Gambit

password hashing scheme

Pintér Krisztián
pinterkr@gmail.com

Autentikáció

- “is” - ki vagyok én
  - ujjlenyomat
  - retina
  - DNS?
- “have” - mim van
  - token
  - fájl
- “know” - mit tudok
  - jelszó
  - gesture

Történet dióhéjban

reuse miatt nem tároljuk (hash)

lookup/rainbow table ellen salt

támadó brute force-ra kényszerül

csakhogy: hashcat 100 mhash/s

brute force megvalósítható (pwd 30-60 bit)

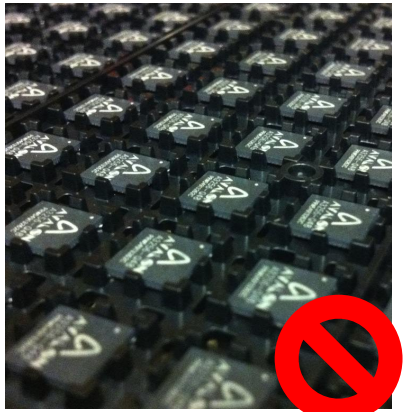
→ lassítás: PBKDF2, bcrypt, scrypt

Barátok és ellenségek

PC gaming master race

hw, firmware,
smartcard,
router, media player

ASIC farm



botnet

Eszköztár

PC

- 8 mag
- GB/mag
- “CISC”

van

GPU

- 200+ mag
- MB/mag
- “RISC”

van

ASIC

- 1 mag
- $/kB/mag
- “xRISC”
- x N

meg kell venni

Ötletek

Mink van, ami nekik nincs?

- memória
- integer aritmetika (szorzás/osztás/modulo)
- lebegőpontos aritmetika
- big num aritmetika
- S-box (anti-GPU, anti-SIMD)
- parallelizáció 4/8/256

“követelmények”

egyszerű (KISS)

side channel

PRF

security proof

128/256 bit security

“áramvonalas”

finom paraméterezhetőség

KISS

- security through obscurity *ellentéte*
- egységnyi analízis többet ér (erdő vs mező)
- implementálás hibalehetősége csökken
- olcsóbb hw/fw

KISS

memória: OK

integer aritmetika: komplex, de kezelhető

lebegőpontos: kerekítés, táblázatok, sorok …

bignum: még komplexebb, de kezelhető

S-box: simd ellenes, gpu ellenes

parallelizáció: komplex, de kezelhető

side channel

memória: olvasás/írás mintázat

integer aritmetika: timing, power

lebegőpontos: timing, power, cache timing

bignum: timing, power, cache timing

S-box: cache timing

parallelizáció: nincs összefüggés

áramvonalas

SHA2(pwd || salt || h_{n-1})

pwd || salt || h_{n-1} || 1...0 || L

adott jelszó hosszhoz:

L konstans, 1..0 konstans, salt konstans

az első round-ban megspórolható

ok: feleslegesen bonyolult primitív

mi maradt

memória: ✓ (fix sorrend)

integer aritmetika: talán

lebegőpontos: x

bignum: x

S-box: x

parallelizáció: talán

memory hard

T(n) idő, S(n) memória

memory hard: $T(n) = O(S(n))$

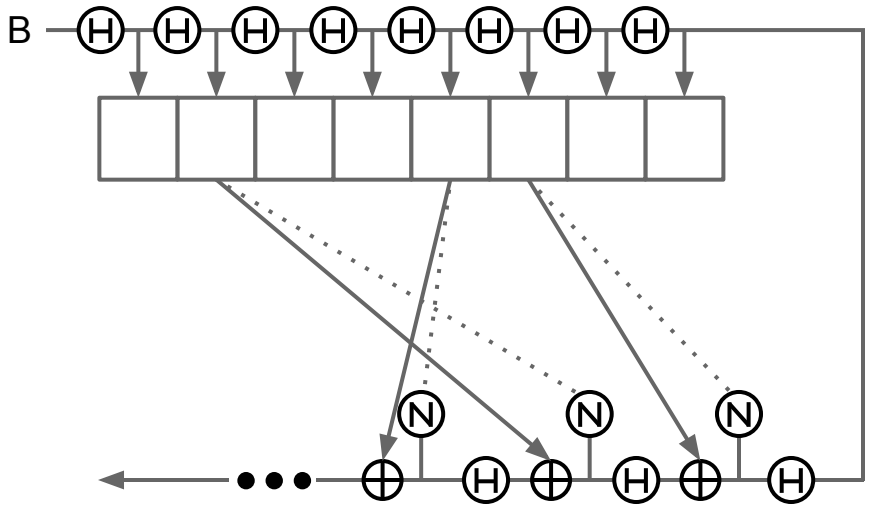
olyan sok memóriát használ, amennyit lehet

