COMPUTER SECURITY

Cryptography: more advanced topics (cont.)

One-way cryptography

Motivation

- «*Hash functions are everywhere in cryptography — everywhere!*»[1]

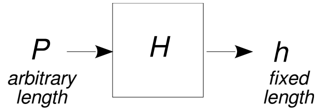
Applications of one-way functions

- data integrity protection
  - P public: $F = h(P)$ is characteristic of P
- confirmation of knowledge
  - P secret: publish $F = h(P)$; later, when P is turned public, F proves previous knowledge of P
- key derivation
  - known $k1$, $k2 = h(k1)$ is new key that does not compromise $k1$!
- pseudo-random number generation
  - *seed* secret: $h^n(seed)$ is apparently random for any successive n
- ...

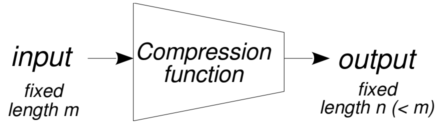
1 *Real-World Cryptography*, D. Wong, Manning, 2021

Definitions[1]

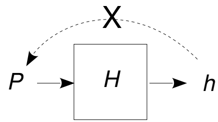
- (minimum) **hash** function *H* [2]
  - compression: maps input *P* of arbitrary finite bit-length, to output *h* of fixed bit-length
  - ease of computation: for any *P*



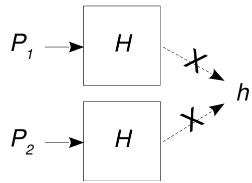
- **compression** function[3]
  - hash function with fixed-size inputs



- **one-way** hash function
  - impractical[4] to invert function



- **collision-resistant** hash function
  - impractical to find two inputs with same output



1 Somewhat based on *Handbook of Applied Cryptography*, A.J. Menezes et. al., 5th Printing, CRC Press, 2001.
2 can use (secret) keys or not... If unkeyed, are also called MDC (Modification Detection Code) functions.
3 this definition is different from the one commonly adopted - see ahead!
4 impractical = currently, computationally infeasible

...One-way cryptography...

Simple examples ($P = P_1 P_2 P_3 ... = P_1 || P_2 || P_3 ...$)

- (minimum) **hash** function (*in*, len(P) ; *out*, len(h))[1]
  - $h = P_1 \oplus P_2 \oplus P_3 \oplus ...$, length (P_i) = length (h)
- **compression** function (*in*: m bits ; *out*: n bits)
  - *out* = (*in*'s first n bits) $\oplus$ (*in*'s last (m-n) bits || (2n-m) 0 bits)
- **one-way** hash function (*in*: m bits ; *out*: n bits)
  - $h = P$ mod len(h)
- **collision-resistant** hash function
  - ?...

1 len --> length

Note on compression function's definition:

- here adopted definition:
  - Compression function (*in*: *m* bits ; *out*: *n* bits)
- common used definition:
  - Common "Compression" function (*in*: *b* bits, *n* bits ; *out*: *n* bits)

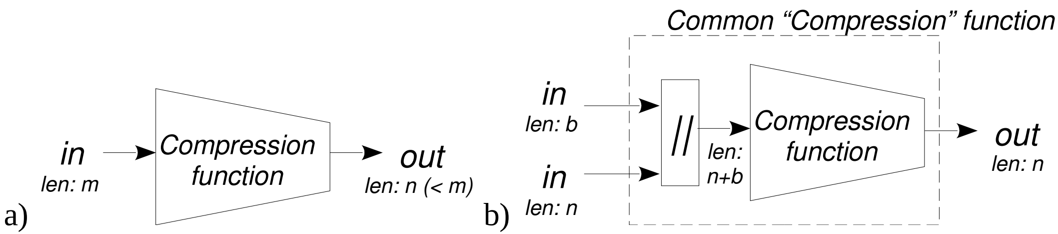
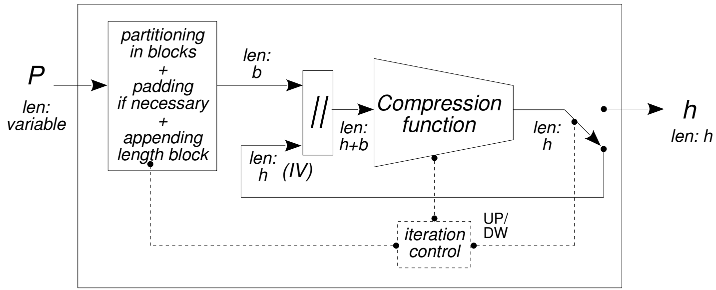


