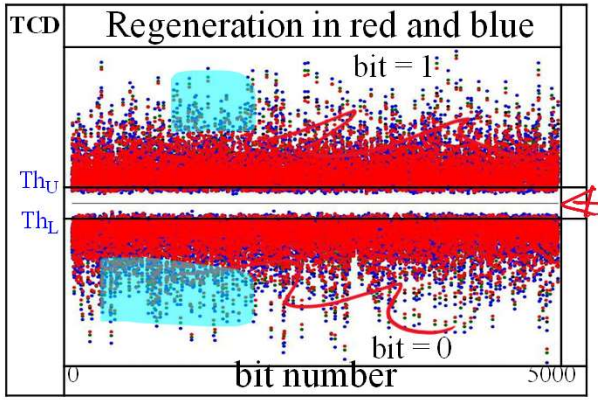


Metal Resistance PUF

Similar to the RO bit generation method, the algorithm used for the metal PUF creates TC differences (TCDs) by randomly selecting pairs of TCs from the distribution



Random set of TCDs created using enrollment data for a chip



J. Ju, R. Chakraborty, C. Lamech and J. Plusquellic, "Stability Analysis of a Physical Unclonable Function based on Metal Resistance Variations", HOST, 2013.

An **error avoidance** scheme is proposed that creates **two thresholds** around the mean of the TCD distribution

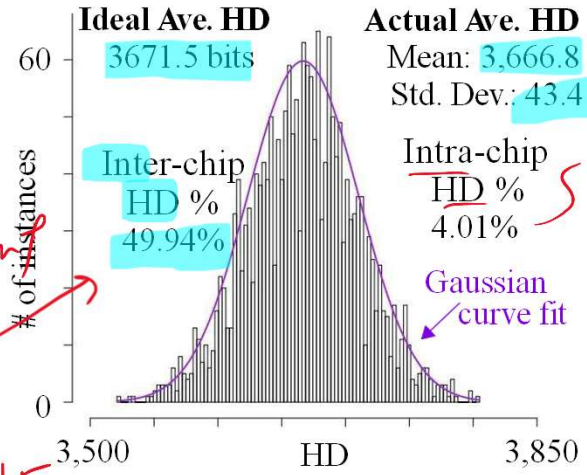
TCDs around the mean are *unstable* and are not permitted to generate a bit in the bit-string/key

The red and blue TCDs illustrate that TV-noise-related variations during regeneration are small enough to prevent bit flip errors

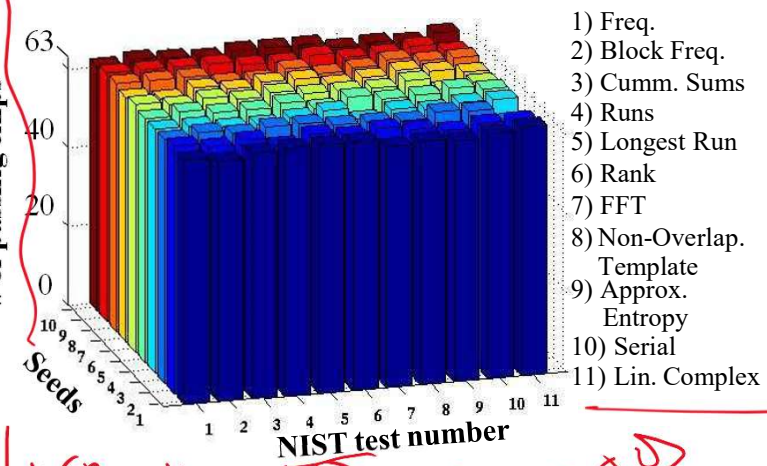
Metal Resistance PUF

Statistical analysis of bitstrings generated from 7343 TCDs and 63 chips

Uniqueness



Randomness



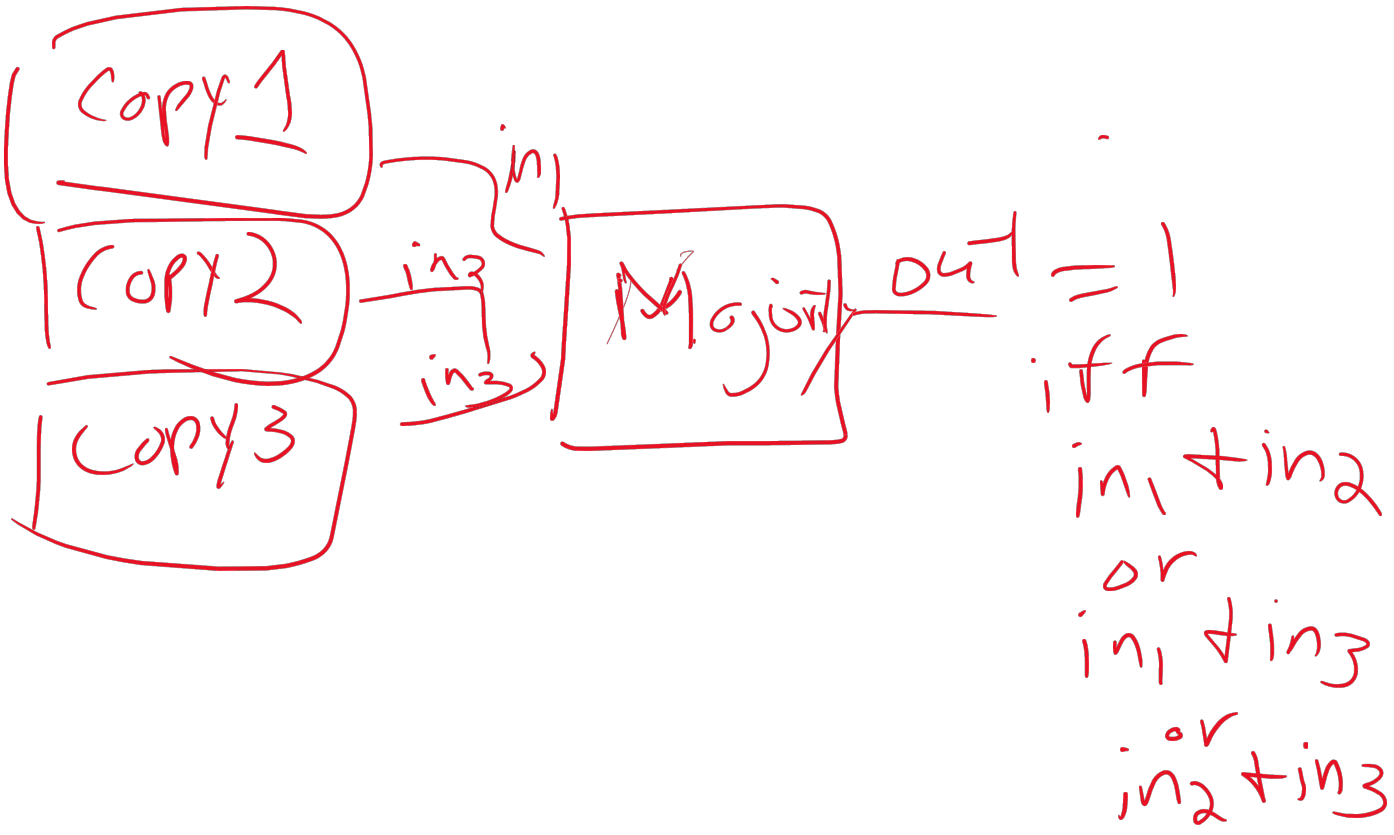
We developed a reliability-enhancing technique called XMR, which creates redundant copies of the bitstring

