

# Sponge Functions

Guido Bertoni, Joan Daemen,  
Michaël Peeters\* and Gilles Van Assche

*\*NXP Semiconductors*

*STMicroelectronics*



# Outline

- Motivation
- Definition
- Properties
- Reference
- Conclusions

# Hash Function

- A hash function maps:
  - Message of variable length
  - To result of fixed length, say  $n$
- Desired properties
  - Collision resistance
  - Pre-image resistance
  - Second pre-image resistance
  - Correlation-freeness
  - Resistance to length-extension attack
  - Chosen target forced prefix pre-image resistance
  - ...

*A good hash function should behave like a random oracle...*

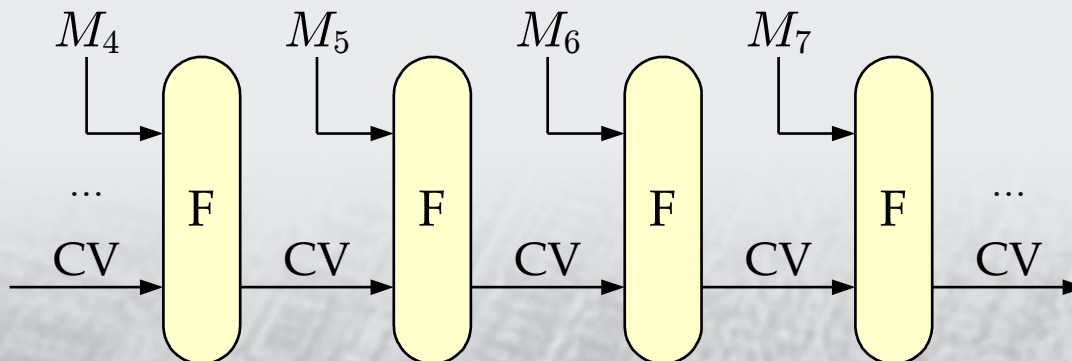


# Random Oracle

- A random oracle (RO) maps: [Bellare-Rogaway 1993]
  - Message of variable length
  - To an infinite output string
- Every input generates a random string
  - Independent and identically distributed bits
  - Two identical inputs have the same output sequence
- Truncated to  $n$  bits, RO has all desired properties of an  $n$ -bit hash function, e.g., effort of
  - Generating a collision:  $2^{n/2}$
  - Finding a (second) pre-image:  $2^n$
  - Finding a chosen-target prefix pre-image:  $2^n$