Fig. a) Adopted definition of Compression function; b) Commonly defined "Compression" function.

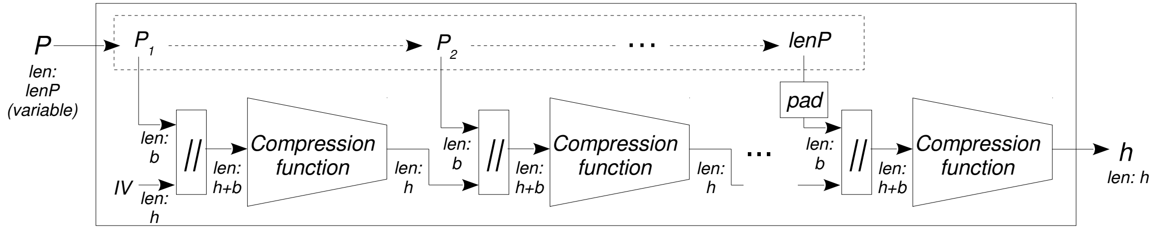
Construction of hash functions

- iterated hash functions (e.g. Merkle–Damgård construction) [FIG]
  - block cipher based hash functions (e.g. Davies-Meyer construction) [FIG]
    - using existing secure cipher functions
  - customized (e.g. SHA-1)
    - specifically designed “from scratch” for optimized performance
  - modular arithmetic based[1] (e.g. MASH-1)
    - quite few implementations as research interest is low:
      - sluggish relative to customized hash functions
      - «*embarrassing history of insecure proposals*» (Menezes et al.)
  - sponge constructions (e.g. SHA-3) [FIG]
    - new paradigm, allowing easy adjustment of output length

1 ISO/IEC 10118-4:1998, Hash-functions using modular arithmetic



a) *H*



b) *H*

Fig. Two views of the Merkle–Damgård construction: a) software-view ; b) time-view.

...One-way cryptography (cont.): Block cipher based - Davies-Meyer construction

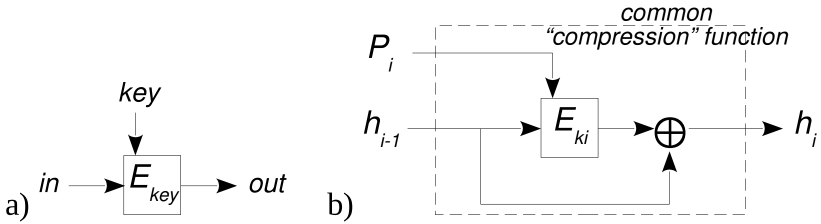


Fig. Structure of Davies-Meyer construction (block-cipher based):
a) enciphering snippet with general idea: if *in* is fixed, E_{key} is one-way for mapping *key* --> *out* !
b) Davies-Meyer construction: final hashing result is iteration for all P_i blocks .

