Closing the performance gap between Swift and C

Who I Am

I’m Paul!

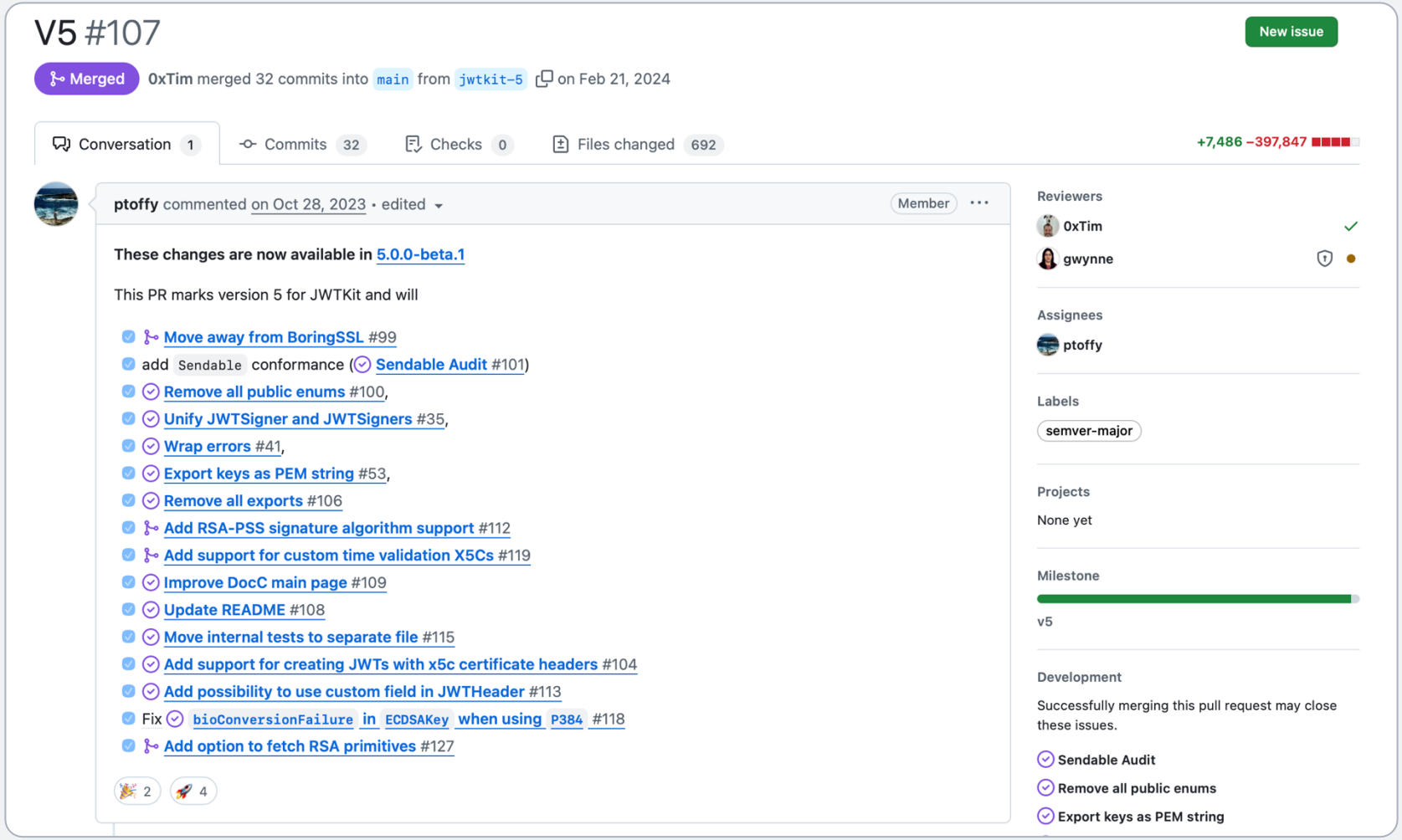
 Swift on server dev @ Broken Hands

 Maintainer @ Vapor

 github.com/ptoffy

 in/paultoffoloni

JWTKit 5



V5 #107

New issue

Merged 0xTim merged 32 commits into main from jwtkit-5 on Feb 21, 2024

Conversation 1 · Commits 32 · Checks 0 · Files changed 692 · +7,486 −397,847

ptoffy commented on Oct 28, 2023 • edited · Member

These changes are now available in 5.0.0-beta.1

This PR marks version 5 for JWTKit and will

- Move away from BoringSSL #99
- add `Sendable` conformance (Sendable Audit #101)
- Remove all public enums #100,
- Unify JWTSigner and JWTSigners #35,
- Wrap errors #41,
- Export keys as PEM string #53,
- Remove all exports #106
- Add RSA-PSS signature algorithm support #112
- Add support for custom time validation X5Cs #119
- Improve DocC main page #109
- Update README #108
- Move internal tests to separate file #115
- Add support for creating JWTs with x5c certificate headers #104
- Add possibility to use custom field in JWTHeader #113
- Fix `bioConversionFailure` in `ECDSAKey` when using `P384` #118
- Add option to fetch RSA primitives #127

