PBKDF2: performance matters

Joseph Birr-Pixton
@jpixton
http://jbp.io/

1. Quick intro to PBKDF2

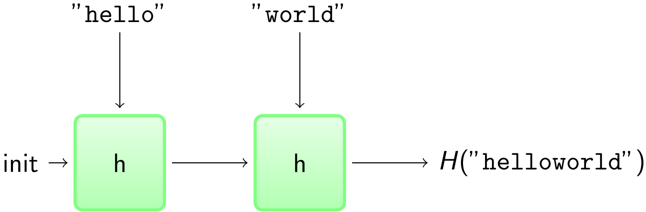
1. Quick intro to PBKDF2
2. The standard is bad

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

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

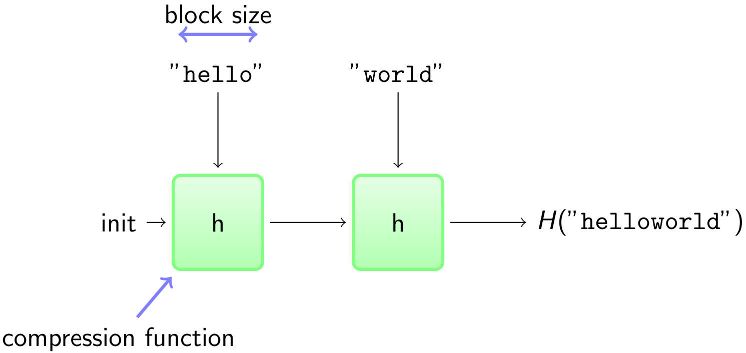
Intro: Merkle-Damgård hash functions

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



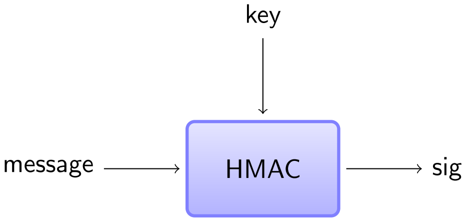
Intro: Merkle-Damgård hash functions

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



Intro: HMAC

Making secure symmetric signatures out of MD hash functions.



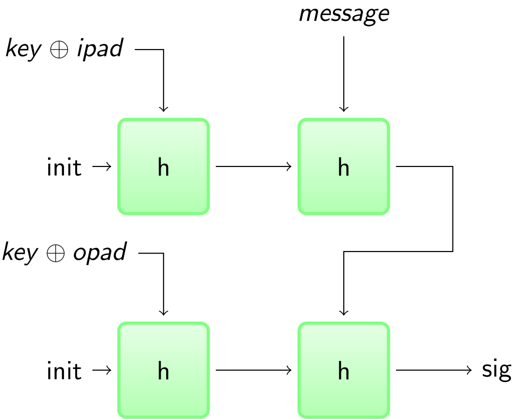
key

message

HMAC

sig

Intro: HMAC innards

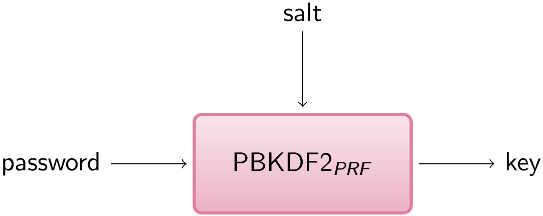


HMAC-H(key, $message$) := H(key ⊕ opad ∥ H(key ⊕ ipad ∥ $message$))

(for messages shorter than a block!)

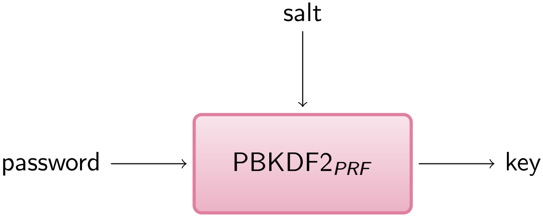
Intro: PBKDF2

Slowly derive a key from a password and salt.



Intro: PBKDF2

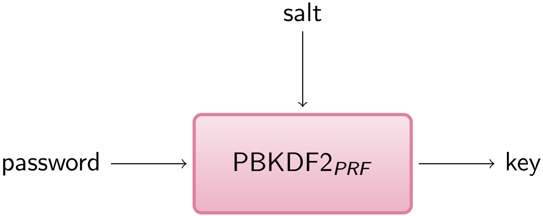
Slowly derive a key from a password and salt.