Case study (simplified): SHA-3 (sponge construction)

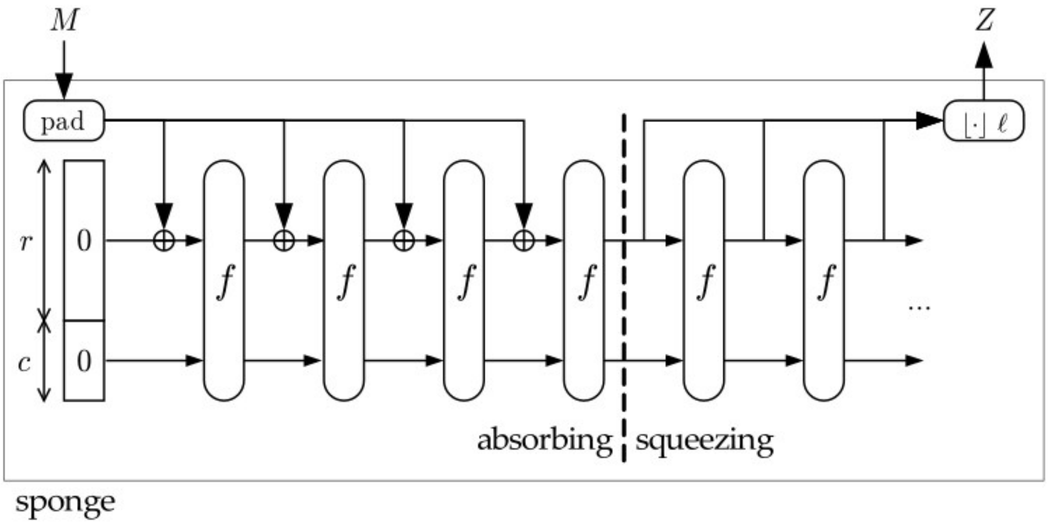
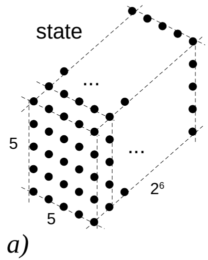


Fig. Sponge construct (time-view): *M* is input that, after padding, is divided in blocks of *r* (rate) bits; *Z* is output of *l* bits of length (specified by input parameter), concatenation of *r* bits' blocks; *c* is capacity, inner, never output, state bits. (*in* keccak.team/sponge_duplex.html)

Sponge construction (cont.)

- function Keccak-*f*[1600][1]:
  - group of permutations on
  - internal state: *b* bits ($5 \times 5 \times 2^6$ bits = 1600)
  - $b = r + c$ bits
    - *r*: bits affected by input
    - *c*: always internal bits
  - group permutation:
    - 12 + 2×6 rounds of five steps:
      θ ρ π χ ι
- specific padding rules



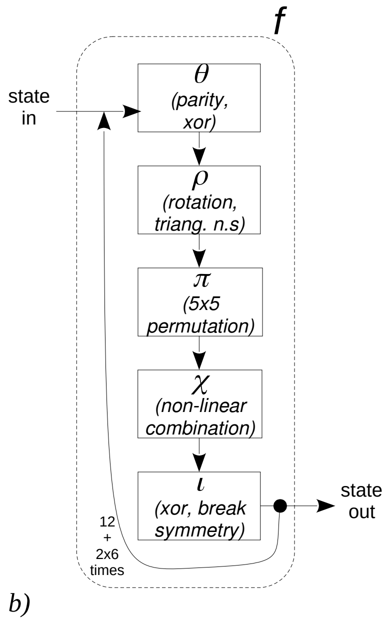


Fig. Inner aspects of sponge structure: a) bits of state; b) sponge function operations.

1 Keccak is pronounced as "ketchak" (keccak.team/keccak_specs_summary.html).

Overall weaknesses of irreversible systems

Problem:

- The number produced by the hashing operation is usually fixed (finite)
    - So, there **have to be** collisions, in an infinite universe of inputs!
    - Will they be likely or easy to cause?

