Hardware & Embedded Security

Prof Daniele Rossi


Via G. Caruso 16, room B-1-03


daniele.rossi1@unipi.it


050 221 7611

Example of Applications of PUFs

Lecture 7 - DR

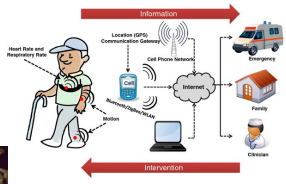
Brief outline

- Design and evaluation principles of PUF
- Quality metrics for PUF evaluation
- PUFs generating/storing cryptographic keys
- **PUFs for entity authentication protocols**
- PUFs for hardware metering

Examples of possible applications

3

Device Authentication



2

4

Another application of PUFs is for auth protocol.
We want to authenticate a device. Lots of situations in which this is needed.
PUFs can be used like a fingerprint, so it can be a good auth instrument we use for our device. Of course here you have an authority that needs to verify identity (authentication), called "verifier". Entity or object whose id needs to be verified is called a prover.
How do you do that?
When prover wants to be auth, it should provide evidence of claimed identity that was only generable by entity itself, and a proof that entity was actively involved in generating this evidence at auth time.
You can identify two stages in entity auth schemes:

↓

Device Authentication

- Numerous cases where in Entity authentication is necessary
- The identity of a physical object needs to be established before a service can be offered
- For example, in the case of cash withdrawal, the bank needs to ensure that the information stored on the card is authentic and consistent with those in its database, before it can release the money
- We refer to the authentication authority as the "***verifier***", for example the government is the verifier of e-passports and the bank is the verifier of the credit cards
- The entity or the physical object to be verified is sometime referred to as the "***prover***"

5

Device Authentication

When a physical entity wants to authenticate itself to a verifier, it needs to provide the following:

1. **Evidence of its claimed identity** which could have only been generated by the entity itself
2. A proof that entity was **actively involved** in generating this evidence at the time of the authentication

Generally, there are **two stages** of an **entity authentication scheme**

1. **Identity Provisioning** wherein **each device obtains a unique identity** such as a serial number or a binary code
2. **Verification** phase wherein the **verifier validate the identity of each entity**

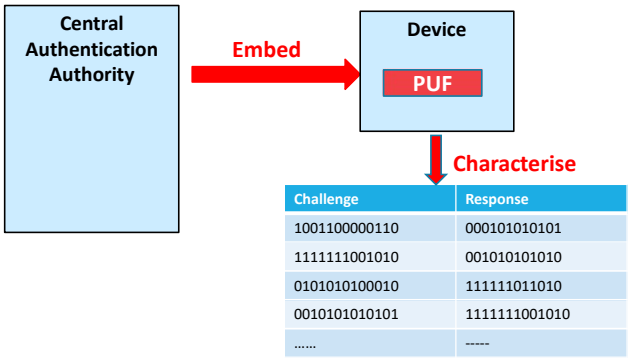
6

↳ can be used in the IoT world

Remote Authentication

Stage 1: Embedding and Characterization

1. The verifier (CAA) or a trusted third party embeds in each **entity** a PUF circuit and gives it a **unique identifier** (**ID**)
2. Verifier **applies a large number of challenges** on each PUF instance and **record the corresponding responses**
3. Verifier creates a **secure database**, in which he **stores the IDs** for all entities with their corresponding **PUF challenge/response pairs**



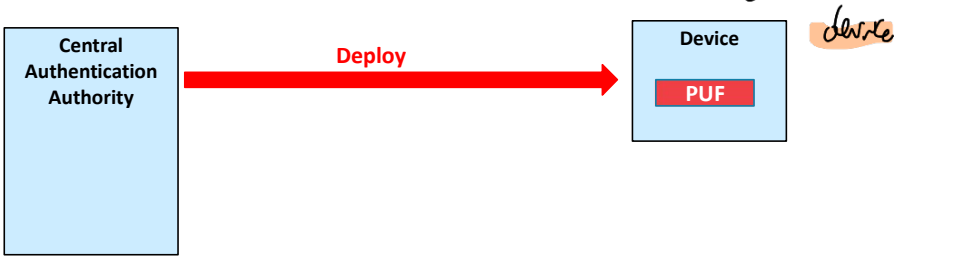
8

- Verifier associates ID to PUF, and in a database we associate IDs with the chl-rsp pairs.

