End-to-end Design of a PUF based Privacy Preserving Authentication Protocol

Aydin Aysu (Virginia Tech)
Ege Gulcan (Virginia Tech)
†Daisuke Moriyama (NICT)
Patrick Schaumont (Virginia Tech)
Moti Yung (Google/Columbia University)
Motivation

PUF is attractive in implementation and theory

*Implementation*

- Investigate new construction
- Analyze PUF’s data
- Check environmental effect
Motivation

PUF is attractive in implementation and theory

Implementation

- Investigate new construction
- Analyze PUF’s data
- Check environmental effect

Theory

- Propose PUF-based protocol
- Provide security model
Motivation

PUF is attractive in implementation and theory

**Implementation**
- Investigate new construction
- Analyze PUF’s data
- Check environmental effect

**Theory**
- Propose PUF-based protocol
- Provide security model

Combine!!!

Development for Realistic Usage
PUF Protocol Design has a GAP

Provide provable security

Propose protocol

GAP!

Program and evaluate
Propose protocol
Program and evaluate
Provide provable security

Question: How can we implement theoretically secure (provably secure) protocol?

Question: Can the PUF-based protocol be worked in a resource-constrained device?

Theory

Imply.

Program and evaluate
This talk

Theory

Provide provable security

Propose protocol

Extract building blocks

Investigate implementation-primitives for computing elements

Estimate bit length for each variable

Imple.

Program and evaluate

PRF, RNG, MAC, Fuzzy extractor,…

AES, BCH, HMAC,…
First Step

Theory

Propose protocol

Extract building blocks

Investigate implementation-primitives for computing elements

Estimate bit length for each variable

Imple.

Program and evaluate
Theoretical Description (core part)...

Server \( R(\{z'_{1,i}, sk_i, T_i\}_{1 \leq i \leq n}) \)

\[ y_1 \leftarrow \{0, 1\}^k \]

Device \( T_i(f, sk, y'_1) \)

\[ z_1 \overset{R}{\leftarrow} f(x, y'_1) \]

\[ \delta \leftarrow \{0, 1\}^k \]

\( (r_1, hd_1) \overset{R}{\leftarrow} FE.Gen(z_1) \)

\[ c := SKE.Enc(sk, hd_1 \parallel \delta) \]

\[ y_2, y'_2 \leftarrow \{0, 1\}^k \]

\( (t_1, \ldots, t_5) := G(r_1, y_1 \parallel y_2) \)

\[ z_2 \overset{R}{\leftarrow} f(x, y'_2) \]

\[ u_1 := z_2 \oplus t_2 \]

\[ s_1 := G'(t_3, c \parallel u_1) \]

For \( 1 \leq i \leq n, \)

\[ hd_1 \parallel \delta := SKE.Dec(sk_i, c) \]

\[ r_1 := FE.Rep(z'_{1,i} \oplus \delta, hd_1) \]

\( (t'_1, \ldots, t'_5) := G(r_1, y_1 \parallel y_2) \)

If \( t'_1 = t_1 \land s_1 = G'(t_3, c \parallel u_1) \)

\[ z'_2 := u_1 \oplus t_2 \]

\[ sk := t_5 \]

If \( t_4 = t'_4, \)

Update \((y'_1, sk)\) to \((y'_2, t_5)\)
Secure Authentication

Server \( \mathcal{R} \) (PUF DB, key DB)

\[
\text{RNG} \rightarrow y_1
\]

For each DB entries (contain all PUFs),

Key DB \rightarrow \text{Decrypt} \rightarrow \text{helper data} \rightarrow \text{PUF DB} \rightarrow \text{Fuzzy extractor} \rightarrow \text{randomness} \rightarrow \text{PRF} \rightarrow (t'_1, \ldots, t'_5)

If \( t'_1 = t_1 \wedge s_1 = \text{PRF}(t'_3, u_1 || c) \), Accept!

