Freie Universität Berlin

Why Attackers Lose: Design and Security Analysis of Arbitrarily Large XOR Arbiter PUFs

Nils Wisiol, Christoph Graebnitz, Marian Margraf, Manuel Oswald, Tudor Soroceanu, and Benjamin Zengin

nils.wisiol@fu-berlin.de · http://idm.mi.fu-berlin.de

PROOFS 2017, 29 Sep 2017, Taipei, Taiwan

Short History of PUFs

- Optical implementation proposed by Pappu et al. in 2002
- For all we know, secure
- Hardly practical

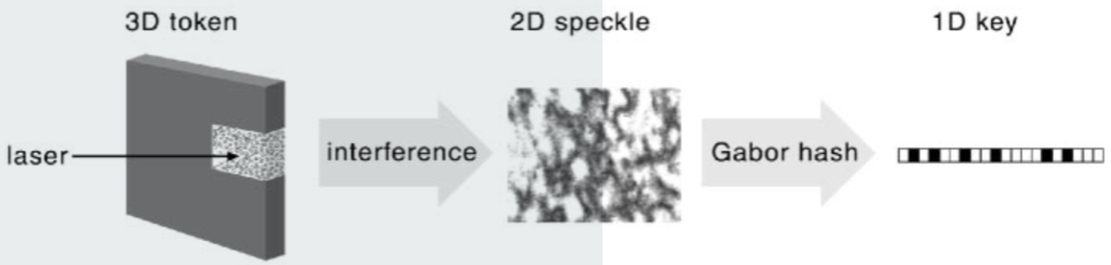
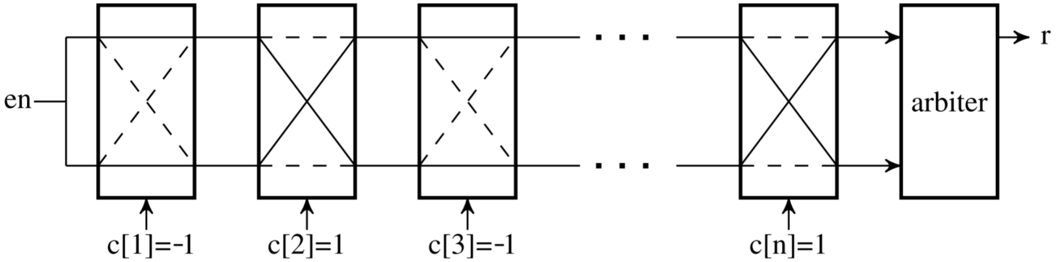


Illustration: Pappu, Ravikanth, et al. "Physical one-way functions." Science 297.5589 (2002): 2026-2030.

Arbiter PUFs



- Easy to build on ASIC
- Response based on signal delays
- Large challenge space
- **Easy to model! ("Linear Model")**

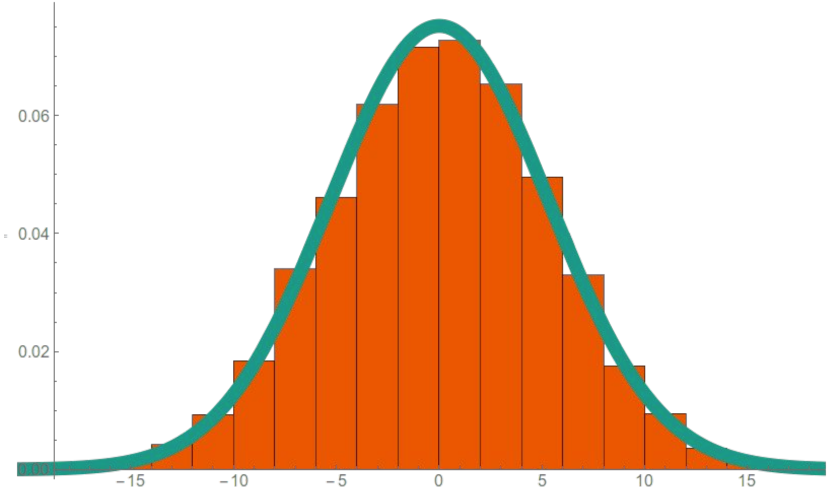
$$\Delta D\left(c\right) = \langle w, x(c) \rangle$$

$$x(c) = (1, c_1 c_2 \cdots c_n, c_2 \cdots c_n, ..., c_n)$$

Illustration (mod.): Tajik, Shahin, et al. "Laser fault attack on physically unclonable functions." Fault Diagnosis and Tolerance in Cryptography (FDTC), 2015 Workshop on. IEEE, 2015.

