CRYPTOGRAPHY LECTURE, 2022

Computer Science and Algorithmics, PWr

Mirosław Kutyłowski

Cryptographic Hash Functions

Goal

- creating a fingerprint of a message M of a fixed size
- the fingerprint of M should not leak in practice any information on M
- Message Authentication Code $\mathrm{MAC}(\mathrm{Key}, M)$

Applications

coin tossing over internet:

i. Alice chooses K and bit a

ii. Alice calculates: $c := \mathrm{Hash}(K, a)$ and sends c to Bob

iii. Bob chooses bit b and sends it to Alice

iv. Alice computes $r := a \otimes b$ and responds with K

v. Bob checks that $c := \mathrm{Hash}(K, a)$ and computes $r := a \otimes b$

Applications

authenticating transmission over a second channel:

i. server A sends to server B a ciphertext C of a large data D (e.g. RSA+CBC AES hybrid mode)

ii. server A computes $h := \mathrm{Hash}(\mathrm{Key}, D)$ where Key is shared by A and B

iii. the operator of A calls the operator of B and dictates h

iv. server B checks whether h corresponds to the message received by recomputing it

Applications

RSA digital signature:

signature creation for message M and secret key e:

$$s := (\mathrm{Hash}(M))^e \bmod n$$

signature verification for message M, public key (n, d) and signature s:

$$s^d = \mathrm{Hash}(M) \bmod n \quad ?$$

properties needed:

i. Hash maps to numbers in the range $(1, n-1)$

ii. for a given M, it is infeasible to find any M' such that $\mathrm{Hash}(M) = \mathrm{Hash}(M')$

Solution with DLP (not very practical but provably secure)

Setup: random generators g and h of a cyclic group G such that $\log_g(h)$ is unknown

Hashing $H\colon \{1, \ldots, q-1\}^2 \to G$

$$H(x_0, x_1) = g^{x_0} \cdot h^{x_1}$$

Properties:

- the size of $H(x_0, x_1)$ is roughly the half of the size of the arguments
- finding a conflict, i.e. $(x_0', x_1') \neq (x_0, x_1)$

$$H(x_0, x_1) = H(x_0', x_1')$$

  is infeasible, as it would lead to breaking the DLP problem:

  $H(x_0, x_1) = H(x_0', x_1')$ means $g^{x_0} \cdot h^{x_1} = g^{x_0'} \cdot h^{x_1'}$, that is

$$h = g^{(x_0' - x_0)/(x_1 - x_1')}$$

Desired properties

one-way function:

given y it is infeasible to find any x such that $y = \mathrm{Hash}(x)$

necessary for using as a MAC for plaintext

Desired properties

second pre-image resistance:

given x and $y = \mathrm{Hash}(x)$ it is infeasible to find any x' such that $y = \mathrm{Hash}(x')$

necessary for commitment schemes:

– Alice commits to m when she presents $y = \mathrm{Hash}(m, r)$ for a random r

– Alice can open commitment y by revealing m and r

Desired properties

conflict freeness:

it is infeasible to find **any** two arguments $x \neq x'$, such that $\mathrm{Hash}(x) = \mathrm{Hash}(x')$

conflicts do exists due to Pigeon Hole Principle

if $\mathrm{Hash}\colon \{0,1\}^m \rightarrow \{0,1\}^n$ and $n < m$, then for a random x there are on average 2^{m-n} strings x' such that $\mathrm{Hash}(x) = \mathrm{Hash}(x')$

Dependencies:

conflict free $\Rightarrow$ 2nd preimage resistant

$\neg$ 2nd preimage resistant $\Rightarrow$ $\neg$ conflict free

2nd preimage resistant $\Rightarrow$ one-way

$\neg$ one-way $\Rightarrow$ $\neg$ 2nd preimage resistant

Attacks complexity and consequences

- functions like MD5 broken (2nd preimage broken in practice)
- SHA-1 conflict freeness broken, but 2nd preimage still not
- SHA-1: chosen prefix attack: one can take $D \neq D'$ and find Z, Z' such that

  SHA-1$(D||Z) =$ SHA-1$(D'||Z')$ (the cost is still very high)

- until 2004 the people belived that SHA-1 is secure, authorities issued recommendations ...
- ... but prof. Xiaoyun Wang and her team did not believe/knew about it and broke MD-4 and consequently MD-5 and SHA-1

NIST competition for SHA-3

process:

- open call in 2009, requirements explicitly stated
- many candidates, final decision in 2012 after rounds of open evaluation and narrowing the set of candidates
- almost transparent process
- hard work done by volunteers

Different approach (Russia): designed in secrecy, later call for attacks

- GOST (weaknesses found, algorithm withdrawn)
- Streebog (Стрибог) – not broken but serious design weaknesses found

