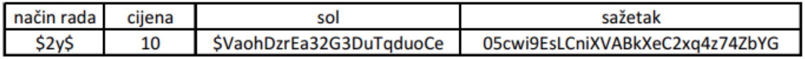
Password hashing

- Introduced in 1970s
- Sense of security which is often false
- Fast hashes
  - MD4, MD5, SHA-1
  - Designed for MACs and digital signatures
  - Must be easy to compute
- Slow hashes
  - bcrypt, PBKDF2, scrypt
  - Designed for password hashing
  - Inefficient and difficult to compute

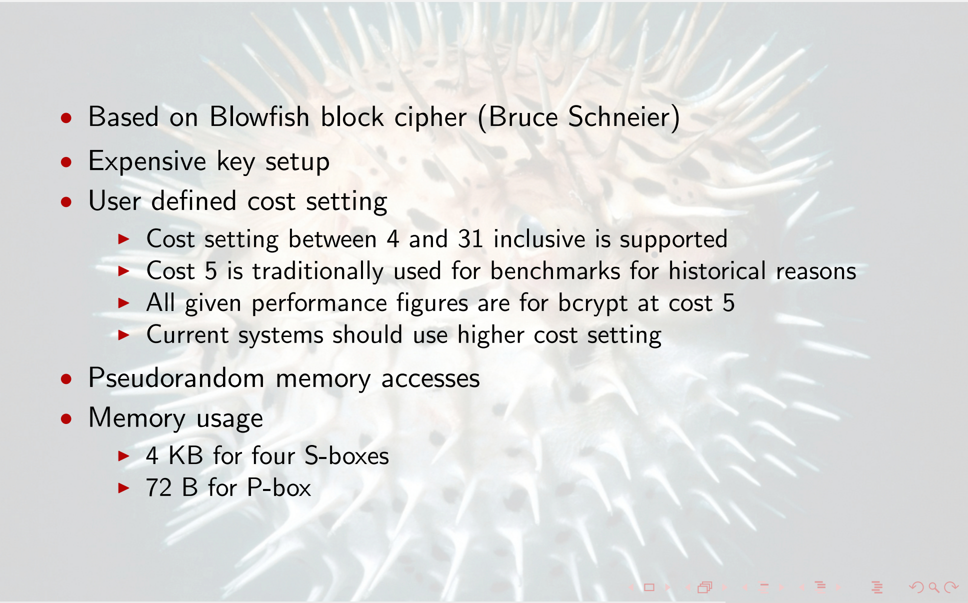


način rada	cijena	sol	sažetak
$2y$	10	$VaohDzrEa32G3DuTqduoCe	05cwi9EsLCniXVABkXeC2xq4z74ZbYG

Motivation

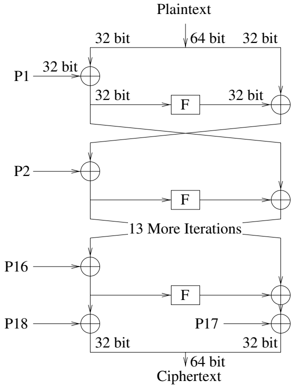
- Bcrypt is:
    - Popular (often used): WWW, OpenBSD, SUSE Linux
    - Slow
    - Sequential
    - Random memory footprint
    - Designed to be resistant to brute force attacks and to remain secure despite hardware improvements
- You could almost think why even bother optimizing

Bcrypt (Provos and Mezieres)



- Based on Blowfish block cipher (Bruce Schneier)
- Expensive key setup
- User defined cost setting
  - Cost setting between 4 and 31 inclusive is supported
  - Cost 5 is traditionally used for benchmarks for historical reasons
  - All given performance figures are for bcrypt at cost 5
  - Current systems should use higher cost setting
- Pseudorandom memory accesses
- Memory usage
  - 4 KB for four S-boxes
  - 72 B for P-box

Blowfish encryption



- Symmetric-key cipher
- 64-bit input block, 32 to 448-bit key
- Feistel network
- Pseudorandom memory accesses
  - 32-bit loads from four 1 KB S-boxes initialized with digits of number π

$$R_i = L_{i-1} \oplus P_i \tag{1}$$

$$L_i = R_{i-1} \oplus F(R_i) \tag{2}$$

$$F(a, b, c, d) = ((S_1[a] + S_2[b]) \oplus S_3[c]) + S_4[d] \tag{3}$$

Niels Provos and David Mazieres, "A Future-Adaptable Password Scheme", The OpenBSD Project, 1999

EksBlowfish

Ekspensive key schedule Blowfish

Algorithm 1 EksBlowfishSetup(cost, salt, key)

```
1: $state \leftarrow InitState()$
2: $state \leftarrow ExpandKey(state, salt, key)$
3: $repeat(2^{cost})$
4:     $state \leftarrow ExpandKey(state, 0, salt)$
5:     $state \leftarrow ExpandKey(state, 0, key)$
6: $return\ state$
```