sequential memory hard:

nem lehet csökkenteni a T(n)*S(n) értéket még osztott memóriás többszálú szgépen sem.

scrypt

ROMix(H, B, N)



Ⓗ: PRF

$\{0,1\}^b \rightarrow \{0,1\}^b$

Ⓝ: $\{0,1\}^b \rightarrow 0..N-1$

scrypt problémák

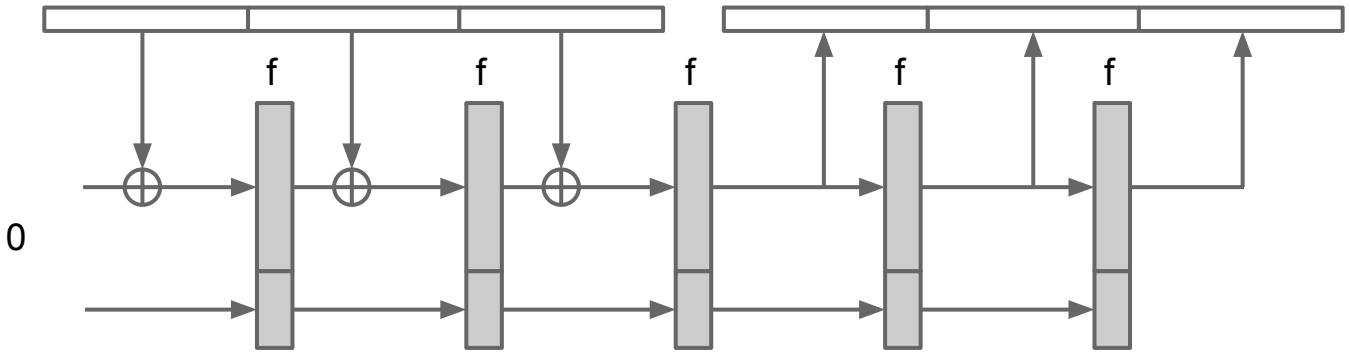
- komplex - 2xPBKDF2, salsa20/8, HMAC-SHA256
- side channel - titkos adattal indexel
- rosszul paraméterezhető - memória és idő egy paraméter bár ez szándékos (memory hard)

Keccak

Keccak-f[25, 50, 100, 200, 400, 800, 1600]

