Sponge-based pseudorandom number generators

Guido Bertoni[1] Joan Daemen[1]
Michaël Peeters[2] Gilles Van Assche[1]

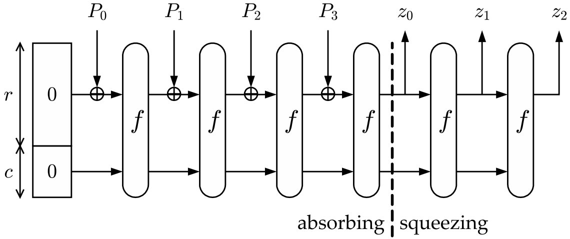
[1]STMicroelectronics

[2]NXP Semiconductors

CHES, Santa Barbara, CA
August 17-20, 2010

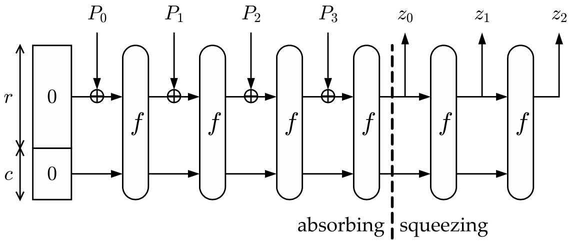
Outline

The sponge construction



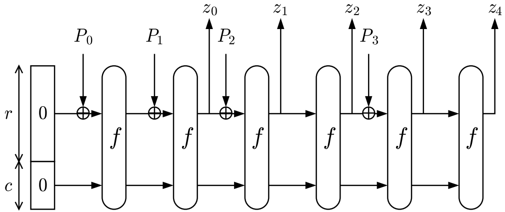
- f: a b-bit permutation with $b = r + c$
- Operating mode:
  - **One** absorbing phase
  - **One** squeezing phase

The sponge construction: security



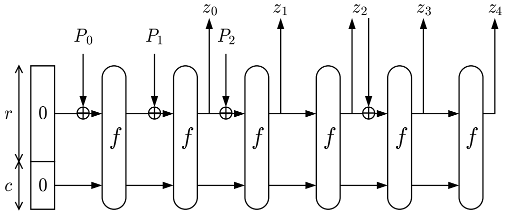
- Indifferentiability proof [Bertoni et al., Eurocrypt 2008]
  - Provably secure against attacks with $< 2^{c/2}$ calls to f
  - Proof assumes f is random permutation
- $\Rightarrow$ Sponge secure if f has no exploitable properties

Sponge-based PRNG: the idea



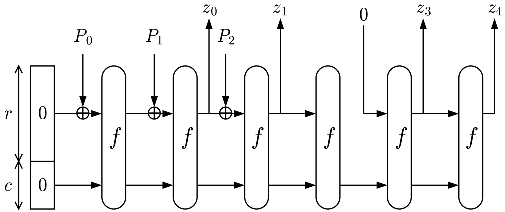
- **Feed** seeding (and reseeding) material P_i
- **Fetch** pseudo-random strings z_i
- Features:
  - f invertible $\Rightarrow$ no entropy loss
  - Forward secrecy: chop state by feeding back z_i

Sponge-based PRNG: the idea



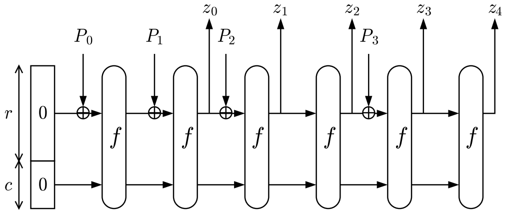
- **Feed** seeding (and reseeding) material P_i
- **Fetch** pseudo-random strings z_i
- Features:
  - f invertible $\Rightarrow$ no entropy loss
  - Forward secrecy: chop state by feeding back z_i

Sponge-based PRNG: the idea



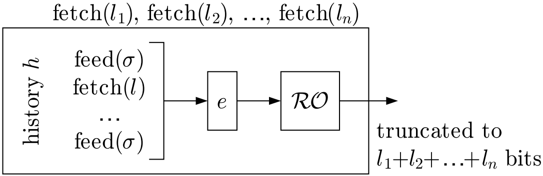
- **Feed** seeding (and reseeding) material P_i
- **Fetch** pseudo-random strings z_i
- Features:
  - f invertible $\Rightarrow$ no entropy loss
  - Forward secrecy: chop state by feeding back z_i

Sponge-based PRNG: security



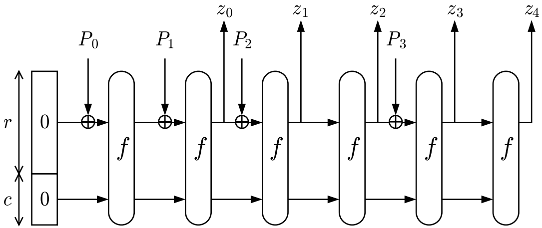
- Multiple absorbing and squeezing phases...?!?
  - Is it secure?
  - What would be the model?

