Secure Key Storage and True Random Number Generation

–

René Struik
(Struik Security Consultancy)

e-mail: rstruik.ext@gmail.com

Outline

1. Putting Trust in Devices
   - Conventional Approach
   - Exploiting “physical” device properties
2. Secure Key Storage via PUFs
   - Main Idea
   - Reliability
   - Randomness
   - Instantiations
3. True Random Number Generation via PUFs
   - Main Idea
   - Randomness Extraction
   - Seeds via PUFs
5. Conclusions & Future Directions

Putting Trust in Devices

Conventional Approach

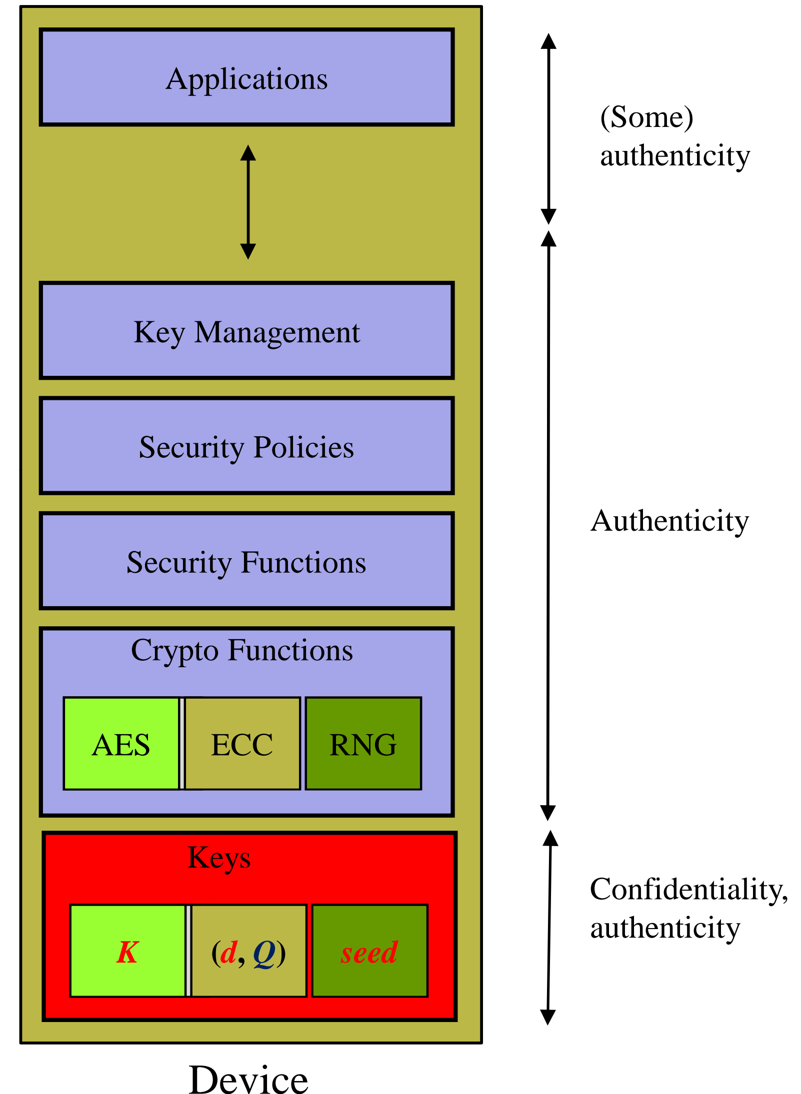
- Trusted implementation of crypto, including side channel resistance
- Trusted security policy routines
- Secure and authentic key storage
- Secure RNG (or RNG seed)

This requires persistent, secure key storage

What to do if *no* persistent storage on device? (or, not yet)

What to do if attacks target

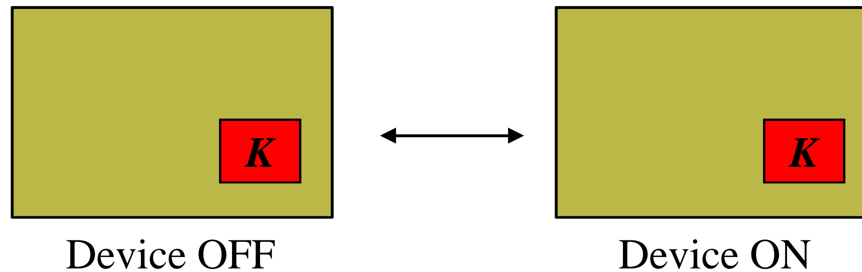
- Key storage location?
- Seeds for random number generator?



