

# script: A new key derivation function

Doing our best to thwart TLAs armed with ASICs

Colin Percival  
Tarsnap  
cperciva@tarsnap.com

May 9, 2009

# script: A new key derivation function

## Making bcrypt obsolete

Colin Percival  
Tarsnap  
cperciva@tarsnap.com

May 9, 2009

# script: A new key derivation function

Are you sure your SSH keys are safe?

Colin Percival

Tarsnap

[cperciva@tarsnap.com](mailto:cperciva@tarsnap.com)

May 9, 2009

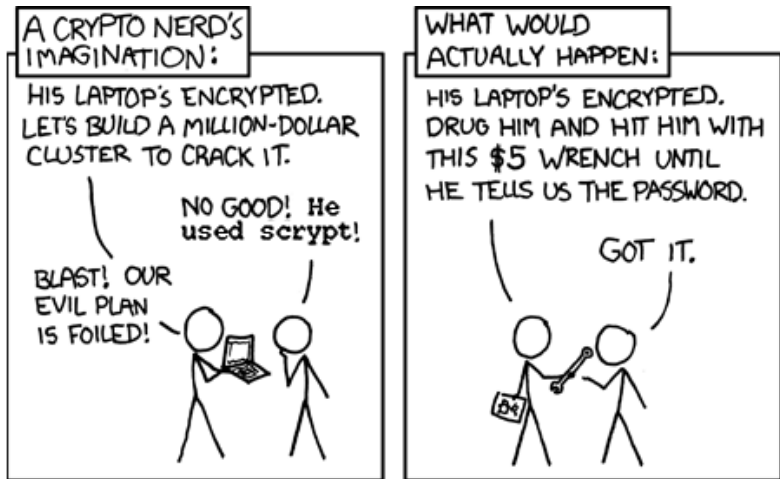
# What are key derivation functions?

- You have a password.
- You want to generate a *derived key* from that password.
  - Verifying passwords for user authentication.
  - Encrypting or signing files.
- In most situations where passwords are used, they are passed to a key derivation function first.
  - In most situations where key derivation functions aren't used, they should be!
- Examples of key derivation functions:
  - DES CRYPT [R. Morris, 1979]
  - MD5 CRYPT [P. H. Kamp, 1994]
  - bcrypt [N. Provos and D. Mazières, 1999]
  - PBKDF2 [B. Kaliski, 2000]
  - MD5 (not really a key derivation function!)

# Security of key derivation functions

- Attack model: Assume that the attacker can mount an offline attack.
  - Attacker has access to `/etc/master.passwd` and wants to find the users' passwords.
  - Attacker has an encrypted file and wants to decrypt it.
- For all reasonable key derivation functions, the only feasible attack is to repeatedly try passwords until you find the right one.
  - This is called a “brute force” attack.
- If it takes twice as long to check if a password is correct, it will take twice as long to find the right password.
  - ... as long as the attacker is using the same software as you.

# Hardware-based brute force attacks



CREDIT: Randall Munroe / xkcd.com

# Hardware-based brute force attacks

- Some organizations have the resources to design and fabricate custom password-cracking integrated circuits.
  - The US National Security Agency
  - The UK Government Communications Headquarters?
  - The Communications Security Establishment of Canada?
  - The Chinese government?
  - Organized crime?
  - The Electronic Frontier Foundation?
- Using ASICs, it is possible to pack many copies of a cryptographic circuit onto a single piece of silicon.
- Moore's law: Every 18–24 months, a new generation of semiconductor manufacturing processes makes CPUs faster.
  - . . . password-cracking ASICs get faster AND can fit more copies of a password-cracking circuit.

# Hardware brute-force attack cost

- The cost of a hardware brute-force attack is measured in dollar-seconds.
  - Password cracking is embarrassingly parallel, so if you use twice as much hardware you can crack the key in half the time.
- Cost of ASICs  $\asymp$  size of ASICs.
  - A strong key derivation function is one which can only be computed by using a large circuit for a long time.
- J. Kelsey, B. Schneier, C. Hall and D. Wagner, 1998: Use “32-bit arithmetic and moderately large amounts of RAM”.
  - An example of a “moderately large amount of RAM”: 1 kB.
- If we use a *ridiculously* large amount of RAM, hardware attacks will be even more expensive.



