PBKDF2: performance matters

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

PBKDF2: quick intro

Purpose

Slowly convert a password + salt into a symmetric key of some length

PBKDF2: quick intro

Purpose

Slowly convert a password + salt into a symmetric key of some length

Origin

RSA labs, 1999. Described in PKCS#5 and then RFC2898

PBKDF2: quick intro

Usage

- Password verification (web sites, network services, etc.)
- Key derivation (disk encryption, key management, etc.)

PBKDF2: quick intro

Usage

- Password verification (web sites, network services, etc.)
- Key derivation (disk encryption, key management, etc.)

Performance

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

PBKDF2: quick intro

Usage

- Password verification (web sites, network services, etc.)
- Key derivation (disk encryption, key management, etc.)

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 from here on in: only considering the first block of output.

PBKDF2: how it was described

$$\mathsf{PBKDF2}_{\mathsf{PRF}}(\mathsf{pw}, \mathsf{salt}, \mathsf{i}) := U_1 \oplus U_2 \oplus \cdots \oplus U_{\mathsf{i}}$$

PBKDF2: how it was described

$$\mathsf{PBKDF2}_{\mathsf{PRF}}(\mathsf{pw}, \mathsf{salt}, \mathsf{i}) := U_1 \oplus U_2 \oplus \cdots \oplus U_{\mathsf{i}}$$

where

$$U_1 := \mathsf{PRF}(\mathsf{pw}, \mathsf{salt} \parallel 0_{32})$$

$$U_n := \mathsf{PRF}(\mathsf{pw}, U_{n-1})$$

PBKDF2: how it was described

$$\mathsf{PBKDF2}_{\mathsf{PRF}}(\mathsf{pw}, \mathsf{salt}, \mathsf{i}) := U_1 \oplus U_2 \oplus \cdots \oplus U_{\mathsf{i}}$$

where

$$U_1 := \mathsf{PRF}(\mathsf{pw}, \mathsf{salt} \parallel 0_{32})$$

$$U_n := \mathsf{PRF}(\mathsf{pw}, U_{n-1})$$

and typically

$$\mathsf{PRF}(\mathsf{pw}, \mathsf{x}) = \mathsf{HMAC\text{-}H}(\mathsf{pw}, \mathsf{x})$$

$$H = \text{SHA-1, SHA-256, SHA-512, or ...}$$

Zoom, enhance

The function $\text{PBKDF2}_{\text{HMAC-SHA-256}}$ is slow because it executes the SHA-256 compression function many times.

Zoom, enhance

The function $\text{PBKDF2}_{\text{HMAC-SHA-256}}$ is slow because it executes the SHA-256 compression function many times.

How many times?

Assumption: password and salt much shorter than SHA-256's 64-byte block size.

$$\text{HMAC-H}(k, m) := \text{H}(k \oplus \text{opad} \parallel \text{H}(k \oplus \text{ipad} \parallel m))$$

Zoom, enhance

The function $\text{PBKDF2}_{\text{HMAC-SHA-256}}$ is slow because it executes the SHA-256 compression function many times.

How many times?

Assumption: password and salt much shorter than SHA-256's 64-byte block size.

$$\text{HMAC-H}(k, m) := \text{H}(k \oplus \text{opad} \parallel \text{H}(k \oplus \text{ipad} \parallel m))$$

$$\text{block 1} : k \oplus \text{ipad}$$

Zoom, enhance

The function $\text{PBKDF2}_{\text{HMAC-SHA-256}}$ is slow because it executes the SHA-256 compression function many times.

How many times?

Assumption: password and salt much shorter than SHA-256's 64-byte block size.

$$\text{HMAC-H}(k, m) := \text{H}(k \oplus \text{opad} \parallel \text{H}(k \oplus \text{ipad} \parallel m))$$

block 1 : $k \oplus \text{ipad}$

block 2 : m

Zoom, enhance

The function $\text{PBKDF2}_{\text{HMAC-SHA-256}}$ is slow because it executes the SHA-256 compression function many times.

How many times?

Assumption: password and salt much shorter than SHA-256's 64-byte block size.

$$\text{HMAC-H}(k, m) := \text{H}(k \oplus \text{opad} \parallel \text{H}(k \oplus \text{ipad} \parallel m))$$

block 1 : $k \oplus \text{ipad}$

block 2 : m

block 3 : $k \oplus \text{opad}$

Zoom, enhance

The function $\text{PBKDF2}_{\text{HMAC-SHA-256}}$ is slow because it executes the SHA-256 compression function many times.