🎉 2 🚀 4

Reviewers
0xTim
gwynne

Assignees
ptoffy

Labels
semver-major

Projects
None yet

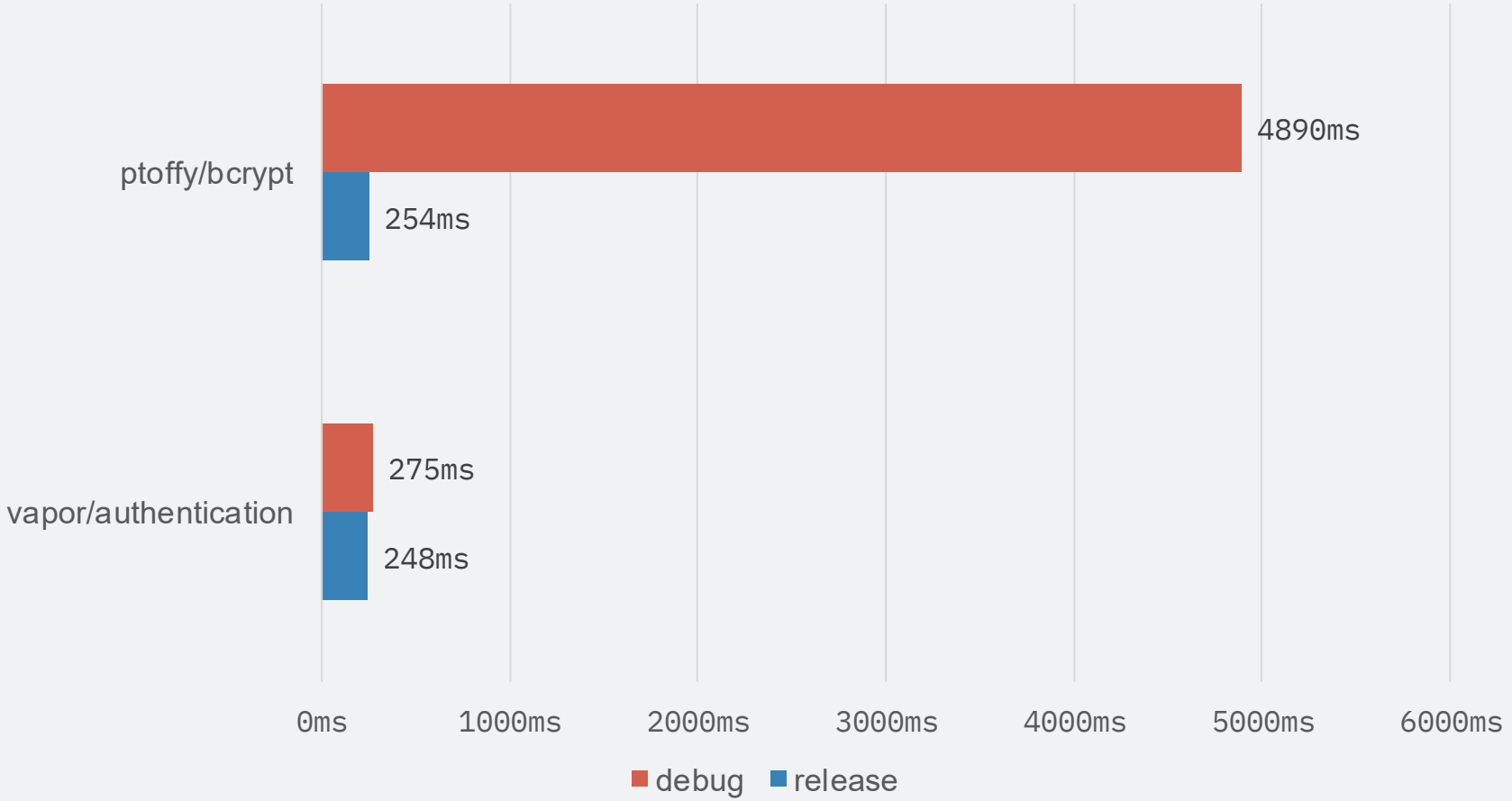
Milestone
v5

Development
Successfully merging this pull request may close these issues.
Sendable Audit
Remove all public enums
Export keys as PEM string

vapor/authentication

- Authentication/Authorisation utilites
- Password hashing
  - bcrypt
  - PBKDF2
- OTP

