Recyclable PUFs: Logically Reconfigurable PUFs

Ünal Kocabaş
unal.kocabas@trust.cased.de
TU Darmstadt (CASED), Germany

Joint Work with
Stefan Katzenbeisser, Vincent van der Leest, Ahmad-Reza Sadeghi,
Geert-Jan Schrijen, Heike Schröder, and Christian Wachsmann
Outline

Physical Security

Physically Unclonable Functions (PUFs)

Logically Reconfigurable PUFs (LR-PUFs)

Application of LR-PUFs
Physical Security

Problem: Cryptography cannot protect against physical attacks
Secrets can be leaked by hardware and/or side-channel attacks

Approach: Use physical properties to build security solutions
- Example: Key exchange based on quantum physics
- Differences to classical cryptography/security:
  - Based on physical instead of computational assumptions
  - Might protect against both algorithmic and physical attacks (tamper-evidence)

Challenge: Find appropriate physical primitives that
- Provide reasonable and verifiable security features
- Are cost-efficient and easy to implement

Promising: Physically Unclonable Functions (PUFs)
Physically Unclonable Functions (PUFs)

Device (e.g., embedded system) → Physically Unclonable Function (noisy function based on physical properties of the device) → Hardware Fingerprint (unique intrinsic device identifier)

- **Inherently Unclonable**
  Due to unpredictable randomness during manufacturing

- **Infeasible to predict**
  Challenge/response behavior is pseudo-random

- **Tamper-evident**
  Tampering with the PUF hardware changes challenge/response behavior
Reconfigurability

Allows to change PUF’s challenge/response behavior after deployment
Ideally, reconfiguration is equivalent to physically replacing the PUF

Extends existing PUF-based security solutions
Example: Secure key erasure/update of secret data bound to PUF
(reconfiguration of PUF “deletes” secrets bound to PUF)

Enables new PUF-based security mechanisms
Example: Protection against software downgrading attacks
(reconfiguration of PUF invalidates software versions bound to pre-reconfigured PUF)

Enables new business models
Example: Recyclable PUF-based access tokens (e.g., RFIDs)
(reconfiguration of PUF allows secure and privacy-preserving re-use of tokens)
In this talk, we present

Logically Reconfigurable PUFs (LR-PUFs)

Formal security model
  Introduces forward and backwards unpredictability
  (specific for reconfigurable PUFs and not covered by previous PUF models)

LR-PUF constructions
  Simple and efficient instantiations and their implementation
  (one optimized for speed and one for area consumption)

Application example
  Recyclable (i.e., re-usable) access tokens based on LR-PUFs
LR-PUF Concept

Logically Reconfigurable PUF

Control Logic
(State S)

Physically Unclonable Function (PUF)

reconfigure
input c
challenge w
response y
output r

A similar concept has been proposed independently by Lao et al. [LP11]
A similar concept has been proposed independently by Lao et al. [LP11]
Assumptions and Adversary Model

Underlying PUF is *unclonable* and *unpredictable*
Can be achieved by using, e.g., a controlled PUF

Algorithm of control logic is publicly known
Typical assumption in cryptography (Kerckhoffs's Principle)

Adversary
- Can adaptively obtain challenge/response pairs of LR-PUF
- Knows current and all previous LR-PUF states
- Cannot set LR-PUF state to a specific value
  *(invasive attacks altering the state of specific memory cells infeasible in practice)*
Security Objectives

Forward Unpredictability
Adversary cannot predict LR-PUF response for previous states

Backward Unpredictability
Adversary cannot predict LR-PUF response for current state
Recyclable PUFs: Logically Reconfigurable PUFs

Definition of Unpredictability Game

A wins the **backward-unpredictability** game if \( r^* \) is a valid LR-PUF response for state \( S_i \) and \( c^* \notin Q_i \)
- For instance, \( A \) may forge a PUF response in an authentication protocol.

A wins the **forward-unpredictability** game if \( r^* \) is a valid LR-PUF response for state \( S_{i-1} \) and \( c^* \notin Q_{i-1} \)
- For instance, \( A \) may recover an old key bound to the PUF.

Formalization follows game-based approach of Armknecht et al. [AMS+11]
Generic Construction

Logically Reconfigurable PUF

Reconfiguration Algorithm
(State $S$)

Input Transformation

Output Transformation

Physically Unclonable Function (PUF)

Reconfigure

input $c$

challenge $w$

response $y$

output $r$
Logically Reconfigurable PUF

Reconfiguration Algorithm (State $S$)

Input Transformation

Physically Unclonable Function (PUF)

Reconfigure

Input $c$

Challenge $w$

Response $y$

Output $r$
Recyclable PUFs: Logically Reconfigurable PUFs

Logically Reconfigurable PUF

Reconfiguration Algorithm
(State $S$)

Hash($S || c$)

64 parallel Arbiter PUFs
(with large CRP space)

challenge $w$

response $y$

output $r$

input $c$

reconfigure

LR-PUF$_S$(c)

$SV \leftarrow E_S(IV)$

$w \leftarrow E_c(SV)$

$y \leftarrow PUF(w)$

$r \leftarrow y$

return $r$
LR-PUFs(c)  
\( SV \leftarrow E_s(IV) \)

**Non-volatile Memory**  
(State \( S \)) + (IV)

**PRESENT Block-cipher**  
Davies-Meyer mode

**64 parallel Arbiter PUFs**  
(with large CRP space)

**Non-volatile Memory**  
(State \( S \)) + (IV)

**PRESENT Block-cipher**  
Davies-Meyer mode

**64 parallel Arbiter PUFs**  
(with large CRP space)

\( IV \) : Initialization vector – 64-bit  
\( SV \) : Session vector – 64-bit
Speed-Optimized Implementation

