Physical Unclonable Functions (PUFs) and Secure Processors

Srini Devadas
Department of EECS and CSAIL
Massachusetts Institute of Technology
Security Challenges

• How to securely authenticate devices at low cost?
  – Keycards, RFIDs, mobile phones
  – Genuine electronics vs. counterfeits

• How to protect sensitive IP on devices that may be physically attacked?
  – Digital content, personal information
  – Software on mobile/embedded systems, routers, etc
Traditional Solution: Authentication Example

- Each IC needs to be **unique**
  - Embed a unique secret key SK in on-chip non-volatile memory

- Use cryptography to authenticate an IC
  - A verifier sends a randomly chosen number
  - An IC signs the number using its secret key so that the verifier can ensure that the IC possesses the secret key

- Cryptographic operations can address other problems such as protecting IP or secure communication

IC with a secret key

Sends a random number

Sign the number with a secret key

→ Only the IC’s key can generate a valid signature
BUT...

• How to generate and store secret keys on ICs in a secure and inexpensive way?
  – Adversaries may physically extract secret keys from non-volatile memory
  – Trusted party must embed and test secret keys in a secure location

• What if cryptography is NOT available?
  – Extremely resource (power) constrained systems such as passive RFIDs
  – Commodity ICs such as FPGAs
Physical Unclonable Functions (PUFs)

- Extract secrets from a complex physical system
- Because of random process variations, no two Integrated Circuits even with the same layouts are identical
  - Variation is inherent in fabrication process
  - Hard to remove or predict
  - Relative variation increases as the fabrication process advances
- Delay-Based Silicon PUF concept (2002)
  - Generate secret keys from unique delay characteristics of each processor chip

![PUF Diagram]

Challenge $n$-bits \rightarrow Combinatorial Circuit \rightarrow Response $m$-bits
Why PUFs?

- **PUF can enable secure, low-cost authentication w/o crypto**
  - Use PUF as a function: challenge $\rightarrow$ response
  - Only an authentic IC can produce a correct response for a challenge
  - Inexpensive: no special fabrication technique

- **PUF can generate a unique secret key / ID**
  - Highly secure: volatile secrets, no need for trusted programming
  - Can integrate key generation into a secure processor

- **Physical security**: PUF secrets are the delays of wires and gates which are harder to extract via microscopy than bits in non-volatile memory
Main Questions

• How to design a PUF circuit for reliability and security?
  – Analog or asynchronous systems are susceptible to noise
  – Need barriers against software modeling attacks (equivalent to cryptanalysis)

• How to use the PUF for authentication and key generation?
Authentication Using PUFs
Low-Cost Authentication

- Protect against IC/FPGA substitution and counterfeits without using cryptographic operations

<table>
<thead>
<tr>
<th>Challenge</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001010</td>
<td>010101</td>
</tr>
<tr>
<td>1011000</td>
<td>101101</td>
</tr>
<tr>
<td>0111001</td>
<td>000110</td>
</tr>
</tbody>
</table>

Database for Device A
Challenge-Response Pairs

• What if an attacker obtains all responses and put them into a fake chip with memory?

• There must be **LOTS** of challenge-response-pairs
  – Use different parts on FPGAs
  – Use configurable delay paths on ASICs
An Arbiter-Based Silicon PUF

- Compare two paths with an **identical delay** in design
  - Random process variation determines which path is faster
  - An arbiter outputs 1-bit digital response

- Multiple bits can be obtained by either duplicate the circuit or use different challenges
  - Each challenge selects a unique pair of delay paths
Metrics

• **Security**: Show that different PUFs (ICs) generate different bits
  – *Inter-chip* variation: how many PUF bits (in %) are different between PUF A and PUF B?
  – Ideally, inter-chip variation should be close to 50%

• **Reliability**: Show that a given PUF (IC) can re-generate the same bits consistently
  – *Intra-chip* variation: how many bits flip when re-generated again from a single PUF
  – Environments (voltage, temperature, etc.) can change
  – Ideally, intra-chip variation should be 0%
Arbiter PUF Experiments: 64 and 512 stages

PUF Response: Average Code Distances
128 (2x64) bit, RFID MUX PUF Rev.Ax1 M3 vs. Rev.Ax8 M3 @ +25°C

64 stage

512 stage
Arbiter PUF is not a PUF (clonable!)

