yescrypt

password hashing scalable beyond bcrypt and scrypt

Solar Designer <solar@openwall.com>

@solardiz @Openwall

http://www.openwall.com

May 2014

Why passwords? Why hashing?

- Passwords remain a convenient and ubiquitous authentication factor
  - "Something you know" in 2FA
- Proper password hashing mitigates the impact of user database leaks
  - Saves a percentage of accounts from compromise until passwords are forcibly changed (as they should be after a known database leak)
  - Mitigates the impact on the users' accounts on other sites, where the same or similar passwords may have been reused (whether we like it or not)
- Password hashing is not a perfect security measure, yet it is a must
  - To make it effective, password policy enforcement is also highly desirable
- A closely related concept is password- or passphrase-based cryptographic key derivation (for data encryption), which also remains relevant

Password hashing scheme selection criteria

- Password hashing schemes and key derivation functions are not all equally good for all use cases
- Need to be efficient for defenders and inefficient for attackers
  - To the extent practical
- Efficient on common defenders' hardware
  - Commodity servers, desktops, or/and mobile devices as appropriate for the use case
- Inefficient on other types of hardware
  - GPUs, typical botnet nodes (vs. defenders' servers), FPGAs, ASICs
  - OK, can't be literally "inefficient" on ASICs, but can minimize the advantage
- Numerous other aspects matter too

ASIC/FPGA attacks on modern password hashes

- PBKDF2-HMAC-SHA-1
- PBKDF2-HMAC-SHA-256, sha256crypt
- PBKDF2-HMAC-SHA-512, sha512crypt
- bcrypt
- scrypt
- yescrypt

▲ Weaker

Stronger ▼

It is a sound approach to consider attacks with ASICs, but in practice attacks with less flexible devices are also relevant

GPU attacks on modern password hashes

- PBKDF2-HMAC-SHA-1
- PBKDF2-HMAC-SHA-256, sha256crypt
- PBKDF2-HMAC-SHA-512, sha512crypt
- scrypt at below 16 MB (misuse)
- bcrypt (uses 4 KB), scrypt at 16 MB
  - YACoin at 16 MB on GTX TITAN 6 GB is on par with CPU
- scrypt at 32+ MB
- yescrypt
  - In native mode with pwxform enabled and non-excessive parallelism

▲ Weaker

Stronger ▼

yescrypt optional improvements from scrypt

- Time-memory trade-off (TMTO) resistance
  - Reduces attacker's flexibility, and increases attacker's area-time cost by 2x to 4x
- Quicker attacker's ASIC area-time growth per defender's running time
- Running time tunable separately from memory usage and parallelism
- bcrypt-like GPU unfriendliness
  - by including bcrypt-like rapid random reads, typically from defender CPU's L1 data cache
- Multiplication latency hardening (efficient on common x86 and ARM CPUs)
  - (Integer) multiplication takes about as much time on common CPUs as it does in ASIC, limiting possible attack speedups (for same memory usage) even with much faster memory

    This differs from scrypt's Salsa20/8, which is optimally computed in a lot fewer clock cycles in ASIC than on CPU

    32x32 to 64-bit: [I]MUL on x86[-64], [V]PMULUDQ with SSE2/AVX/AVX2; UMLAL [or VMLAL] on ARM [with NEON]

yescrypt improved scalability

- Scalable from kilobytes (RAM) to terabytes (ROM) and beyond
  - while providing adequate security vs. alternatives for same defender's {time, memory} cost
  - Configurable entirely via runtime settings
  - A read-only lookup table (ROM) can be arbitrarily large regardless of running time
  - This differs from scrypt, where memory usage for a given running time is limited, so scrypt's memory usage might not be high enough when high request rate capacity is desired (16+ MB may be hard to achieve at thousands of requests per second)
- Scalable to arbitrary SIMD vector width and instruction-level parallelism
  - To be configurable via compile-time settings, or chosen at runtime from common presets
  - Current defaults are for 4x 128-bit SIMD lanes (512 bits)
  - scrypt's `p` could be brought down to instruction level, but it would be sub-optimal

yescrypt Catena-like features

Current:

- Client-side computation of almost final yescrypt hashes (server relief)
  - in a way allowing for a straightforward extension of SCRAM (RFC 5802)