permutáció; xor, and, not, rot

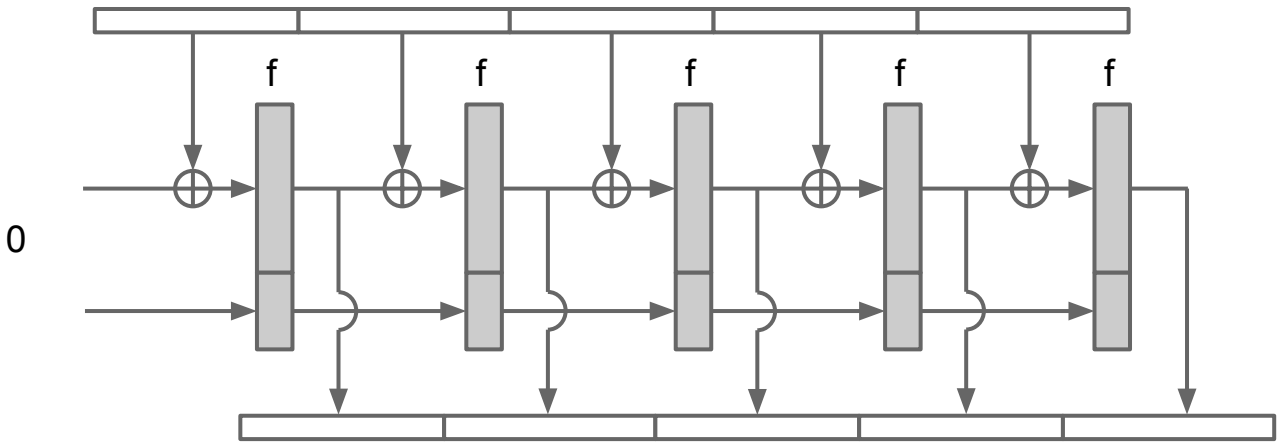
Sponge



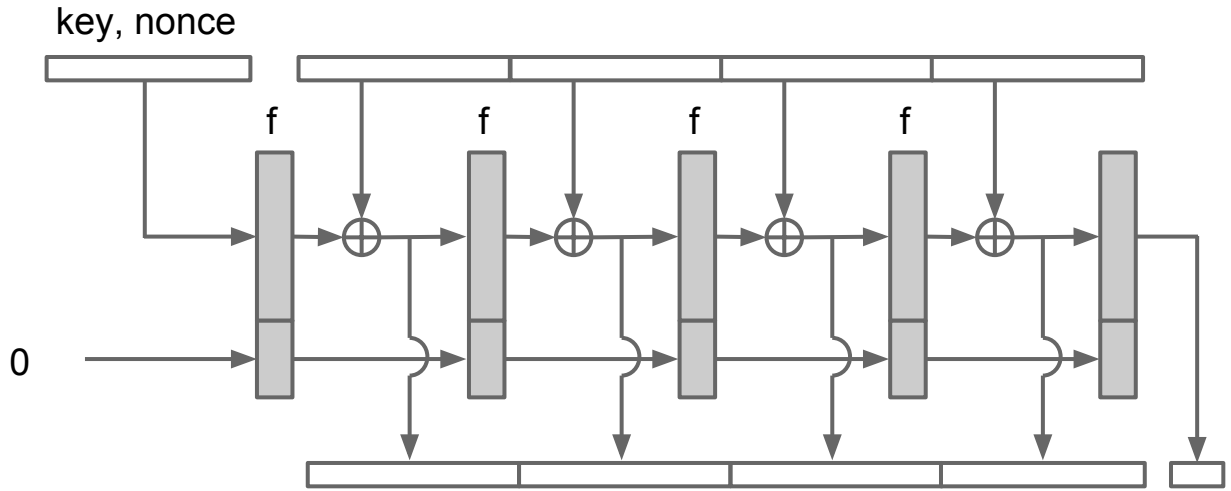
Keccak felhasználások

funkció	**absorb**	**squeeze**
hash	M || 10*1	hash
PRNG	entropy	stream
stream cipher	key, salt	stream
MAC	key, M || 10*1	MAC
KDF	key material	keys
PBKDF	salt, pwd || 10*1 || 0...0	keys

