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

Theory

Provide provable security

Propose protocol

Imple.

Program and evaluate

PUF Protocol Design has a GAP

Provide provable security

Theory

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

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

Imple.

Program and evaluate

This talk

Provide provable security

Theory

↑

Propose protocol

↓

Extract building blocks

PRF, RNG, MAC, Fuzzy extractor,…

↓

Investigate implementation-primitives for computing elements

AES, BCH, HMAC,…

↓

Estimate bit length for each variable

↓

Imple.

Program and evaluate

First Step

Provide provable security

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 $\mathcal{R}(\{z'_{1.i}, sk_i, \mathcal{T}_i\}_{1 \le i \le n})$

Device $\mathcal{T}_i(f, sk, y'_1)$

Server:

$$y_1 \overset{\mathsf{U}}{\leftarrow} \{0,1\}^k$$

Server → Device: y_1

Device:

$$z_1 \overset{\mathsf{R}}{\leftarrow} f(x, y'_1)$$
$$\delta \overset{\mathsf{U}}{\leftarrow} \{0,1\}^k$$
$$(r_1, hd_1) \overset{\mathsf{R}}{\leftarrow} \mathsf{FE.Gen}(z_1)$$
$$c := \mathsf{SKE.Enc}(sk, hd_1 \| \delta)$$
$$y_2, y'_2 \overset{\mathsf{U}}{\leftarrow} \{0,1\}^k$$
$$(t_1, \ldots, t_5) := \mathcal{G}(r_1, y_1 \| y_2)$$
$$z_2 \overset{\mathsf{R}}{\leftarrow} f(x, y'_2)$$
$$u_1 := z_2 \oplus t_2$$
$$s_1 := \mathcal{G}'(t_3, c \| u_1)$$

Device → Server: (c, y_2, t_1, u_1, s_1)

Server:

For $1 \le i \le n$,

$$hd_1 \| \delta := \mathsf{SKE.Dec}(sk_i, c)$$
$$r_1 := \mathsf{FE.Rep}(z'_{1.i} \oplus \delta, hd_1)$$
$$(t'_1, \ldots, t'_5) := \mathcal{G}(r_1, y_1 \| y_2)$$

If $t'_1 = t_1 \land s_1 = \mathcal{G}'(t_3, c \| u_1)$

$$z'_2 := u_1 \oplus t_2$$
$$sk := t_5$$

Server → Device: t'_4

Device:

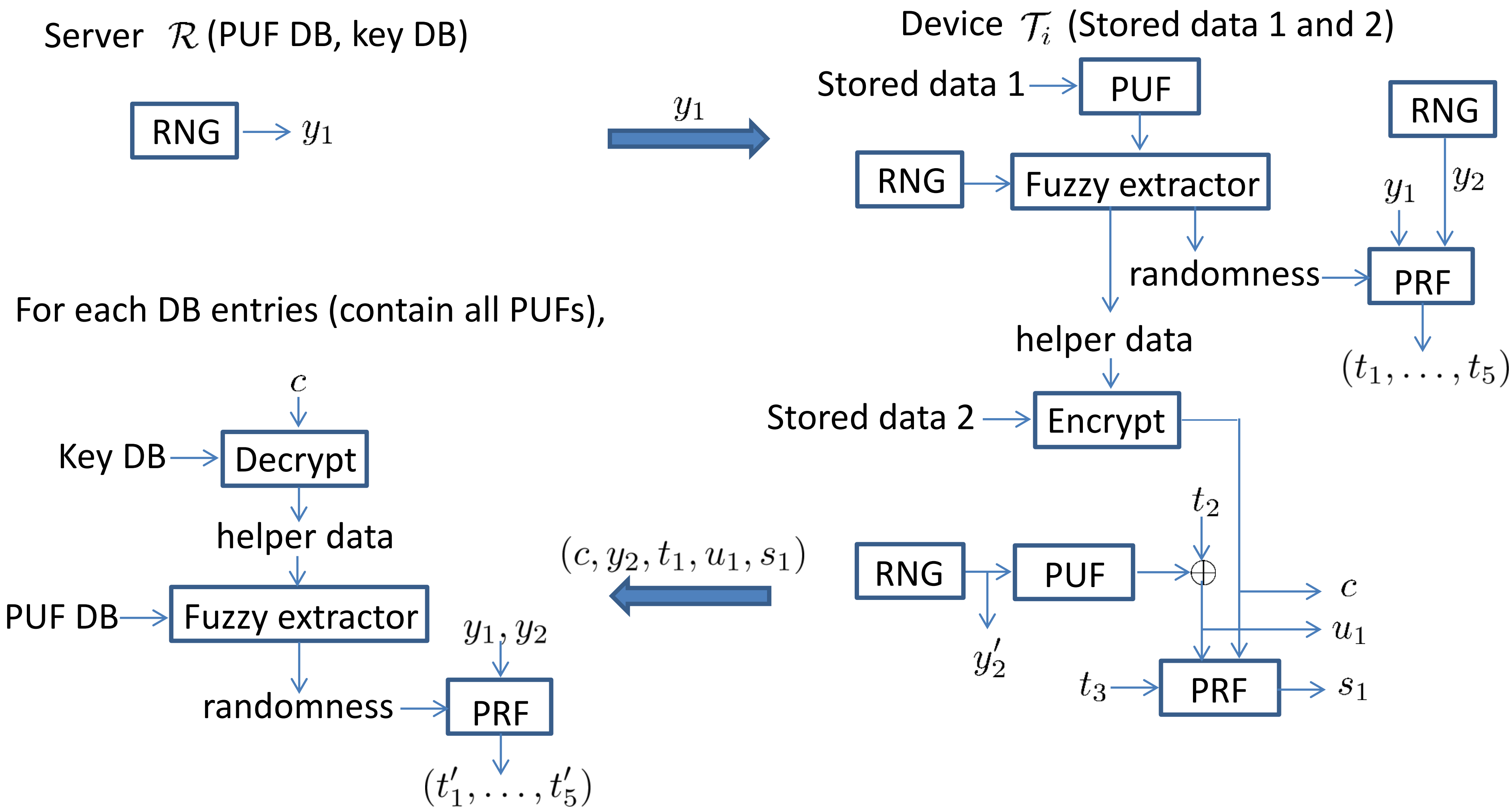
If $t_4 = t'_4$,

Update (y'_1, sk) to (y'_2, t_5)

f : PUF

$\mathcal{G}, \mathcal{G}'$: PRFs

Secure Authentication



If $t'_1 = t_1 \wedge s_1 = \mathsf{PRF}(t'_3, u_1 \| c)$, Accept!
Update DBs to $(t_2 \oplus u_1, t'_5)$