Planned:

- Hash upgrades to higher cost settings without knowledge of passwords
  - Can be defined and implemented without breaking compatibility with current yescrypt
  - Will support over 60% area-time efficiency, as opposed to Catena's over 33%
- Optional full or partial cache timing side-channel resistance
  - This is a trade-off since it weakens TMTO resistance and removes dependency on memory latency (but yescrypt's multiplication latency hardening will help)

yescrypt cryptographic security

- Cryptographic security (collision resistance, preimage and second preimage resistance) is based on that of SHA-256, HMAC, and PBKDF2
- The rest of processing, while crucial for increasing the cost of password cracking attacks, may be considered non-cryptographic
  - There are entropy bypasses to final PBKDF2 step for both password and salt inputs
  - For comparison, scrypt has such entropy bypass for the password input only
- The known unfortunate peculiarities of HMAC and PBKDF2 are fully avoided in the way these primitives are used
  - yescrypt native mode is immune from this scrypt curiosity that could raise some hairs:

```
scrypt(PBKDF2-HMAC-SHA256-fail-affects-scrypt-no-security-issue-bGoDFpr8) =
= scrypt(;`B3nR6wQ2-_LSg"mH  #yszm`[#z8B&L) for any salt, N, r, p
```

yescrypt drawbacks

- Just one major drawback: complexity!
- scrypt was already pretty complicated, and the full yescrypt is more so
  - The full yescrypt can compute classic scrypt hashes and native yescrypt hashes
- The complexity is justified by yescrypt's flexibility and its arsenal of defenses: it's one password hashing scheme for many possible use cases that deals with many kinds of attacks
- The alternative would have been defining multiple simpler schemes, but their total complexity would probably be even greater
- Cut-down yet compatible implementation with partial functionality is planned

yescrypt use cases

- Mass user authentication
  - Social networks, web portals, e-mail service providers
  - Banks, online shopping/payment services
  - e-government, tax return e-filing, tax payment services
- User authentication
  - (Smaller) websites
  - Local operating system accounts
  - Domain accounts
- Encryption key derivation
- Proof-of-work scheme e.g. in a cryptocoin

Mass user authentication

- Dedicated authentication servers
- Can afford a ROM-in-RAM large enough to buy ~5 years of anti-botnet
  - 128 GB to 512 GB of RAM is now easily affordable for servers, but is still way beyond what's found in typical botnet nodes
    - Current desktop CPUs are commonly limited to addressing 32 GB of RAM, and actual botnet node RAM sizes are typically even less (of course, this will be changing)
  - The gap between supported RAM sizes of commodity servers and botnet nodes is likely to persist even as actual sizes grow
- As an option, can use a ROM-on-SSD (terabytes?), which has pros and cons compared to ROM-in-RAM
  - yescrypt supports both kinds, and we ran tests with SSD too