Arbiter PUFs

- Delay values are close to a Gaussian distribution (Berry-Esseen CLT)
- Simplifies analysis



Delay Value Frequencies of a Simulated 32-bit Arbiter PUF
Fitted Gaussian Distribution
(both shown as probability density)

XOR Arbiter PUFs

- Still easy to build in ASIC
  - But limited in size due to noise
- Response based on signal delays
- Large challenge space
- Harder to model when built large

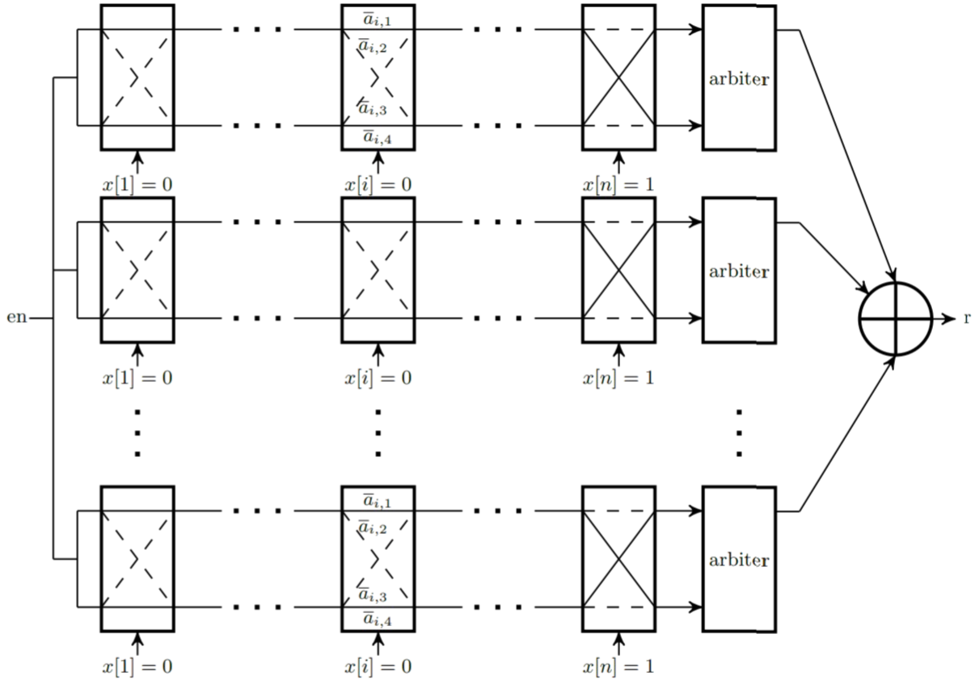


Illustration: Ganji, Fatemeh, et al. "Lattice basis reduction attack against physically unclonable functions." Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. ACM, 2015.

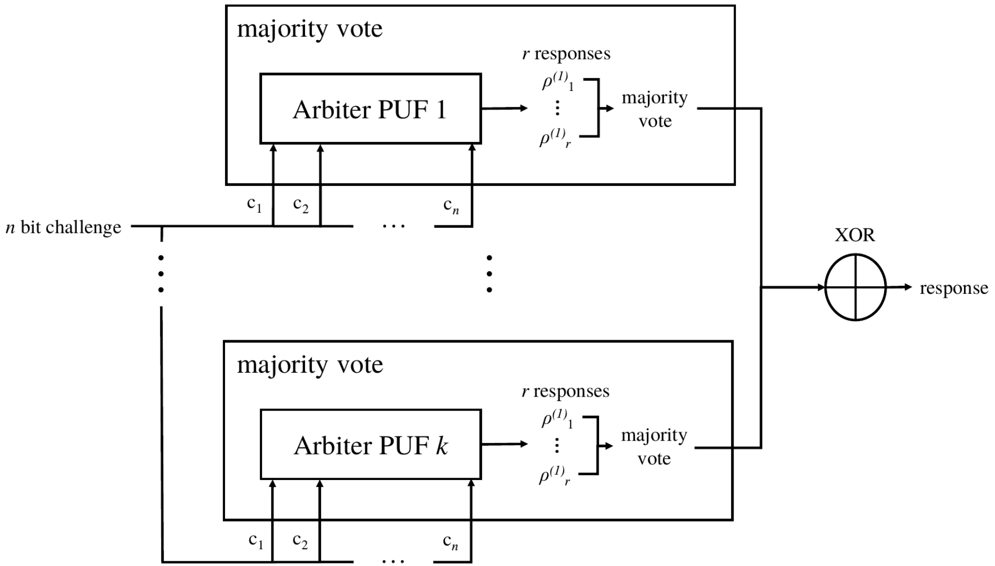
All Feasibly Large XOR Arbiter PUFs Are Insecure

Let's make 'em larger

Becker, Georg T. "The gap between promise and reality: On the insecurity of XOR arbiter PUFs." International Workshop on Cryptographic Hardware and Embedded Systems. Springer, Berlin, Heidelberg, 2015.

