Physical(ly) Unclonable Functions

An introduction to Intrinsic PUFs

Ingrid Verbauwhede

Slide courtesy: Roel Maes
COSIC – K.U.Leuven

Supported by:
IWT and unique

Introduction

Goal of this talk → try to answer the question:

- **What is a PUF?**
- **What is it usage?**
- **Can we use it as an unclonable device identifier?**

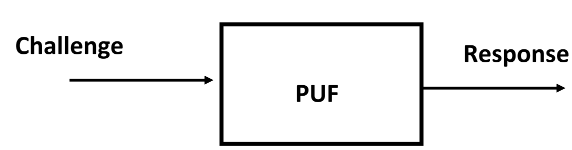
- P.U.F.?
  - P.U.F. → *Physical Unclonable Function*
    a physical function which is unclonable in every sense
  - P.U.F. → *Physical**ly** Unclonable Function*
    a function which is unclonable in a physical sense

Introduction (cont.)

- Need for unique identification of devices or goods
  - Example: RFID tags
  - Secure storage of a key bit string (non volatile memory, battery backed up SRAM, fuses, etc.)
- Problem: personalization step during fabrication
  - Expensive, extra processing steps
  - Post-processing e.g. by blowing fuses
- Idea: use physical uniqueness of devices
- Idea: use CMOS process variations for this
  - Threshold voltage
  - Oxide thickness
  - Metal line shapes

Functional description of PUF

- (how crypto community sees it)
- PUF = (physical) function which is physically unclonable



- Very hard ("impossible") to produce two PUFs with similar challenge-response behavior
- Easy to construct and evaluate a random PUF

Basic properties of PUF

Minimum requirements:

- For two random PUFs, difference between expected responses to same challenge, should be large

  = manufacturing variability is 'large'
- For single random PUF, difference between two measured responses to same challenge, should be small

  = noise, aging, temperature effects,… are limited
- For single random PUF, uncertainty about response to challenge is large, when one does not have access to this PUF instance

  = unpredictability is large

Intrinsic PUFs

- Use inherent manufacturing variability present in CMOS fabrication
- Keep measured PUF responses inside chip

  Useful for secure key storage!

Requirements:

- PUF + measurement circuit + post-processing inside chip
- Standard processing, no extra processing steps

KATHOLIEKE UNIVERSITEIT LEUVEN

Outline

- Introduction
- **A few examples of Intrinsic PUFs**
- PUF properties
- PUF applications
- Conclusion

KATHOLIEKE UNIVERSITEIT LEUVEN

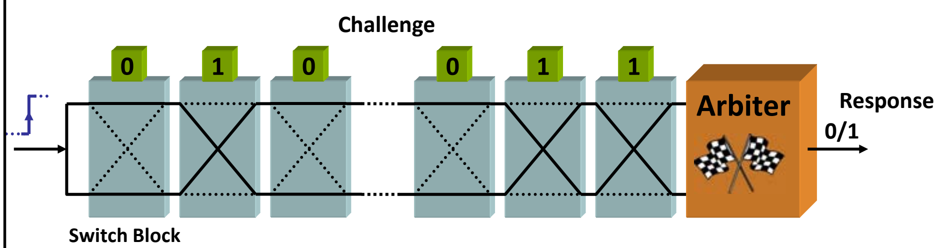
First example: Arbiter PUF

Delay based intrinsic PUF

KATHOLIEKE UNIVERSITEIT LEUVEN

Arbiter PUF: basic operation

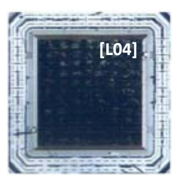
- Initial design [Lee et al, MIT 2004]
  - switch block: e.g. two muxes
  - arbiter: e.g. a latch or a flip-flop
  - *n* switch blocks → 2^n “different” delays

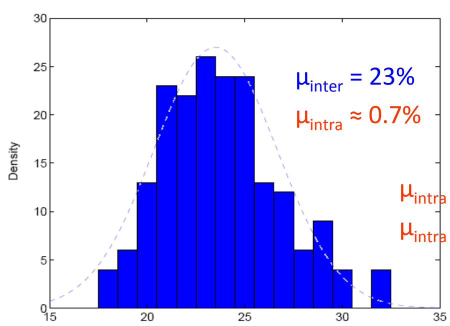


KATHOLIEKE UNIVERSITEIT LEUVEN

Arbiter PUF: experiments [Lee04]