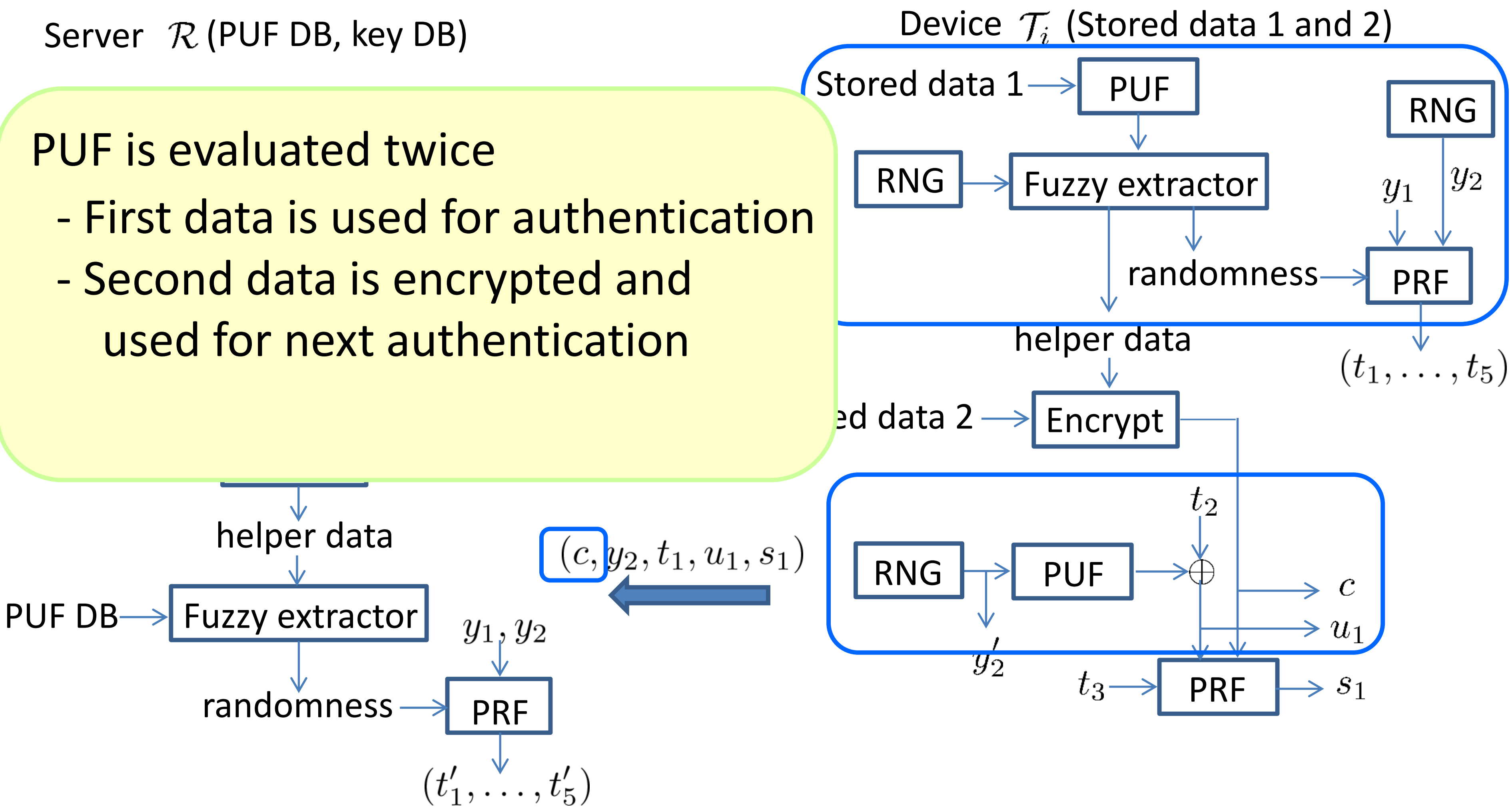
t'_4

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

Secure Authentication

PUF is evaluated twice

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



If $t'_1 = t_1 \wedge s_1 = \mathrm{PRF}(t'_3, u_1 \| c)$, Accept!
Update DBs to $(t_2 \oplus u_1, t'_5)$

t'_4

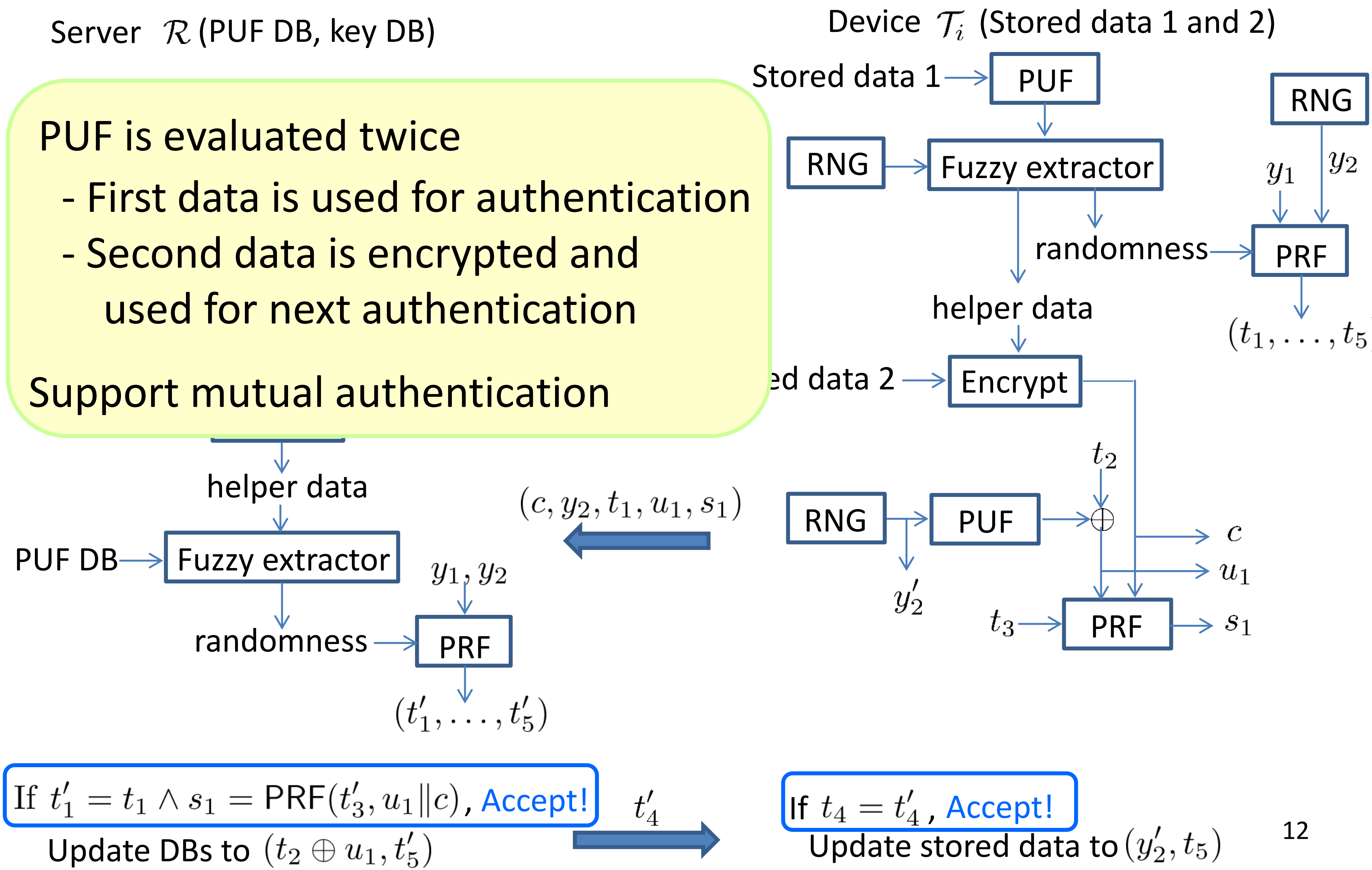
If $t_4 = t'_4$, Accept!
Update stored data to (y'_2, t_5)