Majority voting is then used to 'correct' bit-flip errors

Typical reliability standards target $1e^{-6}$ (1 in a million) to $1e^{-9}$ (1 in a billion)

3MR (TMR) and 5MR provide reliability in this range

In "fault tolerant" design
e.g., Satellite



1 enrollment value
at VDD = nom.
Temp = room.

9 Comp.

9 diff pairs {V1 V2 V3}
low high

{T1 T2 T3}
low room high

---

10 meas.

$\binom{10}{2}$

UAV swarm

The Billion Dollar PUF Question

- Can the underlying physics of a PUF be harnessed to provide the following
  - An exponentially large (as opposed to polynomial) challenge-response space
  - Statistically reliable responses which can be utilized for cryptography
    - Authentication
    - Encryption
  - Sufficient sizes of "*n*" such that an adversary cannot carry out brute-force attacks successfully

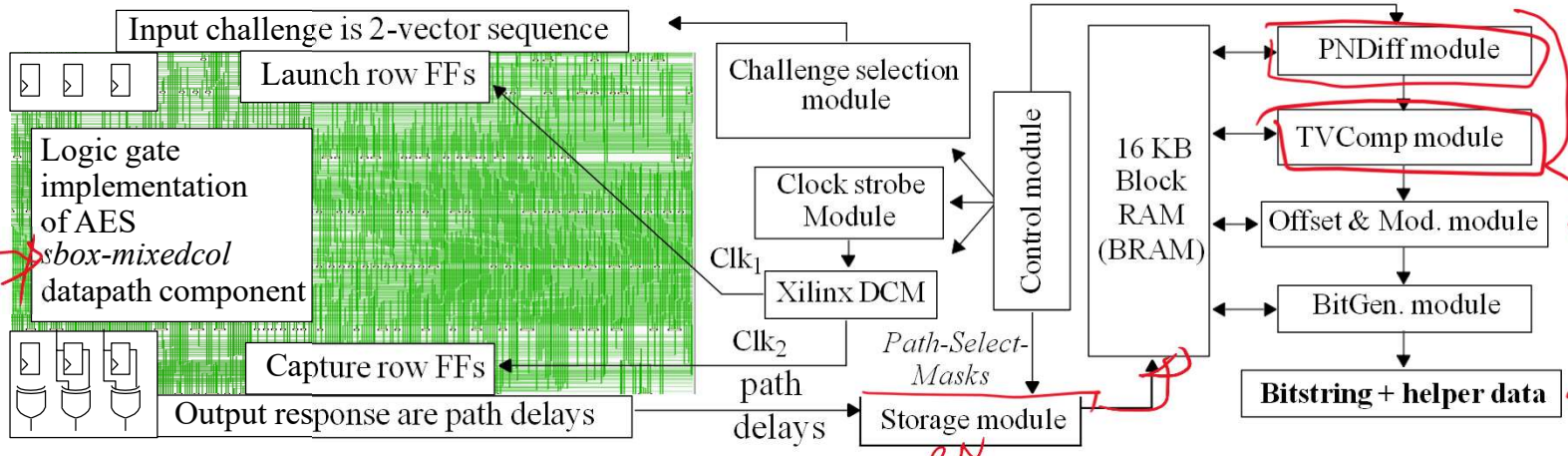
(i) if can argue that the entropy source (physical) has no discernable mathematical relationship, can argue no way to build a model

(ii) if we can produce PRNG AES in CTR

(iii) Sufficient number of physical sources to generate exponential CRP space

Hardware Embedded Delay PUF (HELP)

HELP measures path delays in an on-chip functional unit, e.g., AES, and leverages random **within-die** variations in **propagation delay** as a source of entropy



HELP can be described entirely in an HDL, and therefore can be implemented on FPGAs