## Definition

A *memory-hard* algorithm on a Random Access Machine is an algorithm which uses  $S(n)$  space and  $T(n)$  operations, where  $S(n) \in \Omega(T(n)^{1-\epsilon})$ .

- Conceptually, a memory-hard algorithm is one which comes close to using the largest amount of storage possible for an algorithm with the same running time.
  - ... and consequently the largest circuit area possible.
- The HEKS key derivation algorithm [A.G. Reinhold, 1999] is memory-hard, but it isn't very secure, since it can be effectively parallelized.
- Secure key derivation functions require a large die area *and* a lot of time to compute.

# Sequential memory-hard functions

## Definition

A *sequential memory-hard function* is a function which

- (a) can be computed by a memory-hard algorithm on a Random Access Machine in  $T(n)$  operations; and
- (b) cannot be computed on a Parallel Random Access Machine with  $S^*(n)$  processors and  $S^*(n)$  space in expected time  $T^*(n)$  where  $S^*(n)T^*(n) = O(T(n)^{2-x})$  for any  $x > 0$ .

- Not only do memory-hard functions require lots of storage, but they also cannot be parallelized efficiently.
- If we can find a key derivation function which is sequential memory-hard, it should be very secure against hardware attack.

## Algorithm (ROMix)

Given a hash function  $H$ , an input  $B$ , and an integer parameter  $N$ , compute

$$V_i = H^i(B) \quad 0 \leq i < N$$

and  $X = H^N(B)$ , then iterate

$$j \leftarrow \text{Integerify}(X) \bmod N$$

$$X \leftarrow H(X \oplus V_j)$$

$N$  times; and output  $X$ .

- The function *Integerify* can be any bijection from  $\{0, 1\}^k$  to  $\{0 \dots 2^k - 1\}$ .
- ROMix fills  $V$  with pseudorandom values, then accesses them in a pseudorandom order.

## Theorem

*Under the random oracle model, the class of functions ROMix are sequential memory-hard.*

- The random oracle model is a very standard assumption in proofs relating to hash functions.
- The proof is roughly 2 pages long.
  - I could probably spend my entire talk explaining the proof.
  - ... but I won't.
  - If you're interested, go read the paper I wrote about this.
- It's much easier to prove that an algorithm runs in specified time and space than to prove a minimum bound on the time and space used by *any* algorithm which computes a function.

- Use PBKDF2 to convert a password into a bitstream.
- Feed this bitstream to ROMix.
- Feed the output of ROMix back to PBKDF2 to generate the derived key.
- Cryptographic primitives used:
  - HMAC-SHA256
  - Salsa20/8 core
- The Salsa20/8 core outputs lots of bits very fast, which means that scrypt can use lots of memory.
  - Approximately 1 byte of RAM per 10 clock cycles on a Core 2 processor.
  - We can quickly require a large semiconductor area.

# Estimating hardware brute force attack costs

- It's hard to get accurate information about how much it costs to build password-cracking machines.
  - Oddly enough, the NSA doesn't publish this data.
- The best we can do for most KDFs is to count cryptographic operations and assume that they are responsible for most of the time and die area.
  - This is probably a fairly accurate approximation, since key derivation functions only have a very small amount of non-cryptographic computations.
- For scrypt we also need to look at the die area required for storage.

# Estimating hardware brute force attack costs

- Approximate circuit complexity and performance for cryptographic primitives, based on a 130 nm semiconductor process:
  - A DES circuit with  $\approx 4000$  gates of logic can encrypt data at 2000 Mbps.
  - An MD5 circuit with  $\approx 12000$  gates of logic can hash data at 2500 Mbps.
  - A SHA256 circuit with  $\approx 20000$  gates of logic can hash data at 2500 Mbps.
  - A Blowfish circuit with  $\approx 22000$  gates of logic and 4 kiB of SRAM can encrypt data at 1000 Mbps.
  - A Salsa20/8 circuit with  $\approx 24000$  gates of logic can output a keystream at 2000 Mbps.
- I'm using 130 nm as a reference point because this is what I could get the most data for.