Secure Authentication

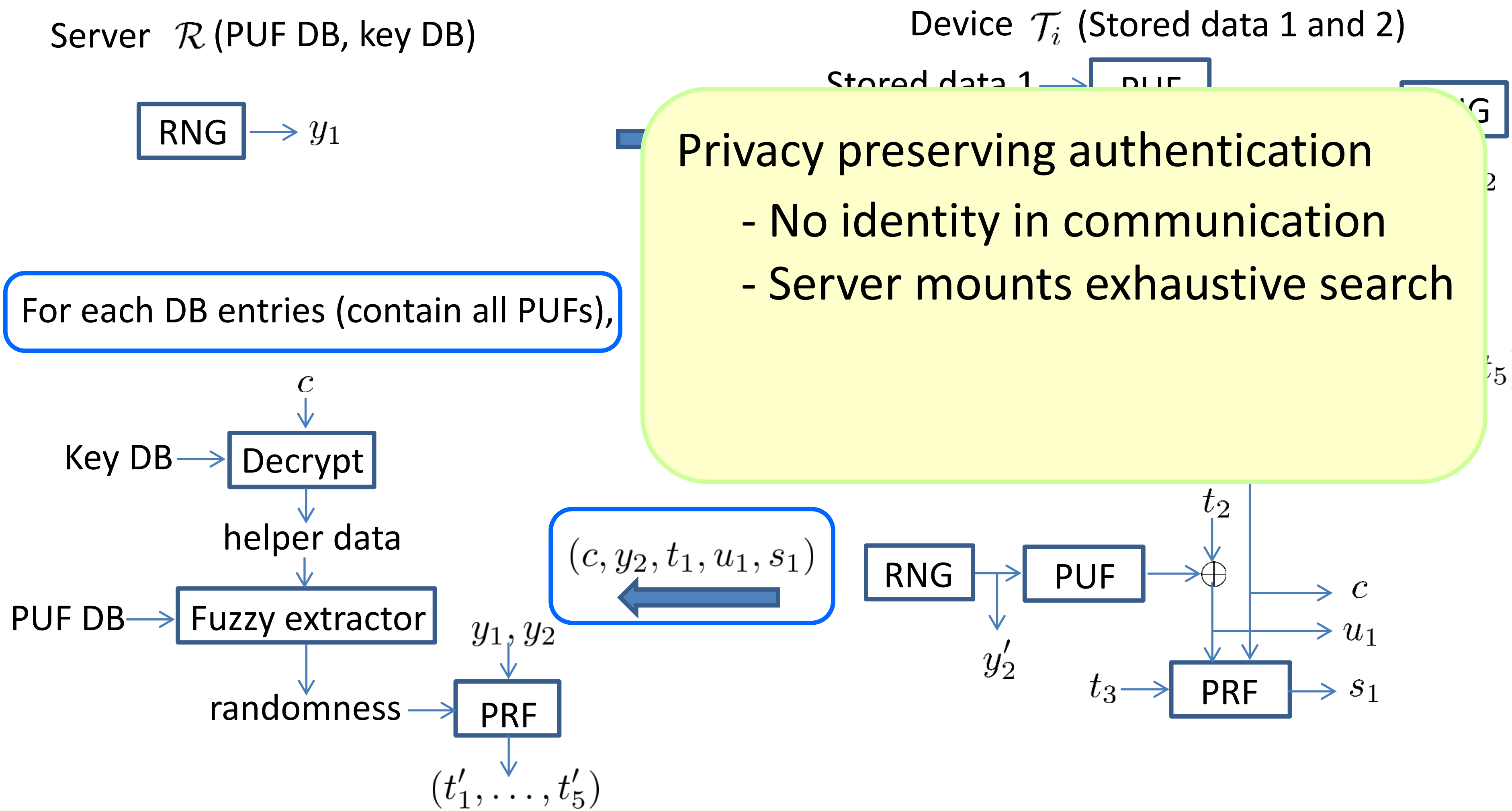
PUF is evaluated twice

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

Support mutual authentication



Secure Authentication

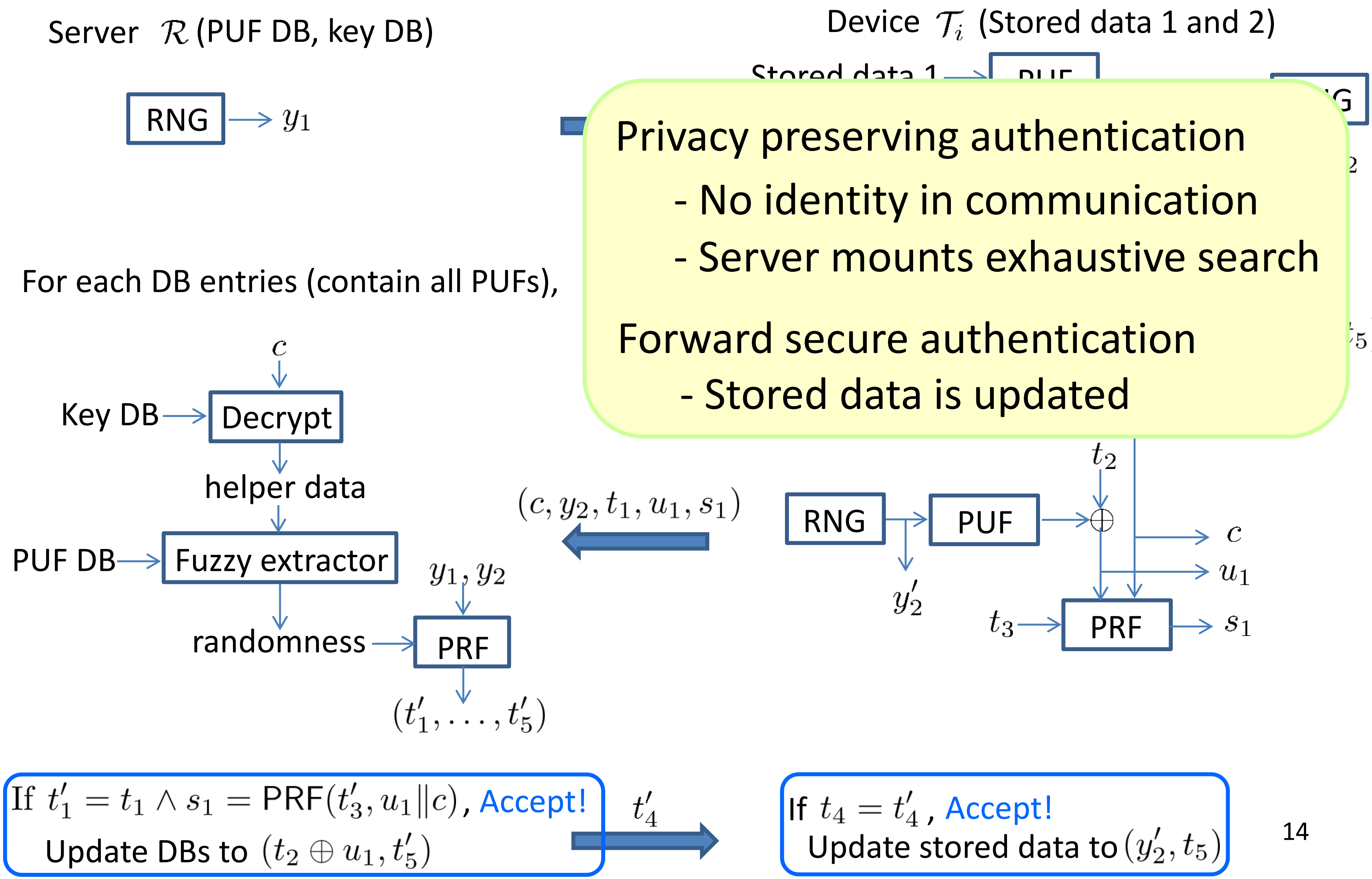


If $t'_1 = t_1 \wedge s_1 = \mathrm{PRF}(t'_3, u_1 \| c)$, Accept!
Update DBs to $(t_2 \oplus u_1, t'_5)$