Update DBs to \( (t_2 \oplus u_1, t'_5) \)

Device \( \mathcal{T}_i \) (Stored data 1 and 2)

\[
\text{RNG} \rightarrow y_1 \rightarrow y_2
\]

\[
\text{RNG} \rightarrow \text{Fuzzy extractor} \rightarrow \text{randomness} \rightarrow \text{PRF} \rightarrow \text{(helper data, (t_1, \ldots, t_5))}
\]

\[
\text{RNG} \rightarrow \text{PUF} \rightarrow \text{Encrypt} \rightarrow (c, y_2, t_1, u_1, s_1)
\]

\[
\text{RNG} \rightarrow \text{PUF} \rightarrow \text{PRF} \rightarrow c \rightarrow u_1 \rightarrow s_1
\]

If \( t_4 = t'_4 \), Accept!

Update stored data to \( (y'_2, t_5) \)
Secure Authentication

Server $\mathcal{R}$ (PUF DB, key DB)

PUF is evaluated twice
- First data is used for authentication
- Second data is encrypted and used for next authentication

If $t_1' = t_1 \land s_1 = \text{PRF}(t_3', u_1 \| c)$, Accept!
Update DBs to $(t_2 \oplus u_1, t_5')$

If $t_4 = t_4'$, Accept!
Update stored data to $(y_2', t_5)$
Secure Authentication

Server \( \mathcal{R} \) (PUF DB, key DB)

PUF is evaluated twice
- First data is used for authentication
- Second data is encrypted and used for next authentication

Support mutual authentication

Device \( \mathcal{T}_i \) (Stored data 1 and 2)

- First data is used for authentication
- Second data is encrypted and used for next authentication

If \( t_1' = t_1 \land s_1 = \text{PRF}(t_3', u_1 \| c) \), Accept!
Update DBs to \((t_2 \oplus u_1, t_5')\)

If \( t_4 = t_4' \), Accept!
Update stored data to \((y_2', t_5)\)
Secure Authentication

Server $\mathcal{R}$ (PUF DB, key DB)

Device $\mathcal{T_i}$ (Stored data 1 and 2)

For each DB entries (contain all PUFs),

- No identity in communication
- Server mounts exhaustive search

Secure Authentication

If $t_1 = t_1 \land s_1 = \text{PRF}(t'_3, u_1 || c)$, Accept!
Update DBs to $(t_2 \oplus u_1, t'_5)$

If $t_4 = t'_4$, Accept!
Update stored data to $(y'_2, t_5)$
Secure Authentication

For each DB entries (contain all PUFs),

If \( t_1' = t_1 \land s_1 = \text{PRF}(t_3', u_1 \parallel c) \), \textbf{Accept!}

Update DBs to \((t_2 \oplus u_1, t_5')\)

If \( t_4 = t_4' \), \textbf{Accept!}

Update stored data to \((y_2', t_5)\)
Abstract Description

Server
- Key/PUF DB
- Protocol

Device
- Non-VM Memory
- Protocol
- PUF

- RNG
- Fuzzy Extractor
- PRF
- Encrypt
Third Step

Theory

- Provide provable security
- Propose protocol
- Extract building blocks
- Investigate implementation-primitives for computing elements
- Estimate bit length for each variable

Imple.

- Program and evaluate
We select SRAM PUF and evaluated with SASEBO-GII (SRAM PUF is area efficient)

To avoid bias, 2-XORed is performed.