Answer:

- that depends
    - on the randomness of the values resulting from the operation
    - on the size of those values (number of bits)
    - on the intended application

Attacks

- certain: only brute force! (if one can live for enough time...)
  - the intention is to find an entry with a specific result?
    - try 2^n inputs (n, number of bits of *hash*)
- likely: perhaps by using certain curious techniques...
  - the intention is to find two entries with the same result?
    - **birthday attack**: try $\sqrt{2^n} = 2^{n/2}$ inputs for 50% chance of success
    - 2 sets of documents with the same *hash*: one "good" set, one "evil"![1]
- possible: scientifically search for construction weaknesses
  - research, research, research
    - MD5: MD5 considered harmful today
    - SHA-1: We have broken SHA-1 in practice
    - ...

1 Diversity of possibilities for trying different documents are as simple as varying the number of spaces between words...

...One-way cryptography: *Irreversible (cont.)*

Ideal strength of hash function of n-bit output:

- security is as good as a random oracle with output truncated to *n* bits
- implies resistance of size:
  - $2^{n/2}$ for strong collision attacks
  - 2^{n} for weak collision attacks

Example: sponge construction (SHA-3) strength

- with random permutation: as strong as a random oracle
- capacity *c* determines resistance size:
  - 2^{c} for both strong and weak collision attacks
    - unfortunately, security is traded for speed, for constant *b* (= *r*+*c*) size
      - higher security (*c*), lower speed (more *r*-bit input blocks to process)

Integrity & Confidentiality protection

- fact: (*confidentiality*) operation modes do not guarantee *integrity* protection[1]
- so, some type of integrity protection must be added
  - basic example: combine secrecy with digital signatures [FIG]
  - in general: use *authenticated encipherment* protocols

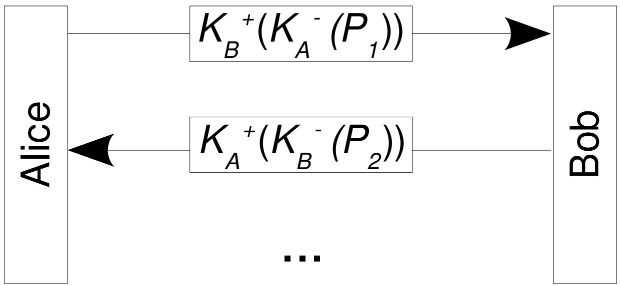


Fig. Confidentiality with integrity protection.

1 E.g. for CBC operation mode, see Kaufman et. al, Network Security, pp. 98-101. Exercise: show the vulnerability with *One time pad*!

Authenticated ciphering protocols (modes)[1]

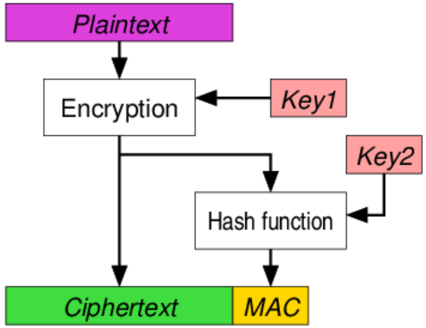
- special protocols developed to aggregate both protections
  - in general, integrity protection is provided by Message Integrity[2] Codes
  - but digital signing can also be used (of course) [previous FIG]
- the main approaches are:
  - (external) combination of protective techniques[3]
    - prone to vulnerabilities due to incorrect implementation
  - "intrinsic" combination
    - several standardized schemes
    - sponge functions can be used in *duplex mode*!
    - *signcryption*: "low-cost" combination of digital signing and ciphering[4]

1 *Authenticated Encryption with Associated Data* (AEAD) applies when it is explicitly necessary to assure integrity protection of plaintext data that is to accompany ciphertext (e.g. network packets might need a visible header that should be integrity protected as well as the secret payload).
2 or Authentication ;-)
3 also called "generic composition" of schemes used separately for achieving confidentiality and integrity protection
4 Digital Signcryption or How to Achieve Cost(Signature & Encryption)..., Y. Zheng, CRYPTO '97

Authenticated Modes - "generic composition"