Our reference model for PRNG



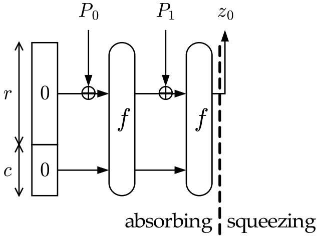
- Using a public **random oracle**
- Input: sequence of *feed* and *fetch* requests
- Output of a *fetch* request
  - must depend on all seed material σ_i thus far
  - may depend on the *fetch* requests
  - $\mathcal{RO}(e(\text{history}))$, where e maps to $\mathbb{Z}_2^*$

Sponge-based PRNG revisited



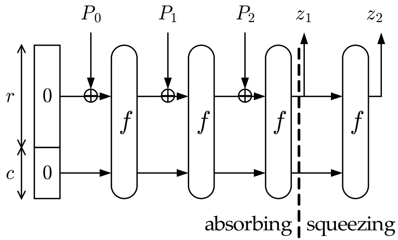
- Can be modeled as multiple calls to a sponge function

Sponge-based PRNG: first call



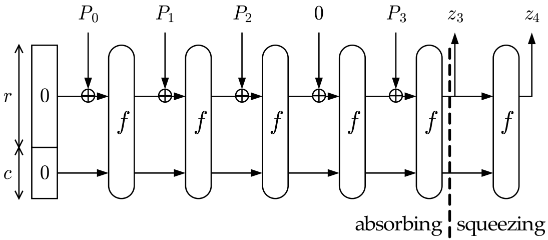
- z_0 output of first call

Sponge-based PRNG: second call



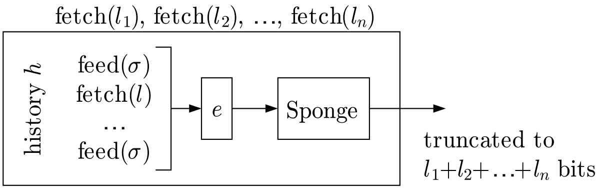
- $z_1 z_2$ output of second call

Sponge-based PRNG: third call



- $z_3 z_4$ output of third call

Sponge-based PRNG: equivalent representation



- Sponge function takes the place of the random oracle
- Indifferentiability $\rightarrow$ secure if f has no exploitable properties

Recent extension: the duplex mode

- Duplex construction [Bertoni et al., Duplexing the sponge, ..., SHA-3 workshop 2010]
  - Sibling to sponge construction, with equivalent security
  - Object with input and output in each call
- Applications include
  - Reseedable PRNG
  - Single-pass authenticated encryption
  - Overwrite mode

Building lightweight implementations

- Width of permutation f: $b = r + c$
- Trade-off between security and efficiency:
  - Security level: $c/2$ bits
  - Efficiency: r pseudorandom/seed bits per call to f
- Optimum trade-off depends on the usage scenarios
- Example 1: QUARK [Aumasson et al., QUARK, ..., CHES 2010]
- Example 2: KECCAK supports : $b \in \{25, 50, 100 \ldots 1600\}$
  - Security level 80 bits implies $c = 160$
  - $b = 200$ gives rate $r = 40$
  - Compact in hardware [Bertoni et al., KECCAK main doc. 2.1]

New security bounds for sponge functions

- Resistance against state recovery [This paper]
  - Expected workload against passive adversaries: 2^c
  - Expected workload against active adversaries: $2^c / \text{data}$
- [Bertoni et al., On the security of the keyed sponge construction, SHA-3 workshop 2010]
  - Generalization of results of this paper
  - Indistinguishable from random oracle if $\text{data} \times \text{time} \leq 2^{c-1}$

Building implementations that are even lighter

- Sponge-based PRNG: passive adversary
  - Security level: c bits
  - Efficiency: r pseudorandom/seed bits per call to f
- Example with KECCAK
  - Security level 80 bits implies $c = 80$ rather than $c = 160$
  - $b = 200$ gives rate $r = 120$: speed $\times 3$
  - $b = 100$ gives rate $r = 20$: area divided by 2

...and secure against side channel attacks

[Bertoni et al., Building power analysis resistant implementations of Keccak, SHA-3 workshop 2010]

- Secret sharing for robust protection against DPA
  - Suited for functions of low algebraic degree
  - Keccak round function: degree 2 in GF(2)
- In software: two shares
  - Roughly doubles RAM usage and computation time
- In dedicated hardware: three shares
  - Trade-off between area and throughput
  - area $\times 4$: no loss of throughput
  - area $\times 2$: maximum throughput divided by 8

Conclusions

- Sponge functions are capable of all symmetric crypto operations:
  - Hashing, encryption, MAC, KDF, MGF (previously known)
  - Reseedable PRNG and authenticated encryption (new)
- Permutation can replace block cipher as crypto primitive!
- Sponge functions are suitable for embedded devices
  - Lightweight: QUARK and small-b KECCAK variants
  - Hardened: protection against DPA