

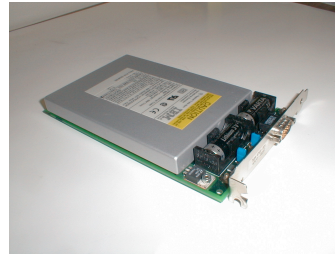
# Physical Unclonable Functions and Applications

Srini Devadas

Contributors: Dwaine Clarke, Blaise Gassend, Daihyun Lim, Jaewook Lee, Marten van Dijk

## Problem:

Storing **digital** information in a device in a way that is resistant to **physical attack** is difficult and expensive.



IBM 4758

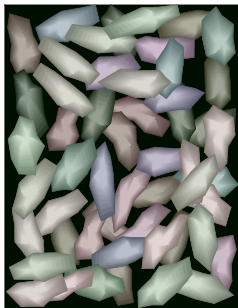
Tamper-proof package containing a secure processor which has a secret key and memory  
Tens of sensors, resistance, temperature, voltage, etc.

Continually battery-powered

~ \$3000 for a 99 MHz processor and 128MB of memory

## Our Solution:

Extract key information from a complex physical system.



## Definition

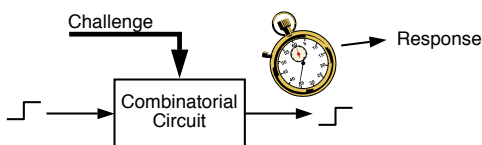


A Physical Random Function or **Physical Unclonable Function (PUF)** is a function that is:

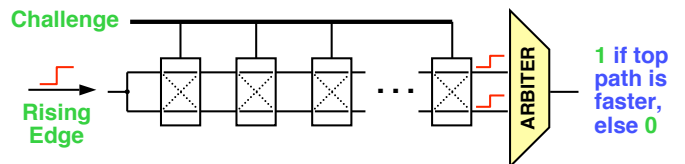
- Based on a physical system
- Easy to evaluate (using the physical system)
- Its output looks like a random function
- Unpredictable even for an attacker with physical access

## Silicon PUF – Proof of Concept

- Because of process variations, **no two Integrated Circuits are identical**
- Experiments in which **identical circuits with identical layouts** were placed on different FPGAs show that path delays vary enough across ICs to use them for identification.



## A Candidate Silicon PUF

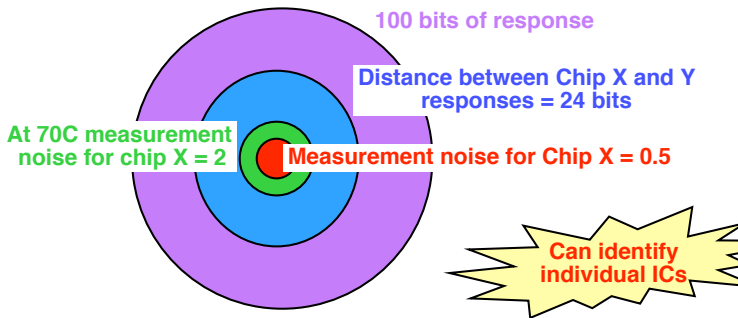


Each challenge creates two paths through the circuit that are excited simultaneously. The digital response is based on a **(timing) comparison of the path delays**.

Path delays in an IC are **statistically distributed** due to random manufacturing variations.

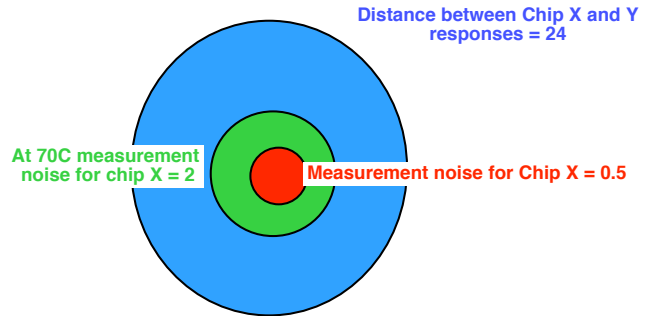
## Experiments

- Fabricated candidate PUF on multiple IC's, 0.18 $\mu$  TSMC
- Apply 100 random challenges and observe response



