Introduction

We discussed the basic tenets of information security, including confidentiality, data integrity, authentication and non-repudiation

Algorithms have been developed that provide these security functions, including unkeyed hash functions, block ciphers, MACs and digital signatures

These algorithms assume a *black box* implementation, where users can **only** interact with the algorithm through its inputs and outputs

The following assumptions are often made (from Maes text):

- **Secure key generation**: A secure, i.e., random, unique and unpredictable, key can be generated for security primitives such as block ciphers
- **Secure key storage**: The key can be stored and retrieved by the instantiation without being revealed
- **Secure execution**: The instantiation of the primitive can execute without revealing any information about the key or internal intermediate results

  And without an adversary being able to influence the internal execution

Introduction

Unfortunately, these assumptions are no longer true and **physical layer** countermeasures are now needed

For example, secure key storage requires specialized technology to provide *secure NVMs*, but recent work shows that even these are vulnerable

Similarly, secure execution requires special design techniques to thwart *side-channel attacks*

Physical layer security is implemented using primitives and methods including:

- **True Random Number Generators (TRNGs)**: Distillation of random numbers from physical random sources for protocols and algorithms

- **Design Styles**: Implementations that minimize and ideally eliminate certain physical side channels leakages and vulnerabilities

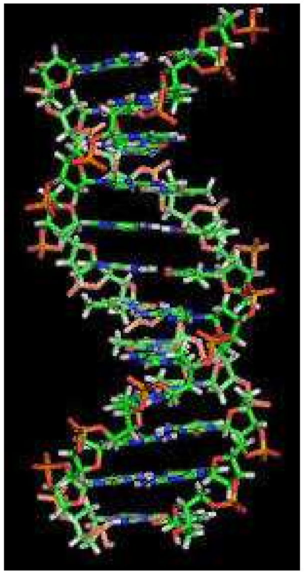
- **Physical Unclonable Functions (PUFs)**: Primitives that produce unpredictable, reliable and instance-specific bitstrings, without the need for NVM

Introduction

PUF definition: ***An inherent and unclonable instance-specific feature of a physical object***

Akin to biometric features in humans, such as fingerprints, iris characteristics and DNA





PUF Constructions: What do they look like and what do they leverage?

PUFs take advantage of *technical limitations* that exist in the physical process of fabricating integrated circuits

PUF Constructions

Even with *extreme* control over a fabrication process, no two physically identical instances of a chip can be created b/c of random and uncontrollable effects

The differences are typically very small, i.e., they exist at the nanometer scale, and require high-precision techniques to measure them

A PUF is defined as a combination of

- A *physical source of randomness* (**Entropy**), i.e., an integrated circuit component that exhibits *within-die* variations
- A *measurement technique* that can convert small analog signal differences introduced by chip-to-chip/within-die variations into unique digital bitstrings

Variations refer to geometrical and chemical *imperfections* that exist in nanometer-sized components on the chip

Makes multiple *designer-drawn* exact replicas of a component *slightly different*

These physical imperfections manifest as changes in the *electrical characteristics* of the component, which is typically what the PUF measurement technique targets

PUF Constructions

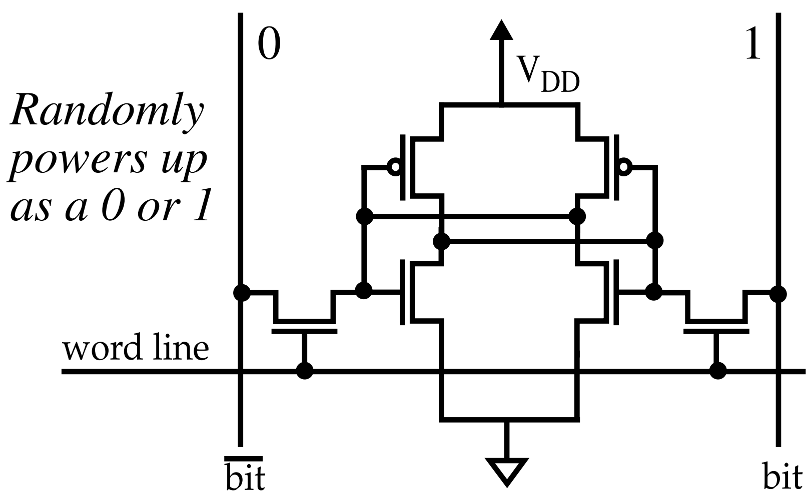
The number of proposed PUF constructions has increased *exponentially*