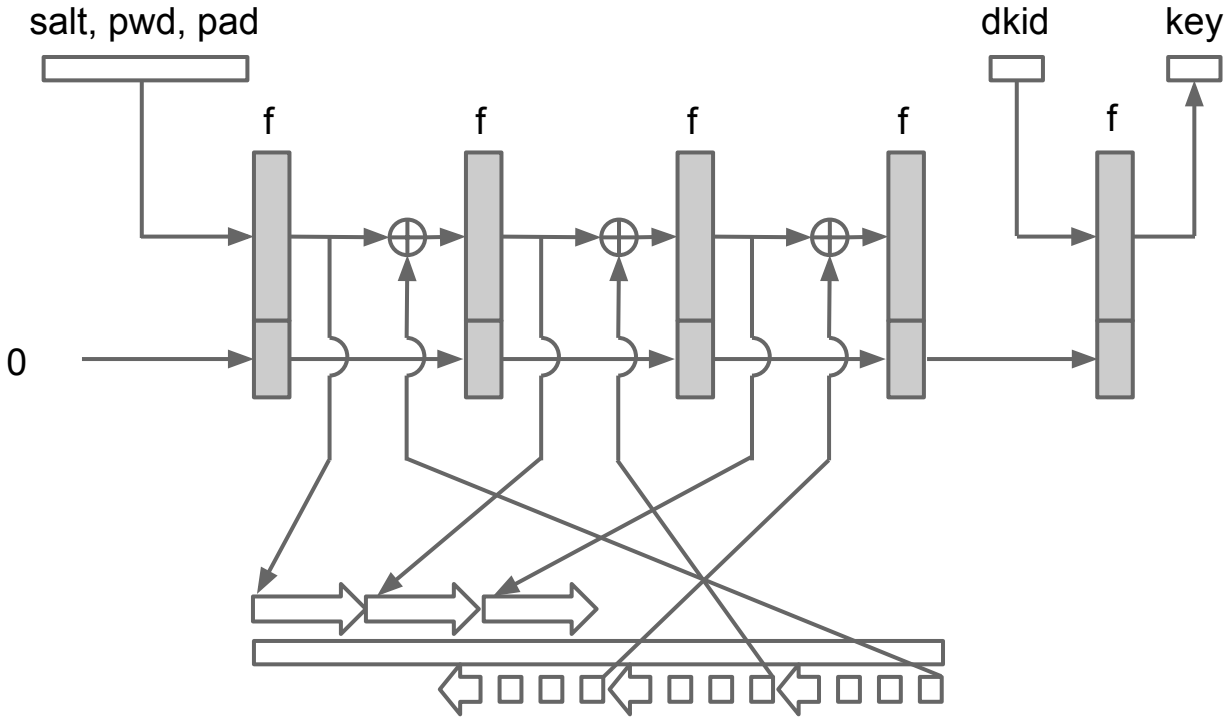
Keccak Duplex



(Authenticated encryption)



Gambit, séma



dkid

sokféle id (corporate?), nem kell egyeztetés

session key funkció

state megsemmisítés (keccak-f permutáció)

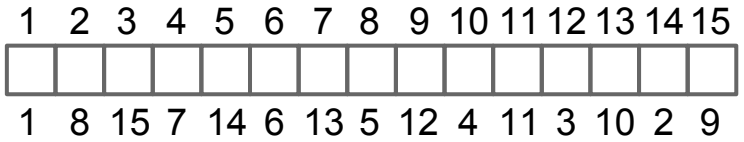
server relief

Írás/olvasás sorrendje

i-edik (0 … n-1) lépésben:

írás: m_i

olvasás: m_j



lehet több kör, ilyenkor lehet, hogy az előző körben írt adatot olvassuk.

Naív módszer

$j = (i-m) \bmod n$

vezessük be a “kort”. kérdés: m_j értéke melyik i lépésből való?

kor: $k = j - i$

$j = i - m \Rightarrow k = m$

törés: párhuzamosan futó példány, m-mel lemaradva. kétszeres CPU, 0 RAM

Naív módszer 2

$j = (i-m_1) \bmod n$ ha i páros

$(i-m_2) \bmod n$ ha i páratlan

$k = m_1$ vagy m_2

törés: két párhuzamosan futó példány, m_1-gyel és m_2-vel lemaradva. 3x CPU, 0 RAM

Naív módszer 3

$j = 2i \bmod n$

$k = 0..n-1$ mind (n páratlan)

törés: két párhuzamosan futó példány, kettőt lép

4x CPU, 0 RAM

Első közelítés

j = -i mod n

k = 0..n-1 mind (n páratlan)

törés: visszafelé futó példány (Keccak-f[] permutáció).