- Parameterised with a PRF, usually HMAC.

Intro: PBKDF2

Slowly derive a key from a password and salt.



salt

password

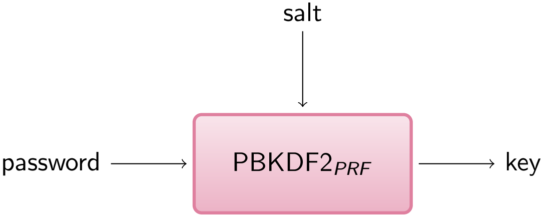
PBKDF2_{PRF}

key

- Parameterised with a PRF, usually HMAC.
- Tunable computation cost, with iteration count.

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

Iteration count choice

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

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.

Intro: PBKDF2

Iteration count choice

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

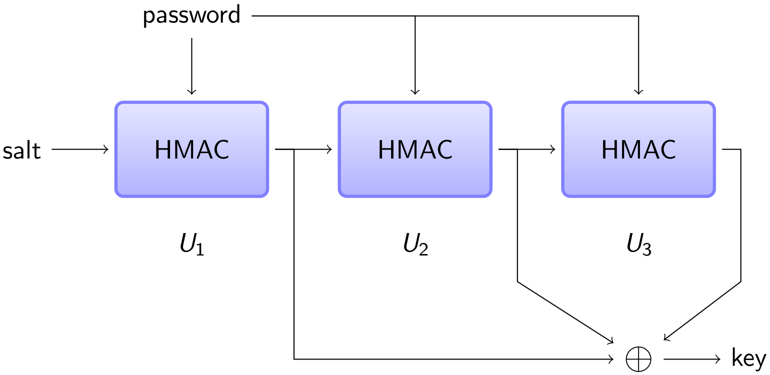
Performance

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

Simplification

PBKDF2 can produce arbitrary length output.
We're going to ignore this capability: assume it produces the same length output as the underlying hash.

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



$$\mathrm{PBKDF2}_{\mathrm{HMAC}}(\mathrm{password}, \mathrm{salt}, \mathrm{i}) := U_1 \oplus U_2 \oplus \cdots \oplus U_{\mathrm{i}}$$

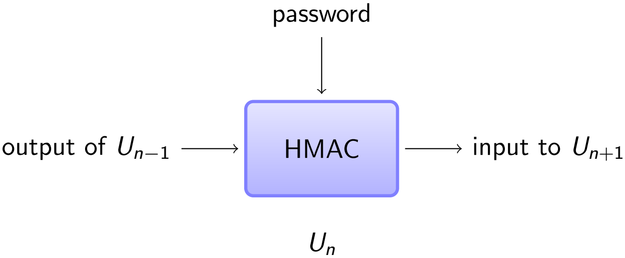
where

$$U_1 := \mathrm{HMAC}(\mathrm{password}, \mathrm{salt})$$

$$U_n := \mathrm{HMAC}(\mathrm{password}, U_{n-1})$$

PBKDF2: perf vs. iteration count

One HMAC per iteration.



