

Cryptanalysis

Part IV: Sponge Modes

André Schrottenloher

Inria Rennes
Team CAPSULE

Inria



1 Recap

2 Sponge-based Hashing

3 Duplex Sponges

4 Cryptanalysis of Permutations

Recap

Recap

In symmetric cryptography we have two categories of objects:

- **Primitives**: small, fixed-size objects (block ciphers, compression functions, etc.)
 - **Modes**: use the primitives to create true cryptographic functionality: (authenticated) encryption, hashing, etc.
- ⇒ **With actual security goals** like confidentiality and authenticity

Modes of operation have **security proofs** which reduce the security to the one of the primitive.

- So we can focus attacks on the primitives
- We should still use the modes with caution (remember Merkle-Damgård)

Recap: AE

Authenticated encryption:

$$\begin{cases} E : \{0,1\}^k \times \{0,1\}^* \times \underbrace{N}_{\text{Nonce / IV}} \rightarrow \{0,1\}^* \\ D : \{0,1\}^k \times \{0,1\}^* \times N \rightarrow \{0,1\}^* \cup \{\perp\} \end{cases}$$

$\perp \iff$ ciphertext rejected as **invalid**.

AE security

IND-CPA + **ciphertext integrity**: adversary cannot create a new ciphertext that decrypts correctly.

AE security \implies CCA security

“Real” vs. “ideal” model

A proof of security in the “real” model runs as follows:

- Let E be a block cipher;
- Assume that E is a SPRP;
- Then in the security game . . . any adversary making . . . queries has advantage . . .

A proofs of security in the “ideal” model runs as follows:

- Choose a function H uniformly at random;
- Then in the security game . . . any adversary making . . . queries has advantage . . .

⇒ In practice, we will instantiate H with a “real” function.

- **Ideal cipher** model: pick E u.a.r. from all block ciphers;
- **Random oracle** model: pick H u.a.r. from all functions $\{0, 1\}^n \rightarrow \{0, 1\}^n$
- **Random permutation** model: pick Π u.a.r. from all permutations $\{0, 1\}^n \rightarrow \{0, 1\}^n$

The Sponge strategy

Hashing & AE modes which are:

- proven secure in the **random permutation** model;
- instantiated with a fixed **cryptographic permutation**.

Strong security proofs, **no key schedule** / key management unlike a block cipher-based mode.

Sponge-based Hashing

One-round Sponge

Let $P : \{0,1\}^n \rightarrow \{0,1\}^n$ be a cryptographic permutation. We divide $n = r + c$.

- $r = \text{rate}$. Determines the speed of the sponge function.
- $r = \text{capacity}$. Determines the security of the sponge function.

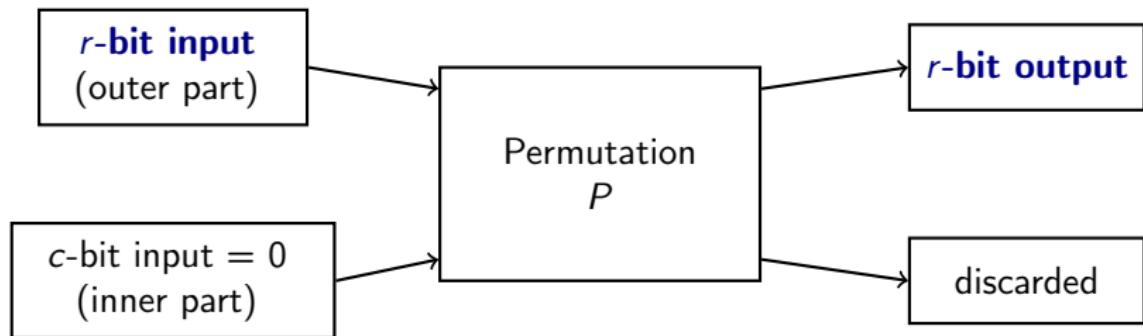
Exercise: Use P to design a random function $\{0,1\}^r \rightarrow \{0,1\}^r$.

One-round Sponge

Let $P : \{0, 1\}^n \rightarrow \{0, 1\}^n$ be a cryptographic permutation. We divide $n = r + c$.

