



1. Quick intro to PBKDF2
2. The standard is bad
3. Your implementation is bad
4. **A faster PBKDF2**

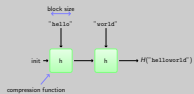
2015-08-05

pbkdf2

Intro: Merkle-Damgård hash functions

Intro: Merkle-Damgård hash functions

Basic construction of most hash functions: MD5, SHA-1, SHA-2.



2015-08-05

pbkdf2

└─ Intro: HMAC

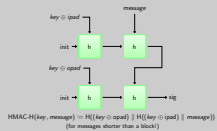
Intro: HMAC

Making secure symmetric signatures out of MD hash functions.



Intro: HMAC innards

Intro: HMAC innards



2015-08-05

pbkdf2

Intro: PBKDF2

Intro: PBKDF2

Slowly derive a key from a password and salt.



- Parameterised with a PRF, usually HMAC.
- Tunable computation cost, with iteration count.
- Origin: RSA labs, 1999. Described in PKCS#5 and then RFC2898.

Intro: PBKDF2

Intro: PBKDF2

Iteration count choice

1. Choose computation budget (say, 50ms).
2. Find iteration count which takes that long with your implementation.

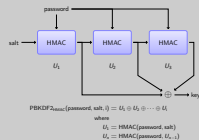
Performance

Performance profile is important for defenders. Aim: to maximise attacker work for defender computation budget.

For simplicity

Assume salts, passwords are less than block size. Assume output length is not more than hash output size.

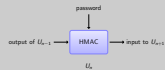
Intro: PBKDF2_{HMAC} with 3 iterations



└ PBKDF2: perf vs. iteration count

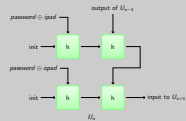
PBKDF2: perf vs. iteration count

One HMAC per iteration.



How many compression function applications?

└ PBKDF2: perf vs. iteration count



Conclusion: $4i$ compression function applications for i iterations.

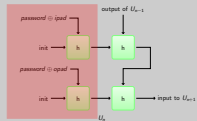
2015-08-05

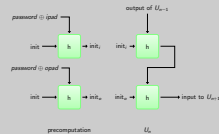
pbkdf2

Nope!

Nope!

This is suboptimal. Neither of the standards mention this, or even describe the expected performance of





Actually $2 + 2i$ compression function applications for i iterations.

- Do HMAC key setup once, reuse that work.
- Better locality of reference too.

Survey of defender implementations

Survey of defender implementations

- FreeBSD 10
- GRUB 2.0
- Truecrypt 7.1a
- Android (disk encryption)
- Android BouncyCastle fork
- Django
- OpenSSL
- Python core (≥ 3.4)
- Python (pypt-pbkdf2)
- Ruby (pbkdf2 gem)
- Go (go.crypto)
- Apple CoreCrypto (disassembly)
- OpenBSD
- PolarSSL/mbedTLS
- CyaSSL/wolfSSL
- SJCL
- Java (OpenJDK)
- Common Lisp (ironclad)
- Perl (Crypt::PBKDF2)
- PHP5
- .NET framework
- scrypt/yescrypt¹
- BouncyCastle

¹never called for scrypt/yescrypt with iterations 1 to 1

└ Our survey says...

Our survey says...

Good: compute 2 + 2/ blocks

- ▶ S/JCL
- ▶ OpenSSL (after Nov 2013)
- ▶ Python core (≥ 3.4)
- ▶ Django (CVE-2013-1443, sc00bz)
- ▶ BouncyCastle (≥ 1.49)
- ▶ Apple CoreCrypto (?)

Slow: compute 4/ blocks

- ▶ FreeBSD 10
- ▶ OpenBSD

Slow: compute 4/ blocks

- ▶ GRUB 2.0
- ▶ Python (pypi pbkdf2)
- ▶ Ruby (pbkdf2 gem)
- ▶ Go (go.crypto)
- ▶ PolarSSL/mbedTLS
- ▶ CyaSSL/wolfSSL
- ▶ Java (OpenJDK)
- ▶ Perl (Crypt::PBKDF2)
- ▶ PHP
- ▶ .NET framework
- ▶ ...

- Note: not blaming implementors.
- Minor structural changes in PBKDF2 would fix this for all impls.
- Failing that, doc changes would likely have improved matters.

Selected performance measurements

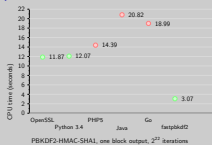
Selected performance measurements

- Question: how much practical difference does this make?
- Let's measure PBKDF2-HMAC-SHA1 for large iteration count (2^{23})

Measured on Intel Atom N2800 (1.86GHz), best of five runs, CPU time in user mode.

└ Selected performance measurements

Selected performance measurements



- OpenSSL is a good baseline to compare against.
- Python3.4 has the same basic impl as OpenSSL = same perf.
- Others are slow.
- Patch for PHP5 is upstream, gives good improvement.
- If we assume similar improvements for others, they end up competitive.
- But we can do better!

2015-08-05

pbkdf2

└ fastpbkdf2

fastpbkdf2

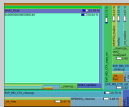
A faster PBKDF2-HMAC-(SHA-1,SHA-256,SHA-512) for defenders.

- ▶ About 400 lines of C99.
- ▶ Uses OpenSSL libcrypto's hash functions.
- ▶ Public domain (CC0).
- ▶ <https://github.com/ctz/fastpbkdf2/>

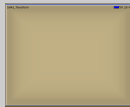
pbkdf2

└ fastpbkdf2

fastpbkdf2



OpenSSL



fastpbkdf2

- boxplot from kcache-grind/valgrind-callgrind.
- Area roughly proportional to cpu time.
- No memory copies, allocations, conversions, padding in inner loop.

└ But wait, there's more!

But wait, there's more!

I lied about not talking about long PBKDF2 outputs.

- ▶ You repeat the whole algorithm (with a counter appended to the salt) and concatenate the outputs.
- ▶ All(?) defender implementations do this sequentially.
- ▶ Attackers often don't need to compute all blocks to win.
- ▶ Really, you don't want this. There are better ways (e.g., run PBKDF2 once to spend time, then use result as input to PBKDF2 with iterations::1 to stretch output to required length).

But, in any case, `fastpbkdf2` optionally parallelises this.

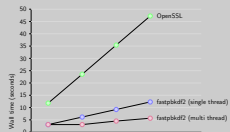
- Attackers can parallelise this freely, or (for cracking database dumps) perhaps don't even need to compute all the blocks at all.
- So you really ought not to ask for more than one block of PBKDF2 output. It's extremely broken.
- However, if you do need that (backwards compat), `fastpbkdf2` optionally parallelises this computation too.
- Uses OpenMP for portability.

2015-08-05

pbkdf2

└ But wait, there's more!

But wait, there's more!



PBKDF2-HMAC-SHA1, one-four blocks output, 2¹⁰ iterations, two cores + HT

└ Parting thoughts...

Parting thoughts...

- PBKDF2 is a poor design, and described in an unhelpful way by its authors.
- Most implementations waste time and power.
- If you use PBKDF2, you can probably drop in a faster implementation (and either increase security margin, or improve time/power performance.)