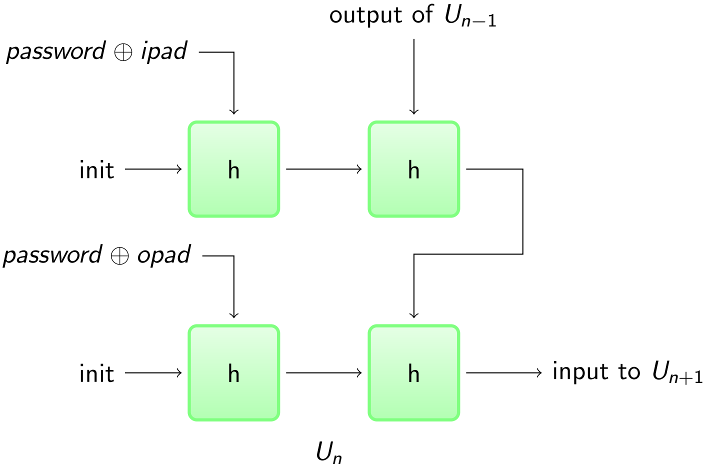
password

output of U_{n-1} ⟶ HMAC ⟶ input to U_{n+1}

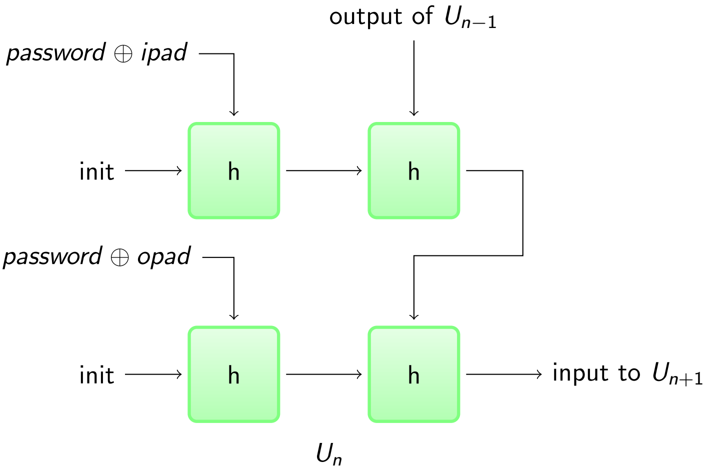
U_n

How many compression function applications?

PBKDF2: perf vs. iteration count



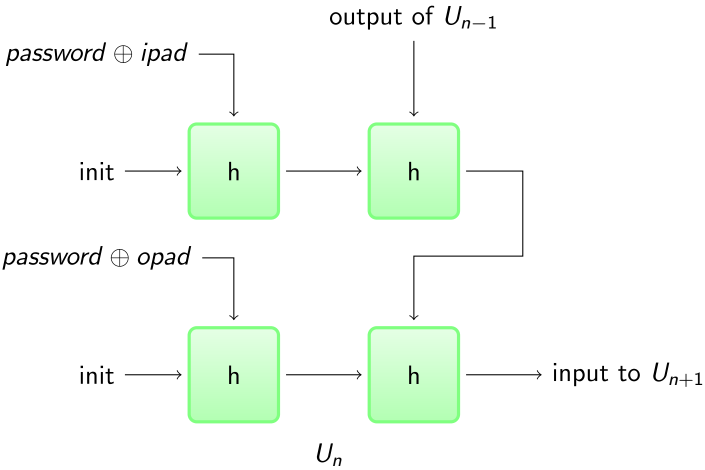
PBKDF2: perf vs. iteration count



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

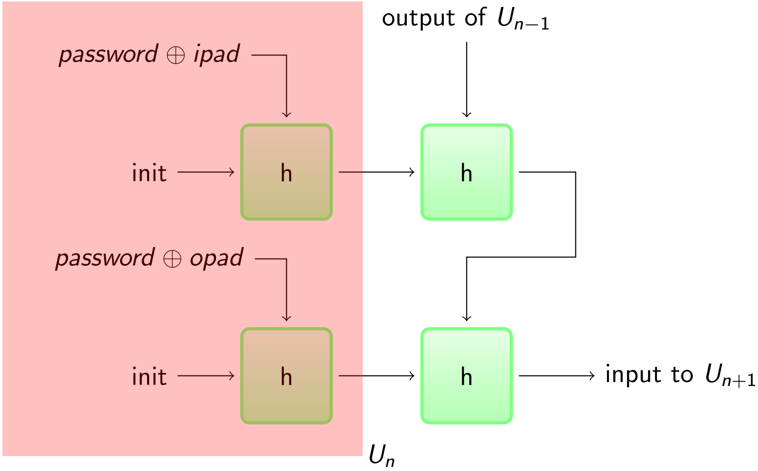
Nope!