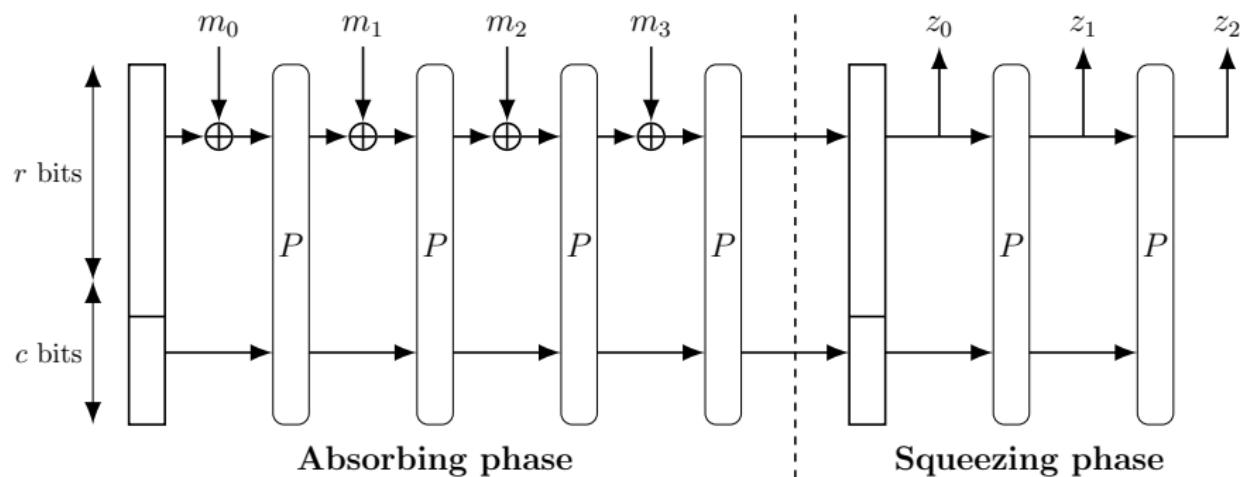
- r = **rate**. Determines the speed of the sponge function.
- r = **capacity**. Determines the security of the sponge function.

Exercise: Use P to design a random function $\{0, 1\}^r \rightarrow \{0, 1\}^r$.



The Sponge: hash function

- Absorb any size of message by splitting into multiple r -bit blocks;
- Output any size of digest by outputting multiple r -bit blocks.



Many variants of sponge exist, this is merely a canonical one.

Security of the Sponge

Collision security

If the output size d is $\leq c$, the security is $d/2$ bits.
If the output size d is $\geq c$, the security is $c/2$ bits.

Security of the Sponge

Collision security

If the output size d is $\leq c$, the security is $d/2$ bits.

If the output size d is $\geq c$, the security is $c/2$ bits.

Attack:

- Find two sequences of messages m_0, \dots, m_ℓ and $m'_0, \dots, m'_{\ell'}$ such that the inner part (c) collides.
- Let r_m, r'_m be the corresponding outer parts (r).
- Then the messages $m_0, \dots, m_\ell, 0$ and $m'_0, \dots, m'_{\ell'}, (r_m \oplus r'_m)$ have a full-state collision \implies all output blocks collide.

Security of the Sponge (ctd.)

Second preimage security

If the output size d is $\leq c/2$, the security is d bits.

If the output size d is $\geq c/2$, the security is $c/2$ bits.

Security of the Sponge (ctd.)

Second preimage security

If the output size d is $\leq c/2$, the security is d bits.

If the output size d is $\geq c/2$, the security is $c/2$ bits.

Attack:

- Determine the n -bit state Z just before the squeezing phase;
- Find two sequences of blocks (going forward from $0||0$ and backward from Z) which collide in the middle.
⇒ $2^{c/2}$ for finding a collision on the inner part, and complete the outer part with another message block.

Security of the Sponge (ctd.)

Preimage security

In general preimage security is higher than second-preimage:

$$\min \left(\max(2^{d-r}, 2^{c/2}), 2^d \right)$$

Security of the Sponge (ctd.)

Preimage security

In general preimage security is higher than second-preimage:

$$\min \left(\max(2^{d-r}, 2^{c/2}), 2^d \right)$$

Attack:

- Exactly like the second preimage, except that finding the state Z before the squeezing phase will cost 2^{d-r} (instead of nothing).
- Why? Because we can match r bits (one rate value) of the image for free, but we have to match the other $d - r$ bits by randomly guessing.

Most often, sponge-based functions (such as SHA-3) will claim $c/2$ security for collision, second preimage and preimage altogether.