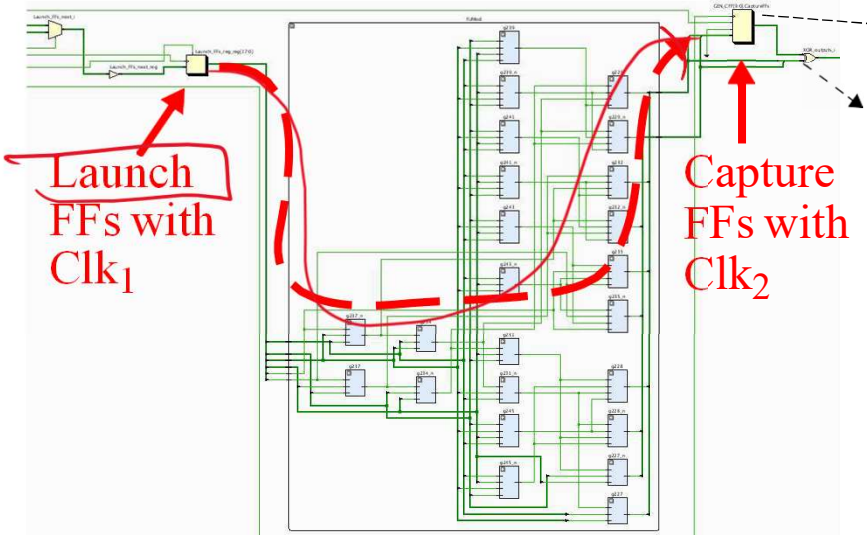
The functional unit (entropy source) is implemented using a specialized logic style that is **hazard-free**

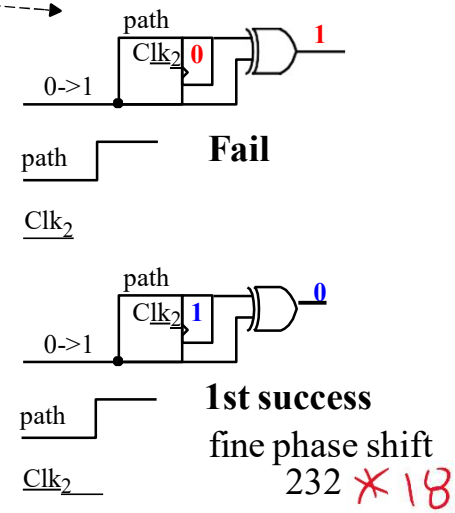
This ensures paths remain *stable*, and can be timed accurately, as TV conditions vary

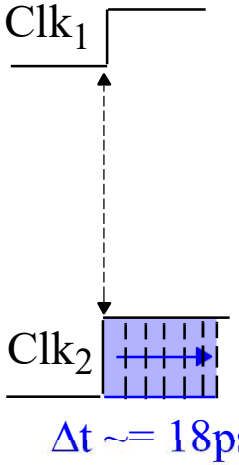
HELP is a STRONG PUF and is capable of generating a large # of random bitstrings

Hardware Embedded Delay PUF (HELP)

HELP uses a *launch-capture* timing mechanism to obtain high-resolution path delay values for combinational logic paths







Path delays can be measured using a **clock strobing** method

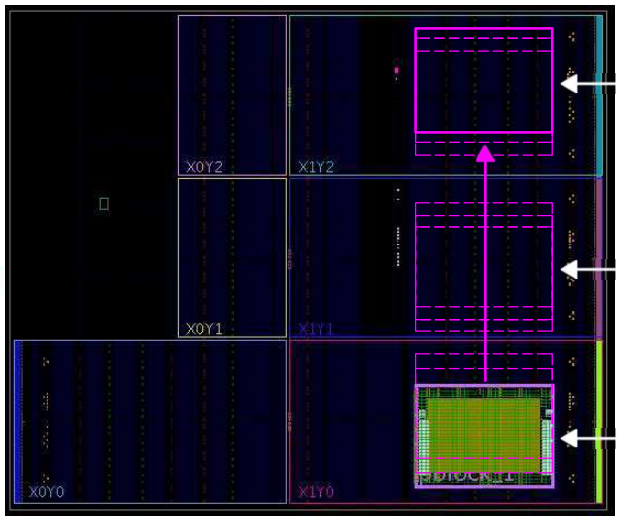
Or using an alternative *flash ADC* method that also works well

The *fine phase shift* feature within modern *digital clock managers* (DCMs) can be used to incrementally tune a capture clock, Clk_2, in a series of launch-capture tests

The integer-based *fine phase shift* value is used as the digitized path delay

HELP Experiments and Features

We implemented HELP on a Xilinx Zynq 7020 and tested 20 chips, with 25 copies of HELP implemented in different locations (but 'fixed') on each of the chips



The total number of paths in the AES functional unit is approx. 8 million (4 million rising paths and 4 million falling paths)

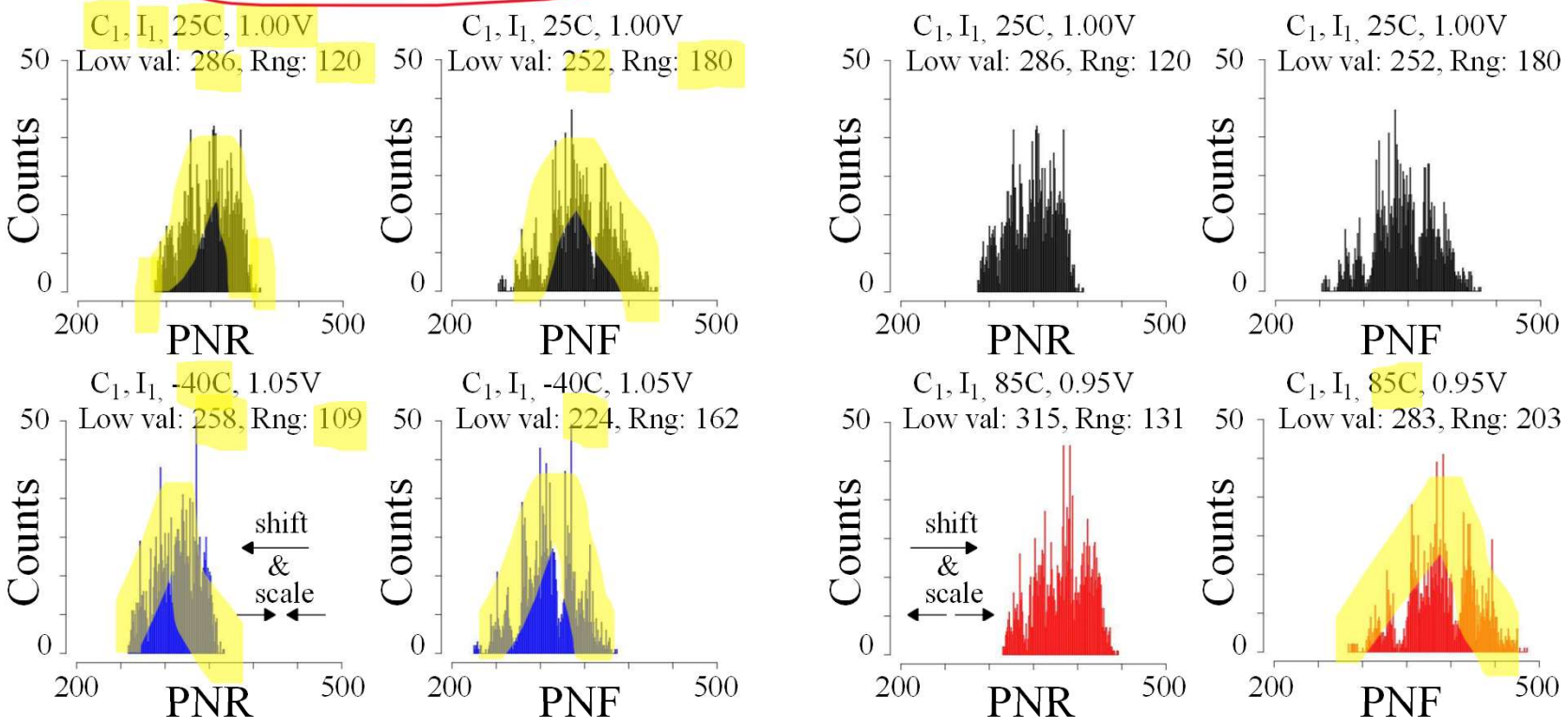
This large # is the first important characteristic that makes HELP a **strong PUF**