How many times?

Assumption: password and salt much shorter than SHA-256's 64-byte block size.

$$\text{HMAC-H}(k, m) := \text{H}(k \oplus \text{opad} \parallel \text{H}(k \oplus \text{ipad} \parallel m))$$

block 1 : $k \oplus \text{ipad}$

block 2 : m

block 3 : $k \oplus \text{opad}$

block 4 : block 2 output

Zoom, enhance

The function $\text{PBKDF2}_{\text{HMAC-SHA-256}}$ is slow because it executes the SHA-256 compression function many times.

How many times?

Assumption: password and salt much shorter than SHA-256's 64-byte block size.

$$\text{HMAC-H}(k, m) := \text{H}(k \oplus \text{opad} \parallel \text{H}(k \oplus \text{ipad} \parallel m))$$

block 1 : $k \oplus \text{ipad}$

block 2 : m

block 3 : $k \oplus \text{opad}$

block 4 : block 2 output

Therefore, we need to compute 4i SHA-256 blocks.

Nope!

This is actually wrong. Neither PKCS#5 nor RFC2898 mention this, or describe the expected performance.

Nope!

This is actually wrong. Neither PKCS#5 nor RFC2898 mention this, or describe the expected performance.

$$U_1 \oplus U_2 \oplus \cdots \oplus U_{\mathrm{i}}$$

Nope!

This is actually wrong. Neither PKCS#5 nor RFC2898 mention this, or describe the expected performance.

$$U_1 \oplus U_2 \oplus \cdots \oplus U_{\mathsf{i}}$$

with

$$U_1 := \text{HMAC-H}(\mathsf{pw}, \mathsf{salt} \parallel 0_{32})$$

$$U_n := \text{HMAC-H}(\mathsf{pw}, U_{n-1})$$

Nope!

This is actually wrong. Neither PKCS#5 nor RFC2898 mention this, or describe the expected performance.

$$U_1 \oplus U_2 \oplus \cdots \oplus U_{\mathrm{i}}$$

with

$$U_1 := \text{HMAC-H}(\text{pw}, \text{salt} \parallel 0_{32})$$

$$U_n := \text{HMAC-H}(\text{pw}, U_{n-1})$$

(or equivalently)

$$U_1 := \text{H}(\text{pw} \oplus \text{opad} \parallel \text{H}(pw \oplus \text{ipad} \parallel \text{salt} \parallel 0_{32}))$$

$$U_n := \text{H}(\text{pw} \oplus \text{opad} \parallel \text{H}(pw \oplus \text{ipad} \parallel U_{n-1}))$$

Nope!

This is actually wrong. Neither PKCS#5 nor RFC2898 mention this, or describe the expected performance.

$$U_1 \oplus U_2 \oplus \cdots \oplus U_{\mathrm{i}}$$

with

$$U_1 := \text{HMAC-H}(\text{pw}, \text{salt} \parallel 0_{32})$$

$$U_n := \text{HMAC-H}(\text{pw}, U_{n-1})$$

(or equivalently)

$$U_1 := \text{H}(\text{pw} \oplus \text{opad} \parallel \text{H}(pw \oplus \text{ipad} \parallel \text{salt} \parallel 0_{32}))$$

$$U_n := \text{H}(\text{pw} \oplus \text{opad} \parallel \text{H}(pw \oplus \text{ipad} \parallel U_{n-1}))$$

We can compute these blocks once

Nope!

This is actually wrong. Neither PKCS#5 nor RFC2898 mention this, or describe the expected performance.

$$U_1 \oplus U_2 \oplus \cdots \oplus U_{\mathrm{i}}$$

with

$$U_1 := \text{HMAC-H}(\text{pw}, \text{salt} \parallel 0_{32})$$

$$U_n := \text{HMAC-H}(\text{pw}, U_{n-1})$$

(or equivalently)

$$U_1 := \text{H}(\text{pw} \oplus \text{opad} \parallel \text{H}(pw \oplus \text{ipad} \parallel \text{salt} \parallel 0_{32}))$$

$$U_n := \text{H}(\text{pw} \oplus \text{opad} \parallel \text{H}(pw \oplus \text{ipad} \parallel U_{n-1}))$$

We can compute these blocks once

How many times?