Encrypt-then-MAC, EtM

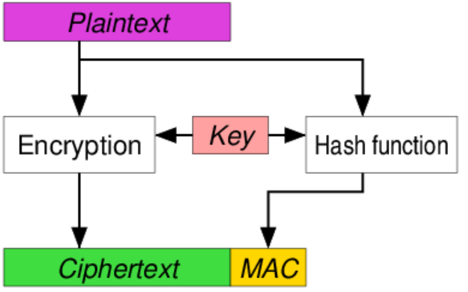
- ISO/IEC 19772:2009
- process: [FIG - in Wikipedia]
  - 1st, encipher; 2nd, calculate MIC
  - non-parallelizable
- different keys K_E, K_{MAC} !
- "normal" padding
- reverse process:
  - verify integrity of ciphertext; decipher to get plaintext
  - parallelizable
- considered the more secure method (compared with the following)[1]



1 see, for instance, Bellare & Namprempre "Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm" (2008)

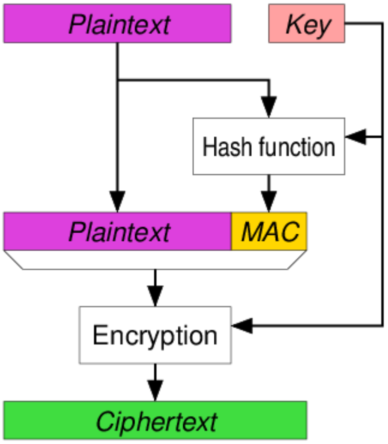
Encrypt-and-MAC (E&M)

- process: [FIG - *in* Wikipedia]
  - encipher; calculate MIC
  - parallelizable
- apparently, a single key is enough!
- "normal" padding
- reverse process:
  - 1st, decipher to get plaintext;
    2nd, verify integrity of plaintext
  - non-parallelizable



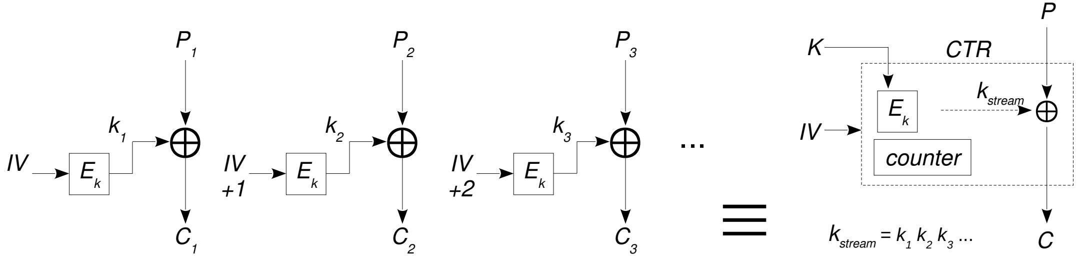
MAC-then-Encrypt (MtE)

- process: [FIG - *in* Wikipedia]
  - 1st, calculate MIC; 2nd, encipher
  - non-parallelizable
- apparently, a single key is enough!
- padding after hashing
- reverse process:
  - 1st, decipher to get plaintext and MAC; 2nd, verify integrity of plaintext
  - non-parallelizable



Authenticated Modes - "intrinsic"

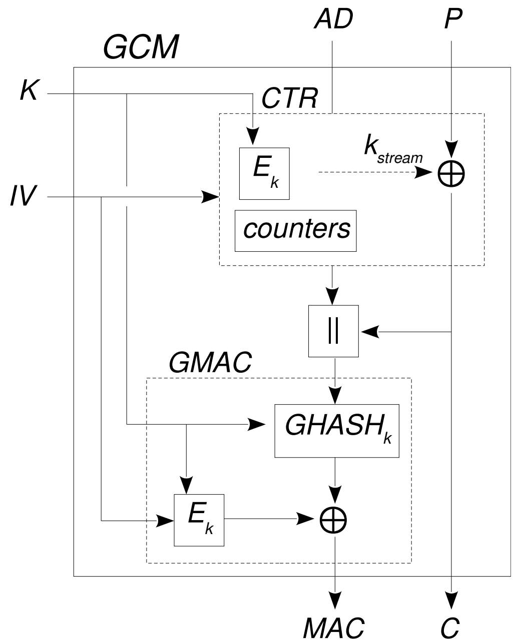
- here, there is an integration of the 2 protections
  - the schemes are built with provision to provide both
- the usual procedure is
  - use a primary key (*seed*) to feed an extended key-generation function
  - use the generated long key, to encipher *P* in *stream* mode
    - typically, a variant of Counter Mode is used [FIG]
  - use part of the generated key to produce a MIC of the ciphered (or plain) text