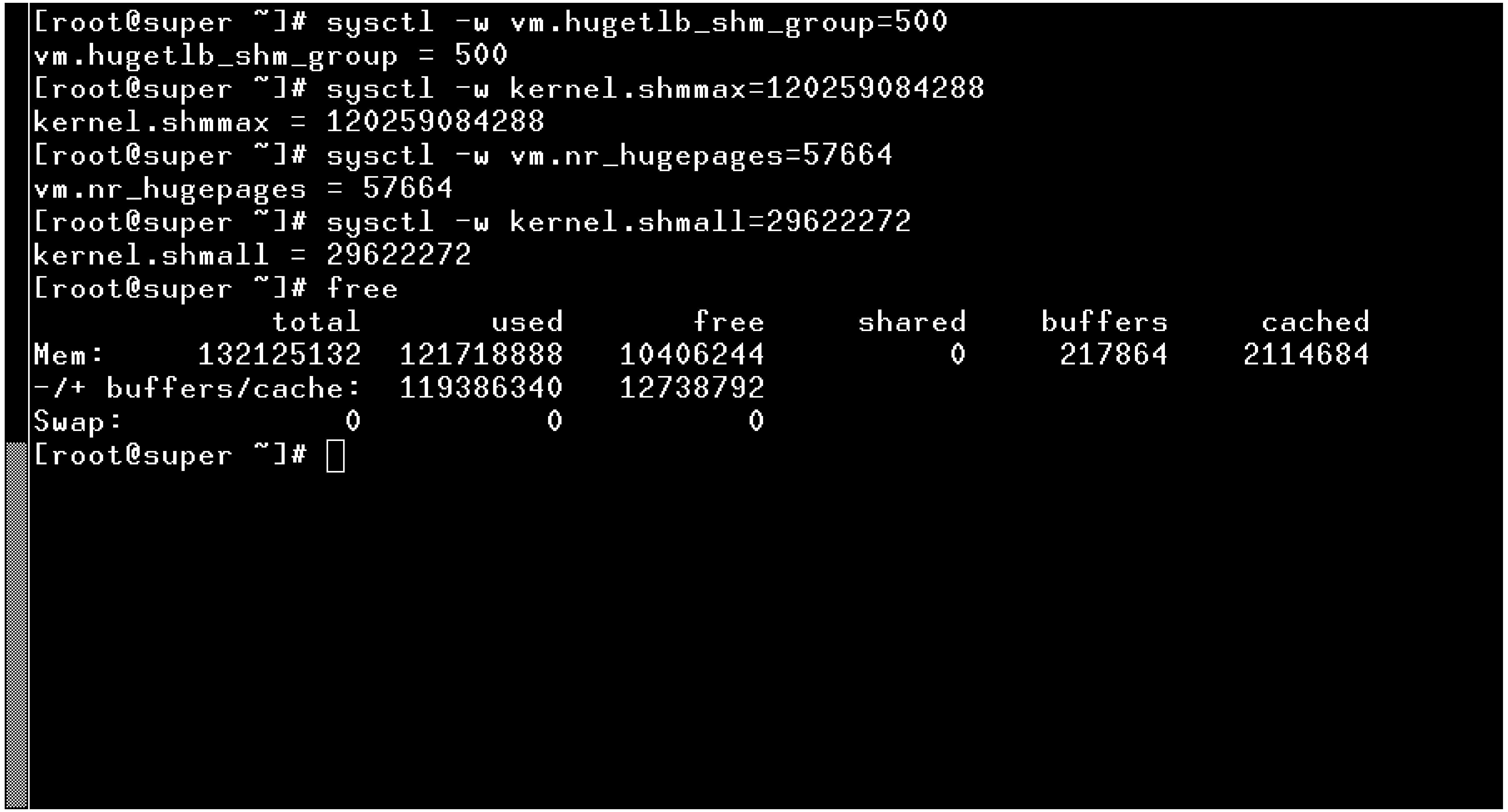
Mass user authentication with ROM-in-RAM: demo

On 2x E5-2670 (16 cores, 32 threads) with 128 GB RAM (8x DDR3-1600)

- 112 GB ROM initialization takes half a minute (on server bootup)
  - yescrypt's TMTO-resistant algorithm discourages recomputation on smaller machines
- ROM is maintained in a SysV shared memory segment
  - This is one good way to do it, although yescrypt API does not mandate it
- Thus, there's no delay in our demo "authentication service" (re)start
  - Important for ease of system administration, and to minimize downtime
- yescrypt is passed a pointer to the ROM via an appropriate API
- Request rate capacity may be tuned, with varying RAM usage settings
  - Attacker's cost varies by our usage of: CPU time, RAM, bandwidth, and by ROM size