Actually, we only need compute $2 + 2i$ SHA-256 blocks.

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)

Bad: compute $4i$ blocks

- FreeBSD
- GRUB
- Android (BouncyCastle)

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)

Bad: compute $4i$ blocks

- FreeBSD
- GRUB
- Android (BouncyCastle)

Bad: compute $4i$ blocks

- Python (pypi pbkdf2)
- Ruby (pbkdf2 gem)
- Go (go.crypto)
- OpenBSD
- PolarSSL
- CyaSSL
- Java (OpenJDK)
- Common Lisp (ironclad)
- Perl (Crypt::PBKDF2)
- PHP
- C#

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

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

Selected performance measurements

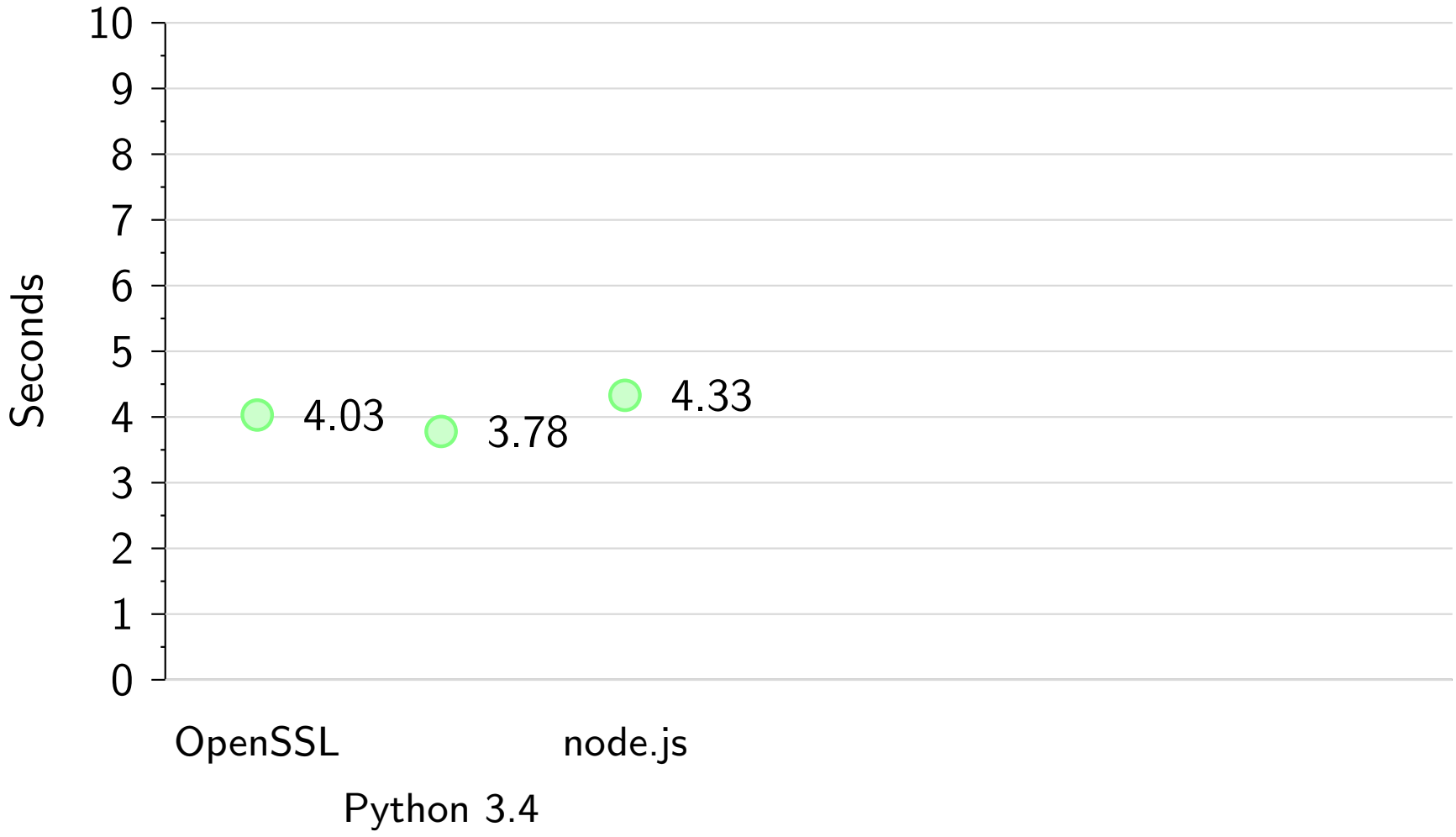

Figure : PBKDF2-HMAC-SHA1, one block output, 2^{22} iterations