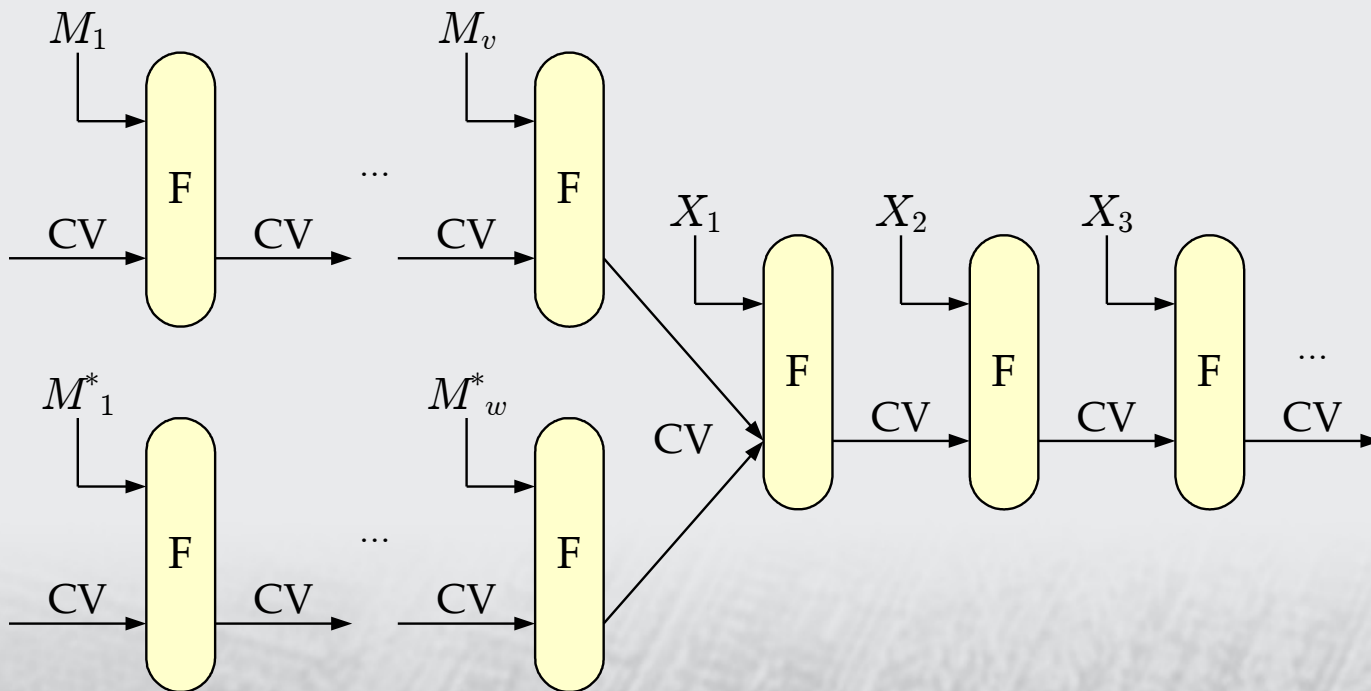
# Iterated Hash Functions

- All practical hash functions are iterated
  - Uses a fixed-length compression function that
  - transforms a  $q$ -bit chaining value while
  - taking a message block  $M_i$
- Output is function of final chaining value



# Internal Collisions!

- Different inputs  $M$  and  $M^*$  giving the same chaining value
- Messages  $M \parallel X$  and  $M^* \parallel X$  always collide for any string  $X$



## Internal Collisions! (cont'd)

- Random oracle has no internal collisions
  - If truncated to  $n$  bits, it does have collisions, say  $M, M^*$
  - But  $M \parallel X$  and  $M^* \parallel X$  collide only with probability  $2^{-n}$

*So the claim to be as strong as a (truncated) random oracle cannot hold for iterated hash functions!*