Remote Authentication

Stage 2: Secure Device Deployment

We deploy PUF embedded device



9

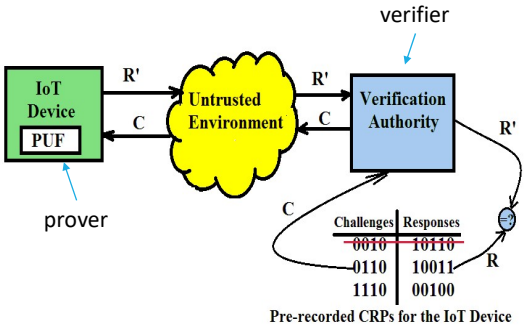
When device wants to access a certain service, prover sends the verifier its identifier. "I am this node". Verification Authority sends the prover a chl after selecting one from the DB. Prover applies chl, records response and sends it to the VA, that will check if response to that chl is the correct one. If match okay, if not, auth failed and response is deleted. After this, chl-rsp pairs is deleted from the ones usable in future, because chl-rsp are sent in clear.

To avoid DoS attacks, IoT device could use encryption that changes at each instance, pseudo random generation to append at end of message etc. Here we don't care about DoS.

Remote Authentication

Stage 3: Authentication

1. Entity (**prover**) starts the process by **sending its ID** to the verifier
2. Verifier looks up the **challenge/response vectors** which corresponds to the received (ID)
3. **Verifier sends a challenge (C) to Entity**
4. Entity applies the received challenge (C) to its PUF and **sends back the generated response** (R')
5. Verifier **compares the received response** (**R'**) with that stored in its data base (**R**), if they are equal then the entity is authenticated, otherwise the authentication request is denied
6. Verifier **deletes the challenge/ response pair used** in the above process to prevent replay attacks



10

A Fuzzy Identification?

Plus, you also have transmissions over a noisy channel
If # of errors is limited, I can assume where was an error, so I assume to work with fuzzy ID.

- PUF responses are **not perfectly reproducible** due to temporal noise
- It is **hard** to construct a PUF-based authentication scheme that requires **100% recovery** of originally recorded responses
- "***Fuzzy***" identification approach can be used to **reduce the potential costs of error correction circuitry**: the verifying authority accepts responses as long as their **Hamming distance** from the original response is **less than a pre-defined threshold**

We want to avoid a false rejection: rejecting a correct auth. Of course I don't want to increase threshold so much to increase false acceptance rate.

11 EX: 128 bits of response. 50 errors is not OK! 3, 4, 5 ... 10 errors are acceptable. Minimize false rejection without increasing false acceptance too much. This threshold depends on application. Sometimes I might want 100% matching.

False Rejection/Acceptance Rates

- In order to choose the appropriate identification threshold, one needs to evaluate the following metrics:
  - **False Acceptance Rate** (FAR): which refers to the probability that a PUF of a physical entity generating a **response identical to that of a PUF implemented on another entity**. This leads to **false identification**
  - **False Rejection Rate** (FRR) is the probability of a **genuine physical entity** generating an **invalid response** due to temporal noise.
  - **Equal Error Rate (EER)** is defined as the maximum of FAR and FRR for a certain identification threshold value

maximum between the two,

$$\mathbf{EER} = \boldsymbol{max}\{\boldsymbol{FAR(t)}, \boldsymbol{FRR(t)}\}$$

12 I could choose to minimise FAR or work directly on EER.

False Rejection/Acceptance Rates

- FAR and FRR are functions of the **identification threshold t**, which is the **max number of bit error** in the PUF response that can be tolerated, i.e., do not cause a false rejection
- To minimise $FRR \rightarrow t$ **needs to be large enough** so the bit flips caused by temporal noise do not lead to rejecting a genuine response
- On the other hand, increasing t can aggravate FAR as it increases the probability of false identification
- In practice, t is chosen to balance FRR and FAR

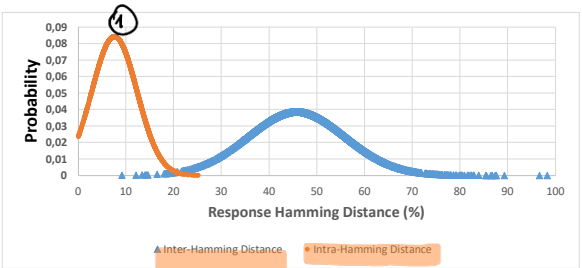


Figure : Intra and Inter Hamming Distance Distributions of a 32-Bit Arbiter PUF

① Same PUF in different working environment.