Secure Key Storage – Main Idea

Conventional Approach

- Persistent, secure key storage

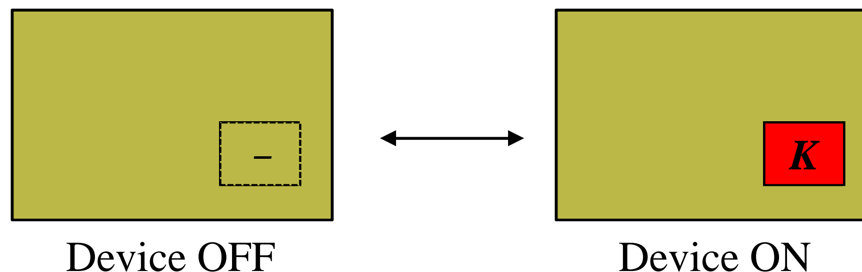


Properties

- Persistent, secure key storage required
- Potential attacks on key storage in "Device OFF" state

Potential approach

- Secure key storage ON/OFF button



The promise…

- No persistent, secure key storage needed
- No attacks on key storage in "Device OFF" state possible

Key *K* is derived from "device properties" upon device start-up (physically unclonable functions – PUFs)

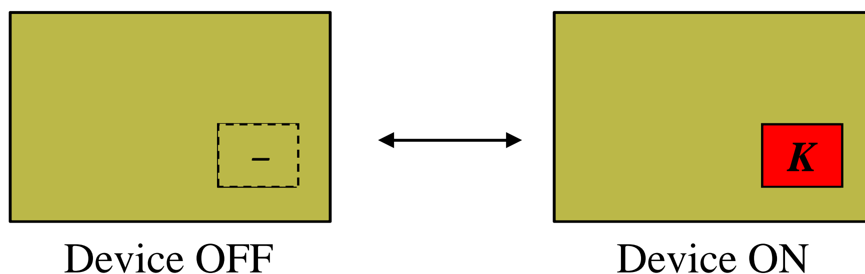
Secure Key Storage via PUFs

– Main Idea
– Reliability
– Randomness
– Instantiations

Secure Key Storage – Reliability of PUFs (1)

Basic approach – uniform, exact distribution

- Secure key storage ON/OFF button

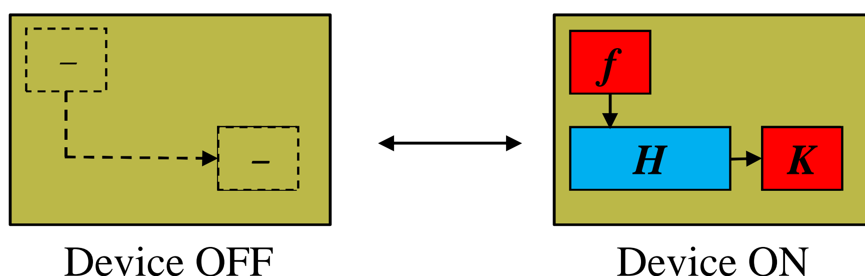


Key *K* is derived from "device properties" upon device start-up

Assumptions on key K

- Key *K* random
- Reliable reconstruction of key *K* possible

Alleviated constraints – biased, exact distribution



Assumptions on PUF f

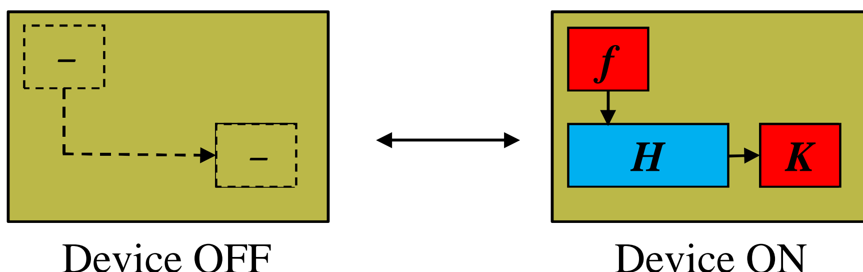
- PUF *f* high min-entropy
- Reliable reconstruction of *f* possible

Key *K* is derived from "device properties" PUF value read-out *f* upon device start-up

Secure Key Storage – Reliability of PUFs (2)

