PUFs 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, MAC 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 every instantiation of the security primitive, e.g., block cipher.

- **Secure key storage**: The key can be assigned to, stored and retrieved by the instantiation without being revealed.

- **Secure execution**: The instantiation of the primitive can execute the crypto algorithm without revealing any (partial) information about the key or internal results, and without an adversary being able to influence the internal execution.
PUFs Introduction

Each of these assumptions are becoming increasingly difficult to meet, and require another layer, called **physical security**, as countermeasures.

Key storage, for example, is typically accomplished using *secure* NVMs, but there is recent research that demonstrates access and theft of the key is possible.

And as we discussed (briefly), secure execution is threatened by *side-channel attacks*.

Physical layer security is needed and is implemented using primitives and techniques including:

- **True Random Number Generators (TRNGs):** Distillation of random numbers from truly physical sources of randomness which can be used to produce highly random keys for crypto apps.

- **Design Styles:** Countermeasures that minimize and ideally eliminate certain physical side channels.

- **Physical Unclonable Functions (PUFs):** Primitives that produce unpredictable and *instance-specific* bitstrings, using implementations that provide physically secure key generation and storage.
PUFs Introduction

PUF definition (Maes): *An expression of an inherent and unclonable instance-specific feature of a physical object*

Akin to biometric features in humans, such as fingerprints or DNA (with the exception of identical twins which I happen to be :)

**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, are 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 analog-to-digitally convert small analog signal differences introduced by within-die variations to unique digital bitstrings

Within-die variations refers to geometrical and chemical *imperfections* that exist in nanometer-sized components on the chip, that makes multiple *designer-drawn* exact replicas of the component *slightly different* even among neighboring copies.

These physical imperfections manifest as changes in the *electrical characteristics* of the component, which is typically what the PUF measurement technique targets.
The number of proposed PUF constructions has increased *exponentially* over the last several years. 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 a measurement and processing mechanism to produce digital bitstrings.

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 $r^c \leftarrow \{0, 1\}^*$, refers to the probabilistic nature of the PUF creation process.
**PUF Constructions**

- **PUF Instance**: A PUF instance created from class $P$ will be referred to as a $puf$
  
  As we will see, most PUF constructions (classes) accept inputs, called *challenges*, that configures 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 $\chi_P$

- **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)
  
  $puf(x).Eval$ refers to the *probabilistic* response of $puf$ under challenge $x$
  
  The set of all possible responses is referred to as $\Upsilon_P$

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 the fixed parameters over time**, *wear-out effects*

Environment *conditions* are denoted by $\alpha$ as $puf(x).Eval^\alpha$
**PUF Constructions**

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 $P$

Response analysis from experiments 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 $N_{puf} \times N_{chal} \times N_{meas}$

$N_{puf}$ refers to the number of PUF instances (chips) while $N_{chal}$ and $N_{meas}$ refer to the number of challenge-response-pairs (CRPs) under condition $\alpha$
PUF Statistical Metrics

As mentioned earlier, PUF responses are affected by environmental conditions $\alpha$.

Beyond temperature and supply voltage variations, measurement noise also introduces changes in a PUF’s responses.

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* ($\text{HD}_{\text{intra}}$): A metric that measures the *resilience* of a PUF to environmental conditions:

$$\text{HD}_{\text{intra}}(x) \equiv \text{dist}[\gamma_i(x);\gamma'_i(x)]$$

where $\gamma_i(x)$ and $\gamma'_i(x)$ are two distinct evaluations of the PUF on $x$. 


PUF Statistical Metrics

$HD_{\text{intra}}$ 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.

$HD_{\text{intra}}$ measures the number of differences (the Hamming distance between the bitstrings) that occur in the bitstring during one or more subsequent regenerations.

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

Therefore, the idea value for $HD_{\text{intra}}$ is 0%.

For example:

<table>
<thead>
<tr>
<th>Chip₀ bitstring during enrollment under conditions $\alpha_1$</th>
<th>Chip₀ bitstring during regeneration under conditions $\alpha_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 1 0 0 1 0 1 1 0</td>
<td>1 0 1 0 1 1 0 1 1 0</td>
</tr>
<tr>
<td>-------------------------------------------------------------</td>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td>0 0 0 0 1 0 0 0 0 0 = $\frac{1}{10} = 10%$ (HD$_{\text{intra}}$)</td>
<td></td>
</tr>
</tbody>
</table>
PUF Statistical Metrics

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

Most published literature on PUFs report $HD_{\text{intra}}$ by carrying out hardware experiments that introduce changes in the environmental conditions $\alpha$.

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 $HD_{\text{intra}}$ from theoretical or simulation experiments is only of limited value, and you should be very skeptical of the results.

The chips on which the PUF is embedded are often classified according to the range of environmental conditions that are tolerant to:

- **Commercial grade**: Typically $0^{\circ}C$ to $85^{\circ}C$, +/- 5% supply voltage
- **Industrial grade**: Typically $-40^{\circ}C$ to $100^{\circ}C$, +/- 10% supply voltage
- **Military grade**: Typically $-60^{\circ}C$ to $125^{\circ}C$, +/- 10% supply voltage
PUF Statistical Metrics

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°C, nominal supply voltage.
- Regeneration typically done at all combinations of temperatures, e.g., 0°C, 25°C and 85°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 (enrollment is typically performed in a secure, controlled environment).
PUF Statistical Metrics

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

$$\mu_{intra} = \overline{HD_{intra}} = \frac{2}{N_{puf} \cdot N_{chal} \cdot N_{meas} \cdot (N_{meas} - 1)} \sum \overline{HD_{intra}}$$

In words, count the number of differences across all 45 pairings of bitstrings for each chip, sum them across all chips and divide by the total number of bitwise 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 produced across all pairings, e.g., $45 \times N_{puf}$.