Mass user authentication with ROM-in-RAM: demo

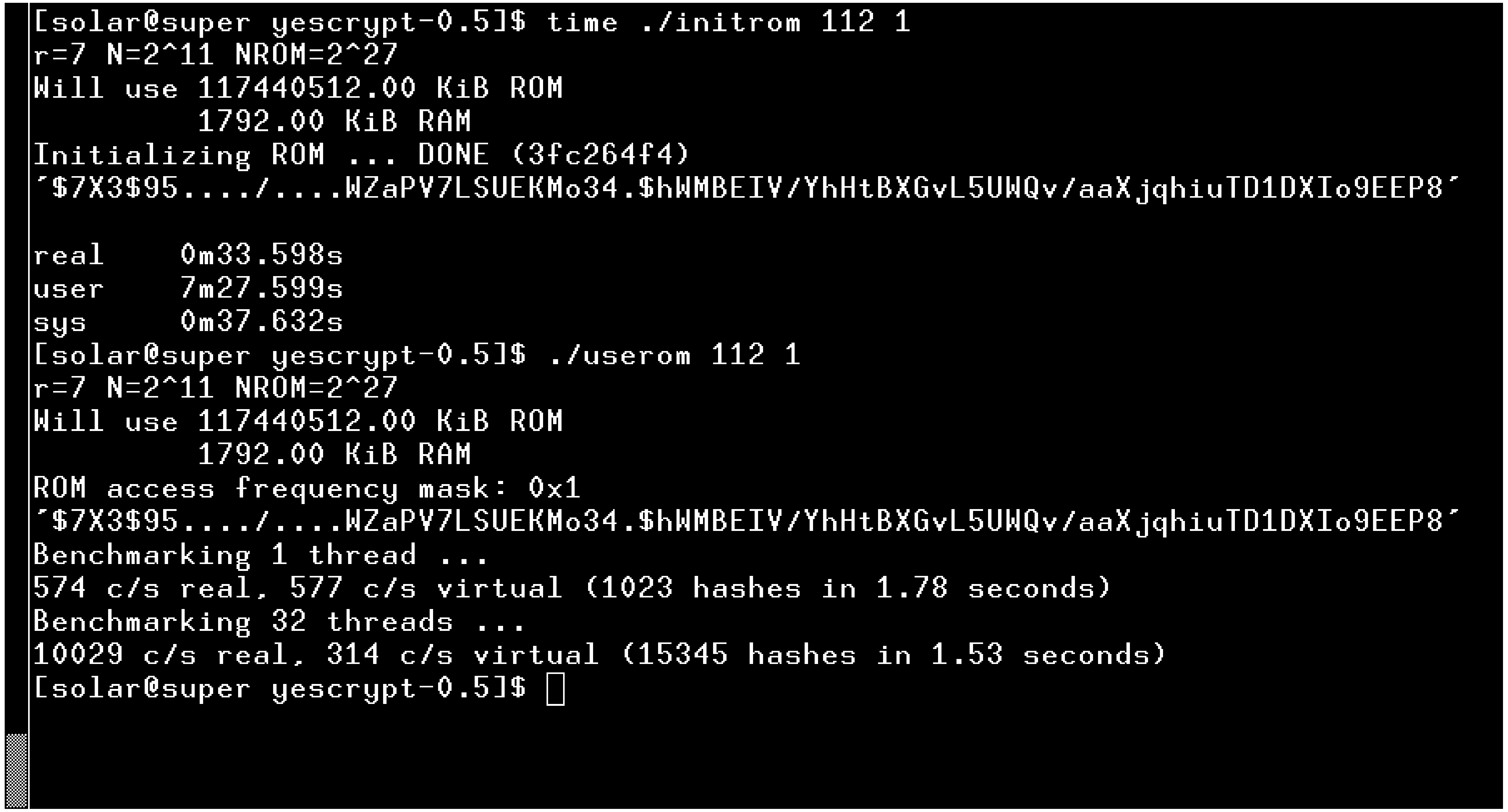
- Linux configuration: a bit more than 112 GiB (120 GB) in "huge pages"



```
[root@super ~]# sysctl -w vm.hugetlb_shm_group=500
vm.hugetlb_shm_group = 500
[root@super ~]# sysctl -w kernel.shmmax=120259084288
kernel.shmmax = 120259084288
[root@super ~]# sysctl -w vm.nr_hugepages=57664
vm.nr_hugepages = 57664
[root@super ~]# sysctl -w kernel.shmall=29622272
kernel.shmall = 29622272
[root@super ~]# free
             total       used       free     shared    buffers     cached
Mem:     132125132  121718888   10406244          0     217864    2114684
-/+ buffers/cache:  119386340   12738792
Swap:            0          0          0
[root@super ~]# 
```