Min-entropy rate: 26%
Noise rate: 10%

8-XORed SRAM data passed NIST random test
Implement Fuzzy Extractor

ECC part: Code-offset with (63,16,23)-BCH code

Correct noise up to 11-bit in 63-bit

Encode (device side)

original PUF data

randomness 16-bit

BCH.Encode

Helper data 63-bit

Decode (server side)

Noisy PUF data 63-bit

BCH.Decode

Original PUF data
Implement Fuzzy Extractor

ECC part: Code-offset with (63,16,23)-BCH code

Min-entropy rate: 26% → 128-bit entropy in 8x63-bit PUF data

Remark: 10% noise rate

Correct one block (63-bit): 97.62%
Correct eight blocks (8x63-bit): 82.61% → Need modification
Implement Fuzzy Extractor

ECC part: Code-offset with (63,16,23)-BCH code

4x63-bit (=252-bit) PUF’s data

Novelty: Apply code-offset for left-rotated PUF’s data
Implement Fuzzy Extractor

ECC part: Code-offset with (63,16,23)-BCH code

Novelty: Apply code-offset for left-rotated PUF’s data

Correctness is improved (> 1 - 10^-6)

Security is also analyzed
Implement Fuzzy Extractor

Randomness extraction part: CBC-MAC based PRF + randomness

504-bit Input data + 256-bit randomness

Secret key (seed)

128-bit output data

PRF and this part are performed by same code

We selected SIMON for the encryption algorithm
Final Step

Theory

- Propose protocol
- Extract building blocks
- Investigate implementation-primitives for computing elements
- Estimate bit length for each variable

Impl.

- Provide provable security
- Program and evaluate
Architecture Design

We provide two versions:

- Soft-core mapping MSP430 in FPGA
- MSP430 w/ Micro-coded hardware implementation
## Implementation Results

<table>
<thead>
<tr>
<th>Category</th>
<th>64-bit SW (MSP430)</th>
<th>128-bit SW (MSP430)</th>
<th>128-bit HW</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text size</td>
<td>6,862</td>
<td>8,104</td>
<td>4,920</td>
<td>Bytes</td>
</tr>
<tr>
<td>Time</td>
<td>562,632</td>
<td>1,859,754</td>
<td>240,814</td>
<td>Cycles</td>
</tr>
</tbody>
</table>

- Fit in real MSP430 (8KB)
- Cycle count includes all procedures
  - In SW, BCH encoding is heavy
  - In HW, write/read from memory is heavy
## Comparison with related works

<table>
<thead>
<tr>
<th></th>
<th>PUFKY (CHES 2012)</th>
<th>Slender (S&amp;P 2012)</th>
<th>Reverse-FE (FC 2012)</th>
<th>This work</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Application</strong></td>
<td>Key Gen</td>
<td>Protocol</td>
<td>Protocol</td>
<td>Protocol</td>
</tr>
<tr>
<td><strong>Privacy</strong></td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Security flaws</strong></td>
<td>No</td>
<td>Yes (ePrint 2014/977)</td>
<td>Yes (ePrint 2014/977)</td>
<td>No</td>
</tr>
<tr>
<td><strong>Cycle count</strong></td>
<td>55,310</td>
<td>-</td>
<td>-</td>
<td>1,859,754 (SW) 240,814 (HW)</td>
</tr>
<tr>
<td><strong>Logic cost</strong></td>
<td>120 Slices</td>
<td>144 LUT, 274 Register</td>
<td>658 LUT, 496 Register</td>
<td>1221 LUT, 442 Register</td>
</tr>
<tr>
<td><strong>PUF</strong></td>
<td>RO-PUF</td>
<td>XOR-Arbit PUF</td>
<td>-</td>
<td>SRAM PUF</td>
</tr>
</tbody>
</table>
Conclusions

• We demonstrated how to bridge theory and implementation

• Implementing secure protocol requires many steps

• The proposed protocol can fit in microcontroller MSP 430: text size < 8KB (further optimization is still possible)
Thank you for your attention!
Appendix: Process of our code-offset