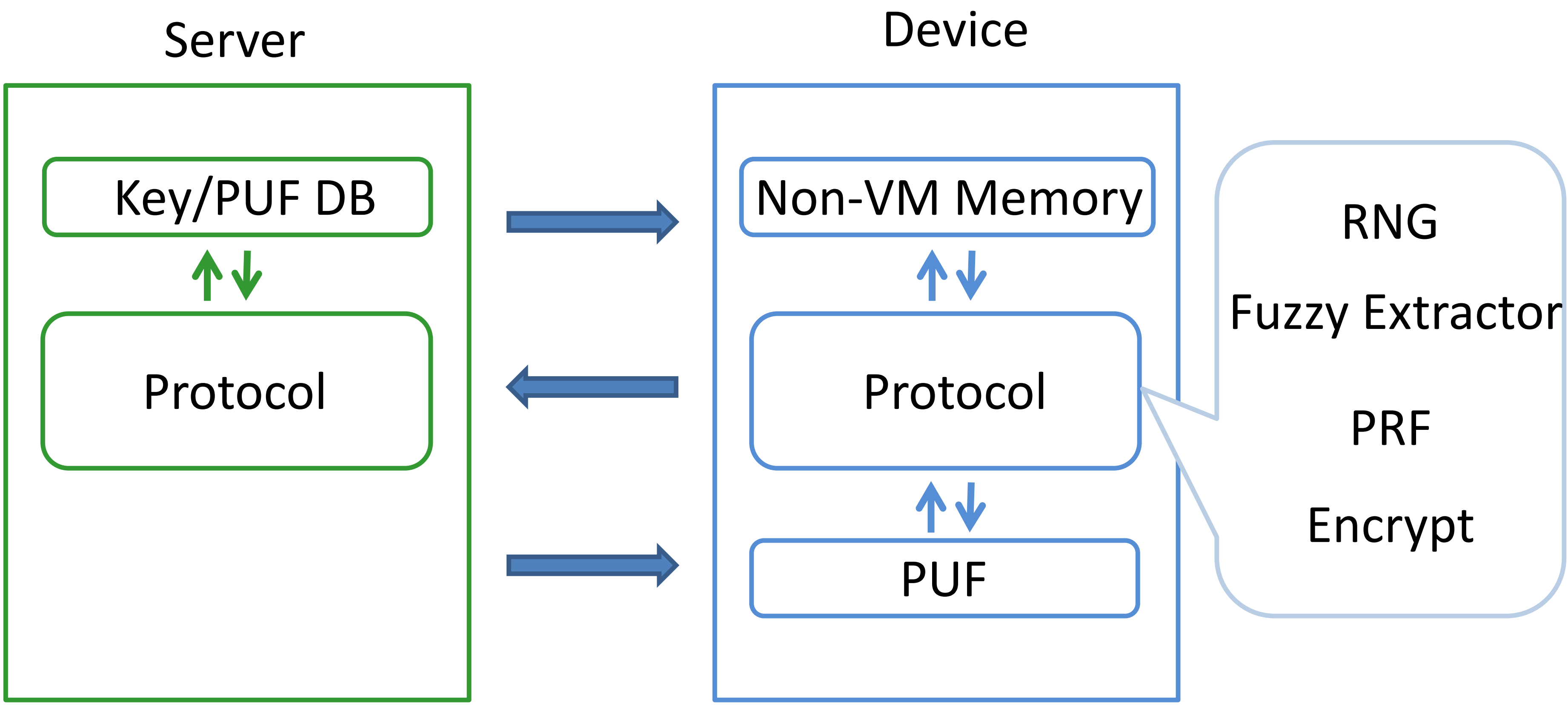
t'_4

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

Secure Authentication



Abstract Description



Third Step

Theory

Provide provable security

Propose protocol

Extract building blocks

Investigate implementation-primitives
for computing elements

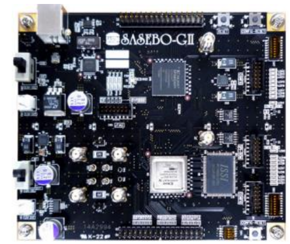
Estimate bit length for each variable

Program and evaluate

Imple.

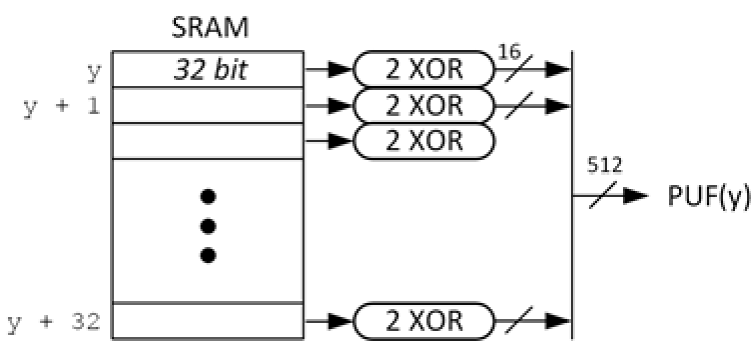
PUF & RNG Construction