bcrypt

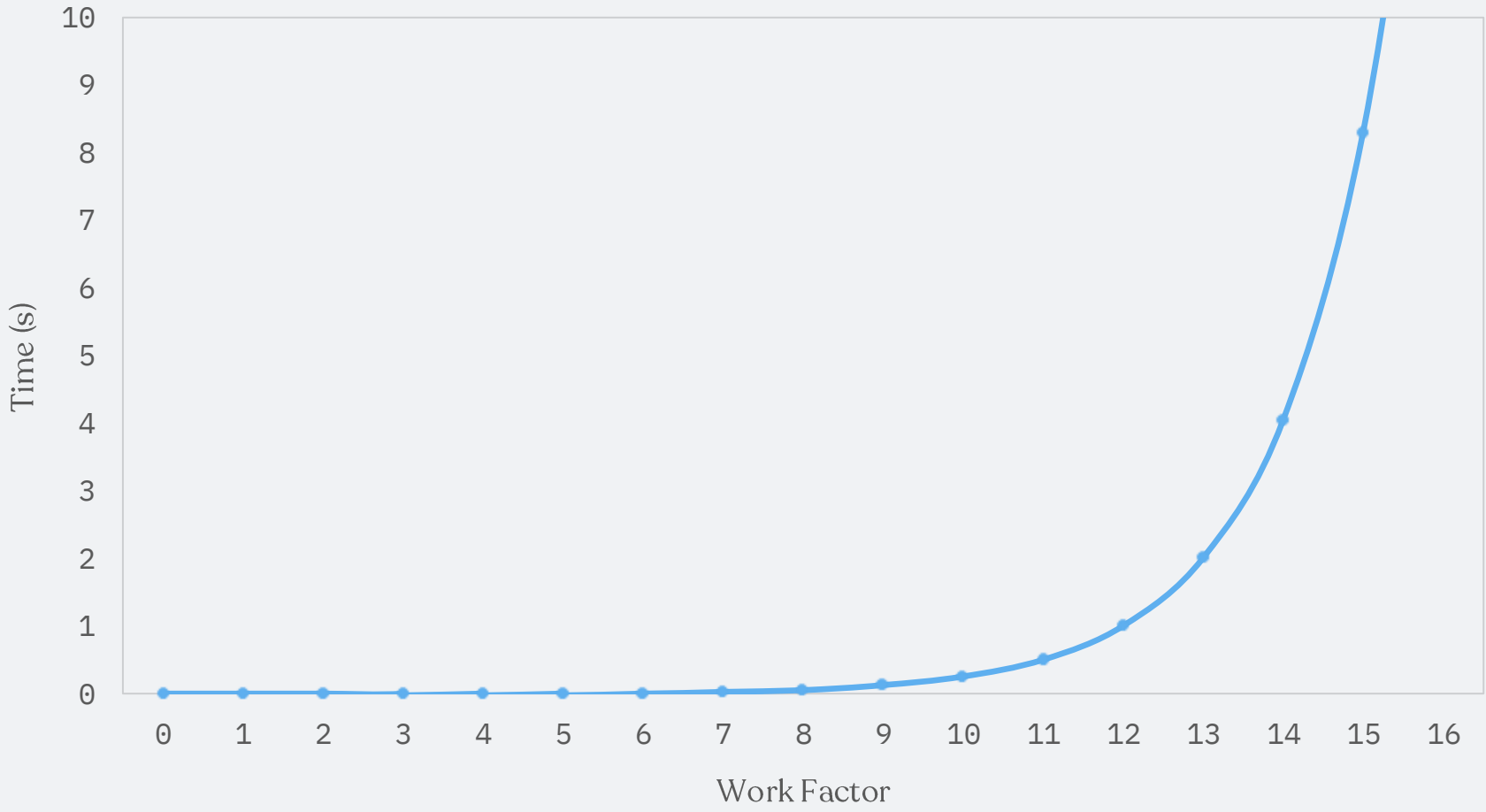

ptoffy/bcrypt
4890ms
254ms
vapor/authentication
275ms
248ms
0ms
1000ms
2000ms
3000ms
4000ms
5000ms
6000ms
debug
release

bcrypt

```
$2a$12$PW2UuE3COpBKn2JGx7i/re
4S6FC7a6jGcmihreTOC.fdaXRB4/sUC
```

- Password-specific hashing function
- Protect against dictionary attacks
- Customisable work factor

bcrypt


Time (s)
10
9
8
7
6
5
4
3
2
1
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Work Factor

bcrypt

```
bcrypt(cost, salt, pwd)
  state < EksBlowfishSetup(cost, salt, key)
  ctext < "OrpheanBeholderScryDoubt"
  repeat(64)
    ctext < EncryptECB(state, ctext)
  return Concatenate(cost, salt, ctext)
```

EksBlowfishSetup

- Prepares the keys for encryption phase
- Repeated encryption:
  - Each output is input for next encryption
  - Non parallelisable
  - 2^cost rounds
- Cost 12 = 4096 encryptions