Block concept

input: a block of a fixed size r (e.g. $r = 1088$)

output: a block of a fixed size d (e.g. $d = 256$) where $d < r$

Output block size and birthday paradox:

attack:

i. choose $2^{d/2}$ arguments at random

ii. compute hash values for them

iii. find a conflict

it suceeds with a fair probability due to the birthday paradox

Corollary: hash functions with output size 80 bits make no sense

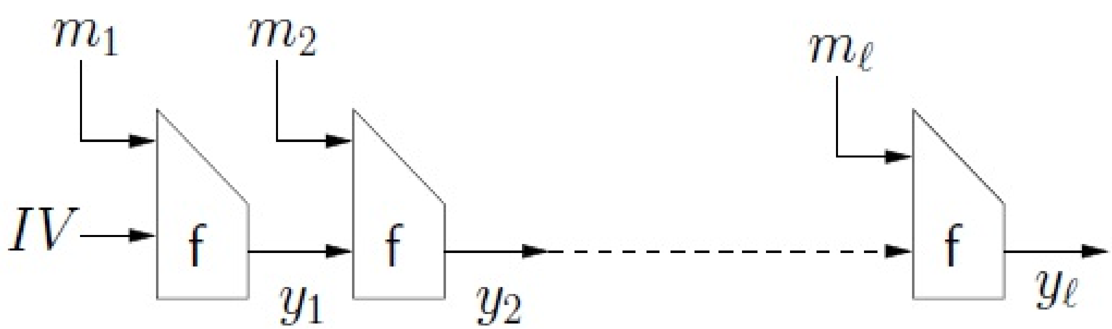
How to hash long messages?

Keccak: sponge construction (phase 1: absorb many blocks, phase 2: squeezing)

invertible

Merkle-Damgård construction:

(picture: Coron et al)



not invertible

Keccak

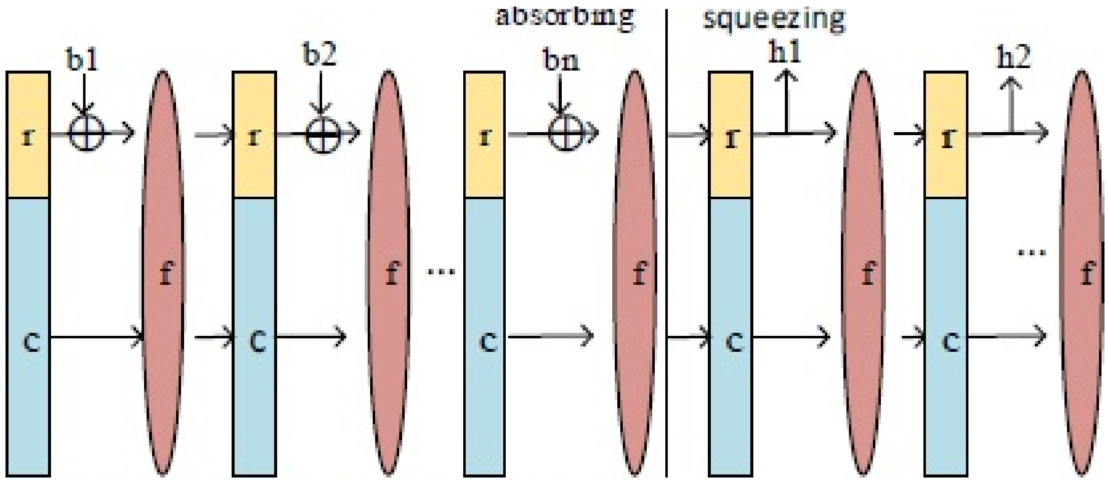


Fig. 9. General structure of Keccak algorithm.