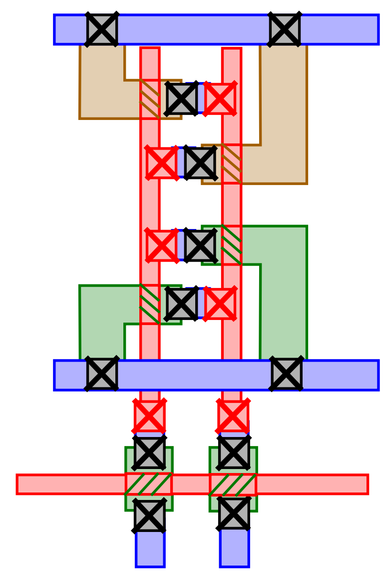
This has occurred because of the vast array of opportunities that exist to construct/configure IC functional components as the source of entropy

Our focus will be on *intrinsic PUFs*

Intrinsic PUFs are defined as those that include both an *entropy source* and an *on-chip measurement method* to produce digital bitstrings

A simple example: SRAM:





Symmetric and identical as drawn

PUF Constructions

We will use the following notations (from Maes text) in reference to PUFs and their properties:

- **PUF Class**: A PUF class will be denoted by *P*, which includes a complete description of a particular PUF construction type

  P.Create is a creation procedure used to create instances of *P*, which refers to the detailed physical fabrication processes used to build an instance of a PUF

  P.Create(r^c), with $r^c \overset{\$}{\leftarrow} \{0, 1\}^*$, refers to the **probabilistic** nature of the PUF creation process

- **PUF Instance**: A PUF *instance* created from class *P* will be referred to as *puf*

  As we will see, most PUF constructions (classes) accept inputs, called ***challenges***, that configure the PUF in a specific state *x*

  Therefore *puf(x)* refers to the application of challenge *x* to a PUF instance *puf*

  The set of all possible challenges for class *P* is denoted χ_P

PUF Constructions

- **PUF Evaluation**: The evaluation of a PUF is referred to as *puf.Eval*

  Evaluation produces a quantitative outcome, i.e., a ***response***, which depends on the state x (the challenge)

  puf(x).Eval represents a *probabilistic* response of *puf* under challenge x

  The set of all possible responses is referred to as Υ_P

Note that the instance-specific response of a PUF is affected by

- **Fixed within-die variations** that occur within the embedding chip
- **Environmental conditions**, e.g., temperature and supply voltage
- Slow changes in transistor parameters over time, **wear-out effects**

Environment *conditions* are denoted by α as $puf(x).Eval^{\alpha}$

The PUF response is generally considered a *random variable* with a characteristic *probability distribution*

The *distribution* is typically determined from simulation or hardware experiments for a given PUF class P

PUF Constructions

A statistical analysis of a PUF response is typically composed of three components (or dimensions):

- Responses from different PUF instances, i.e., different chips (**uniqueness**)
- Responses from the same PUF instance using different challenges (**randomness**)
- Responses from the same PUF instance using the *same* challenges but under different *conditions* (**reliability**)

Definition: An $(N_{puf}, N_{chal}, N_{meas})^{\alpha}$-experiment on a PUF class P is an array of PUF responses of size N_{puf} x N_{chal} x N_{meas}

N_{puf} refers to the number of PUF instances (chips)

N_{chal} refers to the number of challenges (each producing 1 response bit)

N_{meas} refers to the number of evaluations (samples)

PUF Statistical Metrics for Reliability

As mentioned earlier, PUF responses are affected by environmental conditions α

Beyond temperature and supply voltage variations, measurement noise also introduces changes in a PUF's response

This fact makes a PUF a ***probabilistic function*** (as opposed to a real function that always produces the same result for a given input)

Although this feature can be leveraged in cases where the PUF is used as a TRNG, it represents a serious issue for key generation and authentication applications

As we will discuss, a PUF will require **helper data** to accomplish what is normally possible with NVM memories, i.e., precise reproduction of the bitstring

Intra-chip hamming distance (**$\mathbf{HD_{intra}}$**): A metric that measures the *resilience* of a PUF to environmental conditions α and β:

$$HD_{intra}(x) \cong \mathrm{dist}[\Upsilon_i^{\alpha}(x);\Upsilon_i^{\beta}(x)]$$

where $\Upsilon_i^{\alpha}(x)$ and $\Upsilon_i^{\beta}(x)$ are two distinct evaluations of puf_i using x