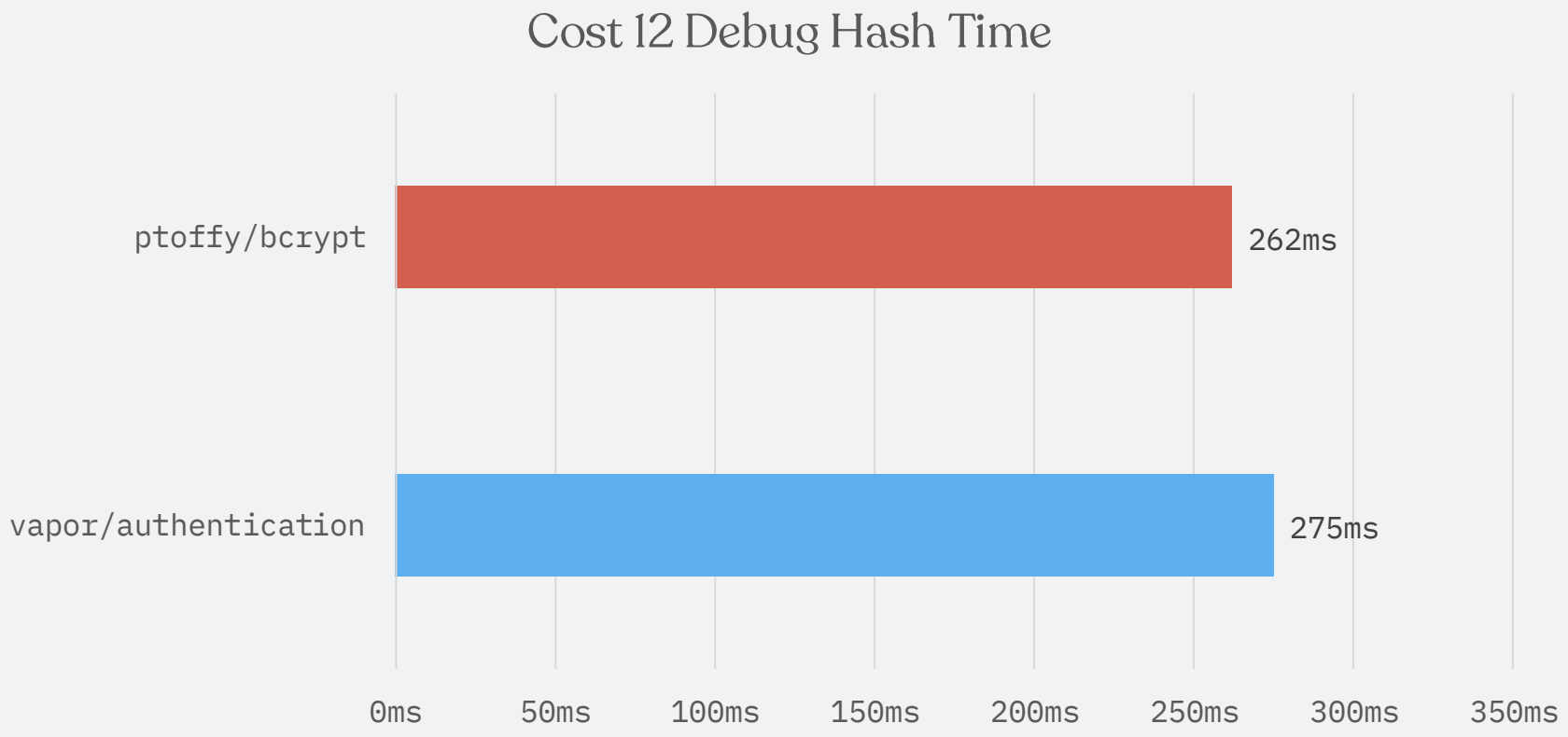
Optimisation Approaches

- -O
- Manual Optimisations
- Span

-O

```
.target(
    name: "Bcrypt",
    swiftSettings: [
        .unsafeFlags(["-O"], .when(configuration: .debug))
    ],
),
```

-O


Cost 12 Debug Hash Time
ptoffy/bcrypt
262ms
vapor/authentication
275ms
0ms
50ms
100ms
150ms
200ms
250ms
300ms
350ms

-O

- Quick and easy
- Bad debug experience
- Possible bugs
- Unusable from other packages due to `unsafeFlags`

Manual Optimisations

```
func F(s: [[UInt32]], x: UInt32) -> UInt32 {
    let a = s[0][Int((x >> 24) & 0xff)]
    let b = s[1][Int((x >> 16) & 0xff)]
    let c = s[2][Int((x >> 8) & 0xff)]
    let d = s[3][Int(x & 0xff)]

    return (a + b) ^ c + d
}
```

```
  Time: 4580ms
C Time:  256ms
```