13 You could have superposition of the two HD you get, in diff conditions
You get problems when distance produced by one chip superposes to distance produced by multiple chips.

FALSE ACCEPTANCE: one entity that doesn't have rights to a service answers on behalf of another. A false acceptance happens when I get authenticated because of a threshold.

False Rejection/Acceptance Rates

Example: A single-challenge/response PUF is used for the entity-authentication of a 4-device network, wherein the unique response of the PUF is used as the device's identifier

The responses of the PUF from each device are generated under four different environmental conditions (i.e. power supply fluctuations and ambiance temperature).

The nominal conditions at which the PUF was enrolled at is ($T = 25^0\text{C}, V_{dd} = 1\ V$) It is assumed that all the listed environment conditions are equally probable

1. Compute the false rejection and admission rates assuming the verifier cannot tolerate any errors i.e. the threshold t=0
2. Repeat the same computations above but with a threshold t=1
3. Which of the threshold values produce the minimum equal error rate (EER)?

14

False Rejection/Acceptance Rates

Environment Conditions \ Device	1	2	3	4
$T = 25^0\text{C}, V_{dd} = 1V$	00000000	00000111	00111000	11111111
$T = 75^0\text{C}, V_{dd} = 1V$	00000000	00000111	00110000	11111000
$T = 25^0\text{C}, V_{dd} = 1.2V$	00000001	10000111	00100000	01111000
$T = 75^0\text{C}, V_{dd} = 1.2V$ ①	00000001	11000111	00000000	00111000

nominal conditions

Same challenge, they answer like this.

Nominal conditions. Responses very close operating conditions

① Power supply noise of 20% (high!)

15

t=0, I expect the response I get in nominal conditions

False Rejection/Acceptance Rates

1) t=0, ① $FRR = \frac{8}{16} = 0.5$ $FAR = \frac{2}{16} = 0.125$ $\boldsymbol{ERR = \frac{8}{16} = 0.5}$

Environment Conditions \ device	1	2	3	4
$T = 25^0C, V_{dd} = 1V$	00000000	00000111	00111000	11111111
$T = 75^0C, V_{dd} = 1V$	00000000	00000111	00110000	11111000
$T = 25^0C, V_{dd} = 1.2V$	00000001	10000111	00100000	01111000
$T = 75^0C, V_{dd} = 1.2V$	00000001	11000111	00000000	00111000

16

For device 1, I have two events of false rejection. Same for device 2, 3, 4.
For 3 and 4 I have a false acceptance too! Mistakenly taken for device 1 and 3.
8 FR, 2 FA. ①

False Rejection/Acceptance Rates

2) t=1, ① $FRR = \frac{2}{16} = 0.125$ $FAR = \frac{4}{16} = 0.25$ $\boldsymbol{ERR = \frac{4}{16} = 0.25}$

Environment Conditions \ Device	1	2	3	4
$T = 25^0C, V_{dd} = 1V$	00000000	00000111	00111000	11111111
$T = 75^0C, V_{dd} = 1V$	00000000	00000111	00110000	11111000
$T = 25^0C, V_{dd} = 1.2V$	00000001	10000111	00100000	01111000
$T = 75^0C, V_{dd} = 1.2V$	00000001	11000111	00000000	00111000

17 I accept patterns that differ 1bit from expected ones. Device 1, all acceptable config.
Device 2 has 3 acceptable configs, last one differs on 2 bits, so deny, 1 FR.
Device 3 has 2 acceptable configs (only 1 difference in bit). But the other 2 differ of 1 bit for device 1 response. So mistakenly accepted as device 1. Same for 4. ①

If I give same weight to FRR and FAR, then F=1 is good. But I increase my FAR. So it depends on what we want.