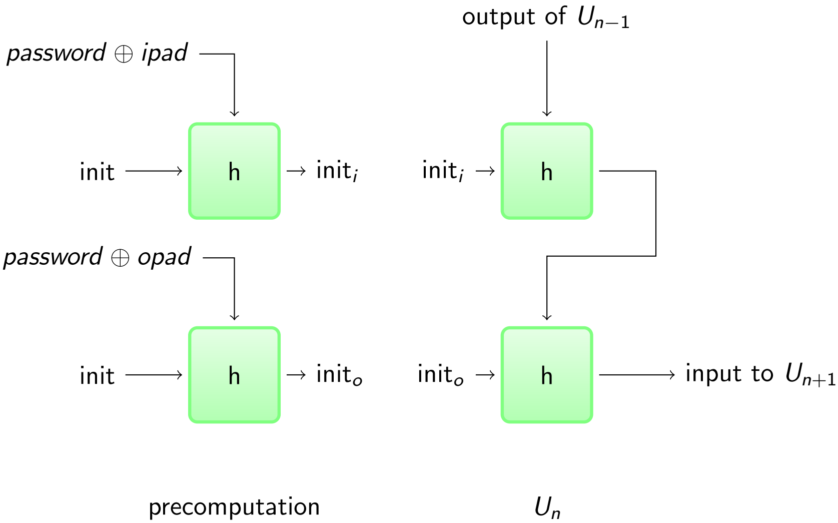
This is suboptimal. Neither of the standards mention this, or even describe the expected performance :(

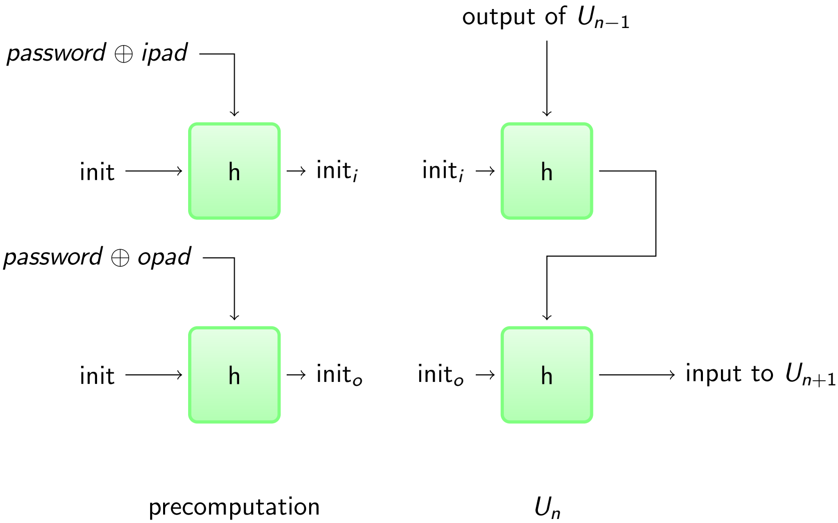


Nope!

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




output of U_{n-1}
$password \oplus ipad$
init
h
→ init_i
init_i →
h
$password \oplus opad$
init
h
→ init_o
init_o →
h
input to U_{n+1}
precomputation
U_n



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

Survey of defender implementations

I looked at the following PBKDF2s:

- FreeBSD 10
- GRUB 2.0
- Truecrypt 7.1a
- Android (disk encryption)
- Android (BouncyCastle)
- Django
- OpenSSL
- Python core ($\geq$3.4)
- Python (pypi pbkdf2)
- Ruby (pbkdf2 gem)
- Go (go.crypto)
- OpenBSD
- PolarSSL/mbedTLS
- CyaSSL/wolfSSL
- SJCL
- Java
- Common Lisp (ironclad)
- Perl (Crypt::PBKDF2)
- PHP5
- .NET framework
- scrypt/yescrypt[1]
- BouncyCastle

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

Our survey says...

Good: compute $2 + 2i$ blocks

- SJCL

Our survey says...

Good: compute $2 + 2i$ blocks

- SJCL
- OpenSSL (after Nov 2013)
- Python core ($\geq$3.4)
- Django (CVE-2013-1443, sc00bz)
- BouncyCastle ($\geq$1.49)

Our survey says...

Good: compute $2 + 2i$ blocks

- SJCL
- OpenSSL (after Nov 2013)
- Python core ($\geq$3.4)
- Django (CVE-2013-1443, sc00bz)
- BouncyCastle ($\geq$1.49)

Slow: compute $4i$ blocks

- FreeBSD 10
- OpenBSD
- GRUB 2.0

Our survey says...

Good: compute $2 + 2i$ blocks

- SJCL
- OpenSSL (after Nov 2013)
- Python core (≥3.4)
- Django (CVE-2013-1443, sc00bz)
- BouncyCastle (≥1.49)

Slow: compute $4i$ blocks

- FreeBSD 10
- OpenBSD
- GRUB 2.0

Slow: compute $4i$ blocks

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

Selected performance measurements

- Question: how much practical difference does this make?