Manual Optimisations

```
func F(s: [UInt32], x: UInt32) -> UInt32 {
    let a = s[Int((x >> 24) & 0xff)]
    let b = s[0x100 + Int((x >> 16) & 0xff)]
    let c = s[0x200 + Int((x >> 8) & 0xff)]
    let d = s[0x300 + Int((x & 0xff))]

    return (a + b) ^ c + d
}
```

```
  Time: 4580 -> 2100ms
C Time:          256ms
```

Manual Optimisations

```
func F(s: UnsafePointer<UInt32>, x: UInt32) -> UInt32 {
   let a = s[Int(tIN: (x &>> 24) & 0xff)]
   let b = s[0x100 &+ Int(tIN: (x &>> 16) & 0xff)]
   let c = s[0x200 &+ Int(tIN: (x &>> 8) & 0xff)]
   let d = s[0x300 &+ Int(tIN: (x & 0xff))]

   return (a &+ b) ^ c &+ d
}
```

* tIN: truncatingIfNeeded:

Time: 2100 -> 492ms
C Time: 256ms

Manual Optimisations

- `.strictMemorySafety()`
- Unsafe?

```
137         Xl ^= p[0]                                          ⚠ Expression uses unsafe constructs but is not marked with 'unsafe'
138
139         var i = 1
140         while i <= 16 {
141             // F(Xr)
142             let a1 = s[Int(truncatingIfNeeded: (Xl &>> 24) & 0xff)]   ⚠ Expression uses unsafe constructs but is not marked with 'unsafe'
143             let b1 = s[0x100 &+ Int(truncatingIfNeeded: (Xl &>> 16) & 0xff)]   ⚠ Expression uses unsafe constructs but is not marked with 'unsafe'
144             let c1 = s[0x200 &+ Int(truncatingIfNeeded: (Xl &>> 8) & 0xff)]   ⚠ Expression uses unsafe constructs but is not marked with 'unsafe'
145             let d1 = s[0x300 &+ Int(truncatingIfNeeded: Xl & 0xff)]   ⚠ Expression uses unsafe constructs but is not marked with 'unsafe'
146             Xr ^= ((a1 &+ b1) ^ c1 &+ d1) ^ p[i]                ⚠ Expression uses unsafe constructs but is not marked with 'unsafe'
147
148             // F(Xl)
149             let a2 = s[Int(truncatingIfNeeded: (Xr &>> 24) & 0xff)]   ⚠ Expression uses unsafe constructs but is not marked with 'unsafe'
150             let b2 = s[0x100 &+ Int(truncatingIfNeeded: (Xr &>> 16) & 0xff)]   ⚠ Expression uses unsafe constructs but is not marked with 'unsafe'
151             let c2 = s[0x200 &+ Int(truncatingIfNeeded: (Xr &>> 8) & 0xff)]   ⚠ Expression uses unsafe constructs but is not marked with 'unsafe'
152             let d2 = s[0x300 &+ Int(truncatingIfNeeded: Xr & 0xff)]   ⚠ Expression uses unsafe constructs but is not marked with 'unsafe'
153             Xl ^= ((a2 &+ b2) ^ c2 &+ d2) ^ p[i &+ 1]           ⚠ Expression uses unsafe constructs but is not marked with 'unsafe'
154
155             i &+= 2
156         }
157
158         xl = Xr ^ p[17]                                         ⚠ Expression uses unsafe constructs but is not marked with 'unsafe'
159         xr = Xl
```

Span

- View into memory
- Replacement for `Unsafe*` constructs
- Lifetime dependent

Span

```
func F(s: UnsafePointer<UInt32>, x: UInt32) -> UInt32 {
   let a = s[Int(tIN: (x &>> 24) & 0xff)]
   let b = s[0x100 &+ Int(tIN: (x &>> 16) & 0xff)]
   let c = s[0x200 &+ Int(tIN: (x &>> 8) & 0xff)]
   let d = s[0x300 &+ Int(tIN: (x & 0xff))]

   return (a &+ b) ^ c &+ d
}
```