Alleviated constraints – biased, exact distribution

- Secure key storage ON/OFF button

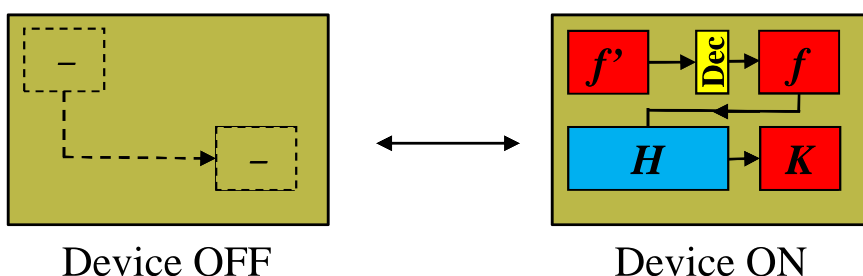


Key *K* is derived from "device properties" upon device start-up

Assumptions on PUF f

- Key *f* high min-entropy
- Reliable reconstruction of *f* possible

Alleviated constraints – biased distribution, allowing errors



Assumptions on PUF f'

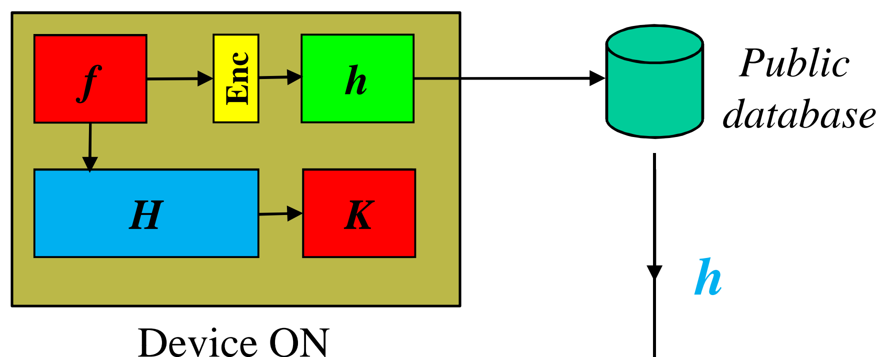
- Key *f* high min-entropy
- Reliable reconstruction of "baseline" *f* from read-out *f'*

Key *K* is derived from "device properties" PUF value read-out *f* upon device start-up

Secure Key Storage – Reliability of PUFs (3)

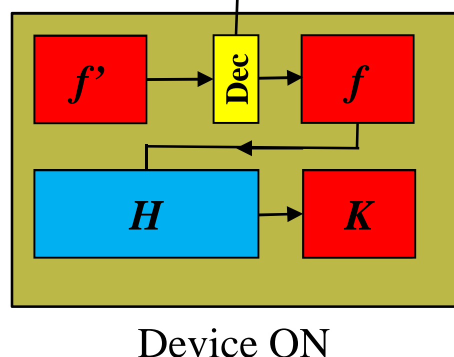
Alleviated constraints – biased distribution, allowing errors (details)

- Device enrollment



Device ON

- Subsequent read-out



Device ON

Properties

- Creates key $K:=H(f)$
- Creates helper data $h:=f+c$ that can be public

(c is random word of t-error-correcting code)

Properties

- Decodes f' to f from $f'+h=(f'+f)+c$, if $f'\approx f$ (less than t errors)
- Creates key $K:=H(f)$

Note: Helper data *does* leak info on PUF value *f*, but *not* (!) where it matters

Secure Key Storage – Reliability of PUFs (4)

Reliability depends on error sources and error-control codes used

Sources of errors with PUF value read-outs

- Chip process technology, PUF details
- Temperature
- Voltage
- Time ("aging")

Typical relative errors with PUFs ≈ 15% or less
Extensive tests with SRAM:

- Cypress 65nm/150nm Virage 90nm/130nm Faraday 130nm

Error-correcting code optimization trade-offs:

- Rate focus: allows use of smaller PUF value
- Complexity focus: use small footprint (repetition, Golay, Reed-Muller codes)

Typical PUF size (with SRAM) ≈ 0.5 kBytes feasible (for 128-bit secret keys)

Secure Key Storage – Randomness of PUFs (1)

Randomness Requirement

- PUF read-out f should have high min-entropy (then, $K:=H(f)$ makes this key random if key extractor H properly picked)
- PUF read-out between different devices should be unpredictable (≈ 50%)

PUF example: SRAM Memory Cell (6T) – on next slides

Typical min-entropy with SRAM-based PUFs ≈ 75% or more
(So, 171 SRAM cells have 128-bits of entropy)

- Dependency on chip process technology
- Smaller technologies tend to have higher min-entropy (≈ 88%)