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



x100

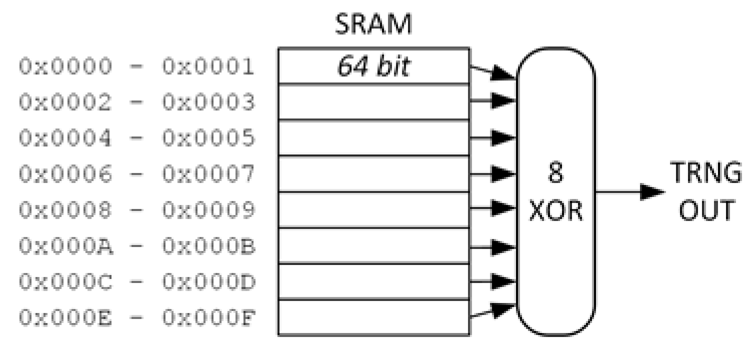
SRAM PUF part



To avoid bias, 2-XORed is performed

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

RNG part



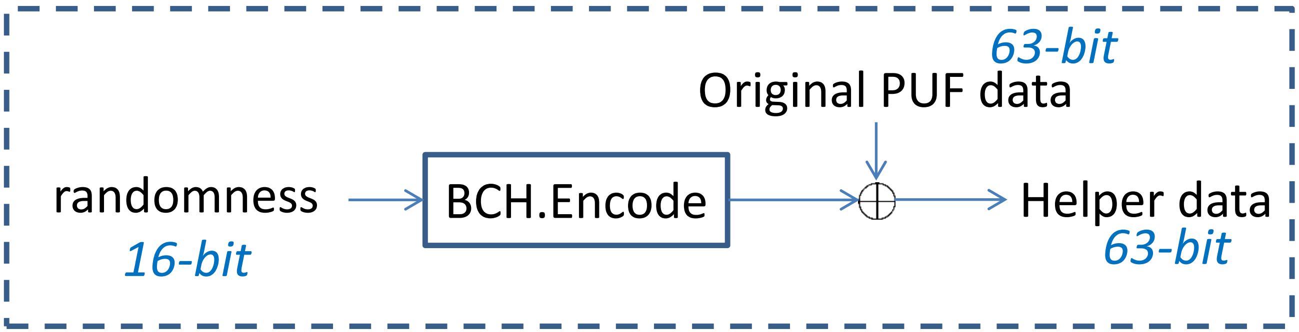
8-XORed SRAM data passed NIST random test

Implement Fuzzy Extractor

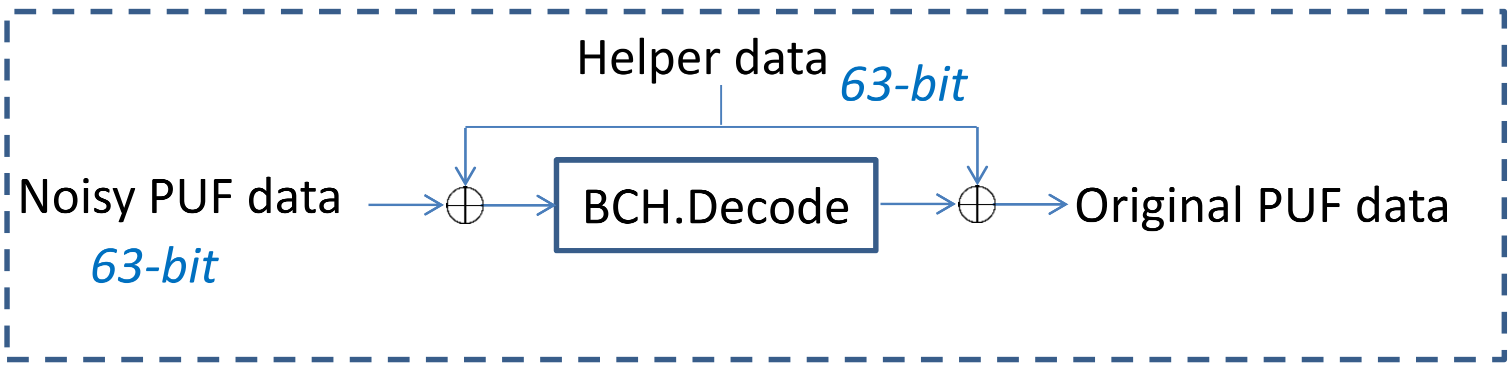
ECC part: Code-offset with (63,16,23)-BCH code

Correct noise up to 11-bit in 63-bit

Encode
(device side)