Brief outline

- Design and evaluation principles of PUF
- Quality metrics for PUF evaluation
- PUFs generating/storing cryptographic keys
- PUFs for entity authentication protocols
- **PUFs for hardware metering**

Examples of possible applications

20

PUF-Based Hardware Metering

- PUF technology can be employed to limit **overproduction** of integrated circuits by malicious factory, which is causing significant financial losses to design houses every year.
  - **Overproduction**: example of counterfeiting consisting in the unauthorised chip production (the untrusted foundry produced more chips than what it should)

    → hardware metering technique to monitor that all authorised chips are produced:
    - PUF can be used in **anticounterfeiting** techniques against overproduction

21

Last application is related to HW metering.

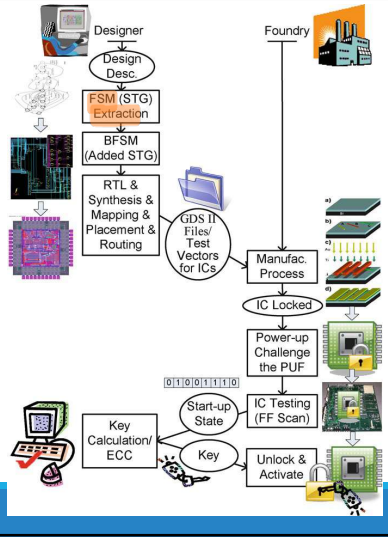
PUF can be used to limit overproduction. I usually let an external fab to produce my chip by giving GDSI file that contains fabrication info.

Why can untrusted manufacturer overproduce? Bro could sell more for itself. And bro would flood market with counterfeited devices and create damage to designer.

1. You need to trust manufacturers. This is the base
2. You can use PUF to control what manufacturer is doing. Designer can activate only a limited number of devices we actually are supposed to get manufactured. ↓

Design-for-Anti-Counterfeit: Overproduction

1. The design house uses the high-level design description to identify the best places to insert a lock
2. The design house embeds a PUF into the design to initialise the systems into a locked state
3. The design then passes subsequent design phases – synthesis, place & route
4. The foundry receives the blueprint of the chip in the form of GDSII files to fabricate the ICs; it also receives a challenge for the PUF
5. Once the chips are fabricated, the foundry will read out the response of the PUF to the designer challenge from each device and send this back to the design house
6. The design house then sends the unlock key to the foundry to unlock the IC
7. The design house keeps track of the number of activated ICs, which helps to prevent overproduction



F. Koushanfar. "Hardware metering: A survey." *Introduction to Hardware Security and Trust*. Springer New York, 2012. 103-122.

22

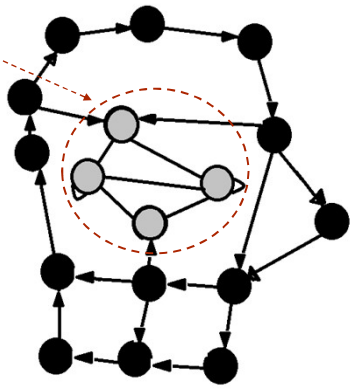
① You exit obfuscated state space only after reading PUF. You use that to unlock device with that. Producer has control over # of unlocked devices. The locked devices are not in a functional state space, so they won't work. And if a locked chip is sold, you can realise that it is locked

Design-for-Anti-Counterfeit : Oveproduction

Example: Sequential Circuits:

1. The design house adds **non-functional states** to the original state machines

If you don't add correct sequence you stay in the obfuscated states.



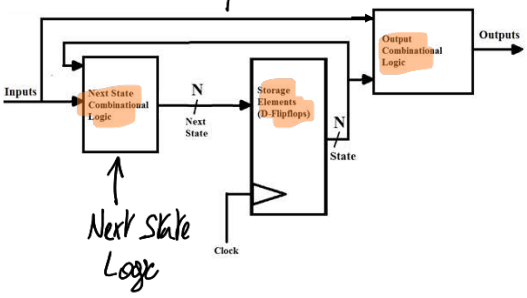
23

You work with high level design and
In FSM stage, I add PUF and get a Boosted FSM. PUF is usually combined with a set of obfuscated states to make attacker's life more complex.

