PBKDF2: how not to do it[1]

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

[1] describe a crypto algorithm

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.

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

$$
\begin{aligned}
\mathrm{PBKDF2}_{\mathrm{PRF}}(\mathrm{pw}, \mathrm{salt}, \mathrm{i}) &:= U_1 \oplus U_2 \oplus \cdots \oplus U_{\mathrm{i}} \\
\text{where} \\
U_1 &:= \mathrm{PRF}(\mathrm{pw}, \mathrm{salt} \parallel 0_{32}) \\
U_n &:= \mathrm{PRF}(\mathrm{pw}, U_{n-1})
\end{aligned}
$$

PBKDF2: how it was described

$$
\begin{aligned}
\mathrm{PBKDF2}_{\mathrm{PRF}}(\mathrm{pw}, \mathrm{salt}, \mathrm{i}) &:= U_1 \oplus U_2 \oplus \cdots \oplus U_{\mathrm{i}} \\
\text{where} & \\
U_1 &:= \mathrm{PRF}(\mathrm{pw}, \mathrm{salt} \parallel 0_{32}) \\
U_n &:= \mathrm{PRF}(\mathrm{pw}, U_{n-1}) \\
\text{and typically} & \\
\mathrm{PRF}(\mathrm{pw}, \mathrm{x}) &:= \text{HMAC-H}(\mathrm{pw}, \mathrm{x}) \\
H &:= \text{SHA-1, SHA-256 or SHA-512}
\end{aligned}
$$

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.

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

$$\begin{aligned} \text{block 1} &: k \oplus \text{ipad} \\ \text{block 2} &: m \end{aligned}$$

Zoom, enhance

The function $\mathrm{PBKDF2}_{\mathrm{HMAC\text{-}SHA\text{-}256}}$ 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.

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

$$\begin{aligned} \text{block 1} &: k \oplus \mathrm{ipad} \\ \text{block 2} &: m \\ \text{block 3} &: k \oplus \mathrm{opad} \end{aligned}$$

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.

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

$$\begin{aligned} \text{block 1} &: k \oplus \text{ipad} \\ \text{block 2} &: m \\ \text{block 3} &: k \oplus \text{opad} \\ \text{block 4} &: \text{block 2 output} \end{aligned}$$

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.

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

$$\begin{aligned} \text{block 1} &: k \oplus \text{ipad} \\ \text{block 2} &: m \\ \text{block 3} &: k \oplus \text{opad} \\ \text{block 4} &: \text{block 2 output} \end{aligned}$$

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

Nope!

This is actually badly wrong. Neither PKCS#5 nor RFC2898 mention this.

Nope!

This is actually badly wrong. Neither PKCS#5 nor RFC2898 mention this.

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

Nope!

This is actually badly wrong. Neither PKCS#5 nor RFC2898 mention this.

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

with

$$\begin{aligned} U_1 &:= \mathrm{HMAC\text{-}H}(\mathrm{pw}, \mathrm{salt} \parallel 0_{32}) \\ U_n &:= \mathrm{HMAC\text{-}H}(\mathrm{pw}, U_{n-1}) \end{aligned}$$

Nope!

This is actually badly wrong. Neither PKCS#5 nor RFC2898 mention this.

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

with

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

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

(or equivalently)

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

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

Nope!

This is actually badly wrong. Neither PKCS#5 nor RFC2898 mention this.

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

with

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

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

(or equivalently)

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

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

We can precompute these blocks!

Nope!

This is actually badly wrong. Neither PKCS#5 nor RFC2898 mention this.

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

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 precompute these blocks!

How many times?

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

Our survey says...

Good: compute $2 + 2i$ blocks

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

Our survey says...

Good: compute $2 + 2i$ blocks

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

Bad: compute $4i$ blocks

- FreeBSD
- GRUB
- Android (BouncyCastle)

Our survey says...

Good: compute $2 + 2i$ blocks

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

Parting thoughts...

- PBKDF2 is not wonderfully designed.

Parting thoughts...

- PBKDF2 is not wonderfully designed.
- Described in an unhelpful way by its authors.

Parting thoughts...

- PBKDF2 is not wonderfully designed.
- Described in an unhelpful way by its authors.
- Most implementations gift a 2x advantage to attackers.

Thank you!

Questions?

Twitter: @jpixton
Mail: jbp@jbp.io
Web: http://jbp.io/