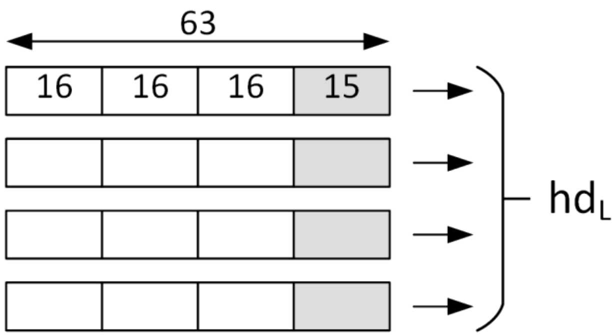


Decode
(server side)



Implement Fuzzy Extractor

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



4x63-bit (=252-bit) PUF's data

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

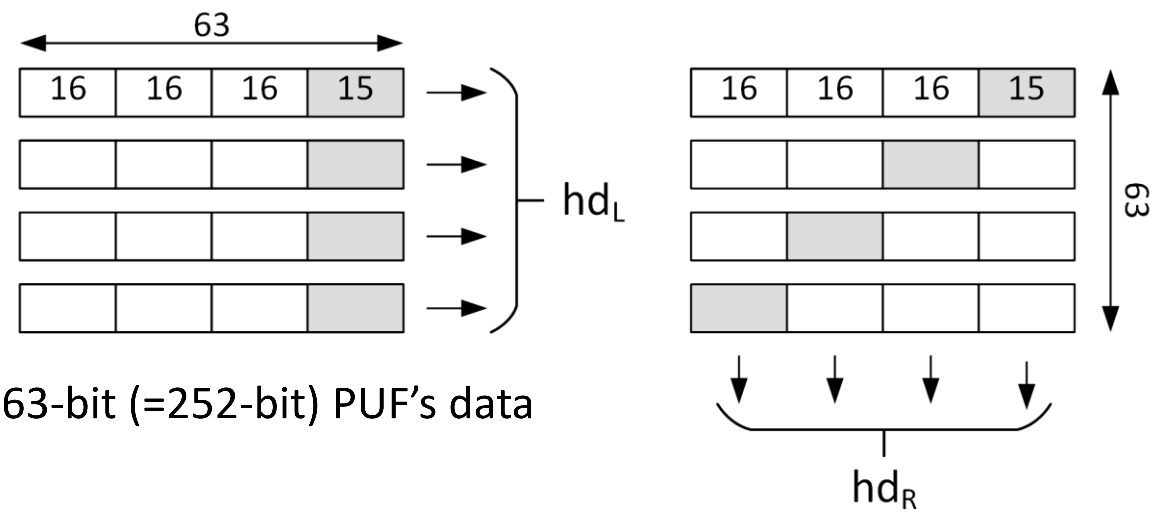
Remark: 10% noise rate

- 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

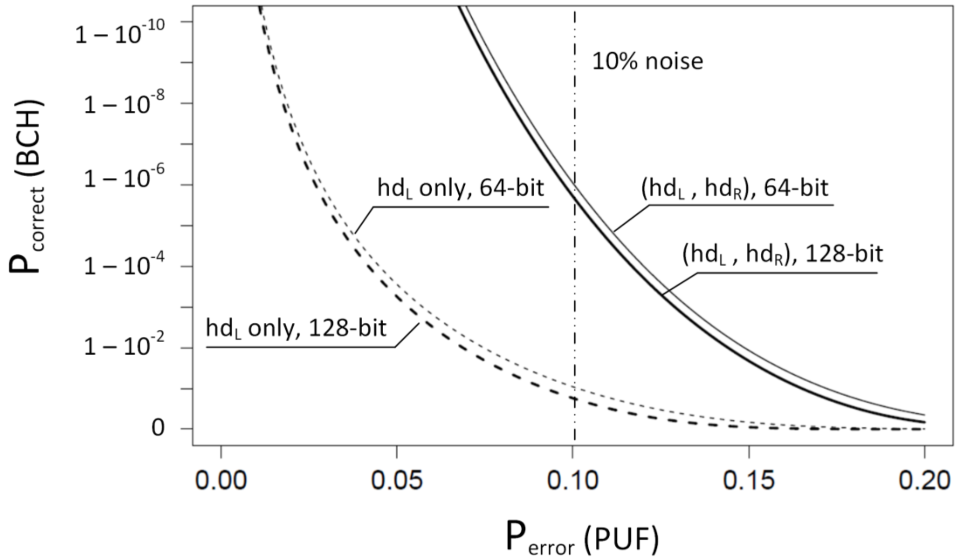


4x63-bit (=252-bit) PUF's data

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

Implement Fuzzy Extractor

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



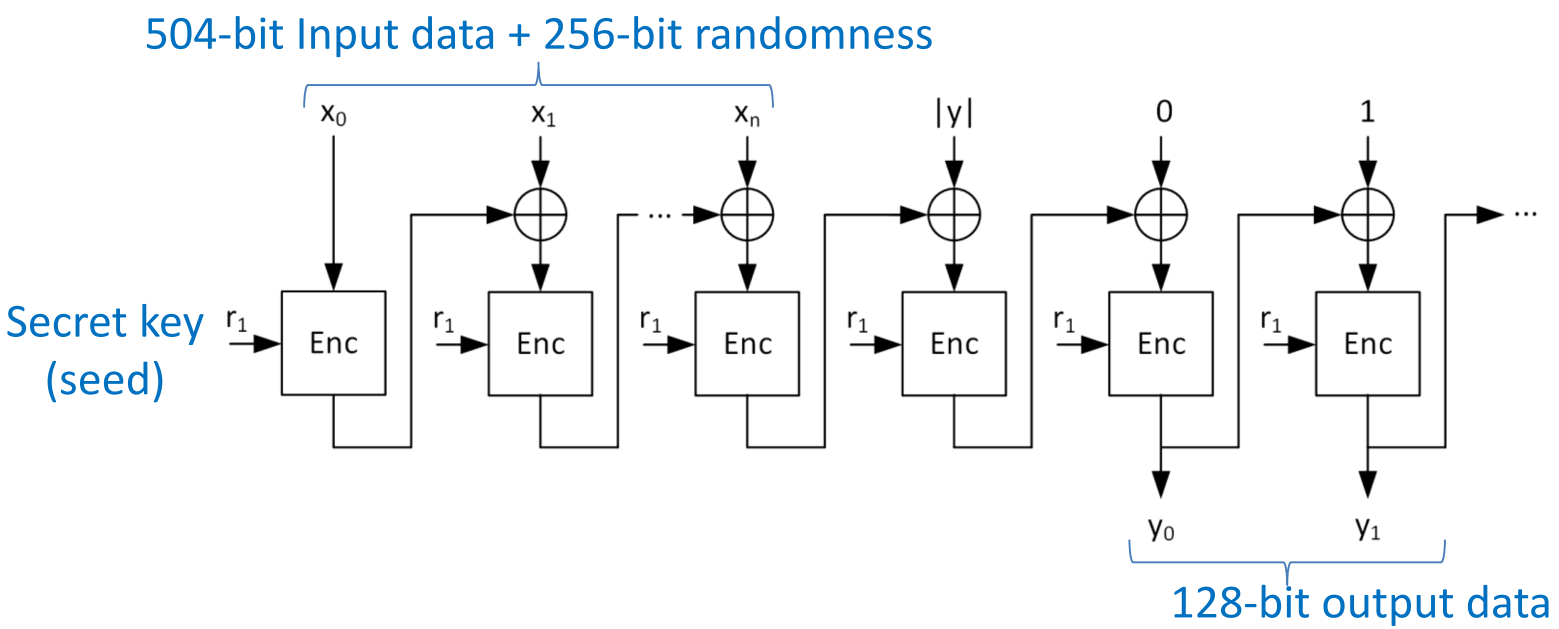
Novelty: Apply code-offset for left-rotated PUF's data