Design-for-Anti-Counterfeit : Oveproduction

Example: Sequential Circuits:

Original circuit modified like :

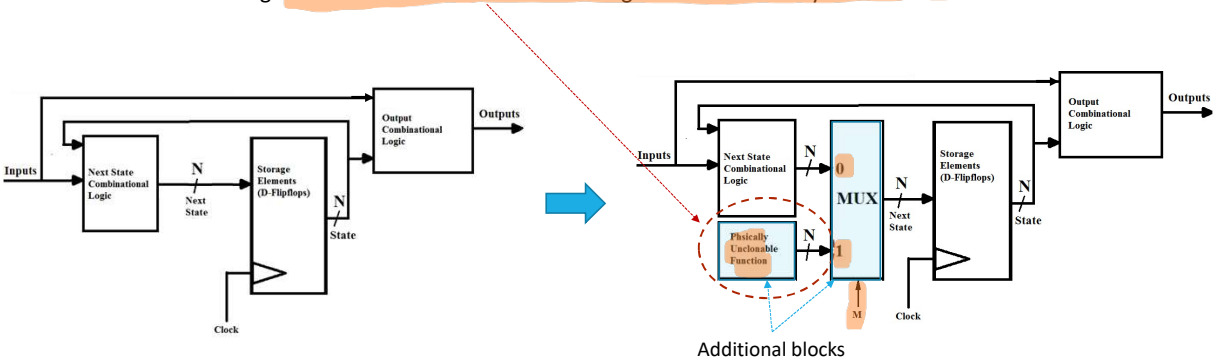


24

Design-for-Anti-Counterfeit : Oveproduction

Example: Sequential Circuits:

2. The design house embeds a **PUF** into the design to initialise the systems into a locked state



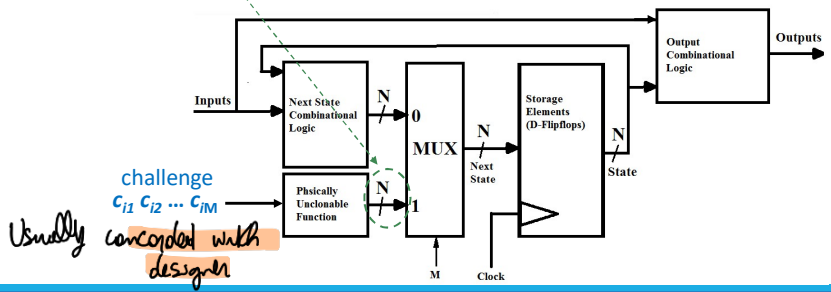
25

Multiplexer allows me to differentiate between next state generated by NSCL and PUF.

Design-for-Anti-Counterfeit

Example: Sequential Circuits:

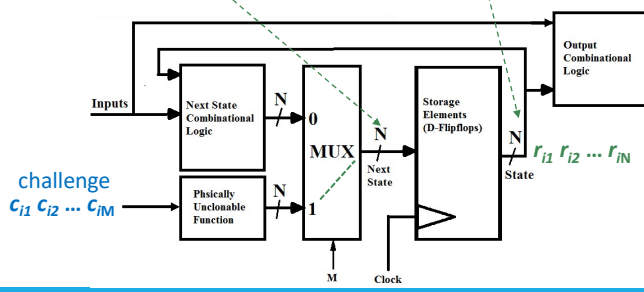
3. Once the chips are fabricated, the foundry will read out the **N-bit response of the PUF** $r_{i1}\ r_{i2}\ \dots\ r_{iN}$ from each device *i* and send this back to the design house.
4. The design house, compute a **key** to unlock each chip based on its PUF response, and sends it back to the manufacturer, to allow chip testing.