100(?)x CPU, 0 RAM

Catena

j = br(i) 0010111 => 1110100

k = br(i) - i

k sokféle értéket felvesz, de nem mindent

bajok:

- szoftverben nem optimális
- 2^n RAM
- broken (Kovratovich; azóta van újabb)

Gambit

j = f*i mod n

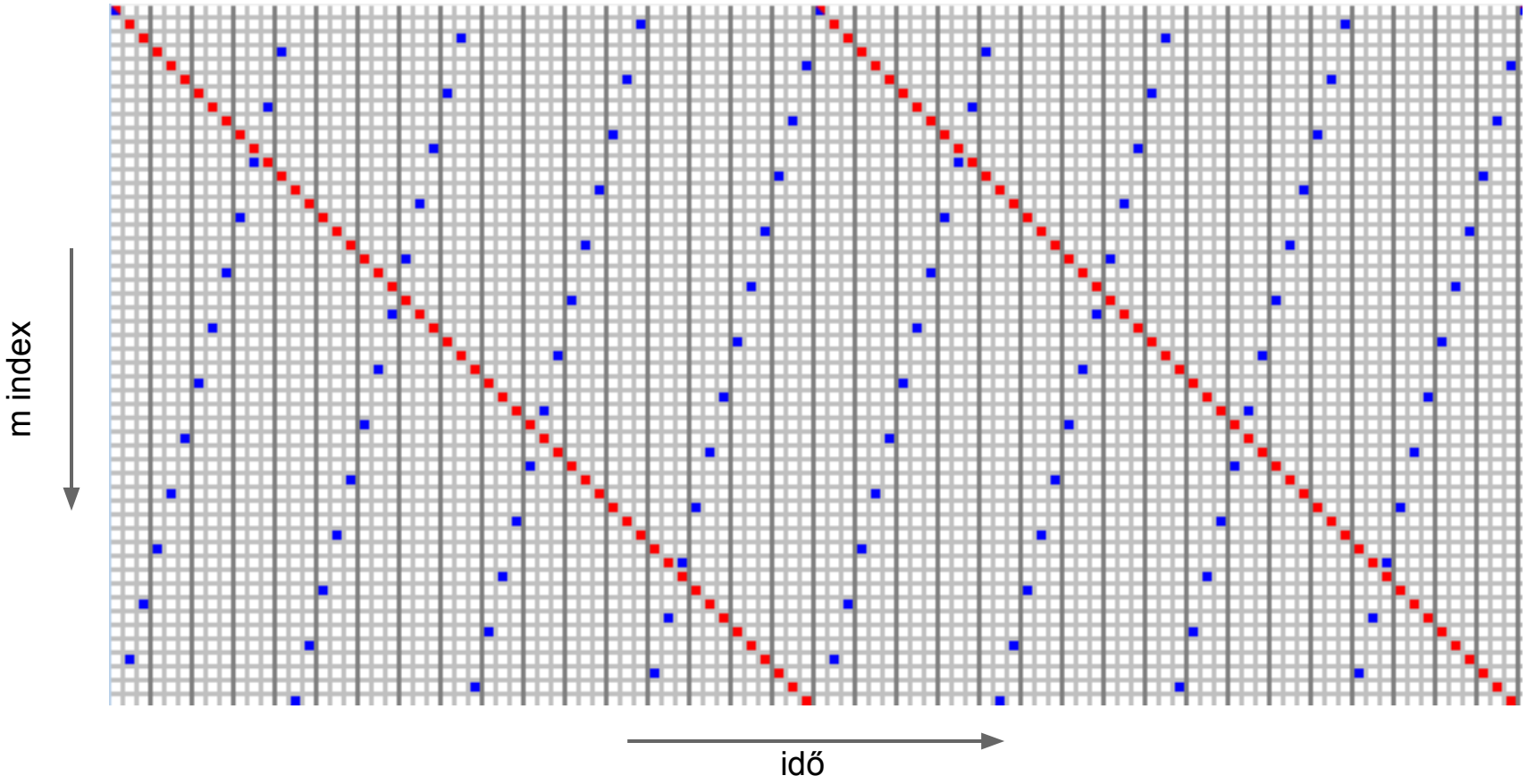
lnko(f, n) = 1

lnko(f-1, n) = 1

f ~ 0.99n = -0.01n

k = 0..n-1 mind

törés: ?