Some "famous" examples

Galois/Counter Mode (GCM)

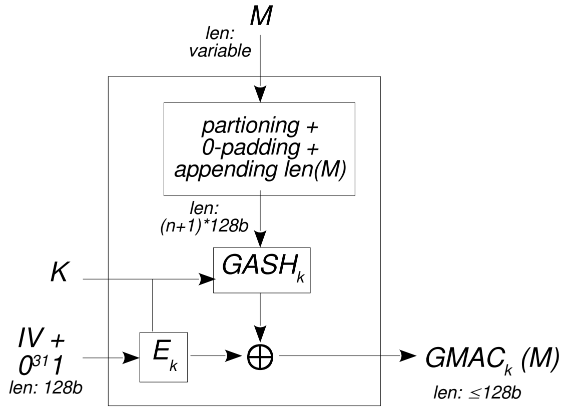
- *NIST 800-38D*
- process: [FIG]
- confidentiality:
  - AES-128b is typical
- integrity protection: GMAC [FIG next page]
  - ciphertext + Associated Data
- apparently, highly performative (parallelization by inter-leaving & pipelining?)
- some obs:
  - *AD*[1] and *C* are padded separately before being concatenated; *IV* is used sequentially in GMAC first and then in CTR; internal intermediate states are to be kept private



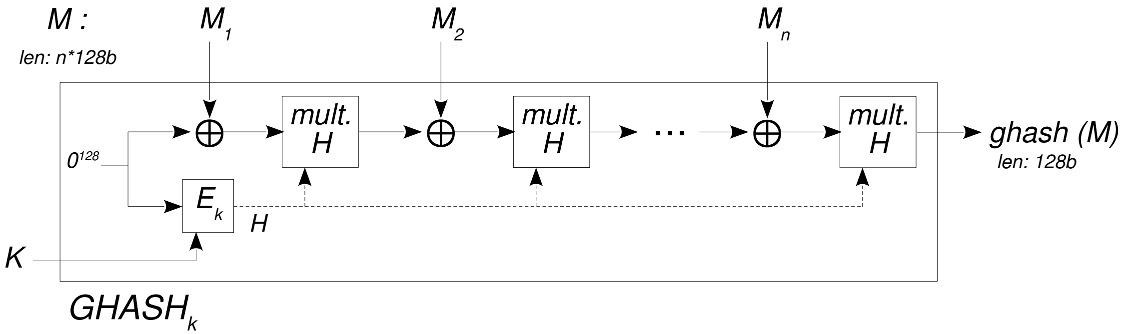
1 Associated Data

Fig. Basic constructs of Galois/Counter Mode for integrity protection (*M* is overall *Message* to protect).

a) Message Authentication Code GMAC;