Logically Reconfigurable PUF

Non-volatile Memory
(State $S'$) + (IV)

PRESENT Block-cipher
Davies-Meyer mode

64 parallel Arbiter PUFs
(with large CRP space)

LR-PUF$_c$(c)
$SV \leftarrow E_s(IV)$
$w \leftarrow E_c(SV)$

$IV$ : Initialization vector – 64-bit
$SV$ : Session vector – 64-bit
Recyclable PUFs: Logically Reconfigurable PUFs

Logically Reconfigurable PUF

Non-volatile Memory
(State $S'$) + (IV)

PRESENT Block-cipher
Davies-Meyer mode

challenge $w$ 
64

64 parallel Arbiter PUFs
(with large CRP space)

response $y$ 
64

output $r$

LR-PUF$_S(c)$
$SV \leftarrow E_S(IV)$
$w \leftarrow E_c(SV)$
$y \leftarrow PUF(w)$
$r \leftarrow y$
return $r$

$IV$ : Initialization vector – 64-bit
$SV$ : Session vector – 64-bit
Recyclable PUFs: Logically Reconfigurable PUFs

Logically Reconfigurable PUF

Non-volatile Memory
(State $S'$) + (IV)

PRESENT Block-cipher
Davies-Meyer mode

Register

reconfigure

$S' \leftarrow \text{Hash}(S)$

$IV$ : Initialization vector – 64-bit
Recyclable PUFs: Logically Reconfigurable PUFs

**Logically Reconfigurable PUF**

- **Non-volatile Memory**
  - (State $S'$) + (IV)

- **PRESENT Block-cipher**
  - Davies-Meyer mode

- **Reconfiguration Algorithm**
  - $S' \leftarrow \text{Hash}(S)
  - $h_1 \leftarrow \text{E}_S(IV)$

- $S$ → 80

- **Initialization vector** – 64-bit
- $h_1$ – hash-1 – 64-bit

---

**Speed-Optimized Implementation**

---

**CHES2011, October 1st 2011**
Recyclable PUFs: Logically Reconfigurable PUFs

Logically Reconfigurable PUF

Non-volatile Memory
(State $S'$) + (IV)

PRESENT Block-cipher
Davies-Meyer mode

$S' \leftarrow \text{Hash}(S)$

$h_1 \leftarrow E_S(\text{IV})$

$h_2 \leftarrow E_{SK}(h_1)$

$IV$ : Initialization vector – 64-bit
$h_1$ : hash-1 – 64-bit
$h_2$ : hash-2 – 64-bit

$S' = h_1[39:0] || h_2[39:0]$
Logically Reconfigurable PUF

Non-volatile Memory
(State $S'$) + (IV)

PRESENT Block-cipher
Davies-Meyer mode

$S' \leftarrow \text{Hash}(S)$

$h_1 \leftarrow \text{E}_S(IV)$
$h_2 \leftarrow \text{E}_{SK}(h_1)$

$S' \leftarrow h_1[39:0]||h_2[39:0]$
Area-Optimized Construction

Logically Reconfigurable PUF

Reconfiguration Algorithm
(State $S$)

Input Transformation

Output Transformation

Single Arbiter PUF
(with small response space)

reconfigure

input $c$

challenge $w_j$

response $y_j$

output $r$

$\ell = y_0 \| \ldots \| y_{63}$
## Performance Results

### Implementation on Xilinx Spartan 6 FPGA
- Based on Arbiter PUFs (take 64 bit challenge, generate 1 bit response each)
- Hash function: PRESENT in Davies-Meyer Mode

<table>
<thead>
<tr>
<th>Optimization</th>
<th>Response time in clock cycles</th>
<th>Area consumption in slices (gate equivalents)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Control logic</td>
</tr>
<tr>
<td>Speed</td>
<td>1,069</td>
<td>166 (1,162 GE)</td>
</tr>
<tr>
<td>Area</td>
<td>64,165</td>
<td>358 (2,506 GE)</td>
</tr>
</tbody>
</table>

**Speed-optimized variant is 63 times faster but 10 times bigger than area-optimized variant**
Use Case: Recyclable Access Tokens

Issuer issues token → User A authenticates using token → Verifier

Issuer returns token after use (gets deposit back)

User B re-issues token (with reconfigured LR-PUF) → User B authenticates using token

B should not be able to use A’s access rights
Recyclable Access Tokens

Save money
No new tokens needed

Can increase security and privacy
Use and re-use small number of advanced tokens instead of a large number of low-cost and constrained one-time tokens

Reduce electronic Waste
Besides obvious ecologic aspects, economic aspect: Governments make vendors of electronic equipment responsible for disposal of their products
Conclusion and Future Work

We presented

- Concept of Logically Reconfigurable PUFs (LR-PUFs)
- Formal security model (backward and forward unpredictability)
- LR-PUF constructions (optimized for speed and area consumption)
- Discussed potential applications

Current and Future work

- Improved LR-PUF constructions allowing for more efficient verification
- Concrete protocols for LR-PUF-based access tokens
Thank you!

Ünal Kocabaş
unal.kocabas@trust.cased.de
TU Darmstadt (CASED), Germany
Back up slide: Controlled PUFs vs. LR-PUFs

**Controlled PUFs**
- Objective: Prevent model building attacks that allow emulating the PUF
- Approach: Hide responses of underlying PUF (e.g., by hashing the PUF response)

**Logically Reconfigurable PUFs**
- Objective: Change challenge/response behavior of underlying PUF after deployment
- Approach: Entangle challenges/responses of underlying PUF with some random state (e.g., by hashing the PUF challenge together with some state)

Although specific instantiations of controlled and logically reconfigurable PUFs look similar, they represent different concepts with different objectives!