Design-for-Anti-Counterfeit

Example: Sequential Circuits:

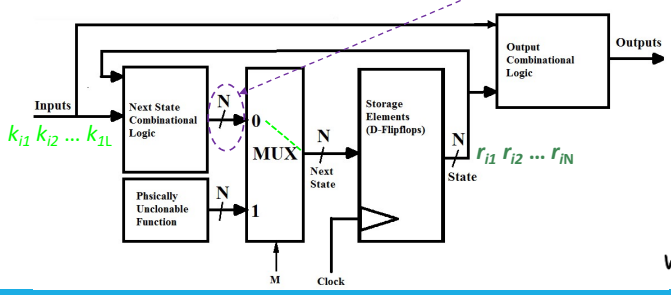
5. For each chip the manufacturer **power up the device**, apply the same challenge to the PUF, set (**M=1**) the PUF response $r_{i1}\ r_{i2}\ \dots\ r_{iN}$ is propagated to the MUX output. The, the manufacturer clock the design once → the design is set in the **PUF-generated initial state** $r_{i1}\ r_{i2}\ \dots\ r_{iN}$.



Design-for-Anti-Counterfeit

Example: Sequential Circuits:

6. The manufacturer then set (M=0) and apply the key to the primary input → the next state is computed as a function of present state $r_{i1}\ r_{i2} \dots r_{iN}$ and primary input $k_{i1}\ k_{i2} \dots k_{1L}$ and then propagated to the MUX output



- Next state is function of PUF state and key input
That's why you need PUF response, next state will be functional

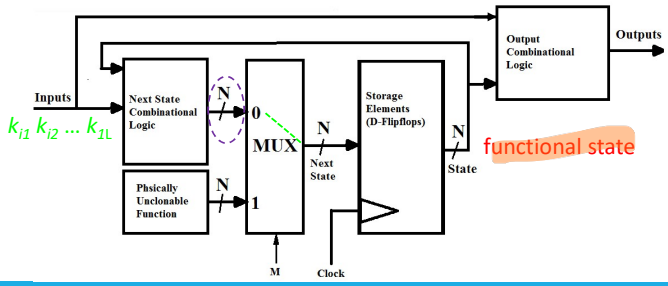
28

Key, given PUF, will be one such that next state will be functional

Design-for-Anti-Counterfeit

Example: Sequential Circuits:

6. The manufacturer then set (M=0) and apply the key to the primary input → the **next state** is computed as a function of present state $r_{i1}\ r_{i2} \dots r_{iN}$ and primary input $k_{i1}\ k_{i2} \dots k_{1L}$ and then propagated to the MUX output
7. The **design is clocked** → This should drive the design into one of the **functional states**.
8. The **chip is now ready for testing**.



Only designer could evaluate the right key to unlock.

29

# of produced keys = # of produced chips.

At runoff, Either device can reload starting state (a functional one), or device should be associated to key challenge response + key. You can just save the functional state in a NVM.

Use of PUF in actual products

Some examples:

- https://www.electronicspecifier.com/products/fpgas/puf-key-storage-tightens-fpga-security
- https://www.ememory.com.tw/en-US/Products/Product?guid=20092115301460
- https://www.intrinsic-id.com/solutions/

33

Any questions?

34

- Possible attacks on silicon PUFs.

We can have a few of them. Last we'll see does it imply these attacks are easy. Practical performing might be hard.

First one: Duplication

1. At least theoretically, attacker could try to fabricate a duplicate of a PUF hoping that eventually we are able to duplicate PUF of interest. Theoretically this would be possible, but in practice, unless PUF is very simple it's hard. Attacker would need to fabricate a massive amount of PUF to get one right.

Note that the main principle behind PUFs we discussed is delay, and reproducing delay behavior is very hard.

2. Model building based on direct measurement: assumption: attacker has unrestricted access to the IC containing the PUF. By probing the circuit the attacker can gain info on the PUF and create a model of the IC. But doing so, opening up the IC, removing several layers would have impact on delay, likely destroying circuit. Not effective.

3. ML using adaptive-chosen chls: idea is build a model of PUF by measuring response of PUF to a subset of the possible challenges that allows you to define a reasonable model. It is not easy to create consistent timing models of PUF just by monitoring subset of responses. Plus, the timing characteristics on which we build PUFs are based on very small delay differences across paths, and to characterize those small differences is hard. But across the three this is the most likely attack to be possible.