Selected performance measurements

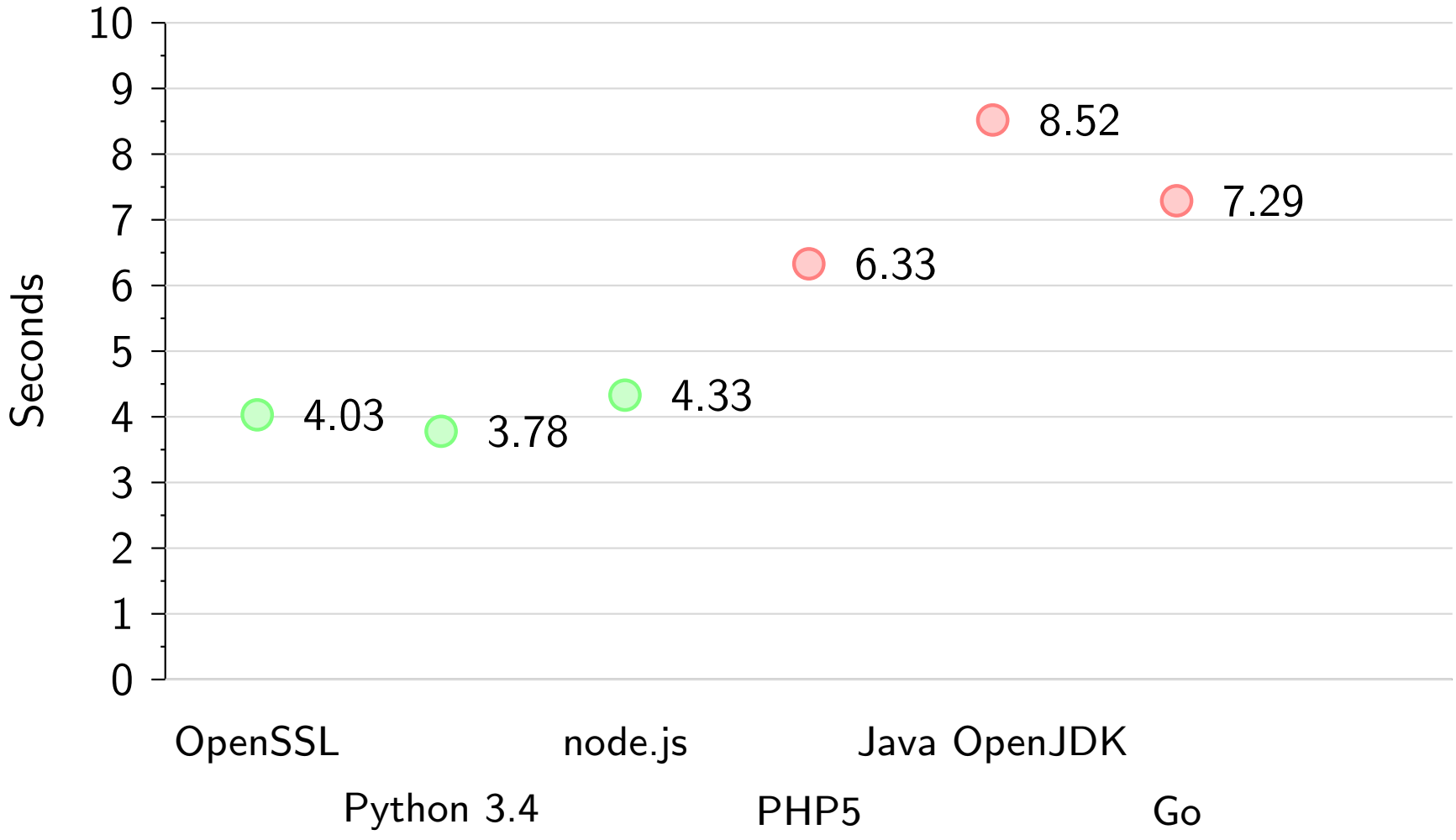

Figure : PBKDF2-HMAC-SHA1, one block output, 2^{22} iterations

Selected performance measurements

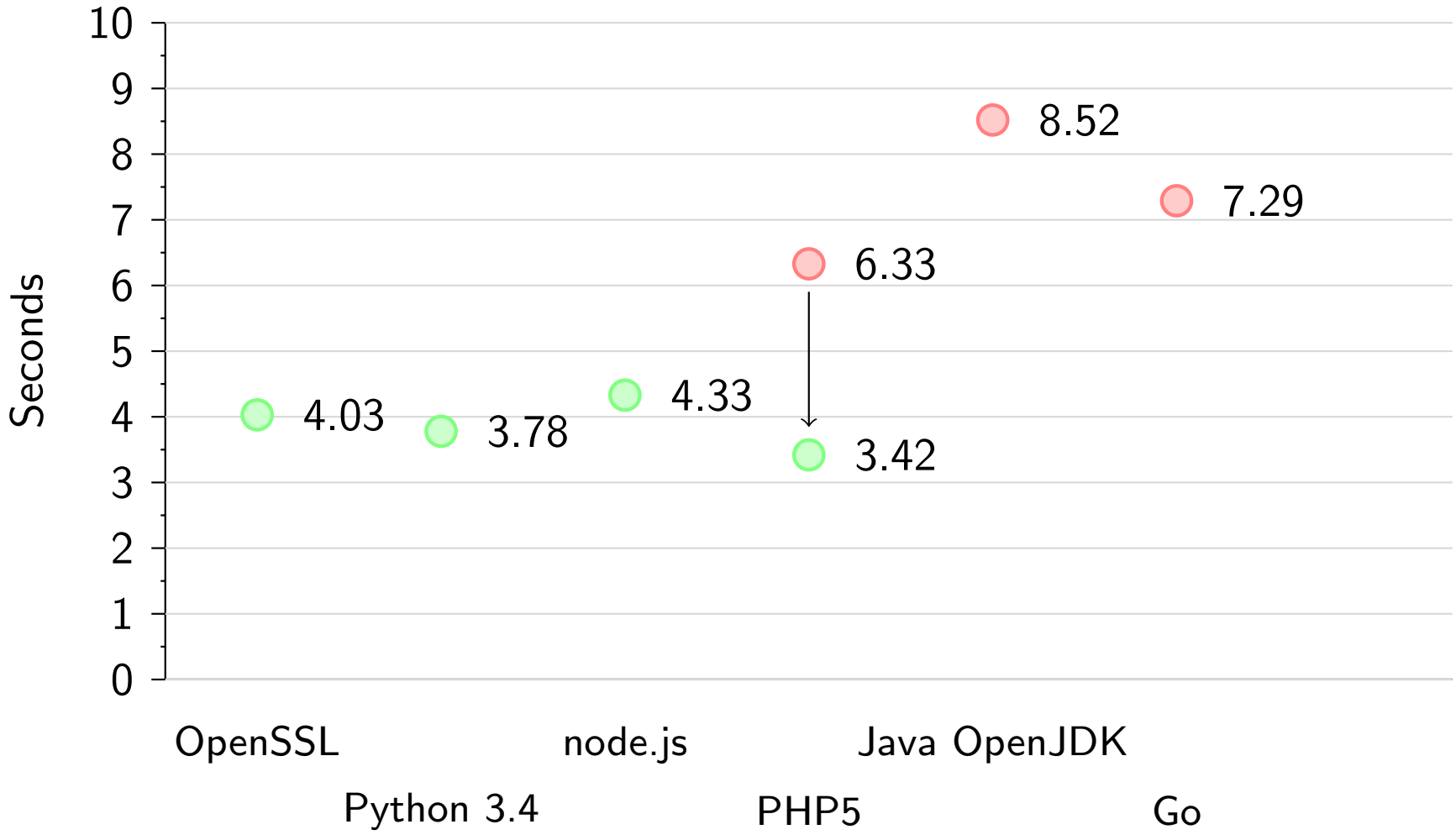

Figure : PBKDF2-HMAC-SHA1, one block output, 2^{22} iterations

Selected performance measurements

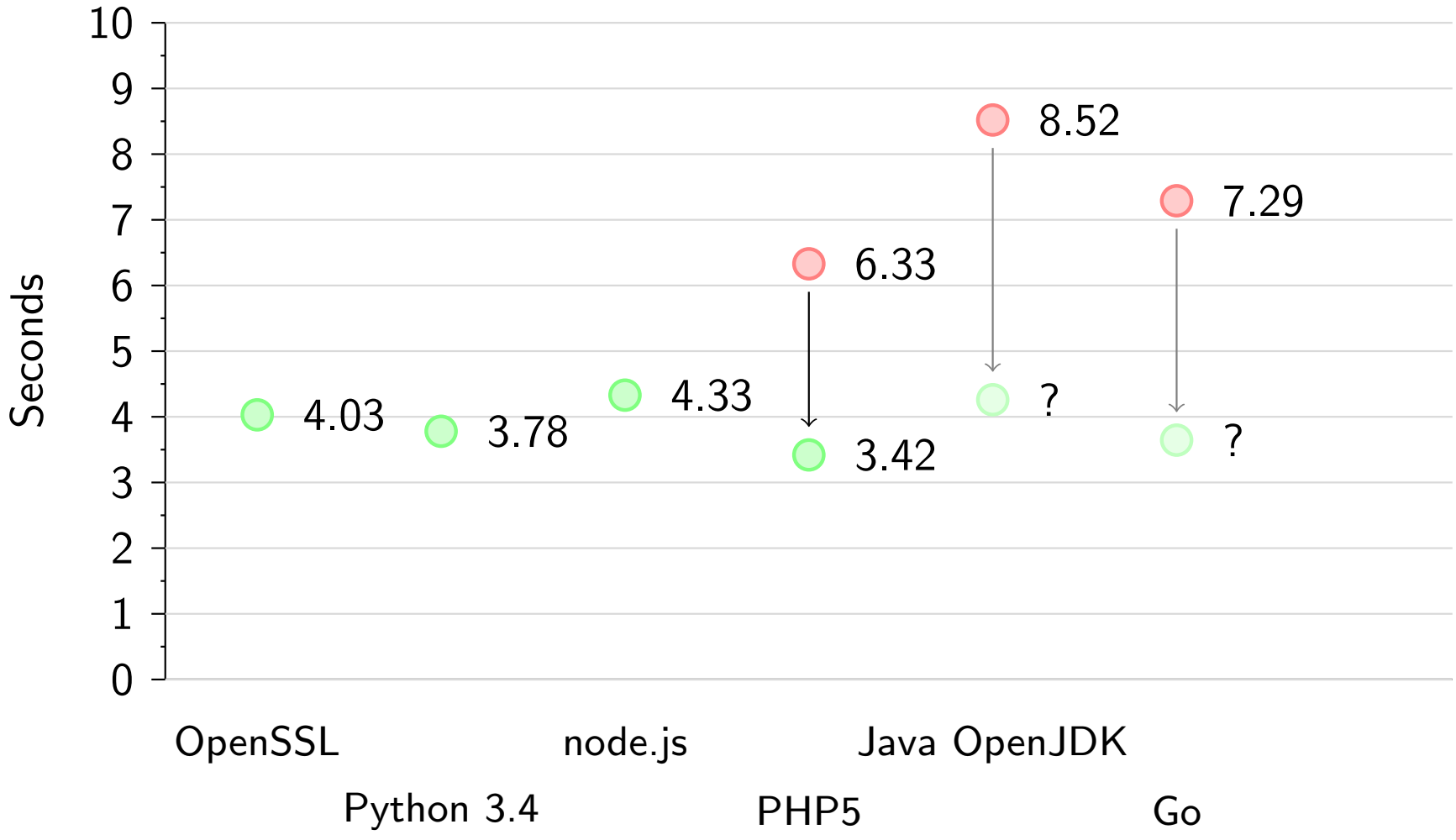