Other features are related to its multi-dimensional CRP space which includes:

- Parameters including two LFSR seeds, μ_{ref} and Rng_{ref}, a Modulus and Margin
- The full set of two vector sequences, *Path-Select masks* and *Distribution Effect*

HELP Processing Steps

STEP 1: Apply a set of challenges to generate 2048 *rising* path delays (called **PNR**) and 2048 *falling* path delays (called **PNF**), with PN for PUFNumber

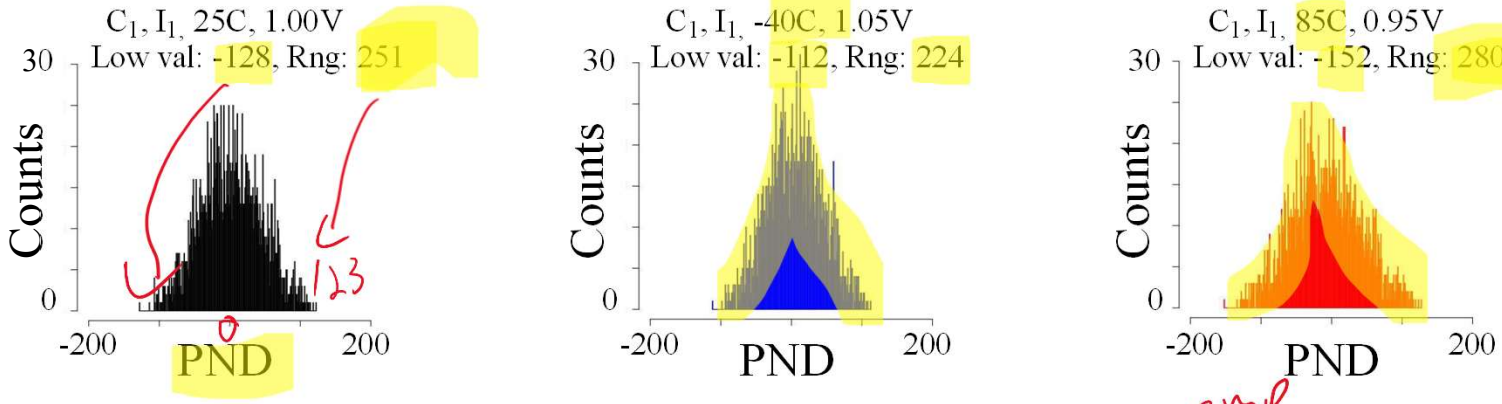


Changes in TV conditions ***shift*** and ***scale*** the digitized path delays

These digitized path delays are **processed as a group**, NOT individually as is true of all other PUFs, i.e., no bits are generated until all group processing is complete

HELP Processing Steps

STEP 2: Create **unique pairing** of rising and falling path delays using two *11-bit* LFSRs, to create PN Differences or **PND**



Shifting and **scaling** of entire distribution is exacerbated, but TV variations are reduced (*partially compensated* for) in the individual PND b/c of common mode

LFSR seeds expand the response space of HELP and allow up to n^2 bits to be generated from *n* PNR and *n* PNF

As we will see later, a **Modulus** operation nearly eliminates the classical *dependencies* that exist when PN are reused

(i) Consider that there are 4096 physical measurements, 2048 PNR values and 2048 PNF values

(ii) One could choose 2 out of 4096 $\binom{4096}{2} = \frac{4096(4095)}{2} = (\text{for } n=2048)\ \frac{2n(2n-1)}{2} = 2n^2 - n$