- Results:
  - 10000 CRPs from 37 ASICs
  - 64-stage arbiter PUF





μ_{intra} ≈ 3.47% (voltage variation)

μ_{intra} ≈ 4.82% (temperature variation)

[Lee04]

  - Results on FPGA [Lee at al 04] : μ_{inter} = 1.05%, μ_{intra} = 0.3%

Arbiter PUF: analysis

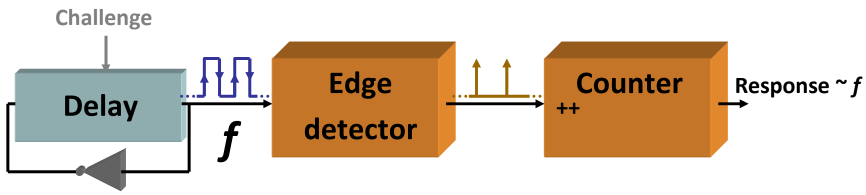
- **delay ≈ additive !**
  - leads to model-building attack (linear programming)
  - also machine-learning techniques (Artificial Neural Networks, Support Vector Machines, …)
- **Attack results:**
  - *ASIC*: 3.55% prediction error with SVM trained with 5000 CRPs (**< μ_{intra} !!!**)
  - *FPGA*: 0.6% prediction error with perceptron trained with 90000 CRPs
- **Extensions [Ozturk et al 2008]**
  - tri-state buffer based delay circuit → comparable to switch-based
  - *Simulation*: prediction error < 3% for linear programming on 4000 CRPs
- **Conclusion**: (Improved) arbiter PUFs can be accurately modelled (= "cloned") from polynomial # known CRPs

Second example: Ring Oscillator PUF

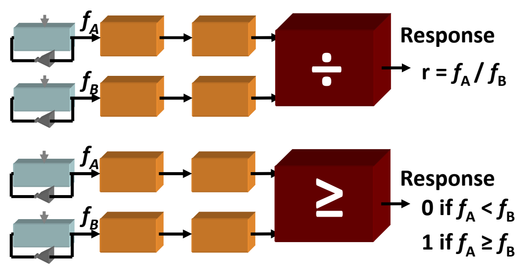
Delay based Intrinsic PUF

Ring Oscillator PUF: basic operation

- Initial design [Gassend et al 2003]



- Compensation
  - To eliminate (scaling) environmental influence
- One-bit compensation [Suh 2007]
  - To avoid costly division

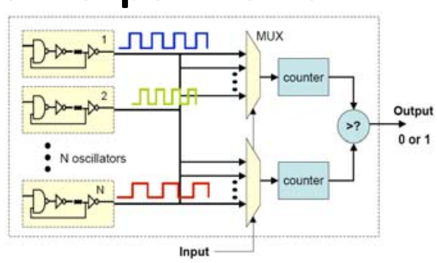


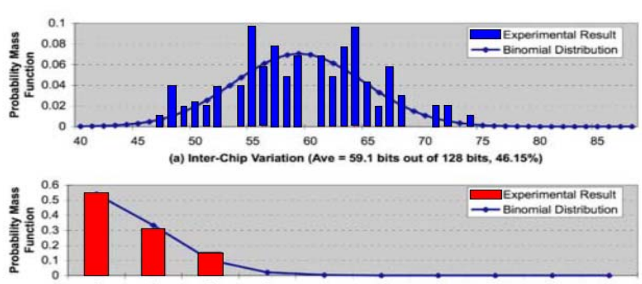
Ring Oscillator PUF: experiments

- Results [Suh2007]:
  - measurements on 15 FPGAs, 1024 loops/FPGA and *1 out of 8 masking*

μ_{inter} = 46.15%

μ_{intra} = 0.48% (with temp./volt. var.)