Introducing: Majority Vote XOR Arbiter PUF

- Vote before XOR
- Increases stability
- Claim: Size can be increased
- Introduces volatile memory
- Evaluation time prolonged



Stability Gain

Introduced by majority vote

VS

Stability Loss

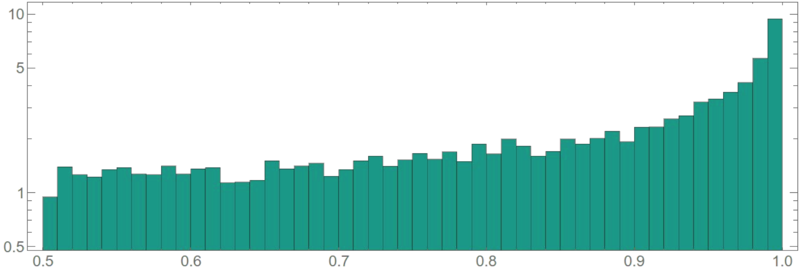
Introduced by huge XOR operation

Notion of Stability

We define: **Stability is the probability to see a noise-free response**

The stability depends on the challenge given

Noise is modelled as Gaussian



Stability Frequencies of a Simulated 64-bit Arbiter PUF
(shown as log-scaled probability density)

Generate histogram data with pypuf: `stability_calculation.py 64 1 1 0.33 10000 200 0xbeef`

Arbiter PUF Noise Analysis

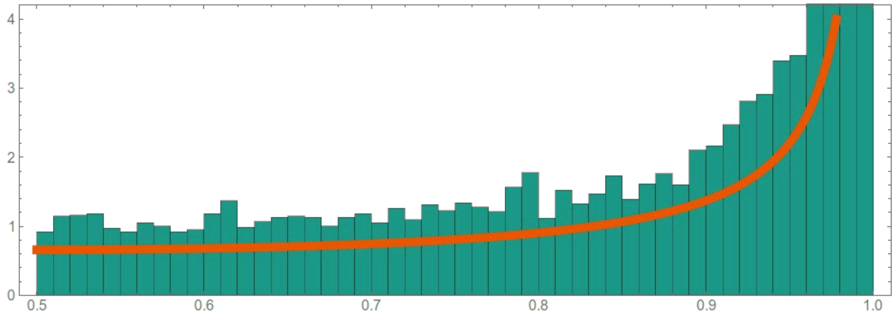
- Fix Arbiter PUF instance and challenge c
- Fix noise parameters
- Analyze stability value for c

$$\begin{aligned}\mathrm{Stab}(c) &= \Pr_{\Delta D_{\mathrm{Noise}}}\left[\mathrm{sgn}\left(\Delta D_{\mathrm{Model}}\left(c\right) + \Delta D_{\mathrm{Noise}}\right) = \mathrm{sgn}\left(\Delta D_{\mathrm{Model}}\right)\right] \\ &= \frac{1}{2} + \frac{1}{2}\,\mathrm{erf}\left(\frac{|\Delta D_{\mathrm{Model}}(c)|}{\sqrt{2}\sigma_{\mathrm{Noise}}}\right)\end{aligned}$$

Arbiter PUF Noise Analysis

Assume Gaussian distributed Stab(c)