• Introduced in 2003 paper, shown in same paper to be susceptible to a machine learning model-building attack

Need to add nonlinearity to circuit
Feed-forward Arbiter

- Also introduced in 2003 paper, conjectured to be hard to learn

- Shown in 2008 (Koushanfar) and 2009 (Ruhrmair) to be susceptible to a model-building attack based on evolutionary algorithm
XOR Arbiter PUF

• Can process and combine outputs of multiple PUFs

• Simplest version: XOR operation
XOR Arbiter PUF Security

• Machine learning complexity appears to grow as $O(n^k)$ for k-way XOR over n-stage PUFs
  – Size of circuit grows as $O(nk)$

• $N = 64$, $k = 4$ is on the edge of being broken

• Can go up to $k = 8$ with reasonable noise levels

• As shown earlier, increasing $n$ decreases noise and allows for larger $k$
4-way XOR Experiments

PUF Response: Average Code Distances
128 (2x64) bit, RFID MUX PUF Rev.A M3 vs. Rev.B COC @-25, 0, +25, +50, +85°C combined

- Intra-chip @ Rev.Ax1
- Inter-chip @ Rev.Ax1
- Intra-chip @ Rev.B
- Inter-chip @ Rev.B

4-way XOR

no XOR
8-way XOR experiments

PUF Response: Average Code Distances
128 (2x64) bit, RFID MUX PUF Rev.B vs. (synthesized) Rev.Bx2XOR @ +25°C

8-way XOR

4-way XOR
PUF as Key Generators
Using a PUF as a Key Generator

• Are only going to generate a **fixed** number of bits from a PUF

• Cannot afford **any** errors!

• **Key question:** How to correct errors guaranteeing limited leakage of information?
  – Need to quantify entropy of PUF
  – Need to analyze/quantify leakage due to redundant bits; these can be syndrome or mask bits
Ring Oscillator

- Ring oscillators are widely used in ICs to generate clocks or characterize performance.
- Each ring oscillator has a unique frequency even if many oscillators are fabricated from the same mask.
“PUF” Key Generator Using Ring Oscillators

Compare frequencies of two oscillators → The faster oscillator is randomly determined by manufacturing variations
Implementation Constraints

- All ring oscillators must be identical
  - Any ring oscillator design will work

- No additional constraints required
  - Everything is standard digital logic
  - No placement/routing constraint outside oscillators
  - Can be implemented even on standard FPGAs

![Diagram showing identical layout and no placement/routing constraint](image)
Key Generation: Initialization

Before First Use: Initialization

- To initialize the circuit, an error correcting syndrome is generated from the reference PUF circuit output
  - Syndrome/error mask is public information
  - Can be stored on-chip, off-chip, or on a remote server

- For example, BCH(127,36,31) code will correct up to 15 errors out of 127 bits to generate 36-bit secret key
  - 91-bit syndrome gives away 91 bits of codeword
  - Failure probability will be dependent on PUF error rate
• There are $P!$ possible cases for ordering $P$ oscillators based on their frequencies
  – Each ordering is equally likely
  – For example, 3 oscillators $R_0, R_1, R_2$ have 6 possible orderings
    $(R_0, R_1, R_2), (R_0, R_2, R_1), (R_1, R_0, R_2), (R_1, R_2, R_0), (R_2, R_0, R_1),$ and $(R_2, R_1, R_0)$

• $P$ oscillators can produce $\log_2(P!)$ independent bits
  – 35 oscillators: 133 bits, 128 oscillators: 716 bits, 256 oscillators: 1687 bits

• For ring oscillator “PUF” adversary can predict relationships between PUF output bits if large number of bits are generated
  – Conservative approach is to use $P = 2N$ ring oscillators to generate $N$ bits; no reuse of ring oscillators, no leakage
Key Generation: In the Field

- In the field, the syndrome will be used to re-generate the same PUF reference output from the circuit.

