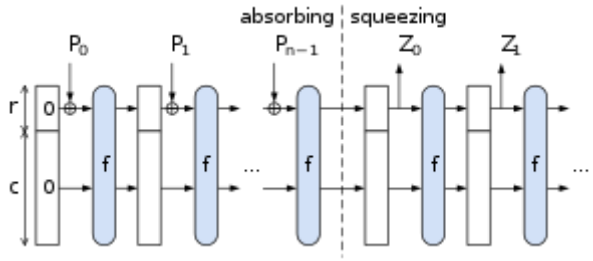
Sponge function

In cryptography, a **sponge function** or **sponge construction** is any of a class of algorithms with finite internal state that take an input bit stream of any length and produce an output bit stream of any desired length. Sponge functions have both theoretical and practical uses. They can be used to model or implement many cryptographic primitives, including cryptographic hashes, message authentication codes, mask generation functions, stream ciphers, pseudo-random number generators, and authenticated encryption.[1]



The sponge construction for hash functions. P_i are blocks of the input string, Z_i are hashed output blocks.

Contents

Construction

A sponge function is built from three components:[2]

- *State*, containing *bits*,
- a function $f : \{0,1\}^{|bits|} \rightarrow \{0,1\}^{|bits|}$ that transforms the state memory (often it is a pseudorandom permutation of the $2^{|bits|}$ state values)
- a padding function *Pad*

The *State* memory is divided into two sections: *Bitrate* and the remaining part the *Capacity*.

Pad appends enough bits to the input string so that the length of the padded input is a whole multiple of $|Bitrate|$. The padded input can thus be broken into $|Bitrate|$-bit blocks.

Operation

The sponge function operates as follows:

- *State* is initialized to zero
- The input string is padded. This means the input is transformed into blocks of $|Bitrate|$ bits using *Pad*.
- for each $|Bitrate|$-bit *Block* of the padded input:

- *Bitrate* is replaced with *Bitrate* XOR *Block* (using bitwise XOR)
- *State* is replaced by *f*(*State*)

This process "absorbs" (in the sponge metaphor) all blocks of the padded input string.

The sponge function output is now ready to be produced ("squeezed out") as follows:

- the *Bitrate* portion of the state memory is output
- repeat until enough bits are output:
  - *State* is replaced by *f*(*State*)
  - the *Bitrate* portion of the state memory is output

If less than $|Bitrate|$ bits remain to be output, then *Bitrate* will be truncated (only part of *Bitrate* will be output).

Another metaphor describes the state memory as an "entropy pool", with input "poured into" the pool, and the transformation function referred to as "stirring the entropy pool".[3]

Note that input bits are never XORed into the *Capacity* portion of the state memory, nor are any bits of *Capacity* ever output directly. The extent to which *Capacity* is altered by the input depends entirely on the transformation function *f*. In hash applications, resistance to collision or preimage attacks depends on *Capacity*, and its size ($|Capacity|$) is typically twice the desired resistance level.

Duplex construction

It is also possible to absorb and squeeze in an alternating fashion.[4] This operation is called the duplex construction or duplexing. It can be the basis of a single pass authenticated encryption system.

- The *State* is initialized to zero
- *Bitrate* is XORed with the first $|Bitrate|$-bit block of the input
- *State* is replaced by *f*(*State*)
- *Bitrate* is now the first $|Bitrate|$ bits of the output.
- *Bitrate* is XORed with the next $|Bitrate|$-bit block of the input
- *State* is replaced by *f*(*State*)
- *Bitrate* is now the next $|Bitrate|$ bits of the output.
- …

Overwrite mode

It is possible to omit the XOR operations during absorption, while still maintaining the chosen security level.[4] In this mode, in the absorbing phase, the next block of the input overwrites the *Bitrate* part of the state. This allows keeping a smaller state between the steps. Since the *Bitrate* part will be overwritten anyway, it can be discarded in advance, only the *Capacity* part must be kept.

Applications

Sponge functions have both theoretical and practical uses. In theoretical cryptanalysis, a **random sponge function** is a sponge construction where *f* is a random permutation or transformation, as appropriate. Random sponge functions capture more of the practical limitations of cryptographic primitives than does the widely used random oracle model, in particular the finite internal state.[5]

The sponge construction can also be used to build practical cryptographic primitives. For example, Keccak cryptographic sponge with a 1600-bit state has been selected by NIST as the winner in the SHA-3 competition. The strength of Keccak derives from the intricate, multi-round permutation *f* that its authors developed.[6] The RC4-redesign called Spritz refers to the sponge-construct to define the algorithm.

For other examples, a sponge function can be used to build authenticated encryption with associated data (AEAD),[3] as well as a password hashing schemes.[7]

References

1. The Keccak Team. "Duplexing The Sponge" (http://sponge.noekeon.org/SpongeDuplex.pdf) (PDF).
2. Guido Bertoni, Joan Daemen, Michaël Peeters and Gilles Van Assche. "Sponge Functions" (http://sponge.noekeon.org/). Ecrypt Hash Workshop 2007.
3. Rivest, Ron; Schuldt, Jacob (2014-10-27). "Spritz – a spongy RC4-like stream cipher and hash function" (http://people.csail.mit.edu/rivest/pubs/RS14.pdf) (PDF). Retrieved 2014-12-29.
4. Guido Bertoni, Joan Daemen, Michaël Peeters and Gilles Van Assche. "Duplexing the sponge: single-pass authenticated encryption and other applications" (http://sponge.noekeon.org/SpongeDuplex.pdf) (PDF).
5. Guido Bertoni, Joan Daemen, Michaël Peeters and Gilles Van Assche. "On the Indifferentiability of the Sponge Construction" (http://sponge.noekeon.org/). EuroCrypt 2008.
6. Boutin, Chad (2 October 2012). "NIST Selects Winner of Secure Hash Algorithm (SHA-3) Competition" (https://www.nist.gov/itl/csd/sha-100212.cfm). NIST. Retrieved 4 October 2012.
7. van Beirendonck, M.; Trudeau, L.; Giard, P.; Balatsoukas-Stimming, A. (2019-05-29). *A Lyra2 FPGA Core for Lyra2REv2-Based Cryptocurrencies*. IEEE International Symposium on Circuits and Systems (ISCAS). Sapporo, Japan: IEEE. pp. 1–5. arXiv:1807.05764 (https://arxiv.org/abs/1807.05764). doi:10.1109/ISCAS.2019.8702498 (https://doi.org/10.1109%2FISCAS.2019.8702498).