Mass user authentication with ROM-in-RAM: demo

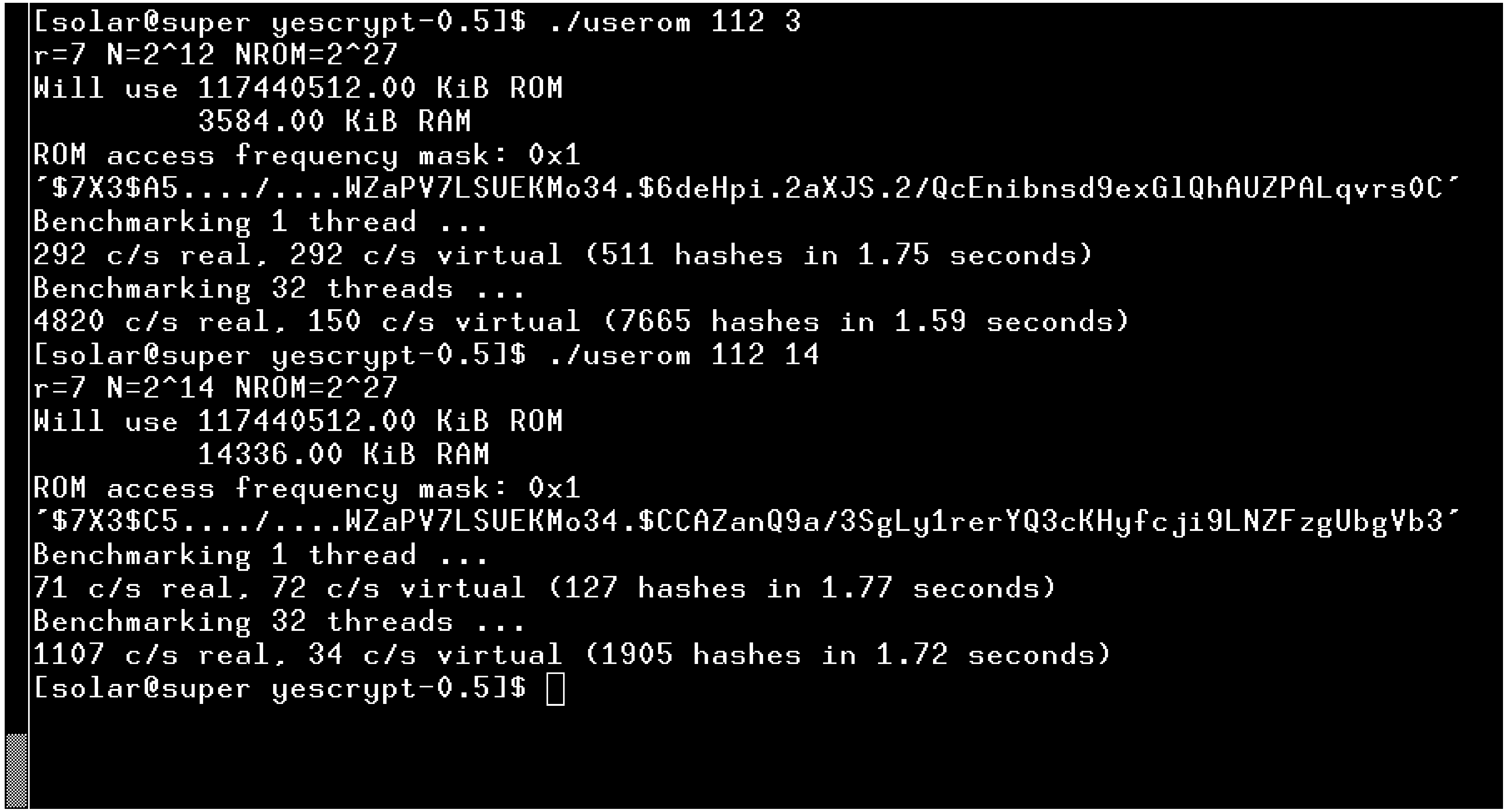
- ROM initialization (34 seconds) and initial test (over 10000 hashes/second)



```
[solar@super yescrypt-0.5]$ time ./initrom 112 1
r=7 N=2^11 NROM=2^27
Will use 117440512.00 KiB ROM
         1792.00 KiB RAM
Initializing ROM ... DONE (3fc264f4)
'$7X3$95..../....WZaPV7LSUEKMo34.$hWMBEIV/YhHtBXGvL5UWQv/aaXjqhiuTD1DXIo9EEP8'

real    0m33.598s
user    7m27.599s
sys     0m37.632s
[solar@super yescrypt-0.5]$ ./userom 112 1
r=7 N=2^11 NROM=2^27
Will use 117440512.00 KiB ROM
         1792.00 KiB RAM
ROM access frequency mask: 0x1
'$7X3$95..../....WZaPV7LSUEKMo34.$hWMBEIV/YhHtBXGvL5UWQv/aaXjqhiuTD1DXIo9EEP8'
Benchmarking 1 thread ...
574 c/s real, 577 c/s virtual (1023 hashes in 1.78 seconds)
Benchmarking 32 threads ...
10029 c/s real, 314 c/s virtual (15345 hashes in 1.53 seconds)
[solar@super yescrypt-0.5]$ 
```