(a) Inter-Chip Variation (Ave = 59.1 bits out of 128 bits, 46.15%)

(b) Intra-Chip Var. under the Worst-Case Env. Change (20C 1.2V vs. 120C 1.08V) (Ave = 0.61 bits out of 128 bits, 0.48%)

Ring Oscillator PUF: analysis

- Modelling attacks?
  - same as arbiter PUFs for basic design
  - not for fixed delay circuit as in [Suh 2007]

    → but much less challenges!
    [$N/2 < \#\text{challenges} < \log_2(N!)$]

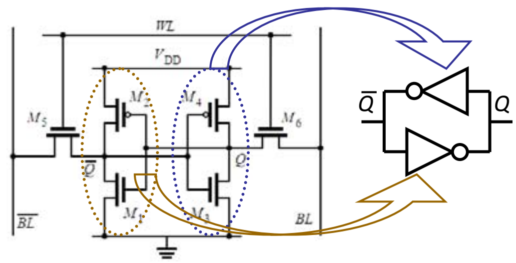
    → inefficient area use

SRAM PUF

Memory based Intrinsic PUF

SRAM PUF: basic operation

- SRAM cell:
  (6T-CMOS)



Q	$\overline{Q}$
0	*1*
1	*0*

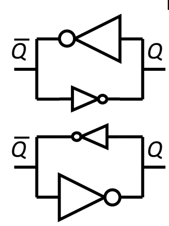
2 possible stable states
→ 1-bit storage

- Which state right after power-up?
  - depends on physical ***mismatch*** between M_2 and M_4
  - power-up state =
    measure for ***manufacturing variability***

SRAM PUF: basic operation

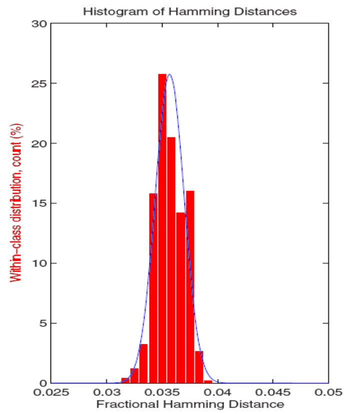
- SRAM PUF
  - **challenge** = SRAM address
  - **response** = power-up state of addressed cell(s)
- Guajardo et al 2007
  - SRAM on FPGA
- Holcomb et al 2007
  - COTS SRAM
  - SRAM on embedded micro-controller

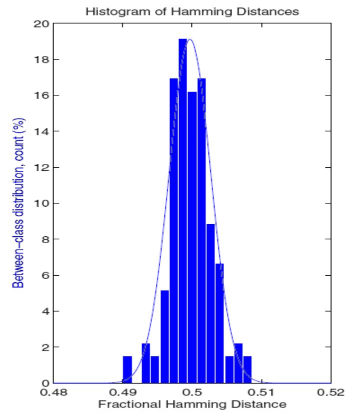
power-up state:



SRAM PUF: experiments

- Results [Guarjado et al CHES 2007]
  - measurements on FPGA, 8190 bytes from different SRAM blocks
  - μ_{intra} = 3.57% , σ_{intra} = 0.13% μ_{inter} = 49.97% , σ_{inter} = 0.3%



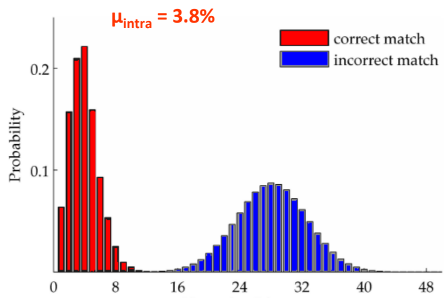


SRAM PUF: experiments

- Results [Holcomb, Burleson, Fu 2009]
  - measurements on two types of devices
  - COTS SRAM chip: 5120 64-bit blocks on 8 ICs
  - Embedded SRAM in µC: 15 64-bit blocks on 3 ICs