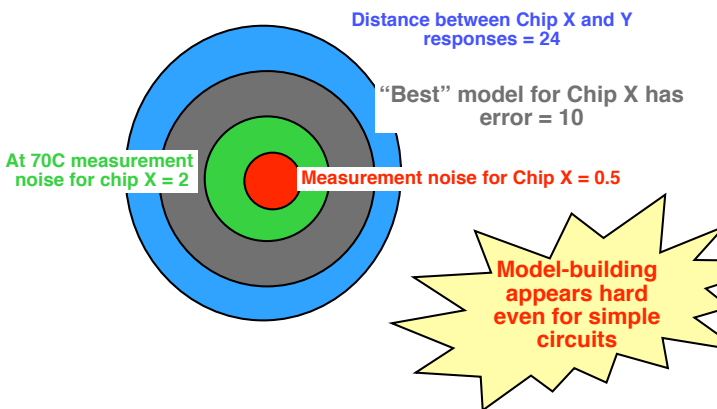
## Measurement Attacks and Software Attacks

Can an adversary create a *software clone* of a given PUF chip?



## Measurement Attacks and Software Attacks

Can an adversary create a *software clone* of a given PUF chip?



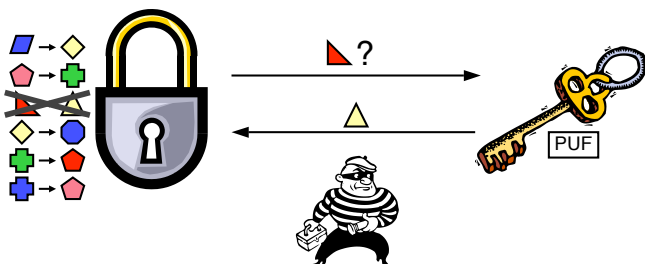
## Physical Attacks

- Make PUF delays depend on overlaid metal layers and package
- Invasive attack (e.g., package removal) changes PUF delays and destroys PUF
- Non-invasive attacks are still possible
  - To find wire delays need to find precise relative timing of transient signals as opposed to looking for 0's and 1's
  - Wire delay is not a number but a function of challenge bits and adjacent wire voltages

## Using a PUF as an Unclonable Key

A Silicon PUF can be used as an unclonable key.

- The lock has a database of challenge-response pairs.
- To open the lock, the key has to show that it knows the response to one or more challenges.

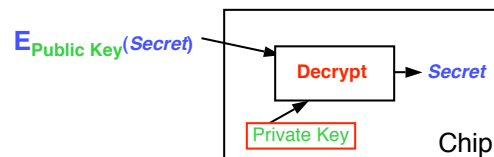


## Private/Public Keys

If a *remote chip* stores a private key, Alice can *share a secret* with the chip since she knows the public key corresponding to the stored private key

Encrypt *Secret* using chip's *public key*

Only the chip can decrypt *Secret* using the stored *private key*



## Applications



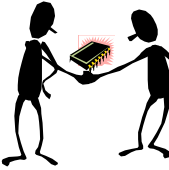
- **Anonymous Computation**

Alice wants to run computations on Bob's computer, and wants to make sure that she is getting correct results. A certificate is returned with her results to show that they were correctly executed.



- **Software Licensing**

Alice wants to sell Bob a program which will only run on Bob's chip (identified by a PUF). The program is copy-protected so it will not run on any other chip.



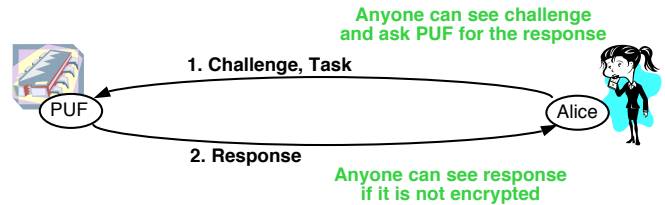
How can we enable the above applications by trusting only a single-chip processor that contains a silicon PUF?

## Sharing a Secret with a Silicon PUF

Suppose Alice wishes to share a secret with the silicon PUF

She has a challenge response pair that no one else knows, which can authenticate the PUF

She asks the PUF for the response to a challenge

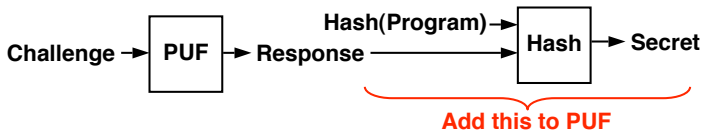


## Restricting Access to the PUF

- To prevent the attack, the man in the middle must be **prevented** from finding out the response.
- Alice's program must be able to establish a shared secret with the PUF, the attacker's program must not be able to get the secret.