m index
idő

Gambit pseudocode

```
function Gambit(pwd, salt, t, m, dkid) returns key is
     S.Init
     Mem[0..m-1] := 0
     S.Absorb salt || pwd || pad
     loop i in 0 .. t-1
          R := S.Squeeze
          loop j in 0 .. r-1
               Mem[i*r + j] ^= Trans(R[j])
               R[j] := (Mem[(i*r + j) * f] ^ ROM[i*r + j])
          end loop
          S.Absorb R
     end loop
     // save S here
     S.AbsorbOvr dkid
     key := S.Squeeze
end
```

Gambit, c++

```
for (; cost_t > 0; cost_t--)
{
      for (unsigned int i = 0; i < r/8; i++)
      {
            mem[wrtp] ^= trans( A[i] );
            wrtp++;
            if (wrtp == cost_m) wrtp = 0;

            A[i] ^= mem[rdp] ^ ROM[romp];
            rdp += f;
            if (rdp >= cost_m) rdp -= cost_m;
            romp++;
            if (romp >= ROM_len) romp = 0;
      }
      keccak-f(A);
}
```

Bizonyítvány

KISS	5
side channel	5(?)
PRF	5
security proof	2
128/256 bit security	5
“áramvonalas”	5
finom paraméterezés	5

Probléma 1

Titok-függő adat terítése több MB RAM-ba.

Támadás: cold boot, 1% pontossággal

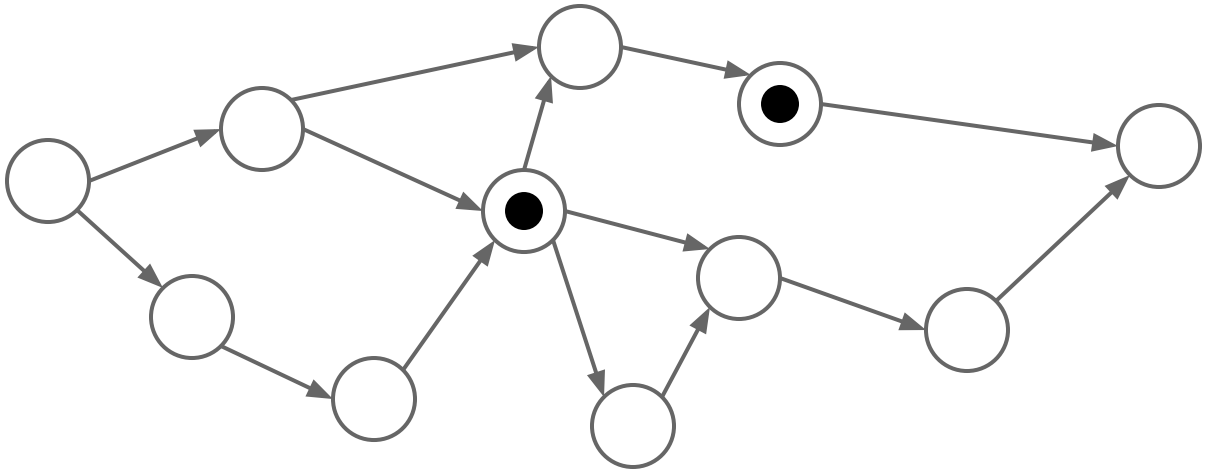
- 128 bit kulcs: 1.28 bit helyreállítva
- 10MB RAM array: teljes helyreállítás

Úgy viselkedik, mint hibajavító algoritmus.

Blinding?

Probléma 2

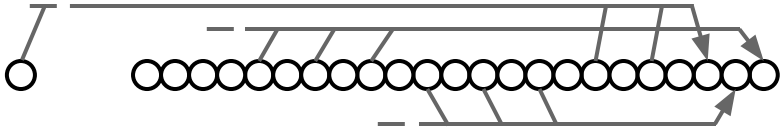
Black pebble game



Van-e hatékony Gambit pebbling?

hogy kell egyáltalán felrajzolni a gráfot?

- csúcs: keccak state



- csúcs: szavak