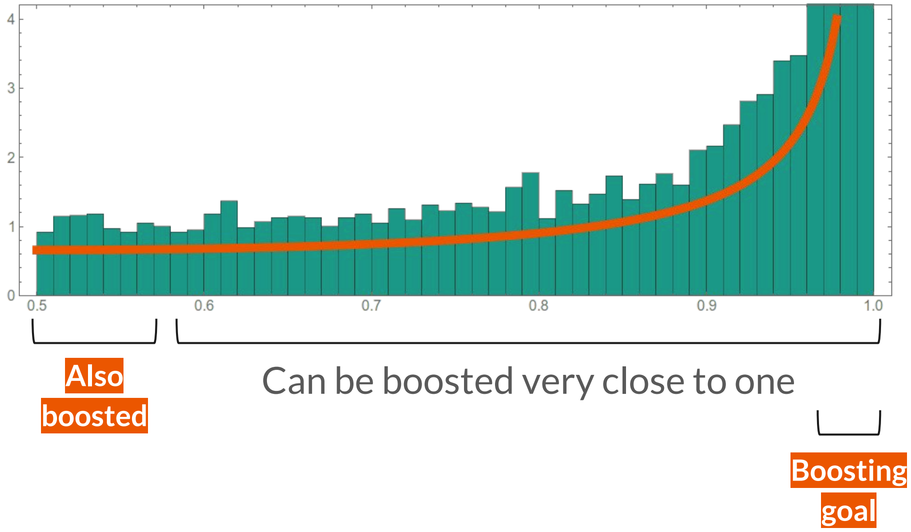
$$\Pr\left[\mathrm{Stab}(c) < z\right] = \mathrm{erf}\left(\frac{\sigma_{\mathrm{Noise}}}{\sigma_{\mathrm{Model}}}\,\mathrm{erf}^{-1}\left(2z-1\right)\right)$$



Stability Frequencies of a Simulated 64-bit Arbiter PUF
Analytic Stability Distribution
(both shown as probability density)

Boosting by Polynomial Majority Vote is Limited

- It's impossible to boost all challenges very close to one
- But it is possible to boost most challenges close to one



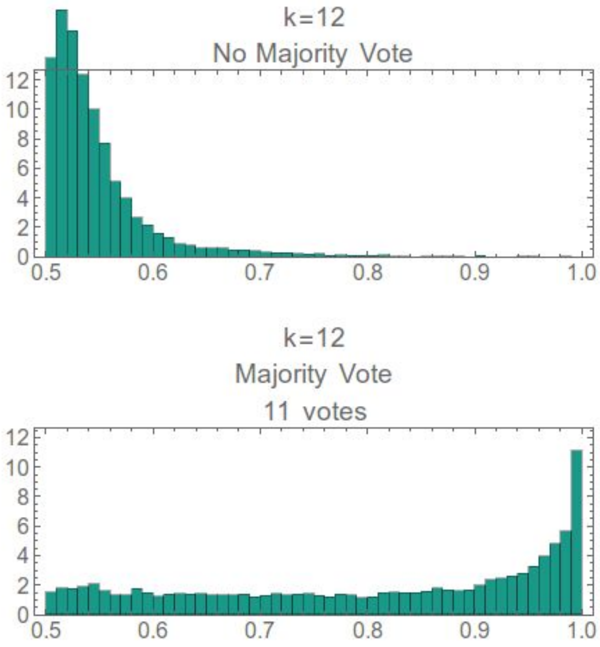
Boosting Result

Assumptions:

- n-bit challenges
- k arbiter chains
- α to select challenges
- α' to set boosting goal

Required votes:

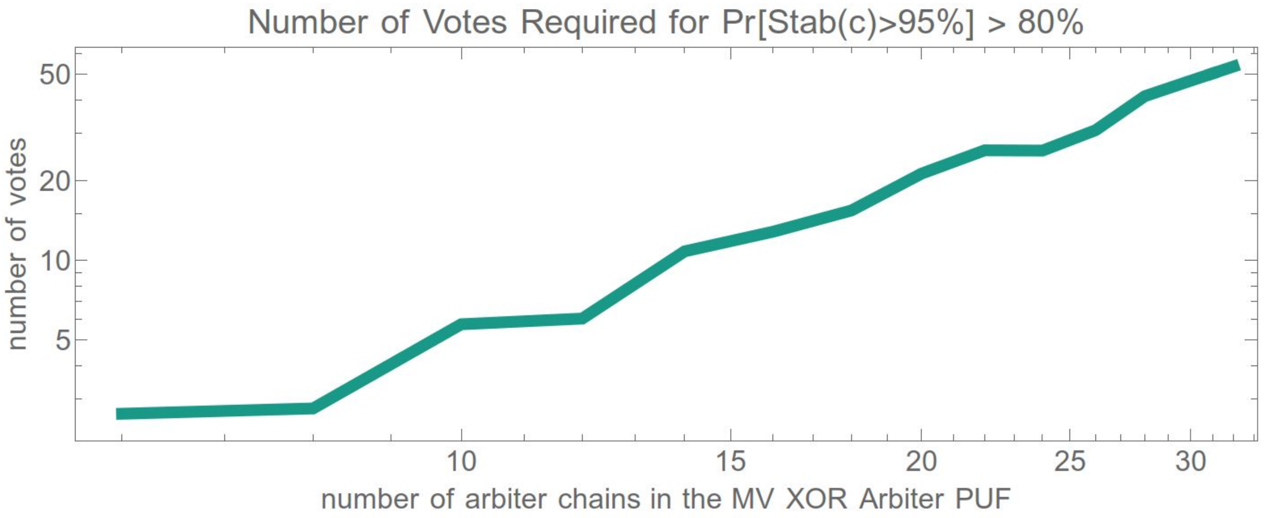
$$r \in O(\alpha^2 \cdot \alpha' \cdot k^2 \cdot \log k)$$