picture: Yakut, Tuncer, Ozer

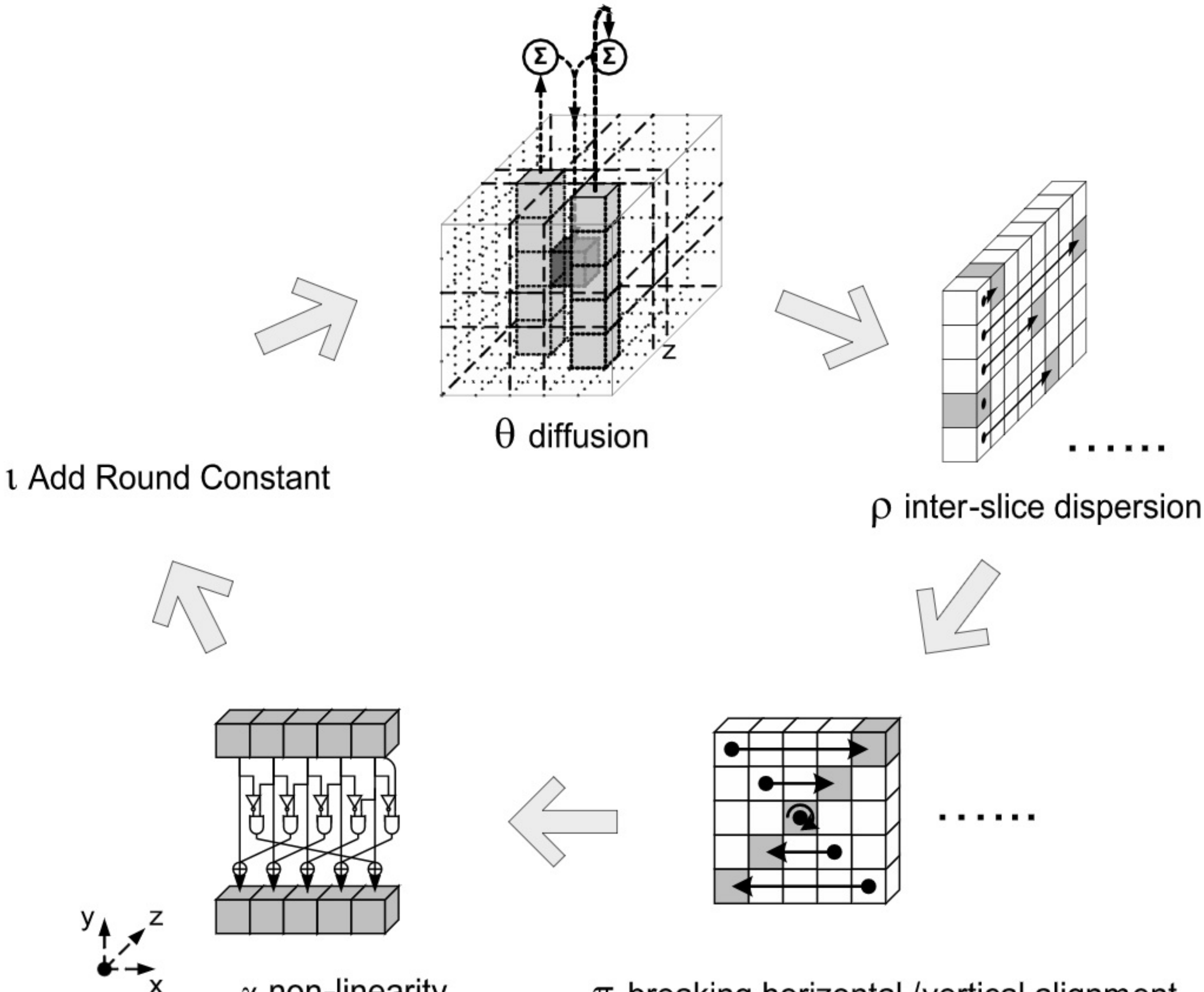
Function f:

only XOR, AND, NOT operations

data viewed as a 3-dimentional structure: 5x5xw

mixing in 3 dimensions (instead of 2 as for AES)

rounds


Σ
Σ
z
θ diffusion
ρ inter-slice dispersion
π breaking horizontal /vertical alignment
χ non-linearity
ι Add Round Constant
y
z
x

θ (theta)

$a[i][j][k] \leftarrow a[i][j][k] \oplus \text{parity}(a[0\ldots4][j-1][k]) \oplus \text{parity}(a[0\ldots4][j+1][k-1])$

ρ (rho)

$0 \leq t < 24,\ a[i][j][k] \leftarrow a[i][j][k-(t+1)(t+2)/2]$,

where $\begin{pmatrix} i \\ j \end{pmatrix} = \begin{pmatrix} 3 & 2 \\ 1 & 0 \end{pmatrix}^t \begin{pmatrix} 0 \\ 1 \end{pmatrix}$

π (pi)

$a[3i+2j][i] \leftarrow a[i][j]$

χ (chi)

$a[i][j][k] \leftarrow a[i][j][k] \oplus (\neg a[i][j+1][k] \,\&\, a[i][j+2][k])$

ι (iota)

Exclusive-or a round constant into one word of the state.

HMAC

message authentication code based on Hash function and a secret key

HMAC computation for message M and key K:

$$h := \mathrm{HMAC}_K(M) = \mathrm{Hash}((K \oplus \mathrm{opad}) \| \mathrm{Hash}((K \oplus \mathrm{ipad}) \| M))$$

upon receiving M and h, the HMAC of M is recomputed and compared with h

Hash mode for encryption function

idea: instead of creating separate functions for computing hashes, reuse the existing strong encryption functions

$\longrightarrow$ important for embedded devices: less code/hardware to be installed: e.g. AES instead of hash *and* symmetric encryption

$\longrightarrow$ encryption is also a pseudorandom function

Be careful: e.g. $\mathrm{Hash}(M_1, M_2)$ defined as $\mathrm{Enc}_{M_1}(M_2)$ would be silly:

i take $H := \mathrm{Enc}_{M_1}(M_2)$

ii choose Z at random and $Y := \mathrm{Dec}_Z(H)$

iii ... and we have a collision on (M_1, M_2) and (Z, Y)

$\longrightarrow$ hardware implementation of AES might be easier than of a hash function (e.g. the number of gates required)

→

AES hash mode

see: on the exercises list

- one of the NIST criteria for AES standard: reuse of encryption for hashing