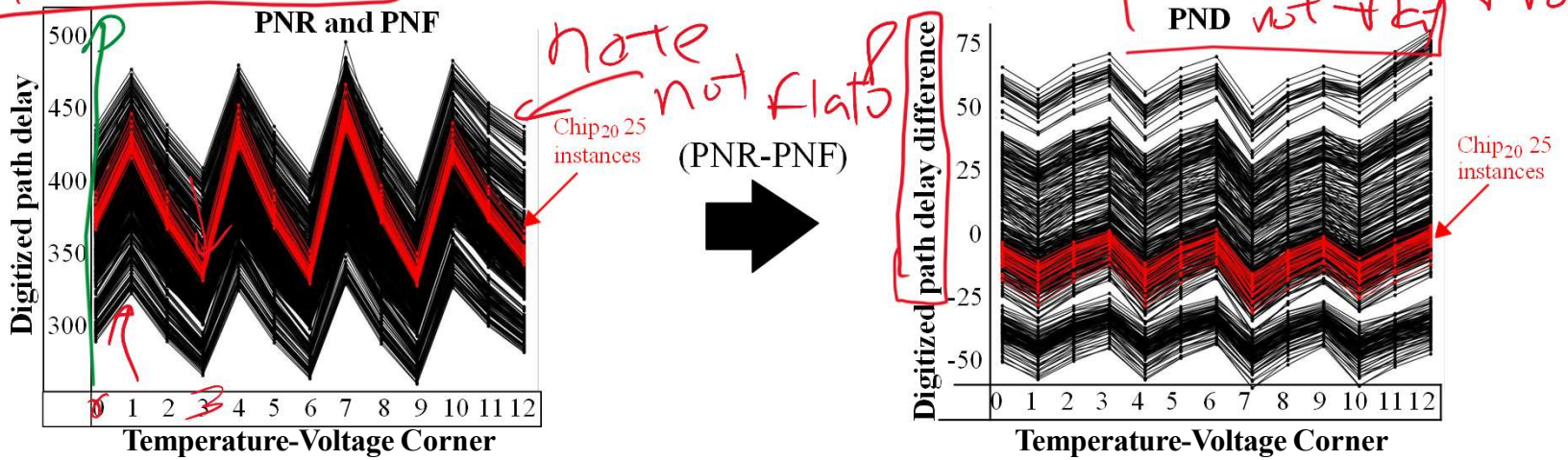
(iii) However, by choosing from a bin of 2048 PNR values and 2048 PNF values $2048^2 = n^2$ combinations possible

(iv) Each LFSR sequences through the LFSR values (an m-bit LFSR has 2^m possible states, $m=11 \Rightarrow 2^{11} = 2048$ values)

(v) details of the two LFSRs are omitted and how they produce PNR + PNF selections are omitted other than that two 11-bit LFSR seeds are user supplied parameters

HELP Processing Steps

Illustration of one PNR and one PNF, collected across 12 TV corners (x-axis) and 500 chips-instances (y-axis)



Legend				
0: 25ºC, 1.00V (enroll)	**1**: 25ºC, 0.95V	**4**: 0ºC, 0.95V	**7**: -40ºC, 0.95V	**10**: 85ºC, 0.95V
	2: 25ºC, 1.00V	**5**: 0ºC, 1.00V	**8**: -40ºC, 1.00V	**11**: 85ºC, 1.00V
(regen)	**3**: 25ºC, 1.05V	**6**: 0ºC, 1.05V	**9**: -40ºC, 1.05V	**12**: 85ºC, 1.05V

Single PNR/PNF illustrate that shifting and scaling is significant, while PND in right plot show reduced *jig-saw* pattern

Goal is to have *flat horizontal* lines, i.e., all TV corners produce same PND

The data from the 25 instances from $Chip_{20}$ are highlighted in red to illustrate performance similarities

The large spread along *y-axis* is largely due to *chip-to-chip* variations

HELP Processing Steps

Its clear that the difference operation is NOT able to remove all of the path delay variation introduced by TV-noise

STEP 3: Apply TVCompensation (**TVComp**) to remove remaining TV-noise

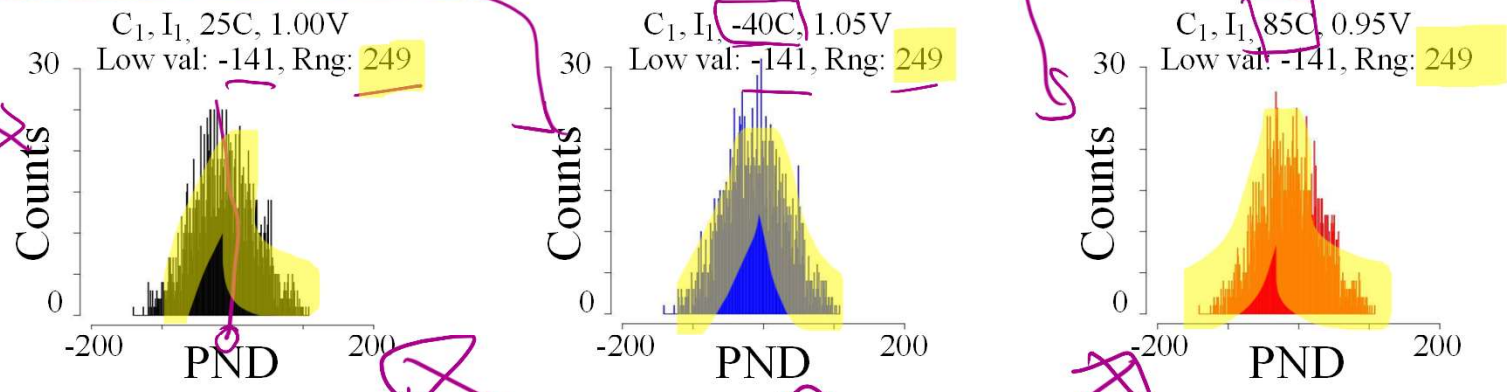
$$zval_i = \frac{(PND_i - \mu_{chip})}{Rng_{chip}}$$