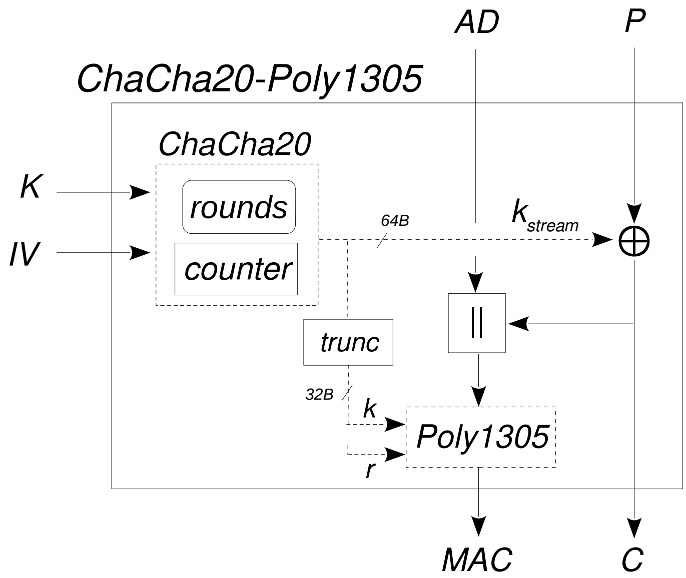


b) hash function GHASH.



ChaCha20-Poly1305

- RFC 8439
- designed by D. J. Bernstein
  - ChaCha20[1] stream cipher
  - Poly1305 *authenticator*
- process: [FIG]
  - key stream feeds
    - first, message integrity code function (counter=0)
    - then, XOR cipher (counter>0)
  - *AD* and *C* are padded separately before being concatenated



1 20 is because it performs 10 times a double set of operations.

ChaCha20-Poly1305 (cont.): Chacha20

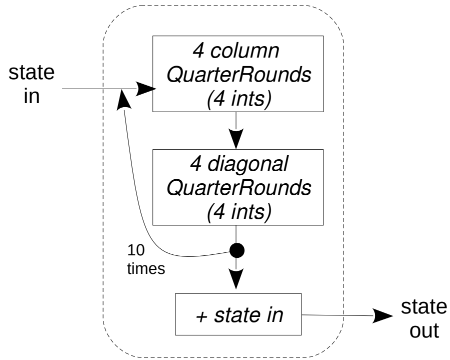
- input: 32B (256b) key, 12B (96b) IV (*nonce*), 4B (32b) counter [FIG]
- output: stream key in 64B (512b) blocks
- internal state: 4 x 4 x 4B (16 32b-integers) = 64 B (512b)
- block function: [FIG]
  - sequence of 10 double[1] "quarter"-rounds
  - quarter-round: set of operations on 4 numbers (addition modulo 2^{32}, XOR, left-shift of n bits)
  - final sum with input
- encipher algorithm:
  - for each iteration (increasing counter), use key stream to cipher 64B block of Plaintext
- deciphering is obvious

state (4x4 32b ints) in:

Cnst	Cnst	Cnst	Cnst
Key	Key	Key	Key
Key	Key	Key	Key
Ctr	IV	IV	IV

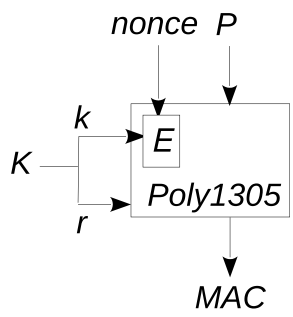
Cnst	Cnst	Cnst	Cnst:
"expa"	"nd 3"	"2-by"	"te k"

Chacha20

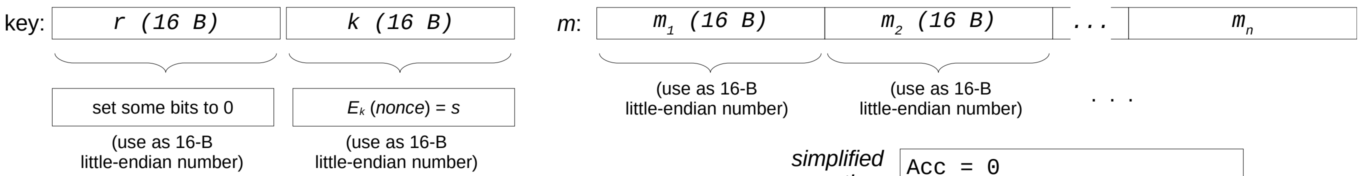


1 10 * 2 = 20 (Chacha20!)