Correctness is improved ($> 1 - 10^{-6}$)

Security is also analyzed

Implement Fuzzy Extractor

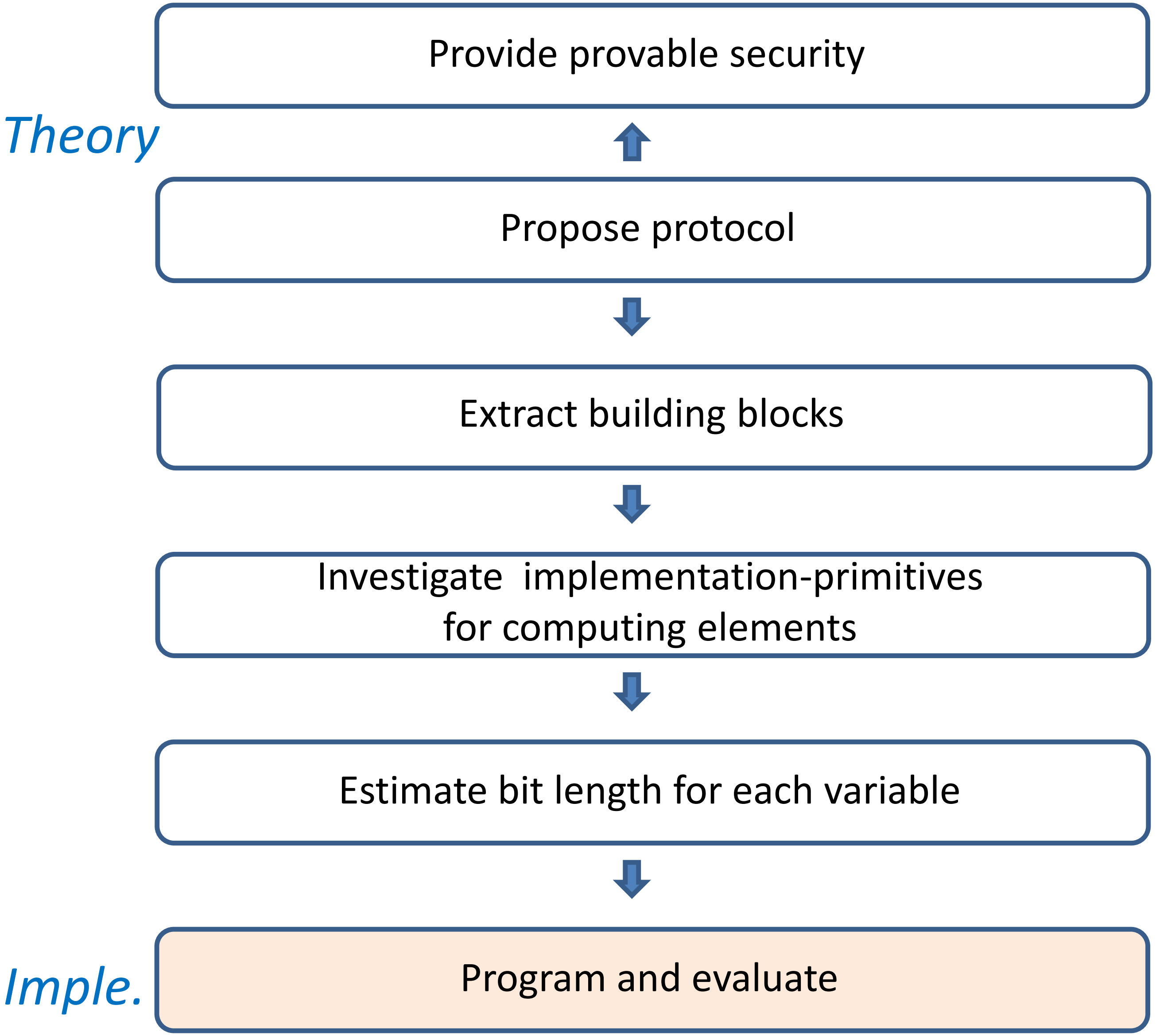
Randomness extraction part: CBC-MAC based PRF + randomness