$$PND_c = zval_i Rng_{ref} + \mu_{ref}$$

The μ_{chip} and Rng_{chip} are computed from a histogram distribution

The *ref* values are *user-specified* parameters

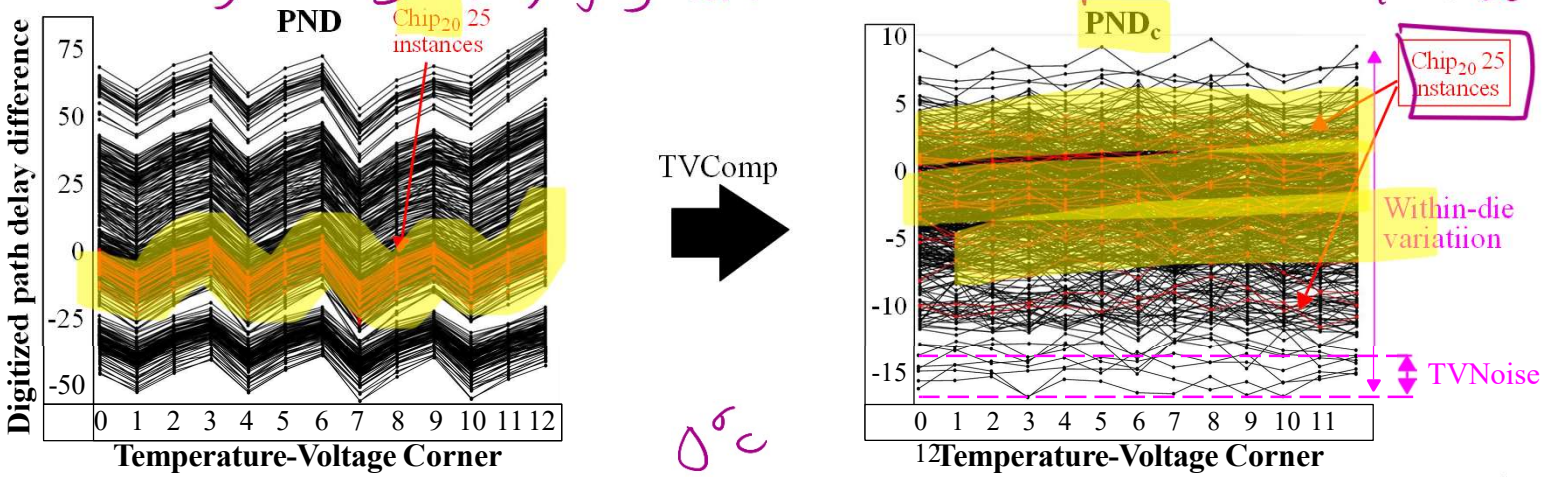
TVComp creates a histogram distribution of PND, and then scales and shifts the path delay distribution to a *reference* distribution



The *reference* distribution values **expand** the response space of HELP in a similar fashion to the 2 LFSR seeds used to create the PND from the PNR and PNF

HELP Processing Steps

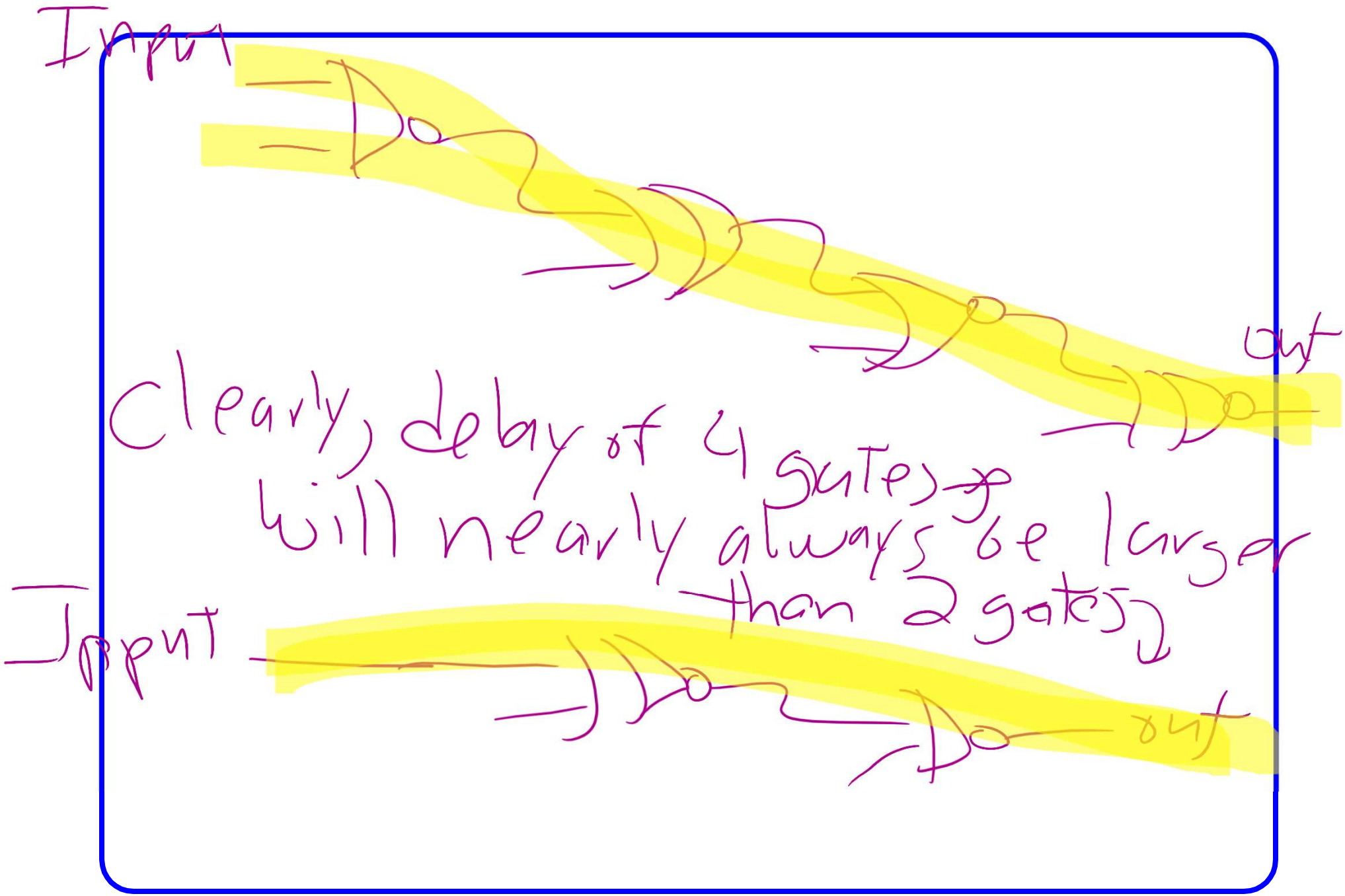
TVComp ELIMINATES all **chip-to-chip** variations, but preserves **within-die** variations