PUF Statistical Metrics for Reliability

HD_{intra} is used to measure the difference in the responses of **one particular PUF instance** evaluated with the **same challenge** *x*

- The process of producing the bitstring the first time is called **enrollment**
- The process of reproducing the bitstring is called **regeneration**
- HD_{intra} measures the *number of differences* (the **Hamming distance** between the bitstrings) that occur in the bitstring during subsequent regenerations

$\mathbf{HD_{intra}}$ expresses the *average noise* in the responses, and reflects **reproducibility** (or **reliability**)

- Therefore, the idea value for HD_{intra} is 0%

For example:

```
1 0 1 0 0 1 0 1 1 0
1 0 1 0 1 1 0 1 1 0
---------------------------
0 0 0 0 1 0 0 0 0 0 = 1/10 = 10%
```

Row 1: ($Chip_0$ bitstring during enrollment under conditions α_1)

Row 2: ($Chip_0$ bitstring during regeneration under conditions α_2)

Result: (HD_{intra})

α_1 might be 25^{o}C, 1.00V while α_2 might be 100^{o}C, 1.05V

PUF Statistical Metrics for Reliability

The HD_{intra} characteristics of a PUF class P are critically important to the practical utility of the PUF

Most published literature on PUFs report HD_{intra} by carrying out **hardware experiments** that introduce changes in the environmental conditions α

Small analog differences in the behavior of the PUF introduced by measurement and temperature/voltage noise (**TV noise**) are very difficult to model accurately

Therefore, predicting HD_{intra} from theoretical or simulation experiments is only OF LIMITED VALUE, and you should be very skeptical of the results

The chips which embed the PUF are often classified according to the range of environmental conditions that they are tolerant to:

- *Commercial grade*: Typically 0^oC to 85^oC, +/- 5% supply voltage
- *Industrial grade*: Typically -40^oC to 100^oC, +/- 10% supply voltage
- *Military grade*: Typically -60^oC to 125^oC, +/- 10% supply voltage

PUF Statistical Metrics for Reliability

Environmental conditions can be controlled using temperature chambers and precision power supplies

A thorough exploration of the HD_{intra} characteristics involves carrying out regeneration across all ***TV corners***

- Enrollment typically done at 25^{o}C, nominal supply voltage
- Regeneration typically done at all combinations of temperatures, e.g., 0^{o}C, 25^{o}C and 85^{o}C, and supply voltages, -5%, nominal and +5%

Therefore, for each chip, a set of 10 bitstrings are produced

HD_{intra} can be computed by counting the number of *bit-wise differences* that occur in the bitstrings using:

- Enrollment and each of the 9 bitstrings from the *TV corners* (9 comparisons) OR
- **All combinations** of the bitstrings, i.e., 10*9/2 (45 comparisons)

Applications such as *encryption* require all combinations, while *authentication* can be relaxed

PUF Statistical Metrics for Reliability

A mean HD_{intra} in a $(N_{puf}, N_{chal}, N_{meas})^{\alpha}$-experiment, where $\alpha = 10$ is computed as follows (when the *all combinations* method is used):

$$\mu_{intra} = \overline{HD}_{intra} = \frac{2}{N_{puf} \bullet N_{chal} \bullet \alpha \bullet (\alpha - 1)} \sum HD_{intra}$$

In words, count the # of differences across all 45 pairings of bitstrings for each chip, sum them across all chips and divide by the total # of bit-wise comparisons

A ***standard deviation*** can be computed in a similar manner

A *distribution* can also be created which plots:

- The number of differences along the x-axis for each pairing
- Against the number of times that difference is observed across all pairings, e.g., 45 * N_{puf}

With N_{puf} = 30 chips, the histogram is created from 1350 HD_{intra} values

HD_{intra} for different PUF classes P vary from 2% to 15% or larger

Error correction/avoidance methods are used to deal with this problem

PUF Statistical Metrics for Uniqueness

***Inter-chip hamming distance* (HD$_{\mathbf{inter}}$)**: A metric that measures the *uniqueness* of a PUF, i.e., how different its responses are when compared to other PUFs:

$$HD_{inter}(x) \cong \mathrm{dist}[\Upsilon_i^{\alpha}(x); \Upsilon_j^{\alpha}(x)]$$

where $\Upsilon_i^{\alpha}(x)$ and $\Upsilon_j^{\alpha}(x)$ are two distinct PUF instances puf_i and puf_j evaluated under environmental conditions α on the same challenges x