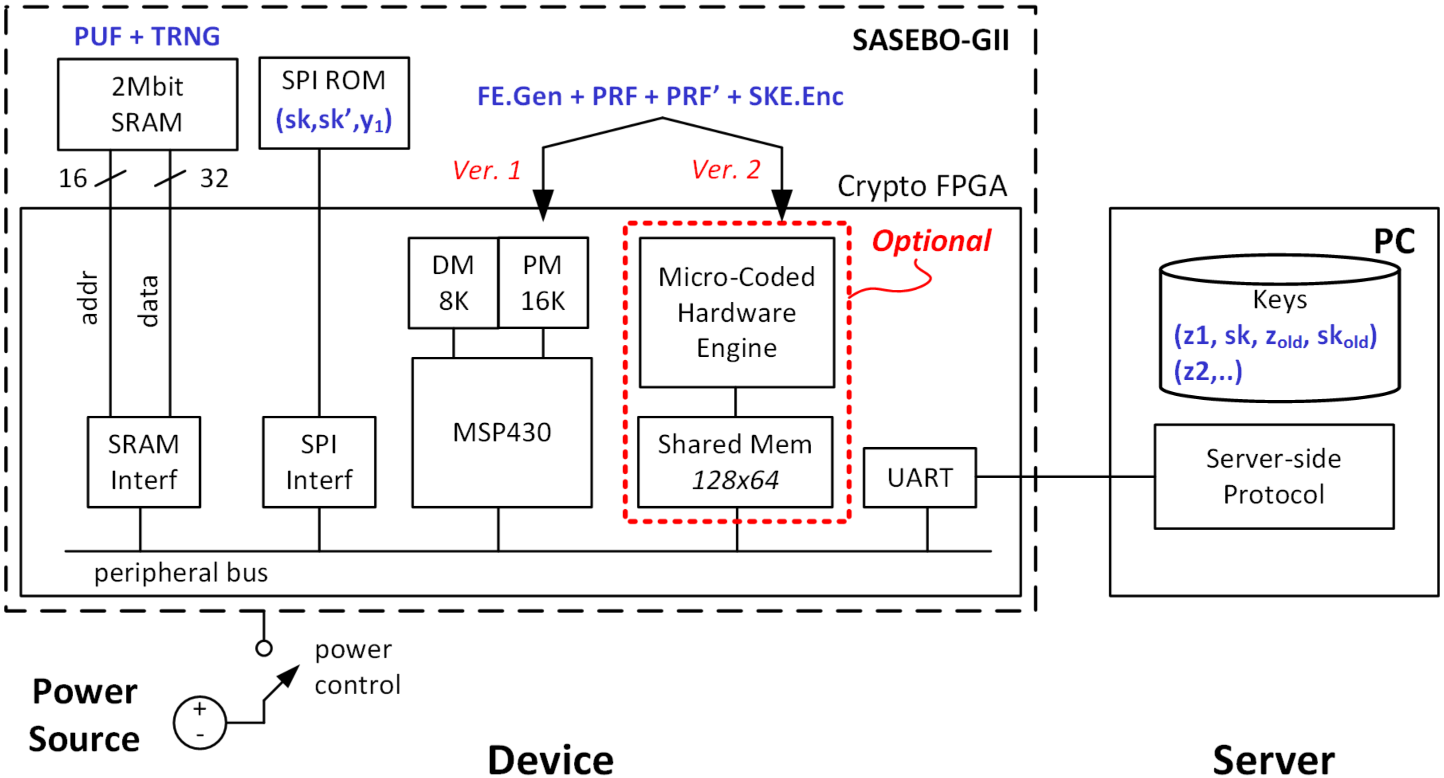
PRF and this part are performed by same code

We selected SIMON for the encryption algorithm

Final Step


Provide provable security
Theory
Propose protocol
Extract building blocks
Investigate implementation-primitives for computing elements
Estimate bit length for each variable
Imple.
Program and evaluate

Architecture Design



We provide two versions:

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

Implementation Results

Category	64-bit SW (MSP430)	128-bit SW (MSP430)	128-bit HW	Unit
Text size	6,862	8,104	4,920	Bytes
Time	562,632	1,859,754	240,814	Cycles

- 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

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

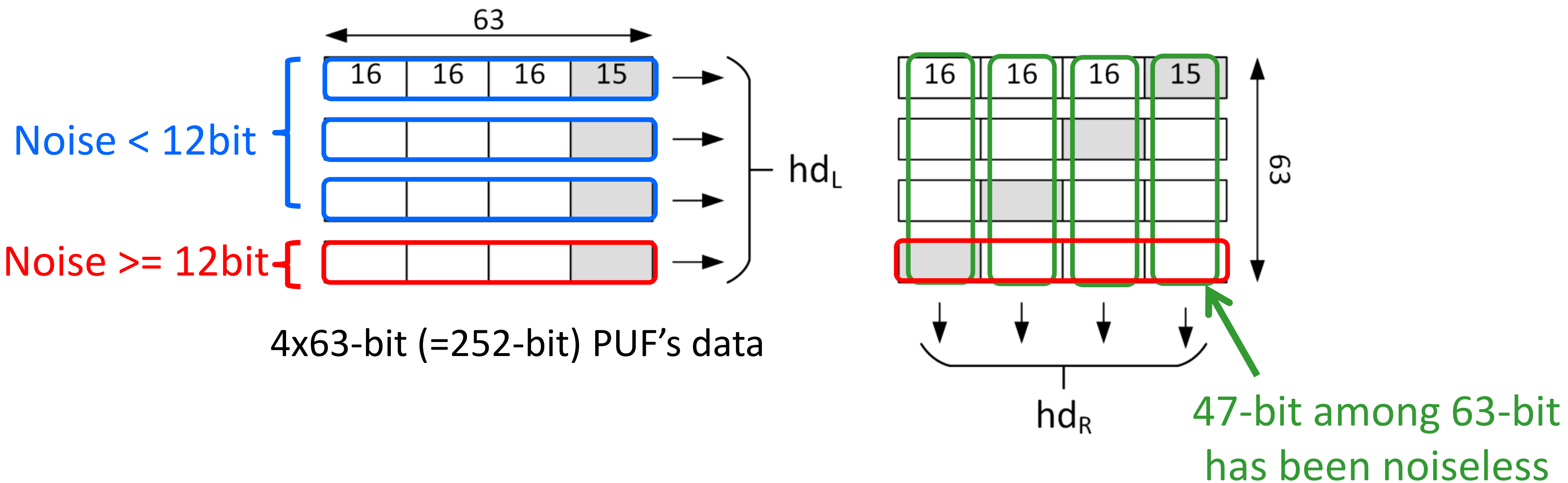
Conclusions

- 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

Appendix: Implementation Cost

Category		64-bit SW (MSP430)	128-bit SW (MSP430)	128-bit HW	Unit
Text	HW abstraction	1,022	1,022	1,398	Bytes
	Communication	496	644	628	Bytes
	SIMON	1604	2,440	0	Bytes
	BCH encoding	1,214	1,214	0	Bytes
	PUF + Fuzzy	562	646	590	Bytes
	RNG	396	456	396	Bytes
	Protocol	1,568	1,682	1,908	Bytes
Total text		6,862	8,104	4,920	Bytes
Data	Variables	424	656	656	Bytes
	Constants	197	197	73	Bytes
Total data		621	853	729	Bytes

Fit into real MSP430 (8KB memory space)

Appendix: Performance details

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

Expensive part in SW: BCH encoding

Expensive part in HW: read/write data