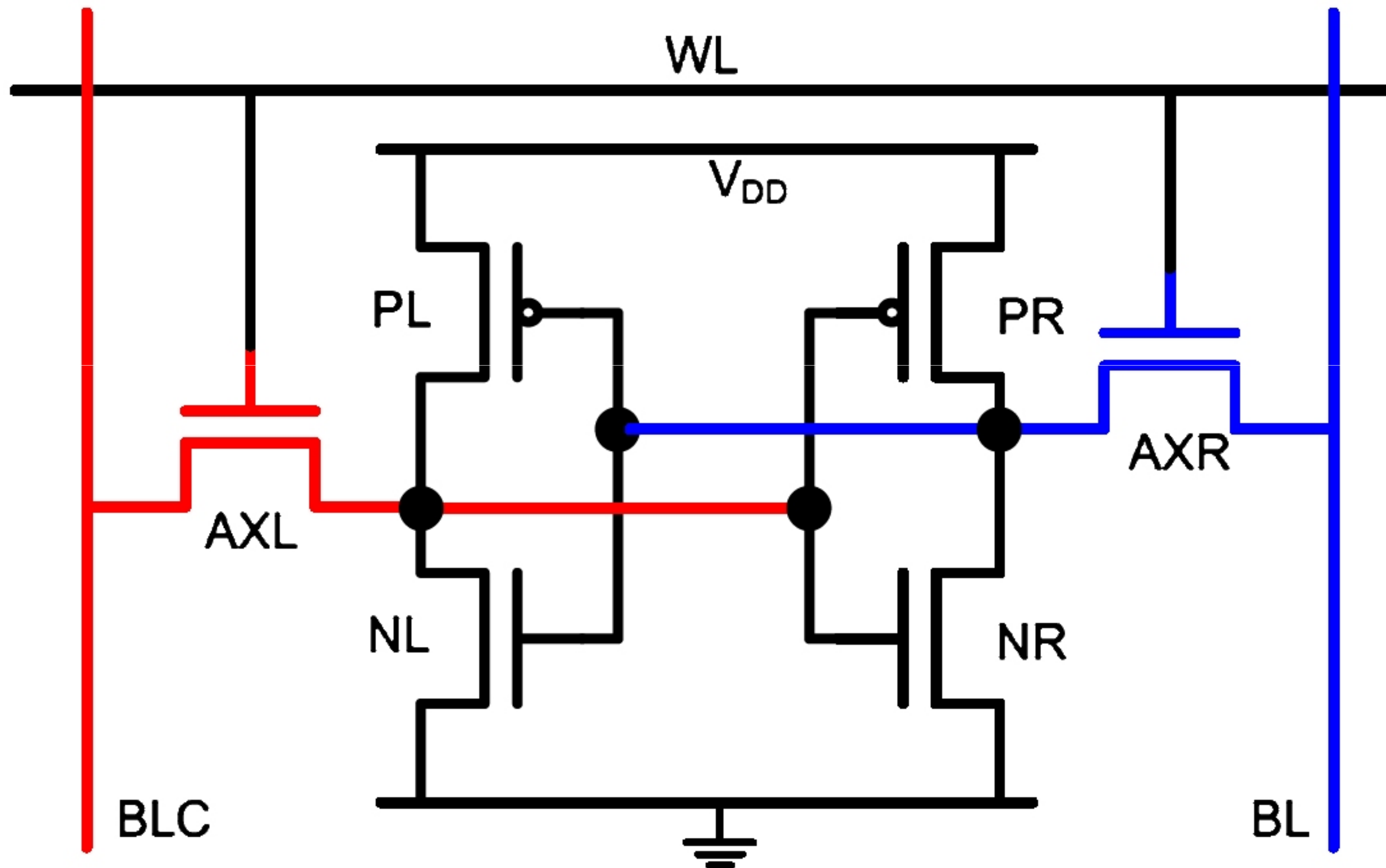
More PUF examples: D-Flip Flop, Buskeeper logic

Examples above are all *standard* semiconductor components

- Easy to integrate (no need to introduce new components)
- Easy to evaluate (digital read-out)