Mass user authentication with ROM-in-RAM: demo

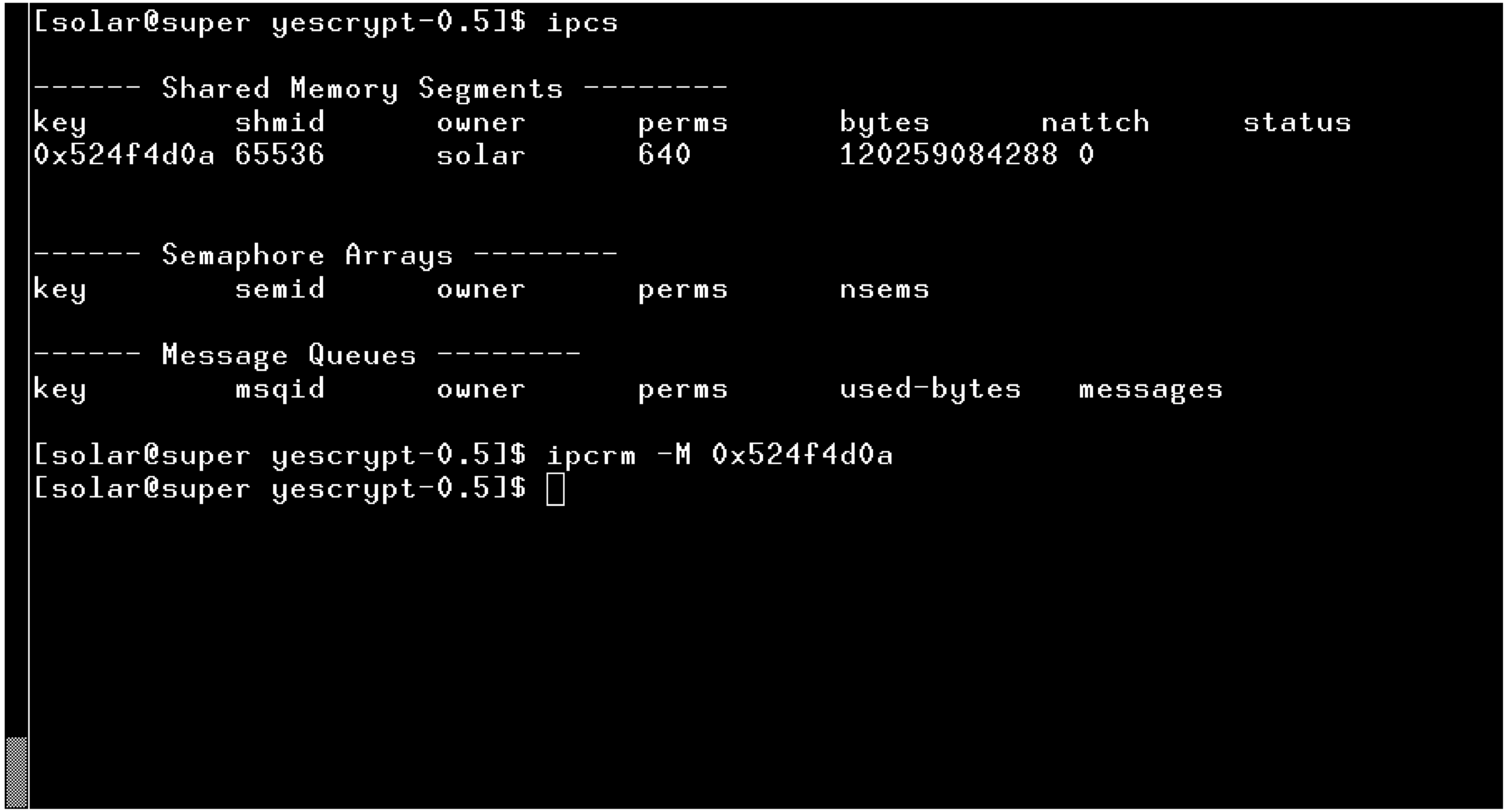
- Higher (per-thread) RAM, lower throughput tests



```
[solar@super yescrypt-0.5]$ ./userom 112 3
r=7 N=2^12 NROM=2^27
Will use 117440512.00 KiB ROM
         3584.00 KiB RAM
ROM access frequency mask: 0x1
'$7X3$A5..../....WZaPV7LSUEKMo34.$6deHpi.2aXJS.2/QcEnibnsd9exGlQhAUZPALqvrs0C'
Benchmarking 1 thread ...
292 c/s real, 292 c/s virtual (511 hashes in 1.75 seconds)
Benchmarking 32 threads ...
4820 c/s real, 150 c/s virtual (7665 hashes in 1.59 seconds)
[solar@super yescrypt-0.5]$ ./userom 112 14
r=7 N=2^14 NROM=2^27
Will use 117440512.00 KiB ROM
         14336.00 KiB RAM
ROM access frequency mask: 0x1
'$7X3$C5..../....WZaPV7LSUEKMo34.$CCAZanQ9a/3SgLy1rerYQ3cKHyfcji9LNZFzgUbgVb3'
Benchmarking 1 thread ...
71 c/s real, 72 c/s virtual (127 hashes in 1.77 seconds)
Benchmarking 32 threads ...
1107 c/s real, 34 c/s virtual (1905 hashes in 1.72 seconds)
[solar@super yescrypt-0.5]$ 
```