Selected performance measurements

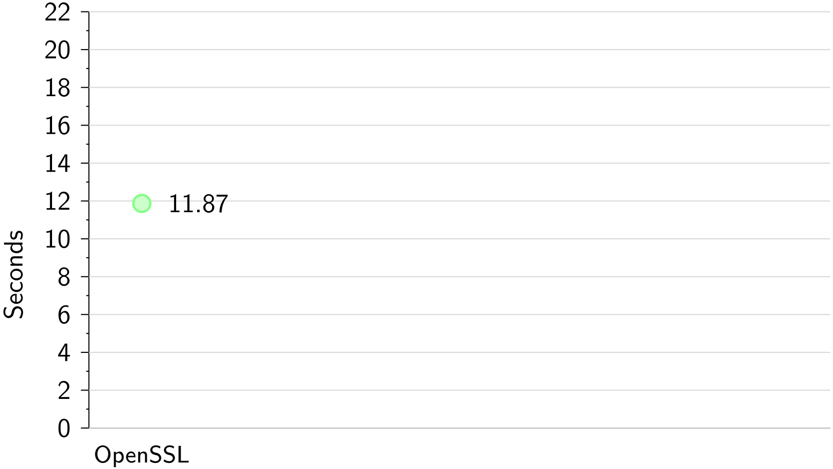
- Question: how much practical difference does this make?
- Let's measure PBKDF2-HMAC-SHA1 for large iteration count (2^{22})

Selected performance measurements

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

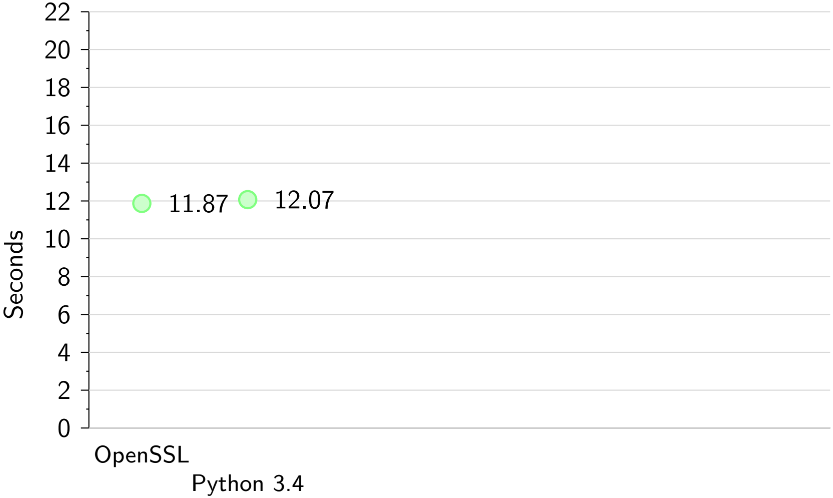
Measured on Intel Atom N2800 (1.86GHz), best of five runs, CPU time in user mode.

Selected performance measurements



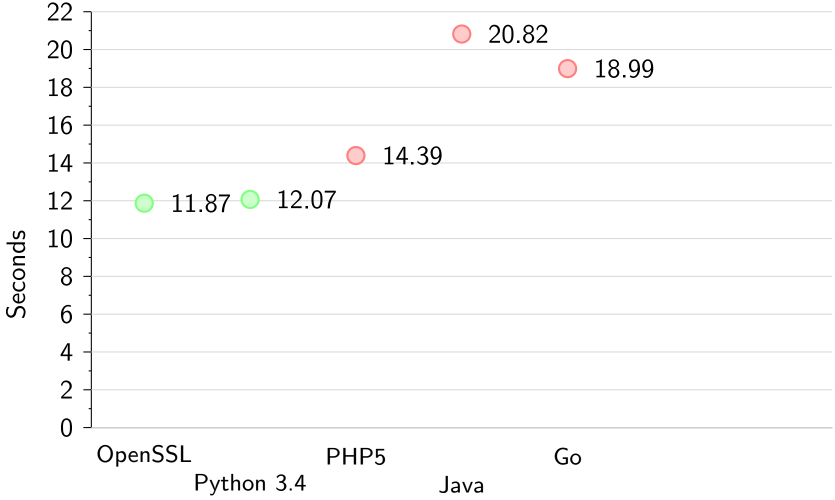
PBKDF2-HMAC-SHA1, one block output, 2^{22} iterations