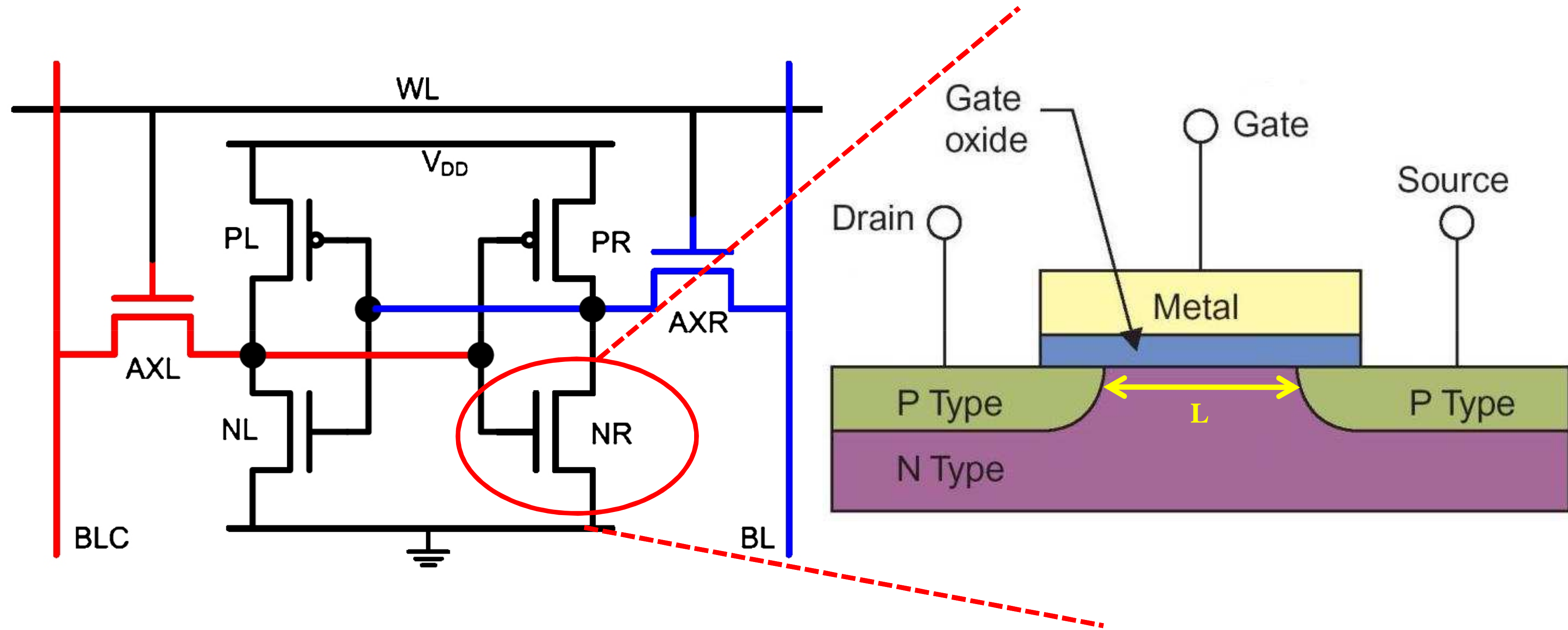
Secure Key Storage – Randomness of PUFs (2)

PUF example: SRAM Memory Cell (6T)



Secure Key Storage – Randomness of PUFs (3)

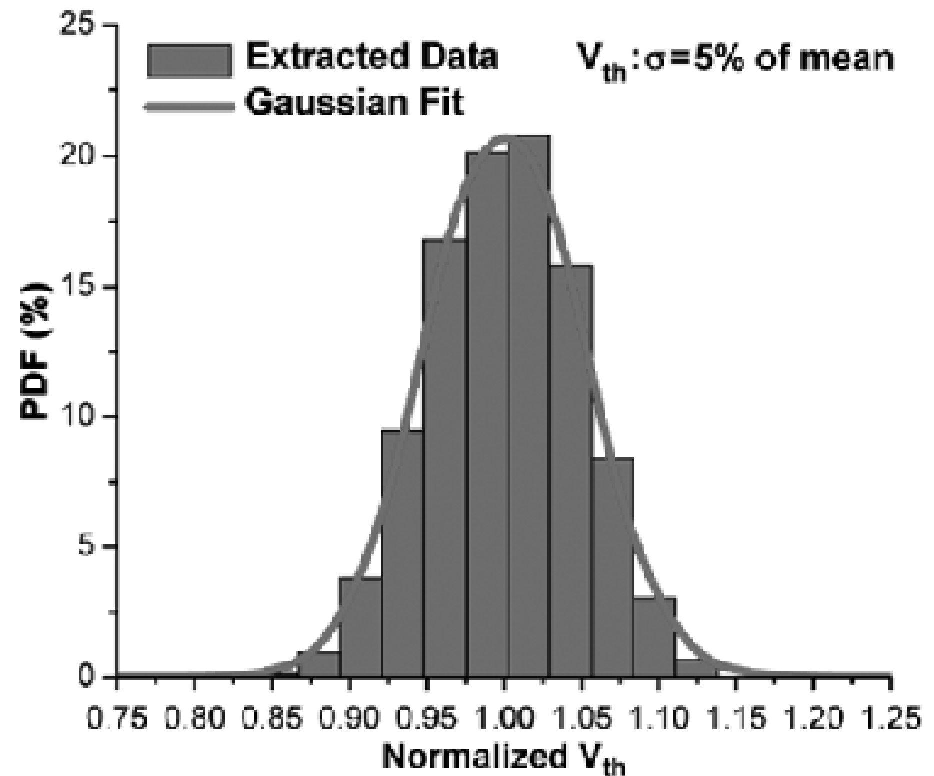
PUF example: SRAM Memory Cell (6T)