Mass user authentication with ROM-in-RAM: demo

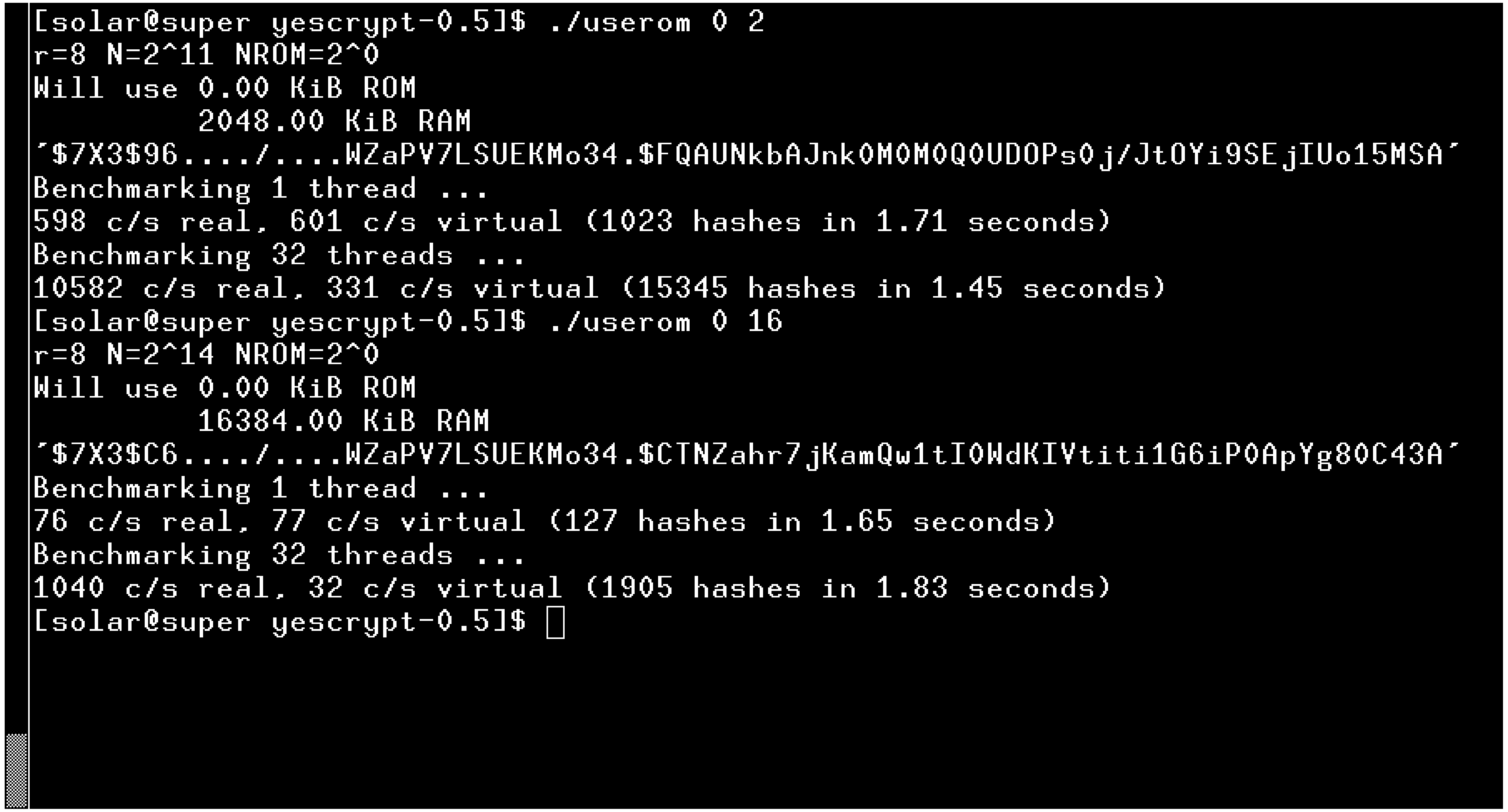
- Here's what the SysV shm segment looked like, and how to destroy it



```
[solar@super yescrypt-0.5]$ ipcs

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x524f4d0a 65536      solar      640        120259084288 0


------ Semaphore Arrays --------
key        semid      owner      perms      nsems

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages

[solar@super yescrypt-0.5]$ ipcrm -M 0x524f4d0a
[solar@super yescrypt-0.5]$ 
```