HD$_{inter}$ is used to measure the difference in the responses of **two PUF instances** evaluated with the **same challenges** *x*

HD$_{\mathbf{inter}}$ expresses the ***uniqueness*** in the responses from different PUF instances

Therefore, the idea value for HD$_{inter}$ is 50%

For example:

```
1 0 1 0 0 1 0 1 1 0
```
(Chip$_0$ bitstring during enrollment under conditions α_1)

```
1 1 0 0 0 1 1 1 0 1
```
(Chip$_1$ bitstring during enrollment under conditions α_1)

```
---------------------------
0 1 1 0 0 0 1 0 1 1 = 5/10 = 50%
```
(HD$_{inter}$)

PUF Statistical Metrics for Uniqueness

A mean HD_{inter} in a $(N_{puf}, N_{chal}, N_{meas})^{\alpha}$-experiment, where $\alpha = 1$ is computed as follows:

$$\mu_{inter} = \overline{HD}_{inter} = \frac{2}{N_{puf} \bullet (N_{puf} - 1) \bullet N_{chal}} \sum HD_{inter}$$

In words, count the # of differences across all combinations of *bitstrings* from different PUF instances and divide by the total # of bit-wise comparisons

Note that usually **enrollment bitstrings** are used but bitstrings generated under any environmental condition α can be evaluated as well

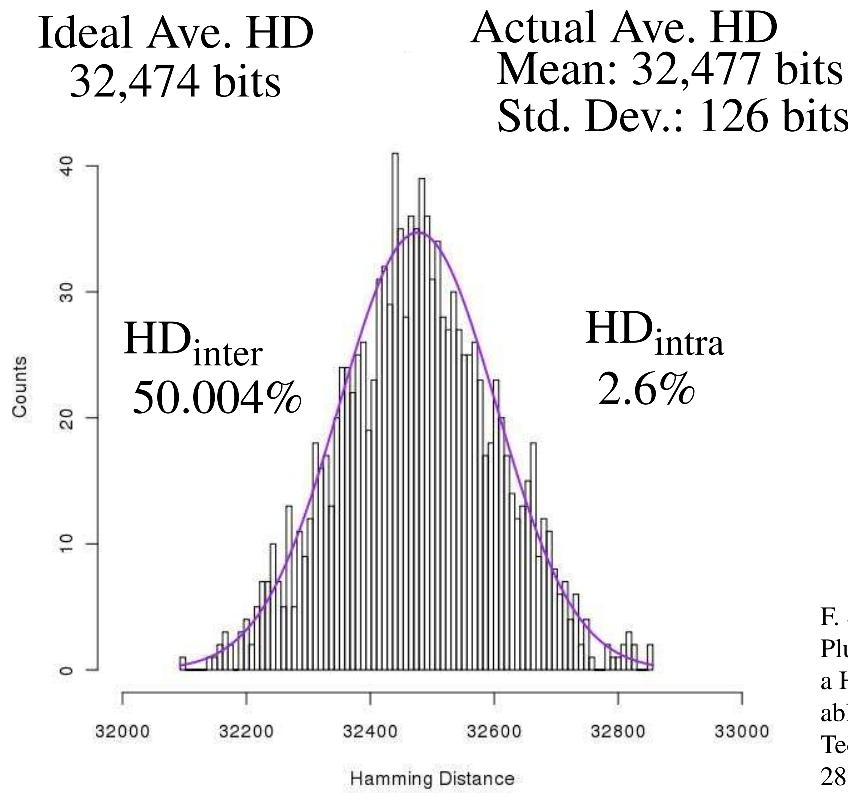
Similar to HD_{intra}, a standard deviation and *distribution* can be created from the

$\frac{N_{puf} \bullet (N_{puf} - 1)}{2}$ combinations

Mean values for different PUFs can vary dramatically from the ideal 50%, and depends heavily on whether *bias effects* are present

PUF Statistical Metrics for Uniqueness

With N_{puf} = 50 chips, the histogram is created from 50*49/2 = 1225 HD_{inter} values:



F. Saqib, M. Areno, J. Aarestad and J. Plusquellic, "An ASIC Implementation of a Hardware-Embedded Physical Unclonable Function", IET Computers & Digital Techniques, Vol. 8, Issue 6, Nov. 2014, pp. 288-299

Note that the distribution is actually characterized as **binomial** and not Gaussian

The expected standard deviation *std* of a binomial is given by

$$\text{std}_{\text{binomial}} = \sqrt{np(1-p)} = \sqrt{64948 \bullet 0.5 \bullet 0.5} = 127.4$$

PUF Statistical Metrics for Randomness

Randomness is more difficult to evaluate than reliability and uniqueness, and requires a suite of tests

Entropy and **MinEntropy** are measures of the disorder or randomness of a random variable X with probabilities p_i, ..., p_n, and are defined as follows:

$$H(X) = -\sum_{i=1}^{n} p_i \log_2 p_i \qquad \textbf{Entropy}$$

$$H_\infty(X) = \min_{i=1}^{n}(-\log_2 p_i) = -\log_2(\max_i(p_i)) \qquad \textbf{MinEntropy}$$

Entropy and MinEntropy measure the information content in a message

Interestingly, the more random a message is, the more information it has

For example, a *compressed* file has much more Entropy than the *uncompressed* version

Patterns in the message, such as those associated with encodings of English characters, can be re-encoded (compressed) using fewer bits

PUF Statistical Metrics for Randomness

For example, assume you analyze a set of 20 binary bits (01110111110101001101) produced by a random variable and obtain the following 'occurrence' results:

- 8 0's (or 8/20 = 0.40)
- 12 1's (or 12/20 = 0.60)

First we recognize that this variable is not ideal, i.e., it does NOT produce both bit values with equal probability of 50%

$$H(X) = -(p_1 \log_2 p_1 + (1 - p_1)\log_2(1 - p_1))$$

Entropy in a binary random variable with prob. of 1 given by p_1

We compute the Entropy using the above formula as:

$0.60*\log_2(0.60) + 0.40*\log_2(0.40) = 0.4422 + 0.5288 = 0.971$

We conclude that this random variable has less than 1 bit of Entropy

As indicated earlier, MinEntropy analyzes the most frequently occurring binary pattern and therefore, measures the worst case behavior of a random variable

In this example, MinEntropy is given as $-\log_2(0.60) = 0.7370$

PUF Statistical Metrics for Randomness

If 'occurence' statistics are known in advance, **Entropy encoding** schemes can be used to optimally encode messages, reducing their length, e.g., *Huffman* coding

There are MANY ways to compute Entropy w.r.t. PUFs, and you will see different methods used in the literature

chip/bit #	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	***H(x)***
C1	0	1	0	0	1	0	1	0	0	0	1	1	1	1	0	0	0	1	1	1	1.000
C2	1	1	0	1	1	1	1	1	0	1	0	0	1	1	0	0	0	1	0	0	0.993
C3	1	1	0	0	1	0	1	0	0	0	0	1	0	1	1	0	0	1	0	0	0.971
C4	1	1	1	1	0	1	0	0	1	1	1	0	0	0	0	0	0	1	1	1	0.993
C5	1	0	0	1	1	0	0	0	1	0	0	1	1	1	0	1	0	1	1	0	1.000
C6	1	1	0	0	0	1	0	1	1	0	1	1	0	1	1	0	1	0	0	0	1.000
C7	0	1	1	0	1	0	1	1	1	1	0	1	1	0	1	0	1	1	0	0	0.971
C8	0	1	1	1	0	1	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0.971
C9	0	0	0	0	1	1	0	1	1	1	0	0	1	0	0	0	0	0	0	0	0.881
C10	1	0	1	0	1	0	1	0	1	1	1	1	0	0	1	0	0	0	0	1	1.000
H(x)	0.97	0.88	0.97	0.97	0.88	1.00	0.97	1.00	0.97	1.00	1.00	0.88	1.00	1.00	0.97	0.47	0.72	0.97	0.88	0.88	

Ideal is for PUF-generated bitstrings to have Entropy of 1 across bitstrings **and** chips

PUF Statistical Metrics for Randomness

Entropy can also be computed over substrings of the bitstring, e.g.,

Second row of table:

1	1	0	1	1	1	1	1	0	1	0	0	1	1	0	0	0	1	0	0

The 4 possible patterns are "00", "01", "10" and "11", which are expected to occur at equal frequencies of 25% when the bitstring is random:

00: 3

01: 3

10: 0

11: 4

Here, Entropy is (note: $\log_2(0)$ is defined to be 0):

$$-0.3*\log_2(0.3) - 0.3*\log_2(0.3) - 0.0*\log_2(0.0) - 0.4*\log_2(0.4) = 1.571/2 \text{ bits} = 0.785 \text{ bits}$$

And MinEntropy is:

$$-\log_2(0.4) = 1.321/2 = 0.661 \text{ bits}$$

Substrings of any size can be analyzed in this fashion

PUF Statistical Metrics for Randomness

Conditional MinEntropy can also be computed using pairs of bits

It is used to determine if correlations exist, i.e., whether the 1st bit is dependent on the 2nd

$$H_{\infty}\langle X|W\rangle = -\log_2\left(max\left(\frac{p_X}{p_W}\right)\right)$$

Here, we compute p_X as usual for the 4 possible patterns

And then divide by p_W which is the probability that the second bit is a ’0’ for patterns "00" and "10" or ’1’ for patterns "01" and "11"

The Conditional MinEntropy for the 10 non-overlapping bit pairs on prev. slide:

```
Prob. 2nd bit is '0' => 0.3
Prob. 2nd bit is '1' => 0.7
```

Find max among 4 computed values of (p_X/p_W) given in this example by 0.3/0.3, 0.3/0.7, 0.0/0.3, 0.4/0.7

$-\log_2(0.3/0.3) = 0.000$ (when 2nd bit 0, so is 1st bit)

PUF Statistical Metrics for Randomness

This material is derived from the NIST published document:

"A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications"

A **random bit sequence** can be interpreted as the result of a sequence of 'flips' of an unbiased (fair) coin

With sides labeled '0' and '1', each flip has probability of exactly 1/2 of producing a '0' or '1'

Also, the 'flip' experiments are **independent** of each another

The fair coin toss experiment is an example of a *perfect* random bit generator because the '0's and '1's are randomly and uniformly distributed

It is not possible to predict the result of the next trial with probability greater than 50%, i.e., the result is *uncertain*

PUF Statistical Metrics for Randomness

Random Number Generators (RNGs)

An RNG uses

- A *non-deterministic* source (the **Entropy source**, e.g., noise in an electrical circuit)
- A processing function called **Entropy distillation** to improve randomness

Distillation is used to overcome any weaknesses in the entropy source that results in generation of non-random sequences (distillation can be done with XOR)

There are an *infinite number* of possible statistical tests that can be applied to a sequence to determine whether 'patterns' exist

Therefore, no *finite* set of tests is deemed complete

Statistical tests are formulated to test a specific **null hypothesis** (H_0)

Here the null hypothesis-under-test is that the sequence being tested is **random**

The antonym to H_0 is the alternative hypothesis (H_a), that the sequence is NOT random

PUF Statistical Metrics for Randomness

Each test has an underlying ***reference distribution*** which is used to develop a **critical value**, e.g., a value out on the tail of the distribution, say at 99%

The **test statistic** computed for the sequence is compared against the critical value, and if larger, the sequence is deemed NOT random (H_0 is rejected)

The premise is that the tested sequence, if random, has a very low probability, e.g., 0.01%, of exceeding the critical value

The probability of a **Type I** error, i.e., the data is actually random but the test statistic exceeds the critical value, is often called the **level of significance**, α

- A commonly used value for α is 0.01

Analogously, the probability of a **Type II** error, i.e., the data is not random but passes the test, is denoted by β

- β (unlike α) is NOT a fixed value because there are an infinite number of ways a sequence can be non-random

PUF Statistical Metrics for Randomness

The NIST tests attempt to minimize the probability of a Type II error

Note that the probabilities α and β are related to each other and to the size n of the tested sequence

And the third parameter is dependent on the other two

Usually sample size n and an α are chosen, and a critical value is computed that minimizes the probability of a Type II error

A **test statistic** S is computed from the data, and is compared to the critical value t to determine whether H_0 is accepted

S is also used to compute a ***P-value***, a measure of the *strength* of the evidence **against** H_0

Technically, the *P-value* is the probability that a perfect RNG would have produced a sequence **less random** than the sequence-under-test