Legend				
0: 25ºC, 1.00V (enroll)	**1**: 25ºC, 0.95V	**4**: 0ºC, 0.95V	**7**: -40ºC, 0.95V	**10**: 85ºC, 0.95V
(regen)	**2**: 25ºC, 1.00V	**5**: 0ºC, 1.00V	**8**: -40ºC, 1.00V	**11**: 85ºC, 1.00V
	3: 25ºC, 1.05V	**6**: 0ºC, 1.05V	**9**: -40ºC, 1.05V	**12**: 85ºC, 1.05V

This fact is illustrated on the right with **PND_c**, which show the data from the 25 instances from $Chip_{20}$ now distributed across entire range of y-axis

In contrast to the grouping of $Chip_{20}$ data on the left, which shows similar performance among the different instances, as expected b/c data is from same chip

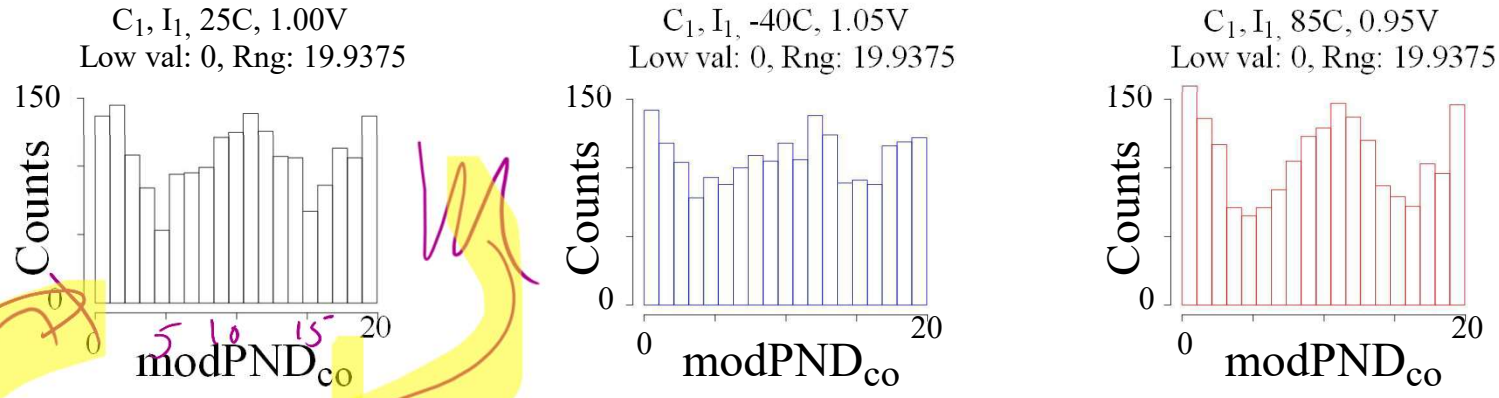

Input
out
clearly, delay of 4 gates
will nearly always be larger
than 2 gates
Input
out

HELP Processing Steps

The PND_c, although *compensated* for TV variations, still possess *path length* bias

Bias is delt with in two ways, first by optionally applying an *Offset* (for fine tuning) and then using a coarse-grained *Modulus* operation

STEP 4: Add server-computed **Offsets** (computed using enrollment data) and then apply a **Modulus** operation to remove path length bias

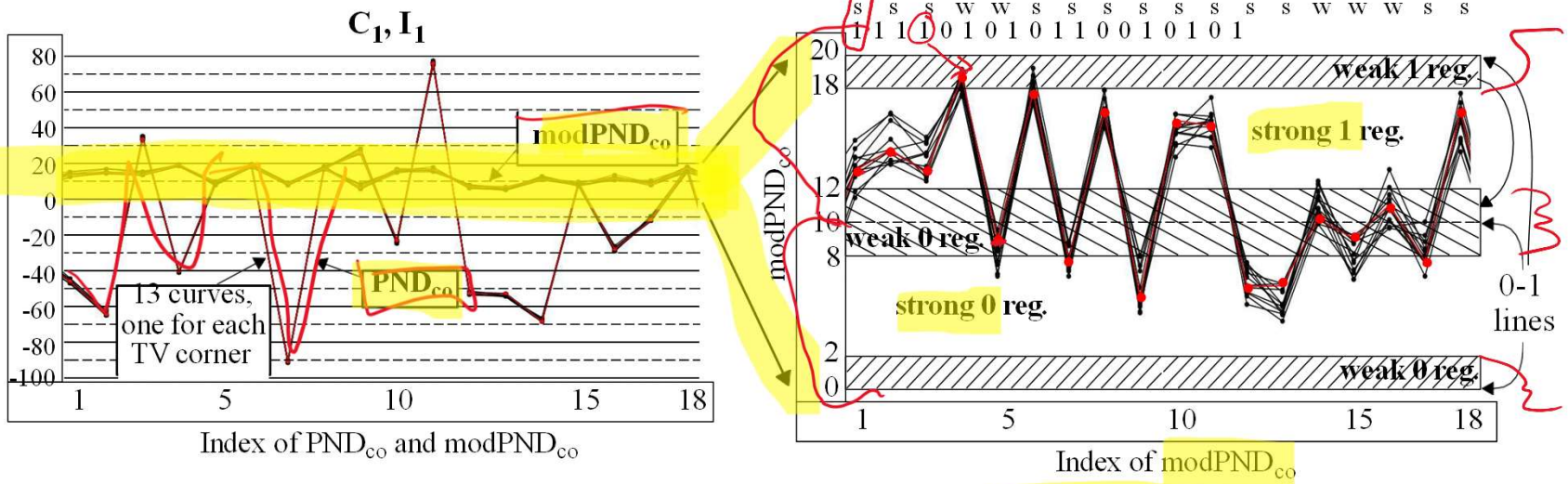


Offsets are computed from the **median of the chip population** and are added to each PND_c, which shifts pop. to a multiple of 10 and then a **Modulus** of 20 is applied