User authentication without ROM: demo

- Similar throughput and adequate security may be achieved without ROM



```
[solar@super yescrypt-0.5]$ ./userom 0 2
r=8 N=2^11 NROM=2^0
Will use 0.00 KiB ROM
         2048.00 KiB RAM
'$7X3$96..../....WZaPV7LSUEKMo34.$FQAUNkbAJnk0M0M0Q0UD0Ps0j/Jt0Yi9SEjIUo15MSA'
Benchmarking 1 thread ...
598 c/s real, 601 c/s virtual (1023 hashes in 1.71 seconds)
Benchmarking 32 threads ...
10582 c/s real, 331 c/s virtual (15345 hashes in 1.45 seconds)
[solar@super yescrypt-0.5]$ ./userom 0 16
r=8 N=2^14 NROM=2^0
Will use 0.00 KiB ROM
         16384.00 KiB RAM
'$7X3$C6..../....WZaPV7LSUEKMo34.$CTNZahr7jKamQw1tI0WdKIVtiti1G6iP0ApYg80C43A'
Benchmarking 1 thread ...
76 c/s real, 77 c/s virtual (127 hashes in 1.65 seconds)
Benchmarking 32 threads ...
1040 c/s real, 32 c/s virtual (1905 hashes in 1.83 seconds)
[solar@super yescrypt-0.5]$
```

yescrypt status (as of May 2014)

It is our duty to point out that:

- (Incompatible) tweaks to yescrypt are expected as we finalize it
- Proper password hashing goes hand in hand with a corresponding password strength policy, and both should be chosen and tuned with a threat model in mind

- We're ready to assist with experimental yescrypt deployments
- We also have a (mature) password policy enforcement tool set, passwdqc
  - libpasswdqc, PAM module, command-line program, PHP wrapper
  - passwdqc for Windows Active Directory in private beta (ask!)

Thanks

We'd like to thank

- Colin Percival, whose scrypt is the basis for yescrypt
- Niels Provos and David Mazieres, whose bcrypt inspired yescrypt's approach at GPU attack resistance at low memory settings
- All contributors to Password Hashing Competition, which yescrypt has been submitted to
  - https://password-hashing.net
  - PHC discussions mailing list has been very helpful in shaping up the design of yescrypt
- DARPA Cyber Fast Track

Questions?

Solar Designer <solar@openwall.com>

@solardiz @Openwall

http://www.openwall.com