PUF Statistical Metrics for Randomness

If the *P-value* is 1, then the sequence appears to have *perfect* randomness, if 0, then its completely non-random, i.e., **larger** *P-values* support randomness

A significance level, α, is chosen and indicates the probability of a Type I error

If the *P-value* >= α, then H_0 is accepted, otherwise it is rejected

If α is 0.01, then one would expect 1 truly random sequence in 100 to be rejected

Two major assumptions:

- **Uniformity**: At **any** point in the generation of a random bit sequence, the number of '0's and '1's is equally likely and is 1/2, i.e., expected number of '1's is *n/2*

- **Scalability**: Any test applicable to a sequence is also applicable to a **subsequence** extracted at random, i.e. all subsequences are also random

PUF Statistical Metrics for Randomness

The NIST Test Suite has 15 tests -- for many of them, it is assumed the bit sequence is large, on order of 10^3 to 10^7

1) Frequency Test:

Counts the number of ‘1’ in a bitstring and assesses the closeness of the fraction of ‘1’s to 0.5 (failing frequency usually means failure of most other tests)

2) Block Frequency Test:

Same except bitstring is partitioned into *M* blocks. Ensures bitstring is ‘locally’ random

3) Runs Test:

Analyzes the total number of *runs*, i.e., uninterrupted sequences of identical bits, and tests whether the oscillation between ‘0’s and ‘1’s is too fast or too slow

4) Longest Run Test:

Analyzes the longest run of ‘1’s within *M-bit* blocks, and tests if it is consistent with the length of the longest run expected in a truly random sequence

PUF Statistical Metrics for Randomness

5) Rank Test:

Analyzes the linear dependence among fixed length substrings, and tests if the number of rows that are linearly independent match the number expected in a truly random sequence

6) Fourier Transform Test:

Analyzes the peak heights in the frequency spectrum of the bitstring, and tests if there are *periodic* features, i.e., repeating patterns close to each other

7&8) Non-overlapping and Overlapping Template Tests:

Analyzes the bitstring for the number of times *pre-specified* target strings occur, to determine if too many occurrences of non-periodic patterns occur

9) Universal Test:

Analyzes the bitstring to determine the *level of compression* that can be achieved without loss of information

NIST Test Suite for Randomness

10) Linear Complexity Test:

Analyzes the bitstring to determine the length of the smallest set of LFSRs needed to reproduce the sequence

11&12) Serial and Approximate Entropy Tests:

Analyzes the bitstring to test the frequency of all possible 2^m overlapping *m-bit* patterns, to determine if the number is uniform for all possible patterns

13&14) Cumulative Sums Test:

Analyzes the bitstring to determine if the cumulative sum of incrementally increasing (decreasing) *partial sequences* is too large or too small

15) Random Excursions Test:

Analyzes the total number of times that a *particular state* occurs in a cumulative sum random walk

The **National Institute of Standards and Technology** (NIST) statistical tools

http://csrc.nist.gov/groups/ST/toolkit/rng/documentation_software.html

NIST Test Suite for Randomness

NIST 'finalAnalysisReport' using HELP ASIC

- 50 chips
- 64,948 bits/chip

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-value	P/F	Proportion	P/F	Statistical test
2	4	5	6	7	5	5	5	5	6	0.956		50/50		Frequency
5	6	8	7	3	7	6	2	4	2	0.494		49/50		Block Frequency
4	2	5	6	5	4	8	7	4	5	0.817		50/50		CumulativeSums
4	1	6	7	8	4	3	4	7	6	0.494		50/50		CumulativeSums
12	3	10	7	2	2	4	5	2	3	0.007		47/50		Runs
5	6	5	6	5	6	4	7	5	1	0.851		49/50		LongestRun
9	8	3	4	4	8	4	3	2	5	0.290		50/50		Rank
8	3	4	5	6	4	5	5	7	3	0.851		50/50		FFT
6	1	5	5	8	2	6	6	6	5	0.575		50/50		NonOverlapping Template
...	...	...	...	...	...	...	...	...	...	...		...	*	...
2	6	5	7	5	4	6	4	6	5	0.936		50/50		ApproximateEntropy
5	6	5	7	6	3	7	4	6	1	0.699		49/50		Serial
7	6	7	2	2	9	7	4	4	2	0.237		50/50		Serial

The minimum pass rate for each statistical test with the exception of the random excursion (variant) test is approximately = 47 for a sample size = 50 binary sequences