Stability Frequencies of a Simulated 64-bit Arbiter PUF
Using no votes and 12 votes, respectively
(both shown as probability density)

Generate histogram data with pypuf: `stability_calculation.py 64` *k votes* `0.33 10000 200 0xbeef`

Number of Required Votes



Generate histogram data with pypuf: `mv_num_of_votes.py .95 .80 32 32 2 .033 2000 200`

Stability Wins! Attackers Lose?

Logistic Regression

Rührmair, Ulrich, et al. "Modeling attacks on physical unclonable functions." Proceedings of the 17th ACM conference on Computer and communications security. ACM, 2010.

- Parameterized model of the XOR Arbiter PUF
- Regression with logistic function
- Depends on random start values
- Runtime increases exponentially with k

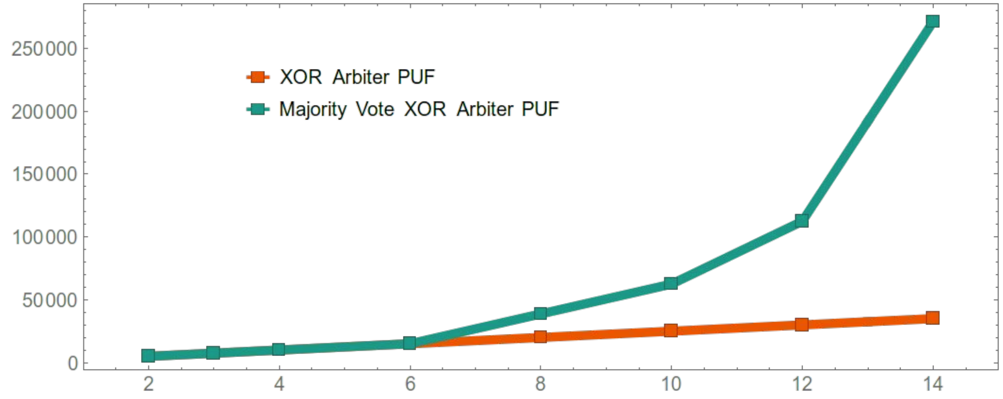
Noise Side-Channel CMA-ES

Becker, Georg T. "The gap between promise and reality: On the insecurity of XOR arbiter PUFs." International Workshop on Cryptographic Hardware and Embedded Systems. Springer, Berlin, Heidelberg, 2015.

- Divide-and-conquer strategy based on a noise side-channel
- Choosing number of votes such that

$$\Pr\left[\mathrm{Stab}(c) \geq 95\%\right] \geq 95\%$$

- Number of required CRPs increases exponentially with k
- **Runtime and required CRPs increases exponentially with k**



Approx. Number of Required CRPs for Successful Attack against increasingly large (Majority Vote) XOR Arbiter PUF

Data generation not yet in pypuf :-(

Take Home Message

- XOR Arbiter PUFs are insecure for all feasible sizes
- Increasing size decreases stability
- Introducing majority vote increases stability
- Stability increase wins with reasonable number of votes
- Mitigate state-of-the-art attacks
- Adding attack surface

Future Work

- CMA-ES attack
- Specialized attacks against Majority Vote XOR Arbiter PUF
- Derivatives of XOR Arbiter PUF
- Avoid low-stability challenges

pypuf

github.com/nils-wisiol/pypuf

- Simulation of PUFs
  - Many flavors of XOR Arbiter PUFs
- Attack on PUFs
  - Logistic Regression
  - CMA-ES (noise side-channel)
  - Some flavors of PAC learning
- Analysis of results

Questions?

Why Attackers Lose: Design and Security Analysis of Arbitrarily Large XOR Arbiter PUFs

Nils Wisiol, Christoph Graebnitz, Marian Margraf, Manuel Oswald, Tudor Soroceanu, and Benjamin Zengin

nils.wisiol@fu-berlin.de
idm.mi.fu-berlin.de

github.com/nils-wisiol/pypuf