- **Main issue**: PUF *maximum* error rates of 15-20% are hard to correct over long code words.
  - Need failure probability to be at part per billion levels.
Error Correction Complexity

• Some examples of BCH codes that are necessary to correct “raw” ring oscillator outputs
  – (127, 36, 31) gives 36 secret bits, corrects 15 errors; need to run 4 times to get 128-bit secret
  – (255, 63, 61) gives 63 secret bits, corrects 30 errors; need to run twice

• BCH engine complexity grows quadratically with code word size

• Raw bits from ring oscillator comparisons have error rates that are too high for efficient error correction
Reducing Error Rate in PUFs

- PUF output bit may “flip” when environment changes significantly

- **Insight**: Comparisons between ring oscillators with significant difference in frequency are stable even when the environment changes

- Use “far apart” oscillators or delay paths to produce bits
  - Mask bits indicate the selection
  - Need to be careful – mask leaks information!
Index-Based Masking

• **Idea**: Use indices to select PUF bits that are less likely to be noisy

• 1 out of k selection using an index of $\log_2 k$ bits
  – Select the most stable bit that corresponds to the two ring oscillators whose frequencies are furthest apart
  – Polarity of bit can be randomly chosen independent of the PUF

```
Index      00   01   10   11
  0   1   1   0
```

• Need to generate $kN$ bits out of ring oscillator “PUF” (and select N bits using indices)
Theoretical Result

• **Theorem (informal version):** Mask does not leak information assuming PUF outputs are i.i.d and polarities of bits are chosen randomly in advance of index-based coding.

• Conservative assumption for i.i.d implies $2kN$ ring oscillators to generate $N$ bits so mask does not leak information
  - Open question: Can we make do with fewer ring oscillators and still prove an equivalent theorem?
FPGA Testing

• 15 FPGAs (Xilinx) with 1 PUF on each FPGA
• +/- 10% voltage variation experiments
• -20C to 120C temperature variation in test chambers
• Combined voltage and temperature variation tests
• Aging of FPGAs performed and experiments re-run
  – Did not change PUF outputs at all
RO “PUF” Characteristics

- 8000 bits from 1024 oscillators, 1 out of 8 selection
Coding Gain using IBS

Maximum number of erroneous bits = 23

BCH(63, 30, 13)

Maximum number of erroneous bits = 6
PUFs in Secure Processors
Private/Public Key Pair Generation

- PUF response is used as a random seed to a private/public key generation algorithm
  - No secret needs to be handled by a manufacturer

- A device generates a key pair on-chip, and outputs a public key
  - The public key can be endorsed at any time
  - No one needs to know private key

- FPGA implementation built and tested
Intellectual Property Protection

Software Encrypted With Symmetric Key K

Public Key PK
Public Syndrome K encrypted with PK

ECC PUF

CPU/ASIC/FPGA

Private Key (SK)
(Never leaves the chip)

Same for all designs

Different for every chip
Summary

• Silicon manufacturing process variations can be turned into a feature rather than a problem

• PUFs can reliably generate unique and unpredictable volatile secrets for each IC
  – Secure authentication of ICs without cryptographic operations
  – Generation of both symmetric and asymmetric keys for cryptographic operations

• PUFs have been demonstrated on FPGAs and ASICs, including passive RFIDs

• Open questions:
  – How strong are PUFs for authentication?
  – How to create circuits with low noise?
  – How to further enhance physical security through tamper-resistant layout?