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 as high as 15%.

The non-zero minimum value drives research on *error correction/avoidance*.
**PUF Statistical Metrics**

*Inter-chip hamming distance* ($\text{HD}_{\text{inter}}$): A metric that measures the *uniqueness* of a PUF:

\[
\text{HD}_{\text{inter}}(x) \equiv \text{dist} [\Upsilon_i(x) ; \Upsilon'_i(x)]
\]

where $\Upsilon_i(x)$ and $\Upsilon'_i(x)$ are two distinct PUF instances evaluated on the same challenge $x$

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

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

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

For example:

```
1 0 1 0 0 1 0 1 1 0  (Chip_0 bitstring during enrollment under conditions \( \alpha_1 \))
1 1 0 0 0 1 1 0 1 1  (Chip_1 bitstring during enrollment under conditions \( \alpha_1 \))
```

```
----------------------------------
0 1 1 0 0 1 0 1 1 = 5/10 = 50%  (\text{HD}_{\text{inter}})
```
PUF Statistical Metrics

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

$$\mu_{\text{inter}} = \frac{\sum HD_{\text{inter}}}{N_{\text{puf}} \cdot (N_{\text{puf}} - 1) \cdot N_{\text{chal}} \cdot N_{\text{meas}}}$$

In words, count the number of differences across all combinations of enrollment bitstrings and divide by the total number of bitwise comparisons.

Note that only enrollment bitstrings are used but enrollment conditions can vary depending on the application, so $\alpha$ evaluations may be needed.

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

$$\frac{N_{\text{puf}} \cdot (N_{\text{puf}} - 1)}{2}$$

combinations.

Mean values for different PUFs can vary dramatically from 20% to a near perfect 50%, and depends heavily on whether bias effects are present.
PUF Statistical Metrics

With $N_{puf} = 50$ chips, the histogram is created from $50 \times 49/2 = 1225$ HD\(_{\text{inter}}\) values:

- **Ideal Ave. HD**
  - 32,474 bits

- **Actual Ave. HD**
  - Mean: 32,477 bits
  - Std. Dev.: 126 bits

![Histogram of HD values](image)

Note that the distribution is actually characterized as binomial and not Gaussian but the two are very similar.

The expected std dev is given by

$$\text{std}_{\text{binomial}} = \sqrt{np(1-p)} = \sqrt{64948 \cdot 0.5 \cdot 0.5} = 127.4$$
**PUF Statistical Metrics**

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

The National Institute of Standards and Technology (NIST) provides a tool kit to evaluate randomness.

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

Please download and install this software on your laptop.

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’.
PUF Statistical Metrics

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

The result of the next trial is IMPOSSIBLE to predict!

Random Number Generators (RNGs)

An RNG uses a non-deterministic source (the entropy source, e.g., noise in an electrical circuit), plus a processing function (the entropy distillation process) to produce randomness

The distillation process is used to overcome any weaknesses in the entropy source that results in production of non-random sequences

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
PUF Statistical Metrics

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.

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 (data is actual random but test statistic exceeds critical value) is often called the level of significance, $\alpha$.

Common values used in crypto are 0.01.
PUF Statistical Metrics

Analogously, the probability of a **Type II** error (data is not random but passes the test) is denoted by $\beta$

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

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

Note that the probabilities $\alpha$ and $\beta$ are related to each other and to the size $n$ of the tested sequence

The third parameter is dependent on the other two

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

A **test statistic**, e.g. $S$ is computed from the data, and is compared to the critical value $t$ to determine whether $H_0$ is accepted
PUF Statistical Metrics

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

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

A significance level, $\alpha$, is chosen and indicates the probability of a Type I error

If the *P-value* $\geq \alpha$, then $H_0$ is accepted, otherwise it is rejected

If $\alpha$ is 0.01, then one would expect 1 truely random sequence in 100 to be rejected

A *P-value* $< 0.01$ indicates that the sequence is non-random with a **confidence** of 99%
**PUF Statistical Metrics**

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

**Entropy**

A measure of the *disorder* or *randomness* in a closed system

The entropy of uncertainty of a random variable $X$ with probabilities $p_1, ..., p_n$ is

$$H(X) = - \sum_{i=1}^{n} p_i \log p_i$$

$$H(X) = \frac{1}{\ln(2)} \log_2 \left( \frac{1}{p} \right) \quad \text{When } p_i = 1/n \text{ (equal probabilities)}$$

$$H_{\infty}(X) = \min_{i=1}^{n} (-\log_2 p_i) = -\log_2 (\max_{i} (p_i)) \quad \text{(min-entropy)}$$

A distribution has a min-entropy of at least $b$ bits if no possible state has prob. $> 2^{-b}$
NIST Test Suite for Randomness

Entropy

So a string of 10 binary values, one with worst case probability of occurrence of $1/500 = 0.002$, yields $-\log_2(0.002) = 8.966$ bits (the best you can achieve)

For NIST tests, if the bit sequence-under-test is non-random, then the calculated test statistic will fall in the *extreme regions* of the reference distribution

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. All other tests assume this test is passed.

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.
NIST Test Suite for Randomness

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.

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