μ_{inter} = 43.16%

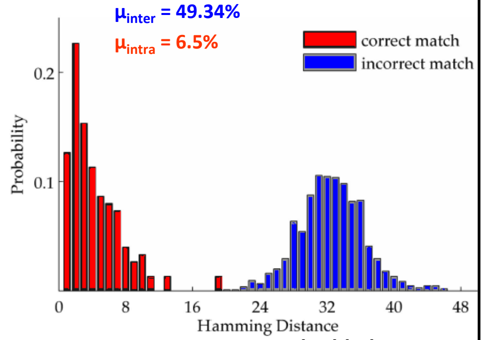
μ_{intra} = 3.8%



SRAM chip

μ_{inter} = 49.34%

μ_{intra} = 6.5%



Embedded SRAM

Outline

- Introduction
- Examples of Intrinsic PUFs
- **PUF properties**
- PUF applications
- Conclusion

Quick overview

- Evaluatable: y = PUF (x) is easy
- Unique: PUF(x) contains some unique information
- Reproducible: PUF() has only small error
- Unclonable: hard to make PUF'(x) given PUF(x)
- Unpredictable: hard to find y_N=PUF(x_N) given other x, y pairs
- One-way: given y and PUF(), cannot find x
- Tamper evident: tampering changes PUF()

Compare PUF constructions

1. **OPT**
   → Optical PUF **[P01][STO06]**
   - *Challenge* = laser orientation
   - *Response* = Gabor hash of speckle pattern
2. **COAT**
   → Coating PUF **[TSSVVW06]**
   - *Challenge* = sensor pair selection
   - *Response* = quantized capacitance measurement
3. **ARB**
   → Basic switch-based arbiter PUF **[L04]**
   - *Challenge* = switch-block delay setting
   - *Response* = one-bit arbiter decision
4. **FF-ARB**
   → Feed-forward arbiter PUF **[L04]**
   - *Challenge* = (reduced) switch-block delay setting
   - *Response* = one-bit arbiter decision
5. **LW-ARB**
   → Lightweight arbiter PUF **[MKP08]**
   - *Challenge* = pre-challenge to input-network
   - *Response* = one-bit XOR of arbiter decisions
6. **RO**
   → Basic switch-based ring oscillator PUF with division compensation **[GCVD02b][B03]**
   - *Challenge* = switch-block delay setting
   - *Response* = ratio of two counter values
7. **1B-RO**
   → Inverter ring oscillator PUF with comparator compensation **[SD07]**
   - *Challenge* = oscillator pair selection
   - *Response* = one-bit comparator output
8. **SRAM/FF**
   → SRAM PUF **[GKST07]**, flip-flop PUF **[MTV08b]**
   - *Challenge* = SRAM cell address
   - *Response* = one-bit power-up state of addressed cell
9. **LATCH/BUTTER**
   → Basic latch PUF **[SHO07]**, butterfly PUF **[KGMST08]**
   - *Challenge* = latch/butterfly cell selection
   - *Response* = one-bit settling state of excited cell
10. **CPUF**
    → Controlled PUF **[GCVD02]**
    - *Challenge* = pre-PUF hash input
    - *Response* = post-PUF hash output
11. **ID**
    → pre-programmed digital identifier
    - *Challenge* = ask identifier
    - *Response* = identifier value
12. **PK**
    → pre-programmed asymmetric private key
    - *Challenge* = random nonce
    - *Response* = signature on nonce
13. **TRNG**
    → true random number generator
    - *Challenge* = ask true random number
    - *Response* = "physically produced" true random number

PUF properties table

	OPT	COAT	ARB	FF-ARB	LW-ARB	RO	1B-RO	SRAM/FF	LATCH/BUTTER	CPUF	ID	PK	TRNG
Eval.	✓ Mechanical procedure	✓ Integrated (extra process)	✓ Integrated	✓ Integrated	✓ Integrated	✓ Integrated	✓ Integrated	✓ Integrated (+ power-up!)	✓ Integrated	✓ Integrated	✓ Very fast	✓ Computation	✓
Unique	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓ If good RNG	✓ If good RNG	✓
Reprod.	✓	✓	✓	✓	✓	✓	✓	✓	✓	⚠ If good ECC	✓ Perfect	✓ Perfect	✗
Phys. Unclon.	✓	✓	✓	✓	✓	✓	✓	✓ ?	✓ ?	⚠ If good PUF	✗ Copy ID	✗ Copy key	✓
Math. Unclon.	✓	✗ Full readout	✗ Modellable	✗ Modellable	✗ Modellable	✗ Modellable	✗ Full readout	✗ Full readout	✗ Full readout	⚠ If good hash	✗ Trivial	✗ Knows private key	✓
One-way	✓ ?	✗ Few challenges	✗ 1-bit response	✗ 1-bit response	✗ 1-bit response	?	✗ 1-bit response	✗ 1-bit response	✗ 1-bit response	⚠ If good hash	✗ Trivial	✗ Knows private key	✓
Unpred.	✓	✓	⚠ If q << 5000	⚠ If q << 50000	⚠ If q << 50000	⚠ If q << 5000	✓	✓	✓	⚠ If good hash	✗ Trivial	✓ If good crypto	⚠ If good TRNG
Tamper Evident	✓	✓	?	?	?	?	?	?	?	?	✗ No protection	✗ No protection	?