- Order of lines 4 and 5 is swapped in implementation

Niels Provos and David Mazieres, "A Future-Adaptable Password Scheme", The OpenBSD Project, 1999

bcrypt

Algorithm 2 bcrypt(cost, salt, pwd)

```
1: $state \leftarrow EksBlowfishSetup(cost, salt, key)$
2: $ctext \leftarrow$ "$OrpheanBeholderScryDoubt$"
3: $repeat(64)$
4:     $ctext \leftarrow EncryptECB(state, ctext)$
5: $return\ Concatenate(cost, salt, ctext)$
```

Output: (bcrypt_indicator, cost, 128-bit base-64 22 chars salt, 184-bit base-64 31 chars hash)

$2a$12$GhvMmNVjRW29ulnudl.LbuAnUtN/LRfe1JsBm1Xu6LE3059z5Tr8m

Niels Provos and David Mazières, "A Future-Adaptable Password Scheme", The OpenBSD Project, 1999

Architecture
Epiphany

- 16/64 32-bit RISC cores operating at up to 1 GHz/800 MHz
  - Chips used in testing operate at 600 MHz
- Pros
  - **Energy-efficient** - 2 W maximum chip power consumption
  - 32 KB of local memory per core
  - 64 registers
  - FPU can be switched to integer mode
  - Dual-register (64-bit) load/store instructions
- Cons
  - FPU in integer mode can issue only add and mul instructions
  - Only simple addressing modes
    - Index scaling would be helpful for S-box lookups

Implementation
Epiphany

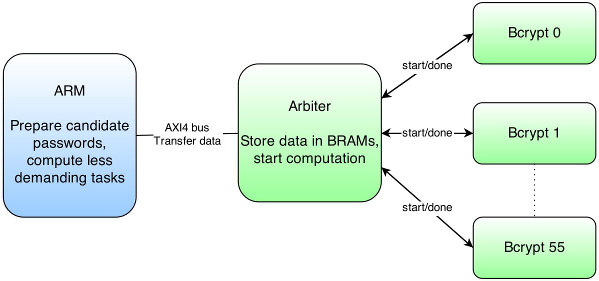
- John the Ripper prepares data on ARM cores
- Bcrypt hashes computed on Epiphany
- Single instance does not have enough instruction level parallelism
  - FPU has four cycle latency
  - FPU does not have bitwise instructions
- Optimized in assembly
- Two instances overlapped to exploit dual-issue architecture
  - Integer ALU
  - FPU in integer mode

Architecture
Zynq 7020

- Heterogeneous device
- Dual ARM Cortex-A9 MPCore
  - 667 MHz
  - 256 KB on-chip memory
- Advanced low power 28nm programmable logic
  - 85 K logic cells
  - 560 KB of block RAM
- AXI buses used for CPU-FPGA communication

Implementation
Zynq 7020

- John the Ripper prepares data on ARM cores
- Bcrypt instances compute hash
- Number of concurrent instances limited by available BRAM
- Large communication overhead for low cost setting
- Hardware defects of ZedBoard limit optimizations


ARM
Prepare candidate passwords, compute less demanding tasks
AXI4 bus
Transfer data
Arbiter
Store data in BRAMs, start computation
start/done
start/done
start/done
Bcrypt 0
Bcrypt 1
Bcrypt 55

Zynq 7045

- Architecture
  - ARM CPU and Zynq reconfigurable logic
  - Roughly 4 times bigger than Zynq 7020
- Implementation
  - ZedBoard implementation ported to a bigger device
  - Bottleneck: CPU-FPGA communication (for low cost setting)
  - Not possible to use all available resources due to hardware defects of ZC706 board

Theoretical Peak Performance Analysis

Theory

$$c/s = \frac{N_{ports} * f}{(2^{cost} * 1024 + 585) * N_{reads} * 16} \tag{4}$$

- N_{ports} - number of available read ports to local memory or L1 cache
- N_{reads} - number of reads per Blowfish round
  - 4 or 5 depending on whether reads from P-boxes go from one of those read ports we've counted or from separate storage such as registers
- $2^{cost} * 1024 + 585$ - number of Blowfish block encryptions in bcrypt hash computation
- f (in Hz) - clock rate

bcrypt(cost, salt, pwd)

```
1: $state \leftarrow InitState()$
2: $state \leftarrow ExpandKey(state, salt, key)$
3: $repeat(2^{cost})$
4:     $state \leftarrow ExpandKey(state, 0, salt)$
5:     $state \leftarrow ExpandKey(state, 0, key)$
6: $ctext \leftarrow$ "OrpheanBeholderScryDoubt"
7: $repeat(64)$
8:     $ctext \leftarrow EncryptECB(state, ctext)$
9: $return\ Concatenate(cost, salt, ctext)$
```

Takeaways

- Many-core low power RISC platforms and FPGAs are capable of exploiting bcrypt peculiarities to achieve comparable performance and higher energy-efficiency
- Higher energy-efficiency enables higher density
  - More chips per board, more boards per system
- It doesn't take ASICs to improve bcrypt cracking energy-efficiency by a factor of 45+
  - Although ASICs would do better yet