Secure Key Storage – Randomness of PUFs (4)

PUF example: SRAM Memory Cell (6T)

SRAM Start-up Behavior
- Dominant factor is Threshold voltage (Simulation results: 2x more dominant than other parameters[1])

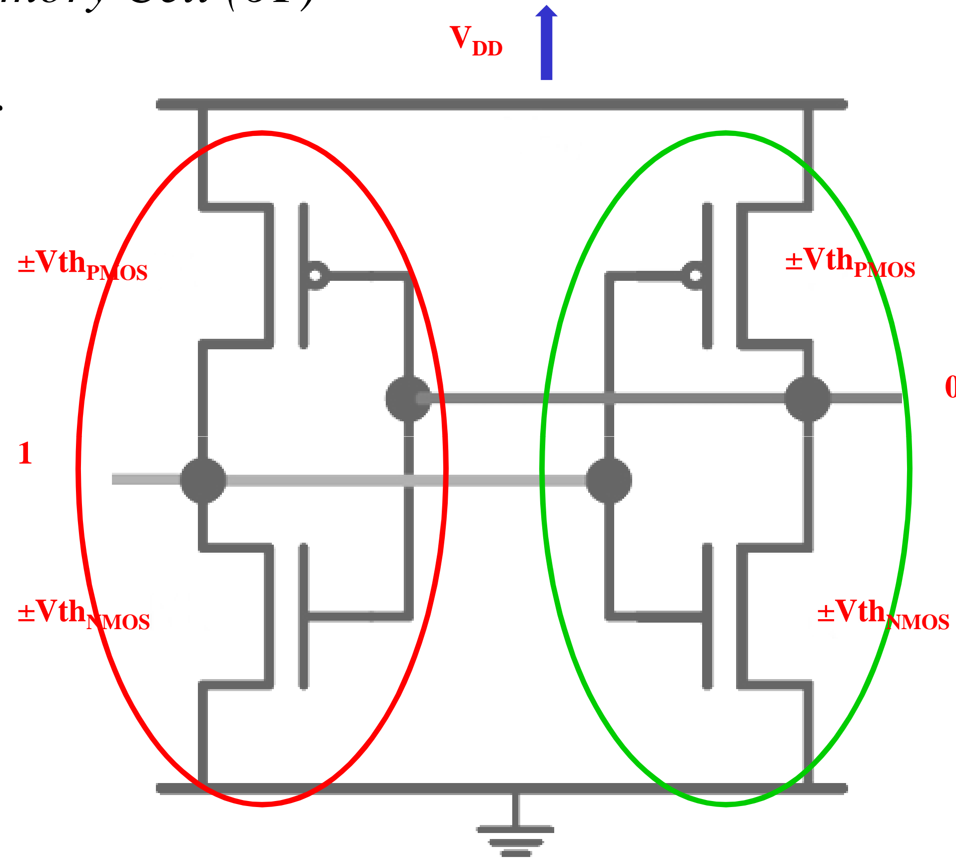


Source: A. Dargar, "Modeling SRAM Start-up Characteristics For Physical Unclonable Functions," Delft University of Technology, MSc. Thesis, 2011.