 "Tight Preimage Resistance of the Sponge Construction", Lefevre & Mennink, CRYPTO 2022

Duplex Sponges

Context

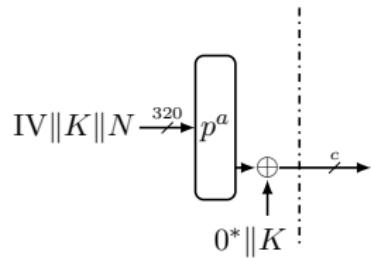
- The Sponge construction has become very popular for hash functions since the 2010s;
- Keccak, winner of the SHA-3 NIST competition, is sponge-based.
- More recently, **duplex modes** have become popular for permutation-based AEAD schemes.
- A majority of the candidates to the **NIST lightweight** competition were duplex-based.
- **Ascon**, new NIST standard for lightweight crypto, is sponge- and duplex-based.



<https://csrc.nist.gov/csrc/media/Presentations/2023/the-ascon-family/images-media/june-21-mendel-the-ascon-family.pdf>

The duplex sponge (Ascon version)

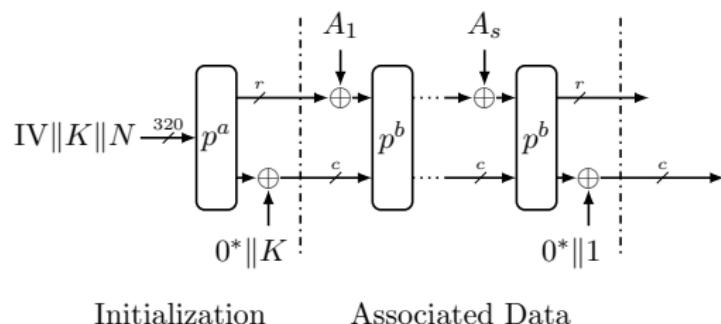
- **Initialization:** load key, IV (constant) and nonce in the internal state.



Initialization

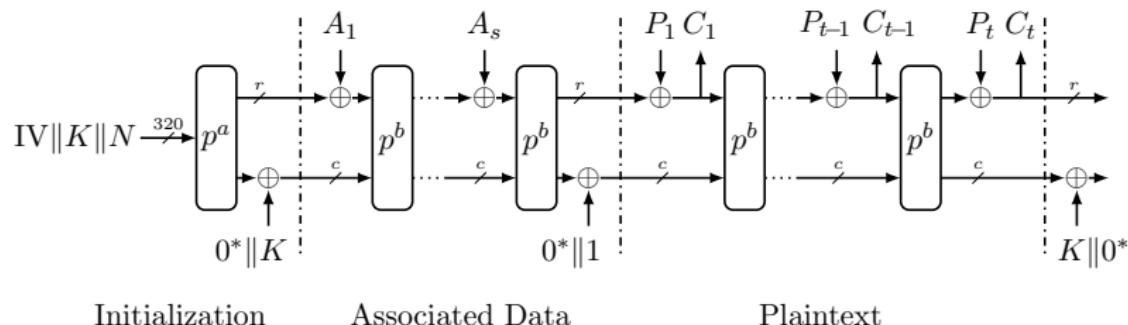
The duplex sponge (Ascon version)

- **Initialization:** load key, IV (constant) and nonce in the internal state.
- **AD processing:** load the AD blocks similarly as in sponge hashing.
(We do not need to recover the AD when decrypting)



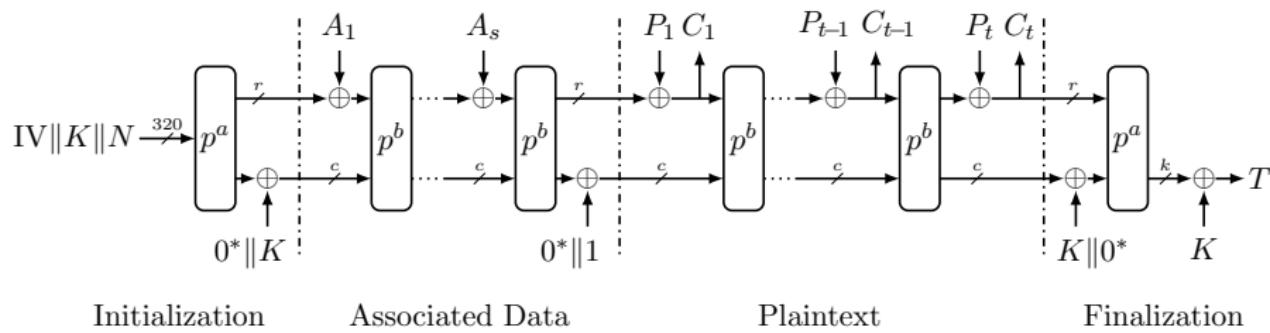
The duplex sponge (Ascon version)

- **Initialization:** load key, IV (constant) and nonce in the internal state.
- **AD processing:** load the AD blocks similarly as in sponge hashing.
(We do not need to recover the AD when decrypting)
- **Plaintext processing:** load the plaintext blocks **and output ciphertext blocks.**