Selected performance measurements



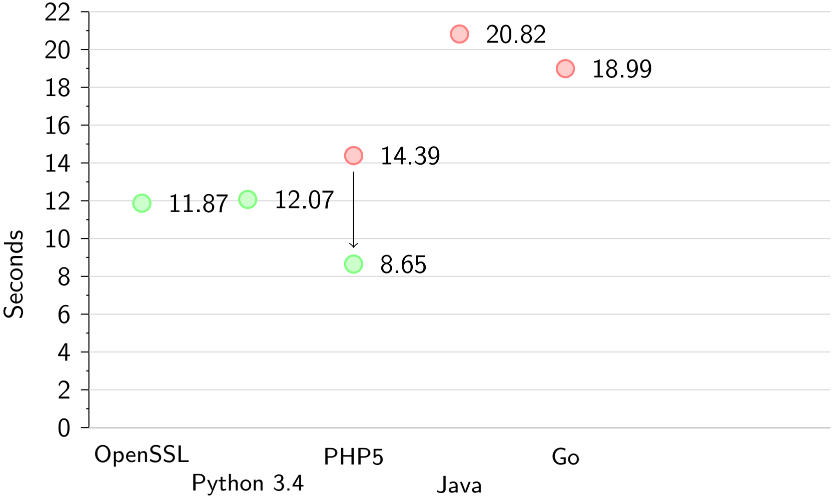
PBKDF2-HMAC-SHA1, one block output, 2^{22} iterations

Selected performance measurements



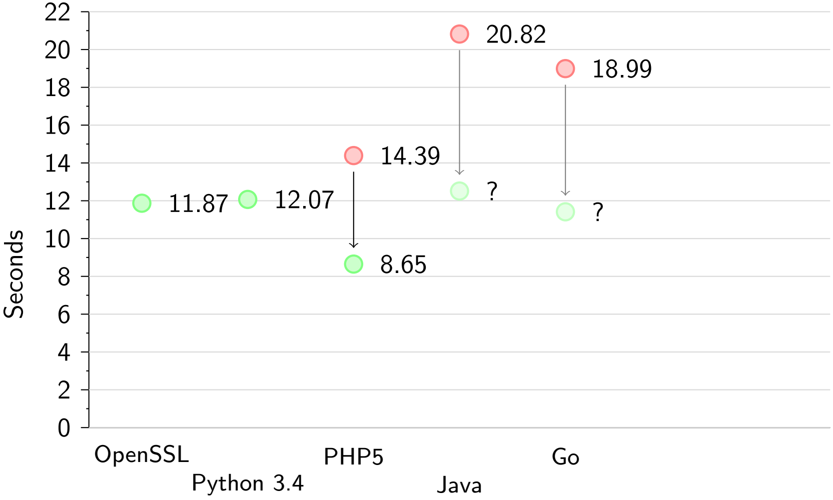
PBKDF2-HMAC-SHA1, one block output, 2^{22} iterations

Selected performance measurements



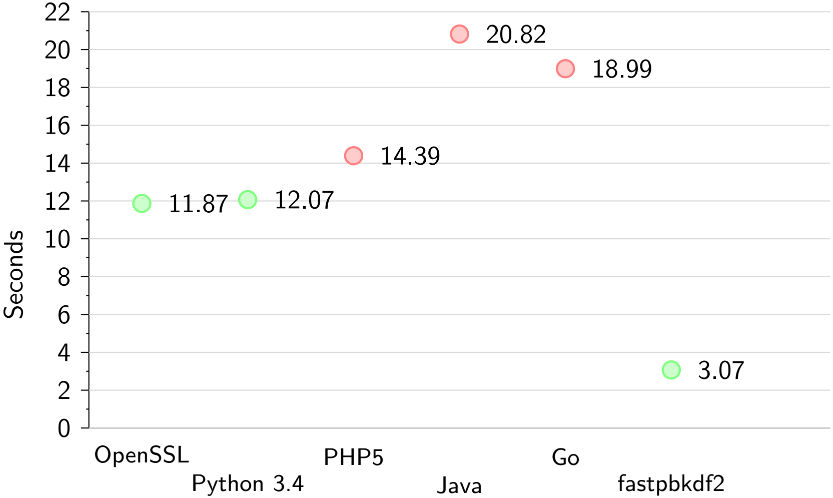
PBKDF2-HMAC-SHA1, one block output, 2^{22} iterations