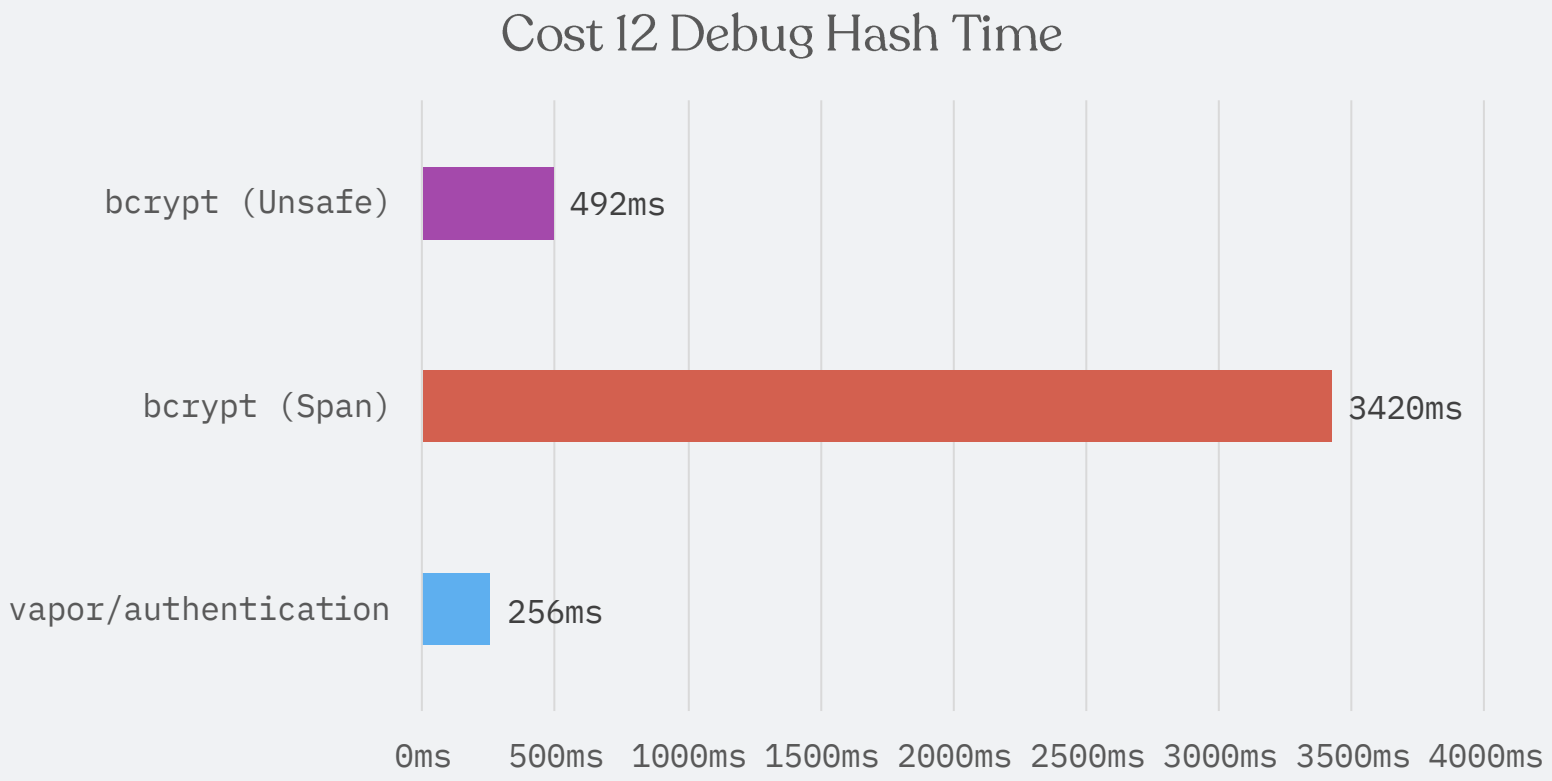
* tIN: truncatingIfNeeded:

Span

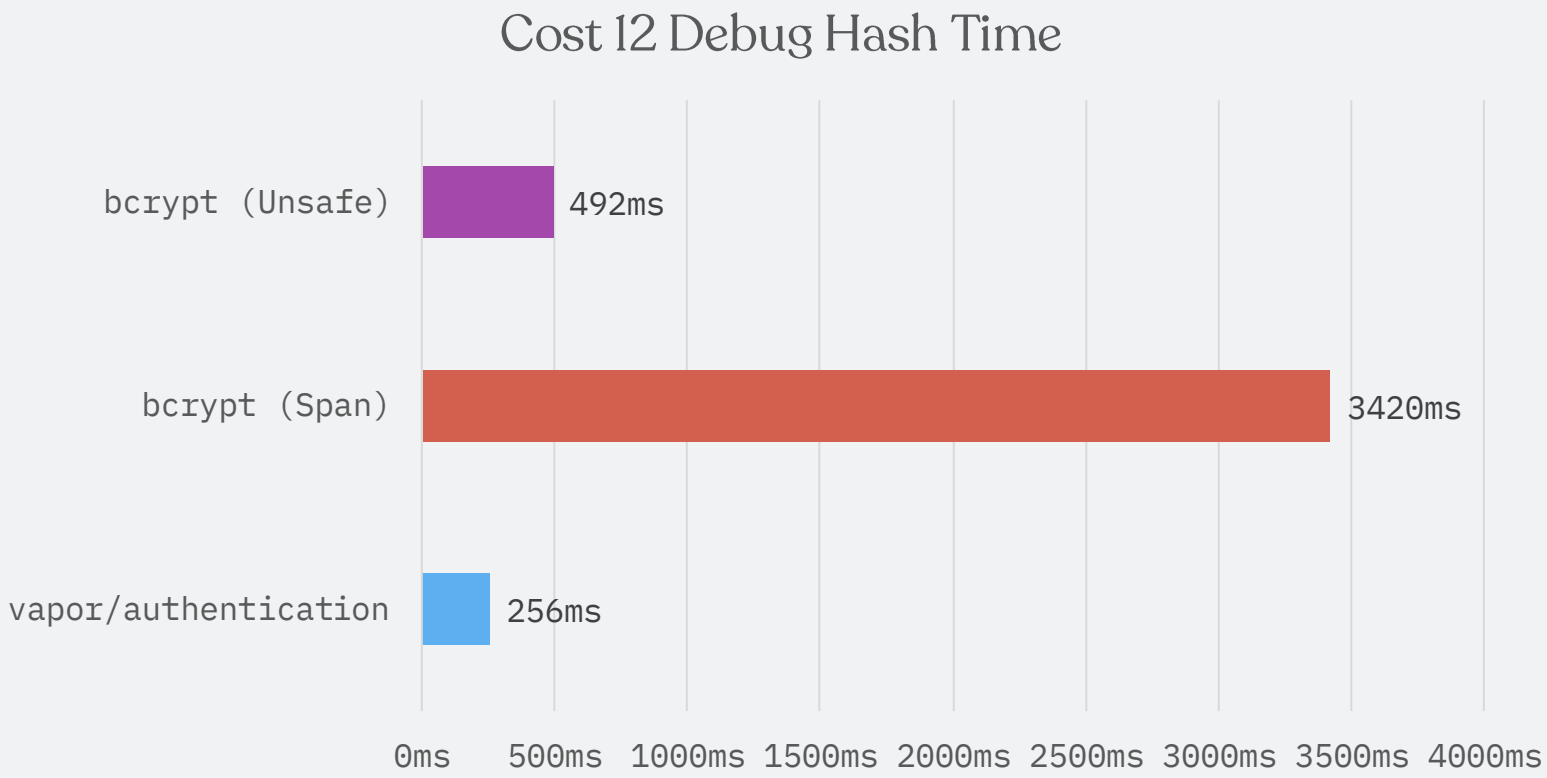
```
func F(s: Span<UInt32>, x: UInt32) -> UInt32 {
    let a = s[Int(tIN: (x &>> 24) & 0xff)]
    let b = s[0x100 &+ Int(tIN: (x &>> 16) & 0xff)]
    let c = s[0x200 &+ Int(tIN: (x &>> 8) & 0xff)]
    let d = s[0x300 &+ Int(tIN: (x & 0xff))]

    return (a &+ b) ^ c &+ d
}

F(s: s.span, x: x)
```

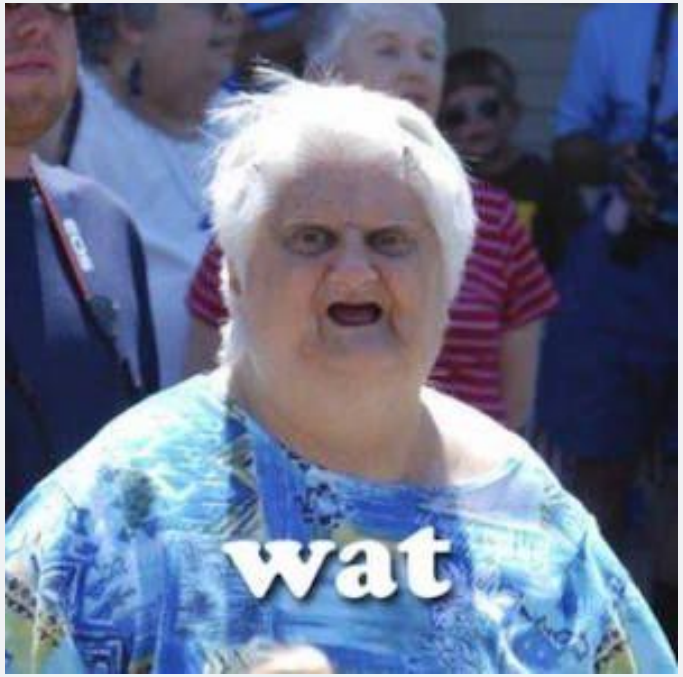
* tIN: truncatingIfNeeded:

Span


Cost 12 Debug Hash Time
bcrypt (Unsafe)
492ms
bcrypt (Span)
3420ms
vapor/authentication
256ms
0ms
500ms
1000ms
1500ms
2000ms
2500ms
3000ms
3500ms
4000ms