- How to defend ourselves? Make this characterization less straightforward ↓

Possible attacks on Silicon PUFs

- There are many possible attacks on PUFs: **duplication**, **model building** using **direct measurement**, and **model building** using **adaptively-chosen challenge** generation

- **Duplication**: The adversary can attempt to duplicate a PUF by fabricating a counterfeit IC containing the PUF
  - Due to statistical variation, unless the PUF is very simple, the adversary will have to fabricate a huge number of IC's and precisely characterize each one → very difficult, and non-cost-effective

35

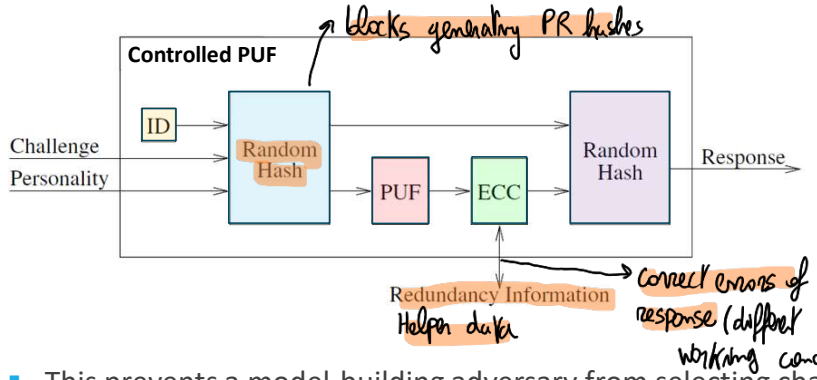
Possible attacks on Silicon PUFs

- **Model building** using **direct measurement**: The adversary has **unrestricted access to the IC** containing the PUF, and so can attempt to create a **model** of the IC by **measuring very precisely the delay** of each device and wire in the IC
  - Direct measurement of device delays requires the adversary to **open the package of the IC**, and **remove several layers**, such as field oxide and metal → this may have a **large impact on delay**, basically **destroying the PUF** → not effective

- **Model building** using **adaptively-chosen challenges**: The adversary could try to **build a model of the PUF** by measuring the **response** of the PUF to a polynomial number of **adaptively-chosen challenges**
  - There is a significant barrier → **creating timing models** of a circuit **accurate** to within measurement error is a **very difficult problem**

36

Attack-resilient PUF: Controlled PUF



- **Controlled PUFs**:

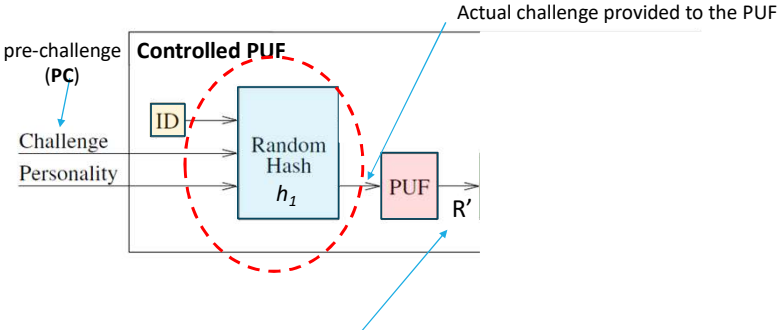
  A random hash function is placed before the PUF → it prevents the adversary from performing a chosen challenge attack on the PUF

- This prevents a model-building adversary from selecting challenges that allow them to extract parameters more easily

B. Gassend, et al., "Controlled physical random functions", Proceedings of the 18th Annual Computer Security Applications Conference, 2002

37

Controlled PUF



Preventing chosen challenge attacks

- For some CRPs, it could be easier for an attacker to **model the behavior** of the PUF (come up with a series of equations that can be solved)