Secure Key Storage – Randomness of PUFs (5)

PUF example: SRAM Memory Cell (6T)

SRAM Start-up Behavior

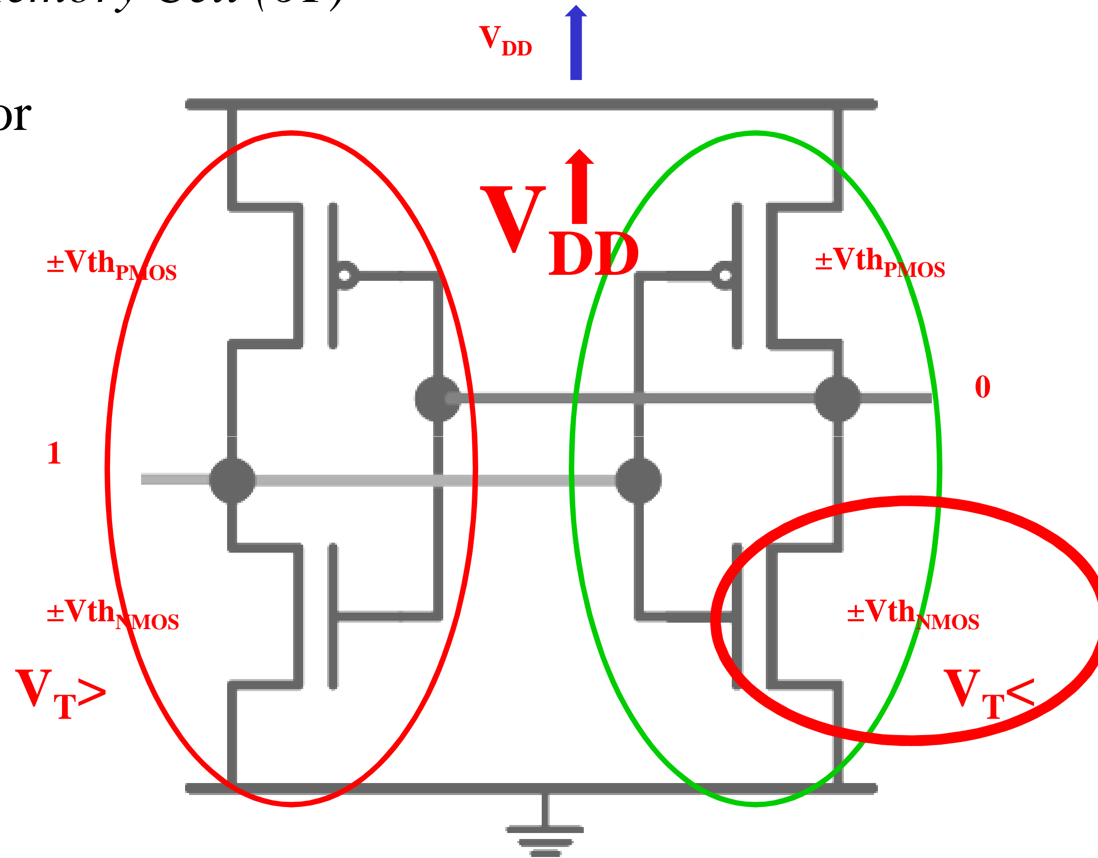


Strongest inverter determines start-up preference of SRAM cell

Secure Key Storage – Randomness of PUFs (6)

PUF example: SRAM Memory Cell (6T)

SRAM Start-up Behavior



Strongest inverter determines start-up preference of SRAM cell

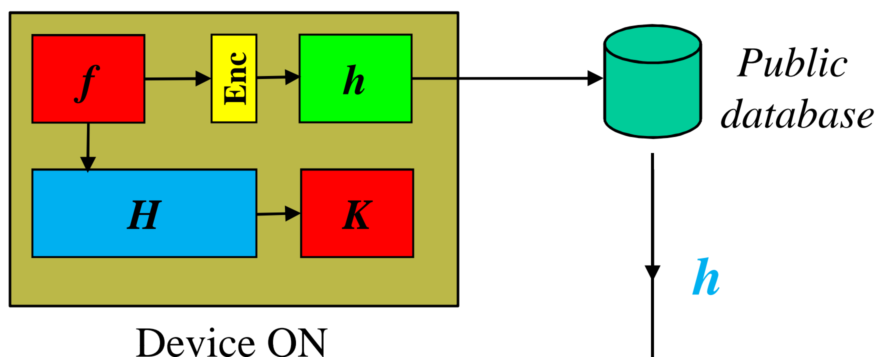
True Random Number Generation via PUFs

- Main Idea
- Randomness Extraction
- Seeds via PUFs

True Random Number Generation – Main Idea

Recap of secure key storage with PUFs

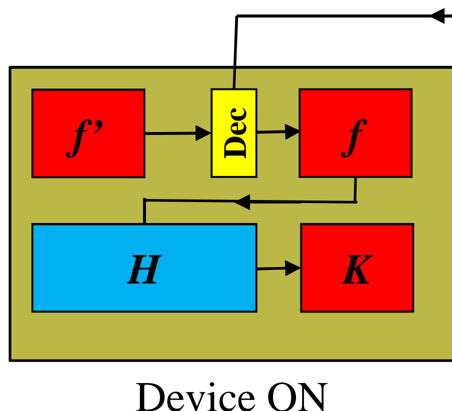
- Device enrollment



h

Device ON

- Subsequent read-out



Device ON

Properties

- Creates key $K:=H(f)$
- Creates helper data $h:=f+c$ that can be public

(c is random word of t-error-correcting code)

Properties

- Decodes f' to f from $f'+h=(f'+f)+c$, if $f'\approx f$ (less than t errors)
- Creates key $K:=H(f)$

Note: Error pattern $e:=f'+f$ is good source of randomness (!)