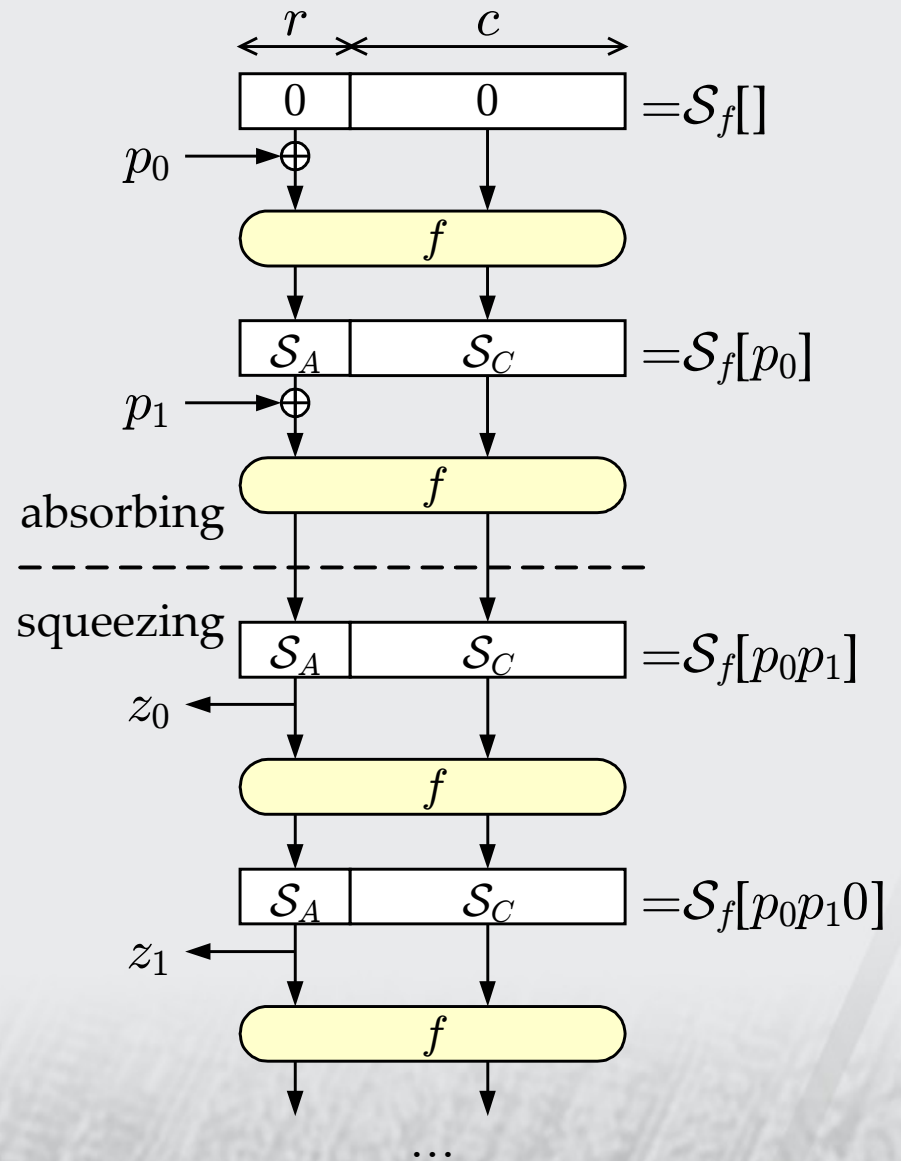
# Motivation for Sponge Functions

- Abandon *iterated* to meet the random oracle ideal
  - In-memory hashing, non-streamable hash functions
  - Example: Zipper Hash [Liskov 2006]
- Learn to live with internal collisions:
  - Stick to the iterated hash function construction
  - Replace the random oracle by **another reference model**
    - To model the impact of **finite memory**
    - Preferably simple to describe, understand and analyze
    - Preferably as close as possible to a random oracle