⇒ **Combine response with hash of program.**

- The PUF can only be accessed via the **GetSecret** function:



## Getting a Challenge-Response Pair

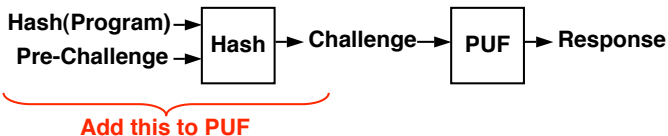
- Now Alice **can** use a Challenge-Response pair to generate a shared **secret** with the PUF equipped device.
- But Alice **can't** get a Challenge-Response pair in the first place since the PUF **never** releases responses directly.

⇒ **An extra function that can return responses is needed.**

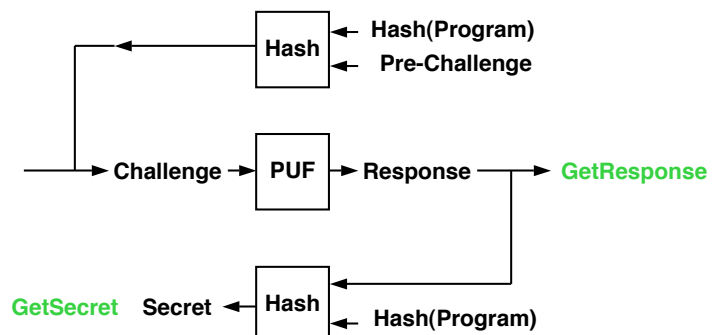
## Getting a Challenge-Response Pair - 2

- Let Alice use a **Pre-Challenge**.
- Use **program hash** to prevent eavesdroppers from using the pre-challenge.

- The PUF has a **GetResponse** function



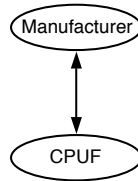
## Controlled PUF Implementation



## Challenge-Response Pair Management: Bootstrapping

When a CPUF has just been produced, the manufacturer wants to generate a challenge-response pair.

1. Manufacturer provides **Pre-challenge** and **Program**.
2. CPUF produces **Response**.
3. Manufacturer gets **Challenge** by computing  $\text{Hash}(\text{Hash}(\text{Program}), \text{PreChallenge})$ .
4. Manufacturer has (**Challenge**, **Response**) pair where **Challenge**, **Program**, and  $\text{Hash}(\text{Program})$  are public, but **Response** is not known to anyone since **Pre-challenge** is thrown away



## Software Licensing

Program (Ecode, Challenge)  
 Secret = GetSecret( Challenge )  
 Code = Decrypt( Ecode, Secret )  
 Run Code

} Hash(Program)

Ecode has been encrypted with Secret by Manufacturer

## Software Licensing

Program (Ecode, Challenge)  
 Secret = GetSecret( Challenge )  
 Code = Decrypt( Ecode, Secret )  
 Run Code

} Hash(Program)

Ecode has been encrypted with Secret by Manufacturer  
 Secret is known to the manufacturer because he knows Response to Challenge and can compute  
 Secret = Hash(Hash(Program), Response)

## Software Licensing

Program (Ecode, Challenge)  
 Secret = GetSecret( Challenge )  
 Code = Decrypt( Ecode, Secret )  
 Run Code

} Hash(Program)

Ecode has been encrypted with Secret by Manufacturer  
 Secret is known to the manufacturer because he knows Response to Challenge and can compute  
 Secret = Hash(Hash(Program), Response)

Adversary cannot determine Secret because he does not know Response or Pre-Challenge

If adversary tries a different program, a different secret will be generated because Hash(Program) is different

## Summary

- PUFs provide secret “key” and CPUFs enable sharing a secret with a hardware device
- CPUFs are not susceptible to model-building attack if we assume physical attacks cannot discover the PUF response
  - Control protects PUF by obfuscating response, and PUF protects the control from attacks by “covering up” the control logic
  - Shared secrets are volatile
- Lots of open questions...