Figure : PBKDF2-HMAC-SHA1, one block output, 2^{22} iterations

Selected performance measurements

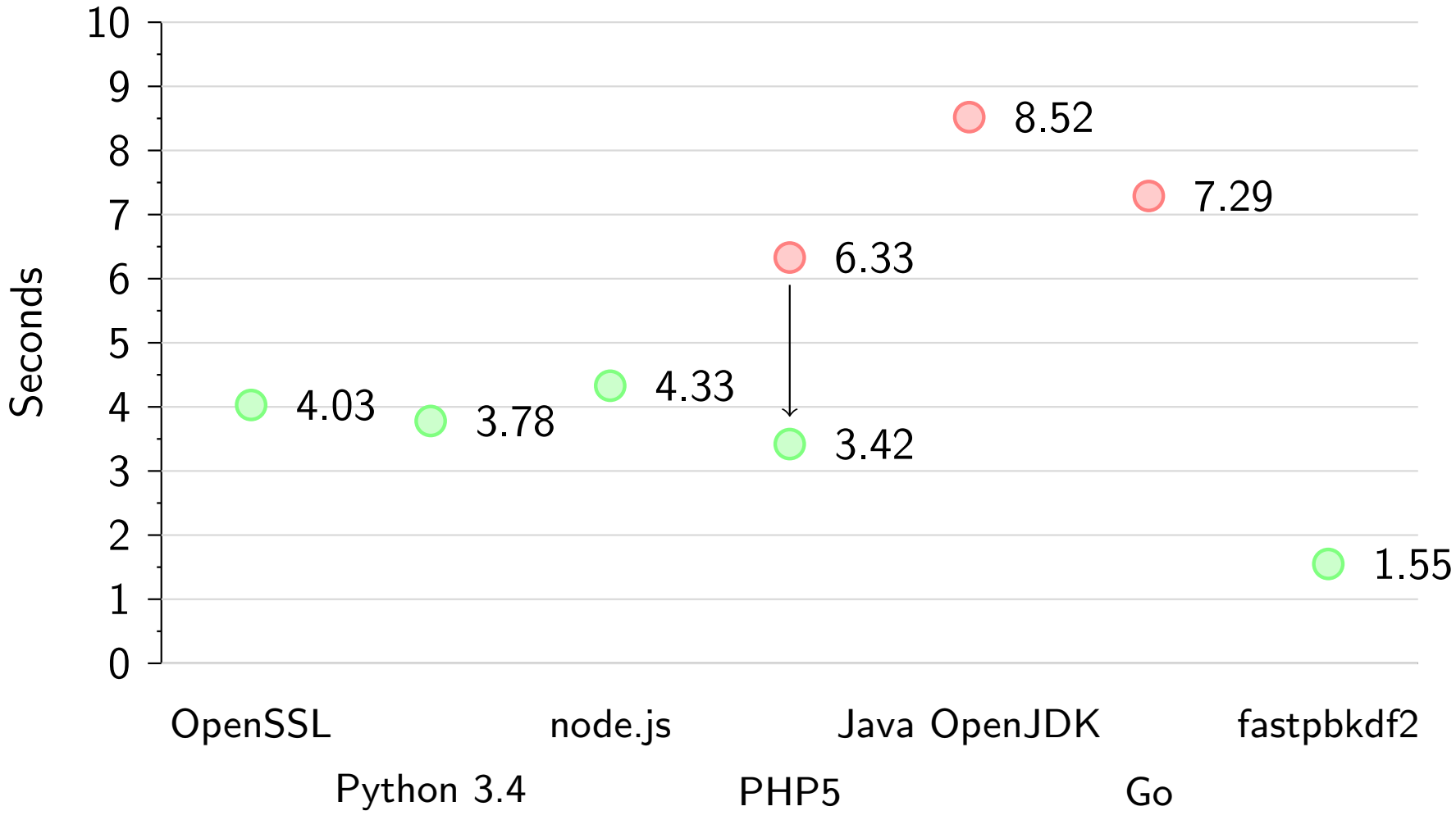

Figure : PBKDF2-HMAC-SHA1, one block output, 2^{22} iterations

fastpbkdf2

A faster PBKDF2 for defenders.

- About 400 lines of C99.

fastpbkdf2

A faster PBKDF2 for defenders.

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

fastpbkdf2

A faster PBKDF2 for defenders.

- About 400 lines of C99.
- Uses OpenSSL libcrypto's hash functions.
- CC0.

fastpbkdf2

A faster PBKDF2 for defenders.

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

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.

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: https://github.com/ctz/talks/
Code: https://github.com/ctz/fastpbkdf2/