# Estimating hardware brute force attack costs

- Very approximate estimates of VLSI area and cost:
  - Each gate of random logic requires  $\approx 5 \mu\text{m}^2$  of VLSI area.
  - Each bit of SRAM requires  $\approx 2.5 \mu\text{m}^2$  of VLSI area.
  - Each bit of DRAM requires  $\approx 0.1 \mu\text{m}^2$  of VLSI area.
  - VLSI circuits cost  $\approx 0.1\$/\text{mm}^2$ .
- These values are based on a 130 nm process circa 2002.
- These values have a very wide error margin.
  - Non-cryptographic parts of ASICs (e.g., I/O), chip packaging, boards, power supplies, and operating costs could increase password-cracking costs by a factor of 10.
  - Improvements in semiconductor technology since 2002 could reduce password-cracking costs by a factor of 10.
  - Improved cryptographic circuits could reduce costs by a factor of 10.



# Key derivation functions

- Non-parameterized KDFs:
  - DES CRYPT
  - MD5 CRYPT
  - MD5
- KDFs tuned for interactive logins ( $t \leq 100$  ms):
  - PBKDF2-HMAC-SHA256,  $c = 86000$
  - bcrypt,  $cost = 11$
  - scrypt,  $N = 2^{14}, r = 8, p = 1$
- KDFs tuned for file encryption ( $t \leq 5$  s):
  - PBKDF2-HMAC-SHA256,  $c = 4300000$
  - bcrypt,  $cost = 16$
  - scrypt,  $N = 2^{20}, r = 8, p = 1$
- Running time based on a 2.5 GHz Core 2 (aka. my laptop).

- 6 lower-case letters; e.g., “sfgroy”.
- 8 lower-case letters; e.g., “ksuvnwyf”.
- 8 characters selected from the 95 printable 7-bit ASCII characters; e.g., “6,uh3y[a”.
- 10 characters selected from the 95 printable 7-bit ASCII characters; e.g., “H.\*W8Jz&r3”.
- A 40-character string of text; e.g., “This is a 40-character string of English”.
  - Entropy estimated according to formula from NIST: 1st character has 4 bits of entropy; 2nd–8th characters have 2 bits of entropy each; 9th–20th characters have 1.5 bits of entropy each; 21st and later characters have 1 bit of entropy each.

# Estimated brute force attack costs

Estimated cost of hardware to crack a password in 1 year.

KDF	6 letters	8 letters	8 chars	10 chars	40-char text
DES CRYPT	< \$1	< \$1	< \$1	< \$1	< \$1
MD5	< \$1	< \$1	< \$1	\$1.1k	\$1
MD5 CRYPT	< \$1	< \$1	\$130	\$1.1M	\$1.4k
PBKDF2 (100 ms)	< \$1	< \$1	\$18k	\$160M	\$200k
bcrypt (95 ms)	< \$1	\$4	\$130k	\$1.2B	\$1.5M
scrypt (64 ms)	< \$1	\$150	\$4.8M	\$43B	\$52M
PBKDF2 (5.0 s)	< \$1	\$29	\$920k	\$8.3B	\$10M
bcrypt (3.0 s)	< \$1	\$130	\$4.3M	\$39B	\$47M
scrypt (3.8 s)	\$900	\$610k	\$19B	\$175T	\$210B

# KDF brute force attack costs

- When used for interactive logins, `scrypt` is ...
  - $\approx 2^5$  times more expensive to attack than `bcrypt`,
  - $\approx 2^8$  times more expensive to attack than `PBKDF2`,
  - and  $\approx 2^{15}$  times more expensive to attack than `MD5 CRYPT`.
- When used for file encryption, `scrypt` is ...
  - $\approx 2^{12}$  times more expensive to attack than `bcrypt`,
  - $\approx 2^{15}$  times more expensive to attack than `PBKDF2`,
  - and  $\approx 2^{37}$  times more expensive to attack than `MD5`.
- `openssl enc` uses `MD5` as a key derivation function.
- `OpenSSH` uses `MD5` as a key derivation function for passphrases on key files.
  - Are you sure that your SSH keys are safe?

- More details at <http://www.tarsnap.com/scrypt/>.
  - Source code for scrypt.
  - A simple file encryption/decryption utility.
  - A 16-page paper.

## Questions?