Selected performance measurements



PBKDF2-HMAC-SHA1, one block output, 2^{22} iterations

Selected performance measurements



PBKDF2-HMAC-SHA1, one block output, 2^{22} iterations

fastpbkdf2

A faster PBKDF2-HMAC-{SHA-1,SHA-256,SHA-512} for defenders.

- About 400 lines of C99.

fastpbkdf2

A faster PBKDF2-HMAC-{SHA-1,SHA-256,SHA-512} for defenders.

- About 400 lines of C99.
- Uses OpenSSL libcrypto's hash functions.

fastpbkdf2

A faster PBKDF2-HMAC-$\{$SHA-1,SHA-256,SHA-512$\}$ for defenders.

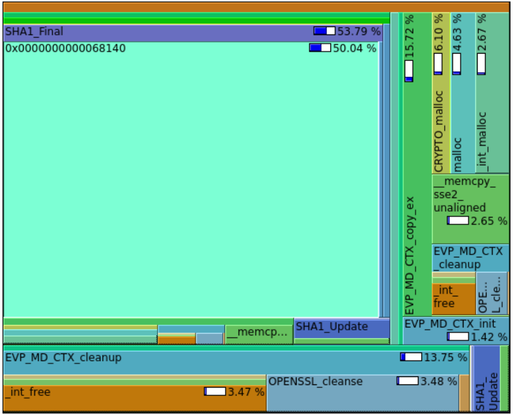
- About 400 lines of C99.
- Uses OpenSSL libcrypto's hash functions.
- Public domain (CC0).

fastpbkdf2

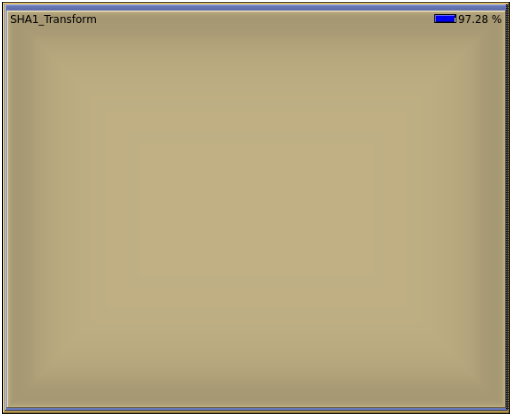
A faster PBKDF2-HMAC-{SHA-1,SHA-256,SHA-512} for defenders.

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

fastpbkdf2



OpenSSL



fastpbkdf2

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.

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.

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.

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[2].

[2] run pbkdf2 once to spend time, then use that result as input to pbkdf2 with iterations=1 to expand output.

But wait, there's more!

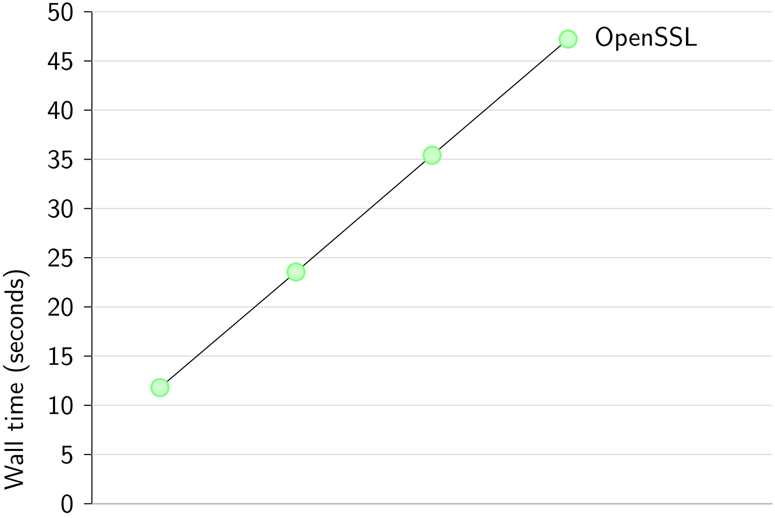
I lied about not talking about long PBKDF2 outputs.