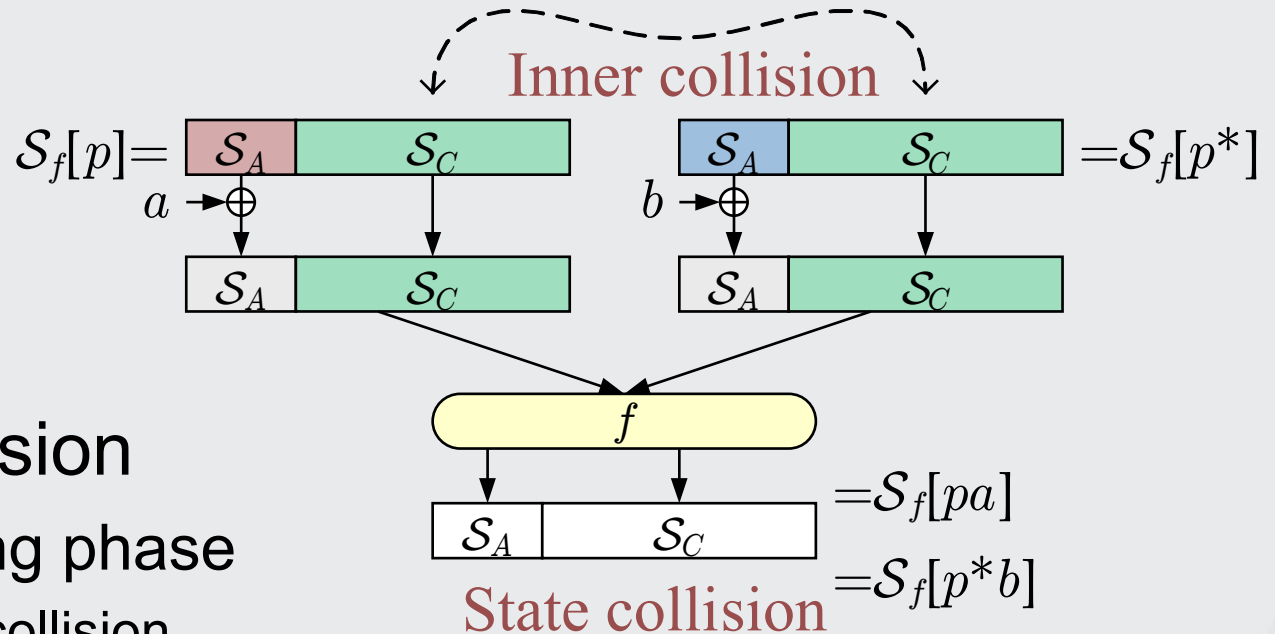
# Sponge Functions



$$z_i = \mathcal{S}_{A,f}[p0^i]$$



# State Collisions, Inner Collisions



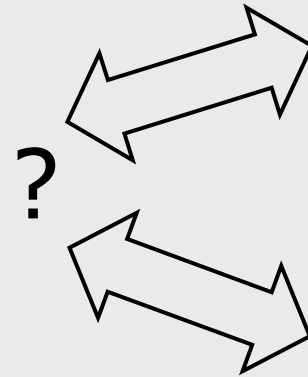
- State collision
  - Absorbing phase
    - Hash collision
  - Squeezing phase
    - Output periodicity

# Random Sponges

- Random **T**-sponge
  - Randomly chosen in  $(2^{c+r})^{2^{c+r}}$  **transformations**  $f$
- Random **P**-sponge
  - Randomly chosen in  $(2^{c+r})!$  **permutations**  $f$

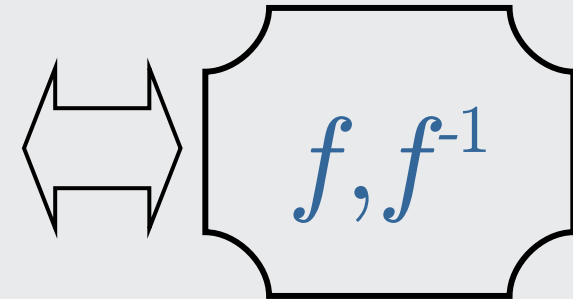
# Distinguishing Random Sponges

- Adversary queries a black box, either RS or RO
  - Budget of  $N$  input and output characters
- **Theorem:** A random sponge can only be distinguished from a random oracle by the presence of **inner collisions**.
  - When  $N \ll 2^{c/2}$ , inner collisions are unlikely



# Attacking Random Sponges

- White box access to  $f$  (and  $f^{-1}$ )
  - Budget of  $N$  function calls
- Expected workload for operations:
  - Producing inner collision
  - Finding a path to inner state
  - Producing output cycles
  - Binding an output string to a state



*The expected workload is a function of: capacity  $c$ ,  $P/T$ -sponge, rate  $r$ .*

# Attacking Random Sponges (cont'd)

- Expected workload for attacks
  - Hash length  $n$
  - Capacity  $c$

|                  | Random<br>T-sponge    | Random<br>P-sponge             |
|------------------|-----------------------|--------------------------------|
| Output collision | $2^{(\min(n,c)+3)/2}$ | $2^{(\min(n,c)+3)/2}$          |
| Pre-image        | $2^n$                 | $\min(2^{n-r} + 2^{c/2}, 2^n)$ |
| Second pre-image | $\min(2^c/l, 2^n)$    | $\min(2^{(c+3)/2}, 2^n)$       |

# Random Sponges as a Reference

- For security claims of hash function designs
  - Parameters
    - Capacity  $c$
    - T/P-sponge
    - Rate  $r$
    - Input/output size limitations
  - External interface (input / output) only
  - Automatically determines the expected workload of attacks
    - E.g., near-collisions, ...
- Flat sponge claim (capacity  $c$ )
  - Expected workload  $\min(\text{Random Oracle}, 2^{c/2})$

# Hashing and Beyond

- Keyed hash modes
  - Prefix the message with the key
- Infinite output
  - Stream ciphers
  - Mask generating functions

# Conclusion

- Random Sponges for security claims

Any  
questions?