The PND_c with offsets are called $\mathbf{PND_{co}}$ and the final values are called $\mathbf{modPND_{co}}$

HELP Processing Steps

STEP 5: Bitstring generation uses a **Margin** parameter, that implements a *bit-flip avoidance* reliability-enhancing scheme



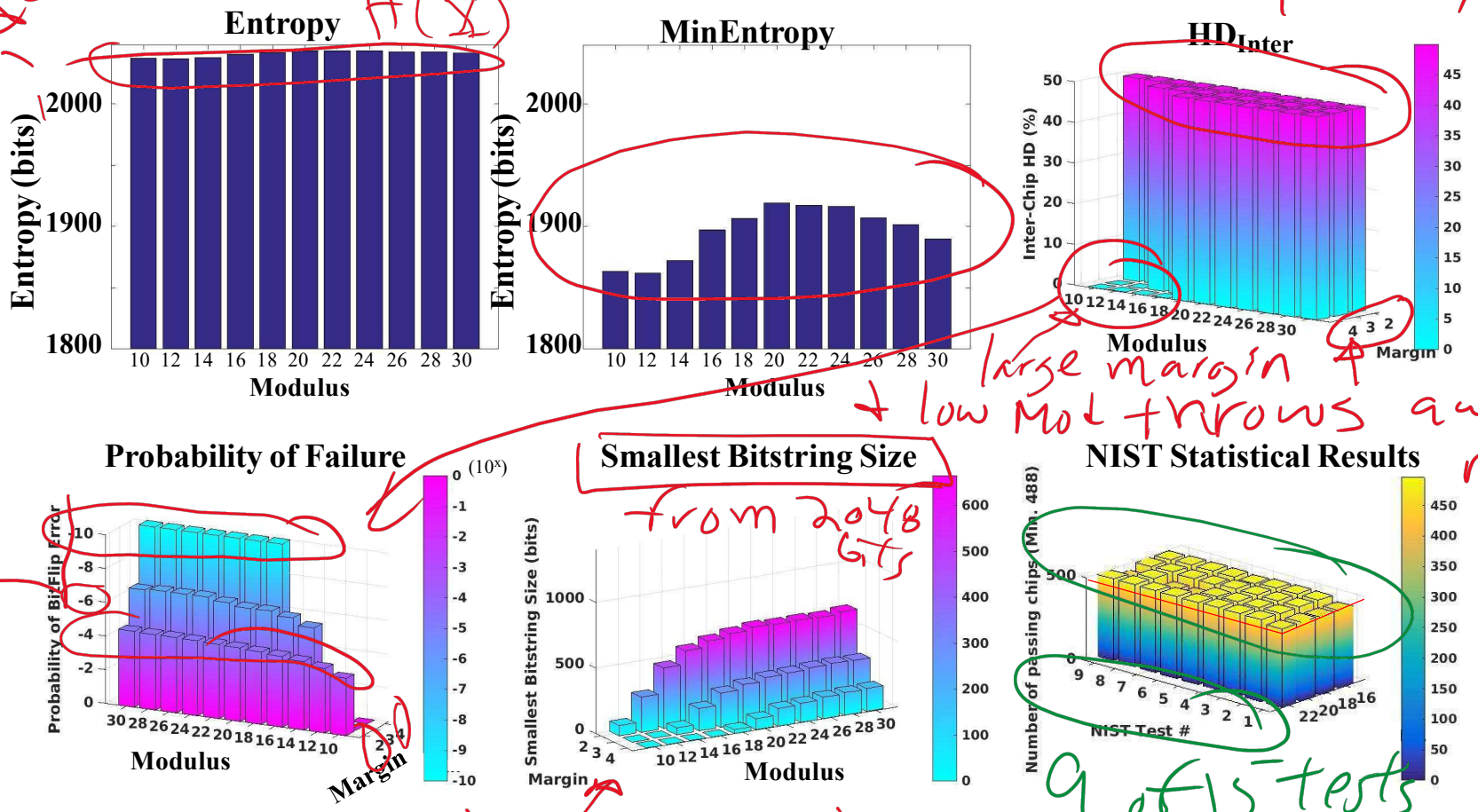
We call this the *Single Helper Data* scheme b/c the Margin scheme is run only by the token during enrollment

We also have a *Dual Helper Data* scheme that combines helper data generated by both the token and server

We have a suite of reliability-enhancing schemes for stand-alone (no server) applications, e.g,, key-encryption-key (**KEK**) mode

HELP Statistical Results

- Statistics using the Offset method



These statistical results indicate the bitstrings generated by HELP are of cryptographic quality

HELP Area Overhead

HELP Module	MUX	Carry	LUTs	FFs
PUF: CollectPNs	15	9	288	79
PUF: ComputeModulus	0	18	194	67
PUF: ComputePNDiffs	0	27	212	101
PUF: DataTransferIn	8	4	513	202
PUF: DataTransferOut	0	0	12	10
PUF: DualHelpBitGen	4	31	346	117
PUF: EvalMod	96	0	299	773
PUF: Entropy Source: (*sbox-mixedcol)* (nets 3564)	0	0	3365	128
PUF: LaunchCaptureEngine	0	0	78	11
PUF: LCTest_Driver	1	7	40	17
PUF: LoadUnLoadMem	0	6	72	19
MstCtrl: Master State Machine	15	38	342	85
PUF: PhaseAdjust	0	7	58	30
PUF: SingleHelpBitGen	0	20	310	98
PUF: SecureKeyEncoder (SKE)	0	15	303	122
PUF: TVComp	0	49	421	155
Totals	**139**	**231**	**6855**	**2014**

Additional resources include 1 MMCM, a 16 KB BRAM and a 24-bit multiplier

Note that this implementation of HELP includes all four functions, including *token authentication*, *verifier authentication*, *session encryption* and *KEK*

Versions dedicated to one function would be smaller in size

Supply Chain: FPGA

Volumes huge &

RoT SecureBoot

if 1st key is

BIOS PUF key

then 1st key

not in NVMB