Outline

- Introduction
- Examples of Intrinsic PUFs
- PUF properties
- **PUF applications**
- Conclusion

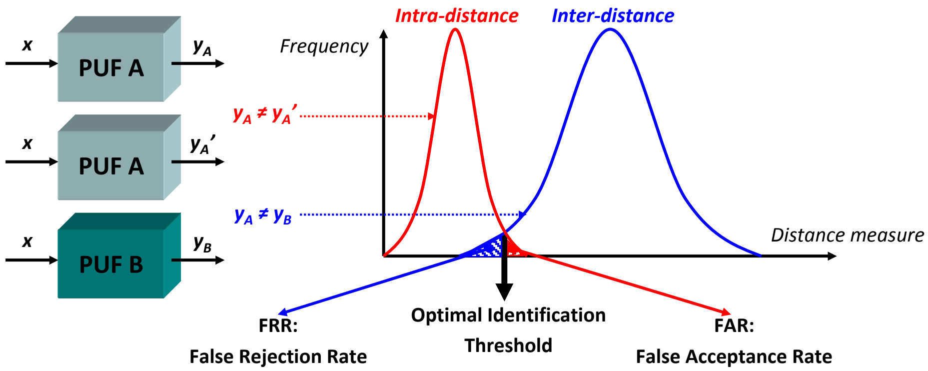
PUF Applications

- System identification
  - anti-counterfeiting
  - hardware binding
  - hardware metering (passive)
- Secret Key Generation
  - secure key storage
  - secure key distribution

  → **key-based crypto → …**
- Hardware Entangled Cryptography
  - side-channel resistance → provable physical security?

System Identification

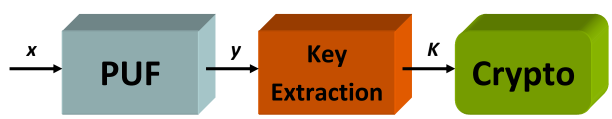
- Very similar to biometrical identification systems



- Easy, fast, straightforward, but limited security
  - e.g. no authentication

Key Generation

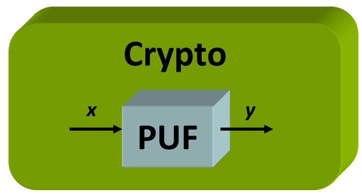
- Integration of PUFs with *keyed* crypto primitives



- Key extraction
- "non-volatile" key storage in a ***"non-digital"*** way
  - key only digitally present when needed in crypto computation
    → no *long-term* digital storage required
- unique key material ***"intrinsically"*** present
  - no key programming step required
    → however, two-phase key generation: enrollment - reproduction

Hardware Entangled Cryptography

- Embedded integration of PUFs in crypto primitives



- Secret parameter = device-unique PUF behavior
  - no digital key present at any point
    → no digital key-storage required whatsoever
  - Basic complexity/cryptographic assumptions about PUFs are needed
- No digital key → constrains application scenarios!
- Topic of active research

Conclusion

- **“What is a PUF?”** is not an easy question
- We identified some *minimal requirements* which
  - …are common to all known PUF constructions (exhaustively)
  - …, but distinguish them from other primitives
- **“Which is the best PUF?”** is an even harder question
  - there are contradicting goals → trade-offs
  - application dependent
- Still many open questions…

For an extended reference list, see website Roel Maes:

http://rmaes.ulyssis.be/pufbib.php