The duplex sponge (Ascon version)

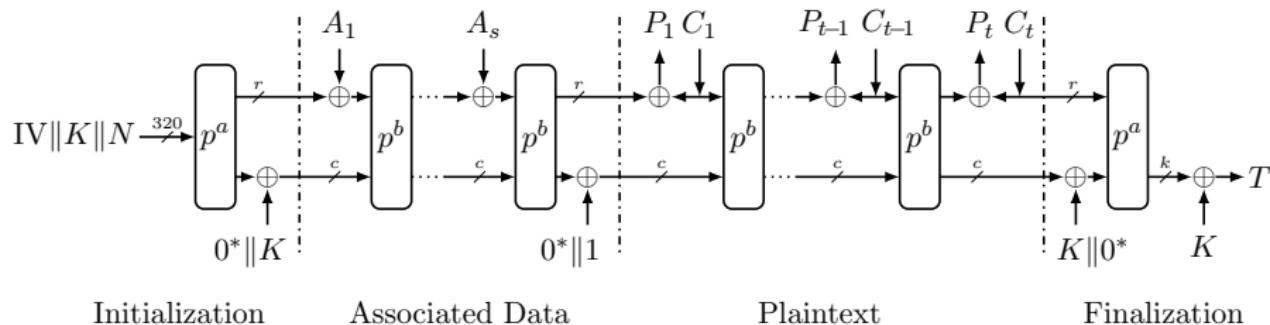
- **Initialization:** load key, IV (constant) and nonce in the internal state.
- **AD processing:** load the AD blocks similarly as in sponge hashing.
(We do not need to recover the AD when decrypting)
- **Plaintext processing:** load the plaintext blocks **and output ciphertext blocks**.
- **Finalization:** output an authentication tag.



Exercise

Do a diagram for the decryption function (two AD blocks, two plaintext blocks).

Ascon: decryption



Parameters of ASCON-AEAD128 (NIST standard)

Security	128
Key	128
Rate	128
Capacity	192
Rounds (a,b)	12, 8
Tag	128



nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-232.pdf

Nonce reuse

The Ascon mode (and Duplex in general) is **nonce-based**: the nonce N **should never be reused**. Why?

 Baudrin, Canteaut, Perrin, "Practical Cube Attack against Nonce-Misused Ascon", ToSC 2022

Nonce reuse

The Ascon mode (and Duplex in general) is **nonce-based**: the nonce N **should never be reused**. Why?

Plaintext-recovery if a nonce is reused between two messages (one known, one unknown).

Dedicated cryptanalysis is often more devastating (full-state recovery for Ascon).

 Baudrin, Canteaut, Perrin, "Practical Cube Attack against Nonce-Misused Ascon", ToSC 2022

Security of the Ascon mode

Like the sponge, the security relies on the fact that **the adversary does not control the inner part**. In the Duplex sponge it will remain secret.

In the single-user setting, with no nonce reuse, the Ascon mode remains AE-secure if the number of encryption / decryption queries Q and the number of permutation queries P satisfy the bounds:

- $P \leq \min(2^k, 2^c)$ (k the key size)
- $Q \leq 2^t$ (t the tag size)
- $PQ \leq 2^{c+r}$



Security of the Ascon mode (ctd.)

Generic (forgery) attacks:

- Make 2^t arbitrary decryption queries with different tags, until success ($Q = 2^t$).
- Make a single encryption query ($Q = 1$). Find the key by exhaustive search (2^k).
- Make a single encryption query ($Q = 1$). Guess the inner part (2^c) somewhere during the encryption phase (= obtain the full state). Compute the full state at the end of the encryption phase. Connect them with a different sequence of messages (time $2^{c/2}$, similar to the sponge attacks). This gives a new message with the same authentication tag.
- Perform many encryption queries. Guess the entire state somewhere during the encryption phase: after $2^{c+r}/Q$ guesses it will match one of the sequences of ciphertext blocks.

Security of Duplex & Sponges in practice

Permutation distinguishers

Find a “non-ideal behavior” of the permutation (algebraic degree bounds, differentials, linear approximations, CICO / structural distinguishers, etc.)

Direct cryptanalysis

Key-recovery / forgery against AEAD, collision / preimage against hash.

- A permutation distinguisher **does not** imply a direct cryptanalysis.
- The security margin may be really different in both cases (ex. in Keccak, or even in Ascon).

Nowadays the permutations used in Sponges tend to be less than “ideal”
... which is why dedicated cryptanalysis matters a lot!