True Random Number Generation – Randomness Seed

Entropy of error "side channel" depends on error sources

Sources of errors with PUF value read-outs

- Chip process technology, PUF details
- Temperature
- Voltage
- Time ("aging")

Typical min-entropy of errors with PUFs ≈ 2-4% or more
Extensive tests with SRAM:

- Cypress 65nm/150nm

Typical PUF size (with SRAM) ≈ 0.8 kBytes feasible (for 128-bit truly random seeds)

Conclusions and Future Directions

Conclusions

- Use of PUFs for secure key storage attractive (≈ 0.5 kBytes for 128-bit keys)
  - No persistent, secure key storage needed
  - No attacks on key storage in "Device OFF" state possible
  - Helper data may be stored outside device (or added later in lifecycle)
- Potential other uses
  - Anti-cloning
  - True random number generation (≈ 0.8 kBytes for 128-bit true seed) (useful with NIST RNG specifications)

Further Reading

Physically Unclonable Functions:

1. Y. Dodis, R. Ostrovsky, L. Reyzin, A. Smith, “Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data,” International Association for Cryptologic Research, IACR ePrint 2003/235, 2003.
2. J. Guajardo, S.S. Kumar, G-J. Schrijen, P. Tuyls, “FPGA Intrinsic PUFs and Their Use for IP Protection,” in *Proceedings of Cryptographic Hardware and Embedded Systems -- CHES 2007*, P. Paillier, I. Verbauwhede, Eds., Lecture Notes in Computer Science, Vol. 4727, pp. 63-80, 2007.
3. R. Maes, I. Verbauwhede, “Physically Unclonable Functions: a Study on the State of the Art and Future Research Directions,” in *Towards Hardware-Intrinsic Security – Foundations and Practice, Part I*, A-R. Sadeghi, D. Naccache, Eds., pp. 3-37, 2010.
4. V. van der Leest, E. van der Sluis, G-J. Schrijen, P. Tuyls, H. Handschuh, “Efficient Implementation of True Random Number Generator Based on SRAM PUFs,‘” in *Cryptography and Security: From Theory to Applications*, Lecture Notes in Computer Science, Vol. 6805, pp. 300-318, 2012.
5. V. van der Leest, B. Preneel, E. van der Sluis, “Soft Decision Error Correction for Compact Memory-Based PUFs Using a Single Enrollment,” to be presented at *Cryptographic Hardware and Embedded Security – CHES 2012*.
6. R. Maes, V. Rozic, I. Verbauwhede, P. Koeberl, E. van der Sluis, V. van der Leest, “Experimental Evaluation of Physically Unclonable Functions in 65 nm CMOS,” in *Proceedings of 38th European Solid-State Circuits Conference – ESSCIRC 2012*, IEEE, 2012.
7. G-J. Schrijen, V. van der Leest, “Comparative Analysis of SRAM Memories Used as PUF Primitives,” DATE 2012.

Further Reading (cont'd)

NIST Standards and Guidelines related to RNGs and to Secure Key Storage:

8. NIST SP 800-90A, *Recommendation for Random Number Generation Using Deterministic Random Bit Generators (Revised)*, January 25, 2012.
9. NIST SP 800-90B, *Recommendation for the Entropy Sources Used for Random Bit Generation*, Draft, September 6, 2012.
10. NIST SP 800-90C, *Recommendation for Random Bit Generator (RBG) Constructions,* Draft, September 6, 2012.
11. NIST SP 800-22, *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*, May 15, 2012.
12. FIPS 140-2, *Implementation Guidance for FIPS Pub 140-2 and the Cryptographic Module Validation Program*, July 15, 2011.
13. FIPS Pub 140-2, *Annex C: Approved Random Number Generators for FIPS Pub 140-2, Security Requirements for Cryptographic Modules*, Draft, February 16, 2012.
14. NIST SP 800-147B, *BIOS Protection Guidelines for Servers*, Draft, July 31 2012.