ECC part: Code-offset with (63,16,23)-BCH code

Novelty: Apply code-offset for left-rotated PUF’s data

Noise < 12bit

Noise >= 12bit

4x63-bit (=252-bit) PUF’s data

47-bit among 63-bit has been noiseless
## Appendix: Implementation Cost

<table>
<thead>
<tr>
<th>Category</th>
<th>64-bit SW (MSP430)</th>
<th>128-bit SW (MSP430)</th>
<th>128-bit HW</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>HW abstraction</td>
<td>1,022</td>
<td>1,022</td>
<td>1,398</td>
<td>Bytes</td>
</tr>
<tr>
<td>Communication</td>
<td>496</td>
<td>644</td>
<td>628</td>
<td>Bytes</td>
</tr>
<tr>
<td>SIMON</td>
<td>1,604</td>
<td>2,440</td>
<td>0</td>
<td>Bytes</td>
</tr>
<tr>
<td>BCH encoding</td>
<td>1,214</td>
<td>1,214</td>
<td>0</td>
<td>Bytes</td>
</tr>
<tr>
<td>PUF + Fuzzy</td>
<td>562</td>
<td>646</td>
<td>590</td>
<td>Bytes</td>
</tr>
<tr>
<td>RNG</td>
<td>396</td>
<td>456</td>
<td>396</td>
<td>Bytes</td>
</tr>
<tr>
<td>Protocol</td>
<td>1,568</td>
<td>1,682</td>
<td>1,908</td>
<td>Bytes</td>
</tr>
<tr>
<td><strong>Total text</strong></td>
<td><strong>6,862</strong></td>
<td><strong>8,104</strong></td>
<td><strong>4,920</strong></td>
<td>Bytes</td>
</tr>
<tr>
<td>Data</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Variables</td>
<td>424</td>
<td>656</td>
<td>656</td>
<td>Bytes</td>
</tr>
<tr>
<td>Constants</td>
<td>197</td>
<td>197</td>
<td>73</td>
<td>Bytes</td>
</tr>
<tr>
<td><strong>Total data</strong></td>
<td><strong>621</strong></td>
<td><strong>853</strong></td>
<td><strong>729</strong></td>
<td>Bytes</td>
</tr>
</tbody>
</table>

Fit into real MSP430 (8KB memory space)
## Appendix: Performance details

<table>
<thead>
<tr>
<th>Category</th>
<th>64-bit SW (MSP430)</th>
<th>128-bit SW (MSP430)</th>
<th>128-bit HW</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read stored data</td>
<td>31,356</td>
<td>61,646</td>
<td>61,646</td>
<td>Cycles</td>
</tr>
<tr>
<td>RNG (SRAM)</td>
<td>11,552</td>
<td>23,341</td>
<td>22,981</td>
<td>Cycles</td>
</tr>
<tr>
<td>SRAM PUF</td>
<td>4,384</td>
<td>9,082</td>
<td>8,741</td>
<td>Cycles</td>
</tr>
<tr>
<td>BCH encoding</td>
<td>268,820</td>
<td>485,094</td>
<td></td>
<td>Cycles</td>
</tr>
<tr>
<td>Fuzzy extractor</td>
<td>28,691</td>
<td>205,080</td>
<td></td>
<td>Cycles</td>
</tr>
<tr>
<td>First PRF</td>
<td>39,583</td>
<td>299,724</td>
<td>18,597</td>
<td>Cycles</td>
</tr>
<tr>
<td>Encrypt</td>
<td>44,355</td>
<td>252,829</td>
<td></td>
<td>Cycles</td>
</tr>
<tr>
<td>Second PRF</td>
<td>57,601</td>
<td>394,129</td>
<td></td>
<td>Cycles</td>
</tr>
<tr>
<td>Write updated data</td>
<td>76,290</td>
<td>128,829</td>
<td>128,849</td>
<td>Cycles</td>
</tr>
<tr>
<td>Total cycles</td>
<td>562,632</td>
<td>1,859,754</td>
<td>240,814</td>
<td>Cycles</td>
</tr>
</tbody>
</table>

Expensive part in SW: BCH encoding
Expensive part in HW: read/write data