ChaCha20-Poly1305 (cont.): Poly1305



- input:
  - two-part key (r (16B) , k (16B)), 16B nonce,
  - arbitrary-length message (sequence of bytes)
- output: 16B (128b) MAC
- arithmetic operations with 16B groups used as numbers



simplified operations

```
Acc = 0
for i: 1..n
      Acc = Acc + m_i
      Acc = Acc x r
Acc = Acc % 2^130 -5
Acc = Acc + s
Acc = Acc % 2^128 --> MAC
```

$$\text{MAC} = ((m_1 r + m_2 r^2 + \ldots + m_n r^n) \bmod 2^{130} - 5 + E_k(nonce)) \bmod 2^{128}$$

Fig. D. J. Bernstein's Poly1305 *authenticator*: 128b MAC.

SpongeWrap

- sponge construct in duplex mode

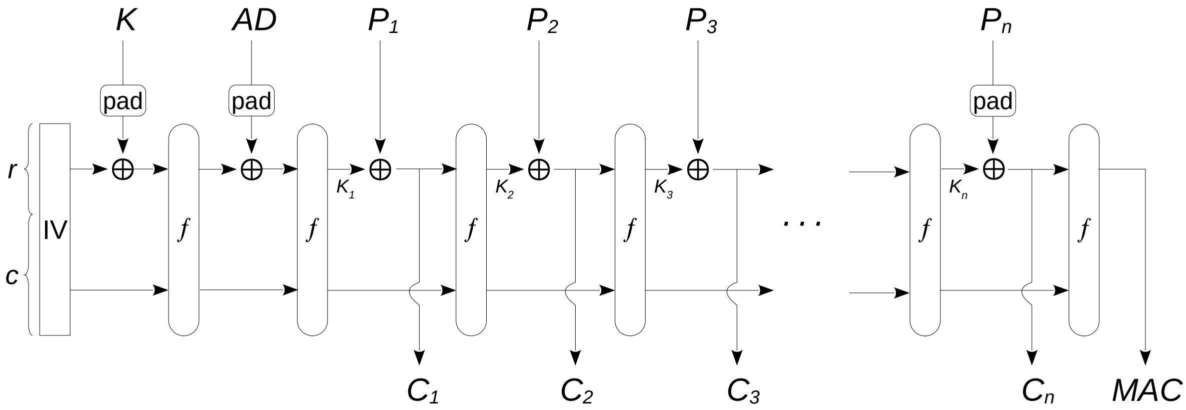


Fig. Sponge construct in duplex-mode for authenticated enciphering (AEAD): notice that plaintext *P* is XORed, block by block, with *f*'s outputs - the *keystream*, k_i *!* The function *pad* is used for padding blocks.

Exercise: adapt the picture to a stream cipher in which the "sponge" generates the key(s).

(could be continued...)

Pointers...

- “**Block cipher mode of operation**”, -2024 – Wikipedia
  - en.wikipedia.org/wiki/Block_cipher_mode_of_operation
- “**Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC**”, 2007 – M. Dworkin, NIST
  - nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf
- “**The Poly1305-AES Message-Authentication Code**”, 2005 – D. Bernstein
  - link.springer.com/content/pdf/10.1007/11502760_3.pdf
- “**ChaCha, a variant of Salsa20**”, 2008 – D. Bernstein
  - cr.yp.to/chacha/chacha-20080120.pdf
- “**Duplexing the sponge: single-pass authenticated encryption...**”, 2011 – G. Bertoni, J. Daemen, M. Peeters, G.Van Assche
  - eprint.iacr.org/2011/499.pdf
- “**The sponge and duplex constructions**”, -2023, G. Bertoni, J. Daemen, S. Hoffert, M. Peeters, G. Van Assche, R. Van Keer
  - keccak.team/sponge_duplex.html