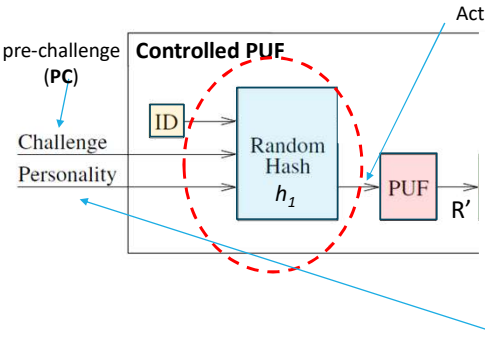
Possible solution: pre-compose the PUF with a (pseudo)random hash function h_1

- Standard PUF: R' = f(PC)
- CPUF: **R' = $f(h_1$(PC))**

38

Input random hash block: it helps us because model attack analyzes correspondence between challenges and responses. If we add an hash, the chl is not directly provided to the PUF. So for a given chl we don't know what will be provided to the PUF.
The external chl is now a pre-challenge (not directly feeded to PUF).
So we are using a function of chl. And we also have other inputs to the hash function. ID can be an external side of IC.
But then we have a personality parameter: a PUF provides a unique side.
There have been users complaining about processors with unique IDE. So this personality allows the user to change and customise behaviour of PUF in many different ways depending on the application. So you can modify the actual challenge if you want to obfuscate PUF behaviour.
We then have ECC. If we have an erroneous response, we can correct it.
Then an output hash function to provide additional randomness in your PUF response.

Controlled PUF



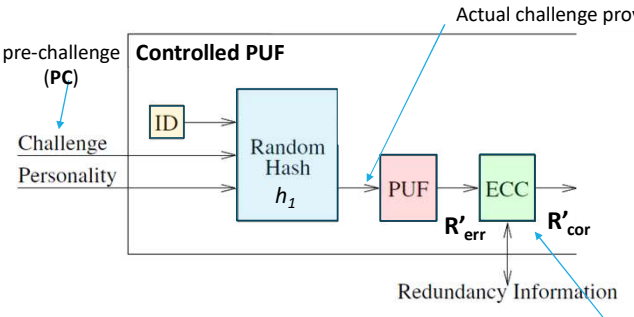
Personality

- Users have often shown to be uncomfortable with processors that have unique identifiers, because they feel they can be tracked
  → they could have the same type of concern with the use of PUFs, since PUFs are a form of **unique identifier**

Possible solution: use of multiple **personalities**. By using this additional parameter, the PUF can show different facets to different applications → challenge is hashed with a user-selected personality number → many "effectively different" PUFs available to the user

39

Controlled PUF

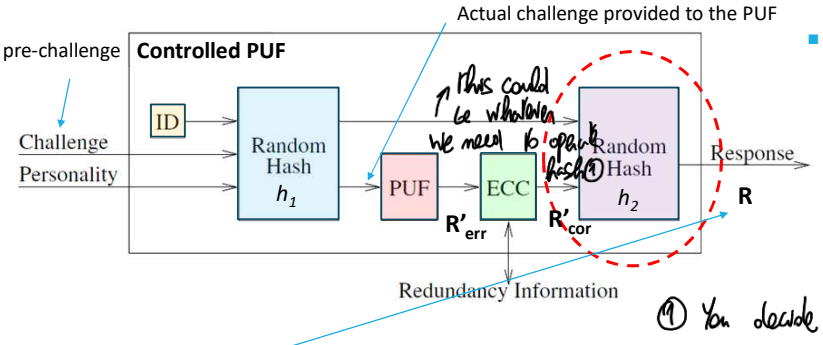


- **Error Correction**
  - Slight variations from one run to the next will cause slight changes in the response of the PUF: R' → R'_{err}

Possible solution: An Error Correcting Code (ECC) can be used to takle noisy physical measurements and turn them into consistent responses → when a challenge-response (PC-R') pair is created, some redundant information is also produced that should allow possible errors to be corrected for

40

Controlled PUF



- **Post-Composition with a random function**
  - PUF output should exhibit as much randomness as possible to prevent an adversary from guessing the response to one challenge by using the response to another challenge

Possible solution: Post-composing the PUF with a random function h_2

- $R = h_2(PC, f(h_1(PC)))$: (pseudo)random **hash function's avalanche-effect** ensures that nearby outputs of f will lead to completely different outputs of the composite function

41

Design more complex and expensive of course.

Thank you!

Any questions?

42