- 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[2].

But, in any case, fastpbkdf2 optionally parallelises this.

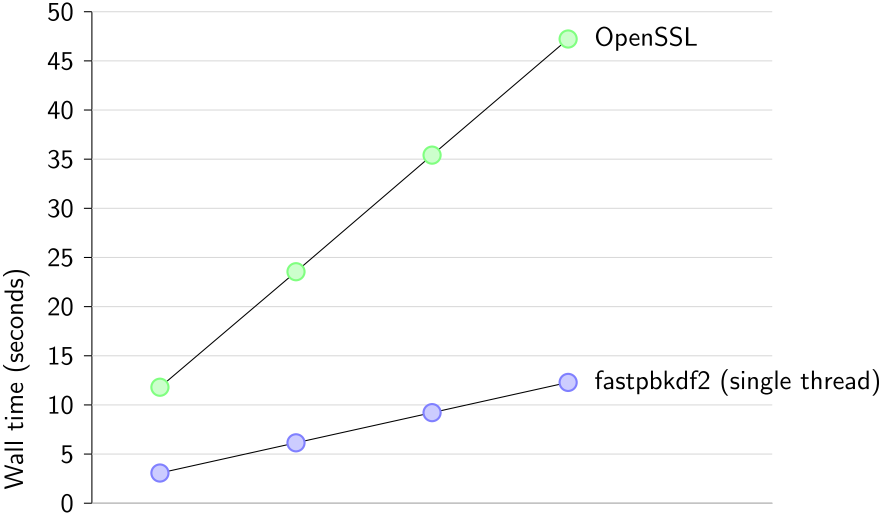
[2] run pbkdf2 once to spend time, then use that result as input to pbkdf2 with iterations=1 to expand output.

But wait, there's more!



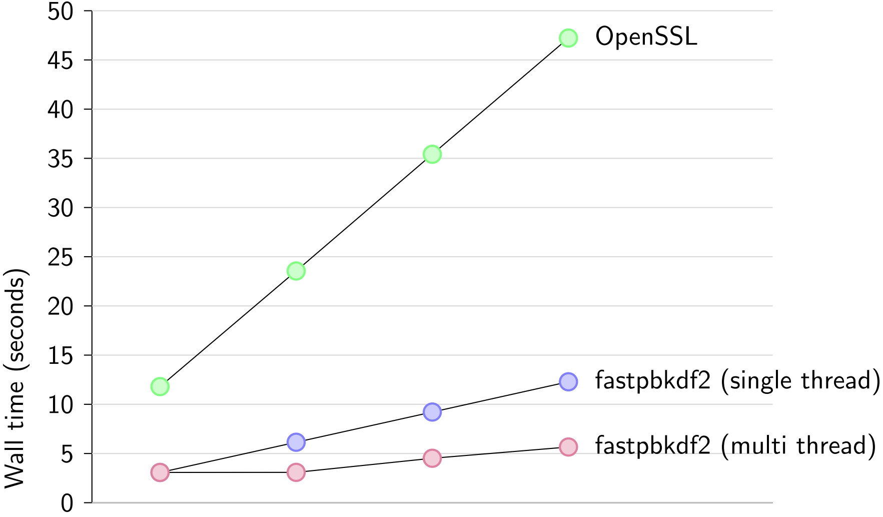
PBKDF2-HMAC-SHA1, one-four blocks output, 2^{22} iterations, two cores + HT

But wait, there's more!



PBKDF2-HMAC-SHA1, one-four blocks output, 2^{22} iterations, two cores + HT

But wait, there's more!



PBKDF2-HMAC-SHA1, one-four blocks output, 2^{22} iterations, two cores + HT

Parting thoughts...

- PBKDF2 is a poor design, and described in an unhelpful way by its authors.

Parting thoughts...

- PBKDF2 is a poor design, and described in an unhelpful way by its authors.
- Most implementations waste time and power.

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.)

Thank you!

Questions?

Twitter: @jpixton
Mail: jbp@jbp.io
Web: https://jbp.io/
Slides and benchmarking code: https://github.com/ctz/talks/
fastpbkdf2 code: https://github.com/ctz/fastpbkdf2/