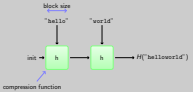
1. Quick intro to PBKDF2
2. The standard is bad
3. Your implementation is bad
4. A faster PBKDF2

└Intro: Merkle-Damgård hash functions

Intro: Merkle-Damgård hash functions

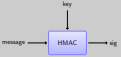
Basic construction of most hash functions: MD5, SHA-1, SHA-2.



└Intro: HMAC

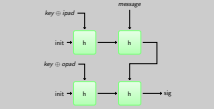
Intro: HMAC

Making secure symmetric signatures out of MD hash functions.



Intro: HMAC innards

Intro: HMAC innards



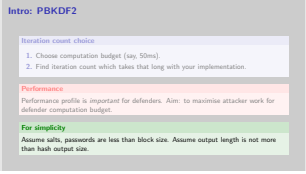
HMAC-H(*key*, *message*) = H((*key* ⊕ opad) ‖ H((*key* ⊕ ipad) ‖ *message*))
(for messages shorter than a block!)

Intro: PBKDF2

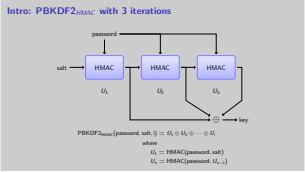
Slowly derive a key from a password and salt.

- 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



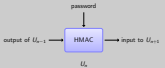
Intro: PBKDF2$_{HMAC}$ with 3 iterations



└PBKDF2: perf vs. iteration count

PBKDF2: perf vs. iteration count

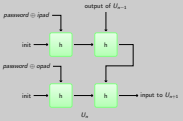
One HMAC per iteration.



How many compression function applications?

└PBKDF2: perf vs. iteration count

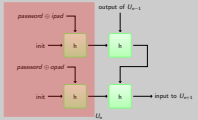
PBKDF2: perf vs. iteration count



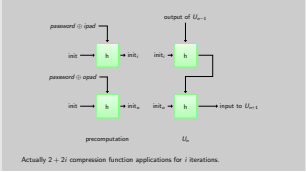
Conclusion: $4i$ compression function applications for i iterations.

Nope!

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



pbkdf2



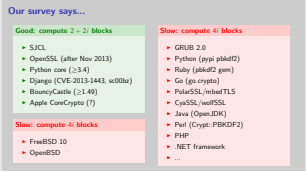
- Do HMAC key setup once, reuse that work.
- Better locality of reference too.

└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 (pypi 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/yescrypt[1]
- BouncyCastle

[1]never called for scrypt/yescrypt with iterations != 1

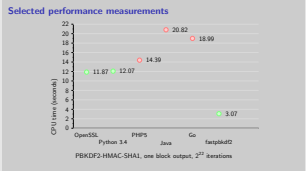


- Note: not blaming implementors.
- Minor structural changes in PBKDF2 would fix this for all impls.
- Failing that, doc changes would likely have improved matters.

Selected performance measurements

- Question: how much practical difference does this make?
- Let's measure PBKDF2-HMAC-SHA1 for large iteration count (2^{22})

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



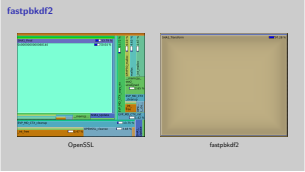
- OpenSSL is a good baseline to compare against.
- Python3.4 has the same basic impl as OpenSSL = same perf.
- Others are slow.
- Patch for PHP5 is upstream, gives good improvement.
- If we assume similar improvements for others, they end up competitive.
- But we can do better!

└fastpbkdf2

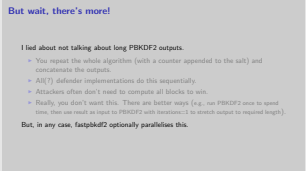
fastpbkdf2

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/

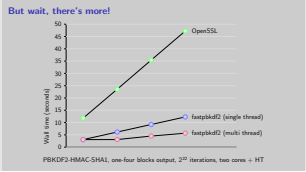


- boxplot from kcachegrind/valgrind-callgrind.
- Area roughly proportional to cpu time.
- No memory copies, allocations, conversions, padding in inner loop.



- 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.

But wait, there's more!



└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.)