Cryptanalysis of Permutations

Cryptanalysis of a permutation

A permutation does not have a key, so we don't have key-recovery attacks.

Typical

Define a non-trivial “generic” property for inputs and outputs of the permutation (which **does not depend** on the permutation).

- There exists a generic algorithm to find such inputs / outputs for a random permutation;
- Does there exist a faster non-generic algorithm?

Examples:

- Find x such that $P(x) = x$
- Find x such that $P(x) = 0 \implies$ oops, that's never difficult!
- Find x such that $P(x\|0)$ has a large zero-prefix (CICO)
- Find x, y such that $x \oplus y$ has a zero-prefix and $P(x) \oplus P(y)$ also (limited-birthday)

Exercise

The Gimli permutation operates on a 384-bit state divided into 4 words of 96 bits A, B, C, D , and each word into 3 sub-words A_x, A_y, A_z , etc.

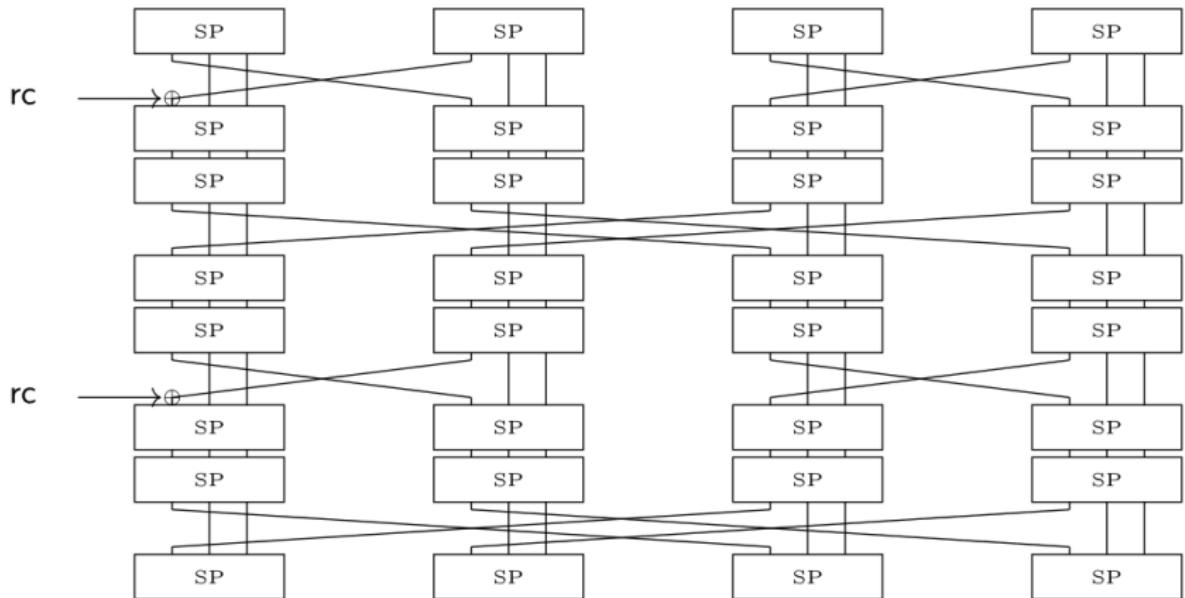
It has a non-linear layer which applies an “SP box” to each word independently, and a linear layer which applies:

- a “small swap” of sub-words x between A, B and C, D ;
- nothing;
- a “big swap” between A, C and B, D ;
- nothing.

There are constant additions in A_x every two rounds. Values of the constants and specifics of the SP-box do not matter.

1. break 2 rounds.
2. break 8 rounds.

Exercise (ctd.)



Solution

There are many ways to “break” this permutation, in a distinguishing sense. We need to come up with a “generic” (not permutation-specific) property over inputs / outputs, and a way to easily satisfy this property.

The easy property in the case of Gimli is an equality between some words. Consider the property: $C = D$.

After 2 rounds this property is preserved, this shouldn't be the case for a random permutation.

After 8 rounds, we just need an equality of 32-bit values to happen twice. The first time this will happen with probability 2^{-32} , the second time with probability 2^{-32} . So by repeating this with random inputs we get an equality in time 2^{64} (less than 2^{96} for a random permutation).

We can do even better by starting from the middle. In the A,B branches we take equal values after the 3 first rounds, so we can ensure that the first equality on the left is always satisfied; then we need the second to be satisfied randomly. Complexity 2^{32} .

Does this break the sponge-based hashing? No. But dedicated attacks in hashing mode reach up to 18 rounds vs. 27 rounds for the permutation.