Span


Cost 12 Debug Hash Time
bcrypt (Unsafe)
492ms
bcrypt (Span)
3420ms
vapor/authentication
256ms
0ms
500ms
1000ms
1500ms
2000ms
2500ms
3000ms
3500ms
4000ms


wat

Span

```
let s: [UInt32] = ...
while ... {
    F(s: s.span, x: x)
}
```

Span

```
let s: [UInt32] = ...
let sSpan: [UInt32] = s.span
while ... {
    F(s: sSpan, x: x)
}
```

Span

- Non-Escapable
- Define ownership: `inout`, `borrowing`, `consuming`
- Define lifetime dependencies

```
.enableUpcomingFeature("Lifetimes")
```

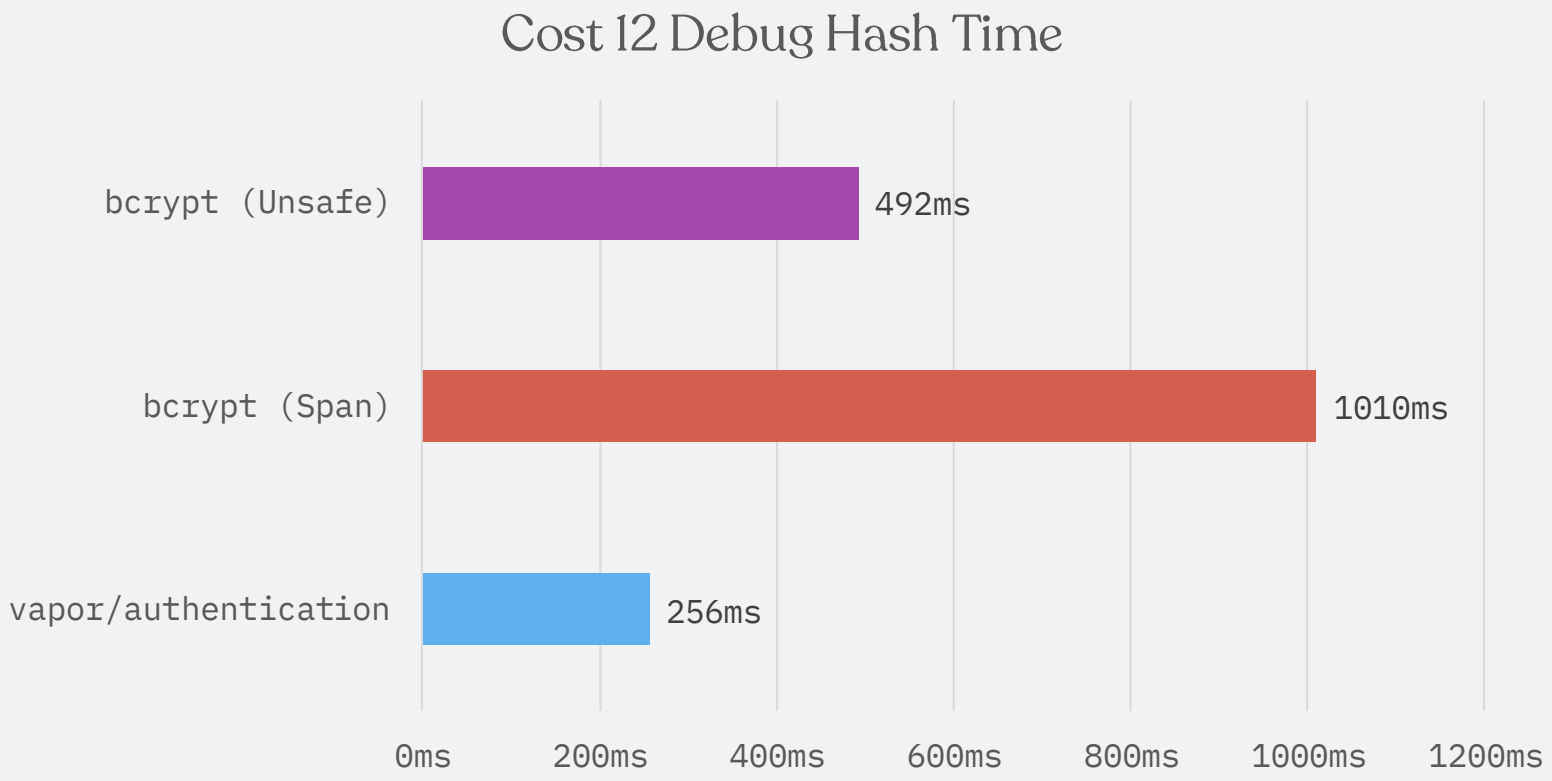
Span

```
@usableFromInline
@_lifetime(&p, &s)
static func expand0State(
   key: [UInt8],
   p: inout MutableSpan<UInt32>,
   s: inout MutableSpan<UInt32>
)
```

Span

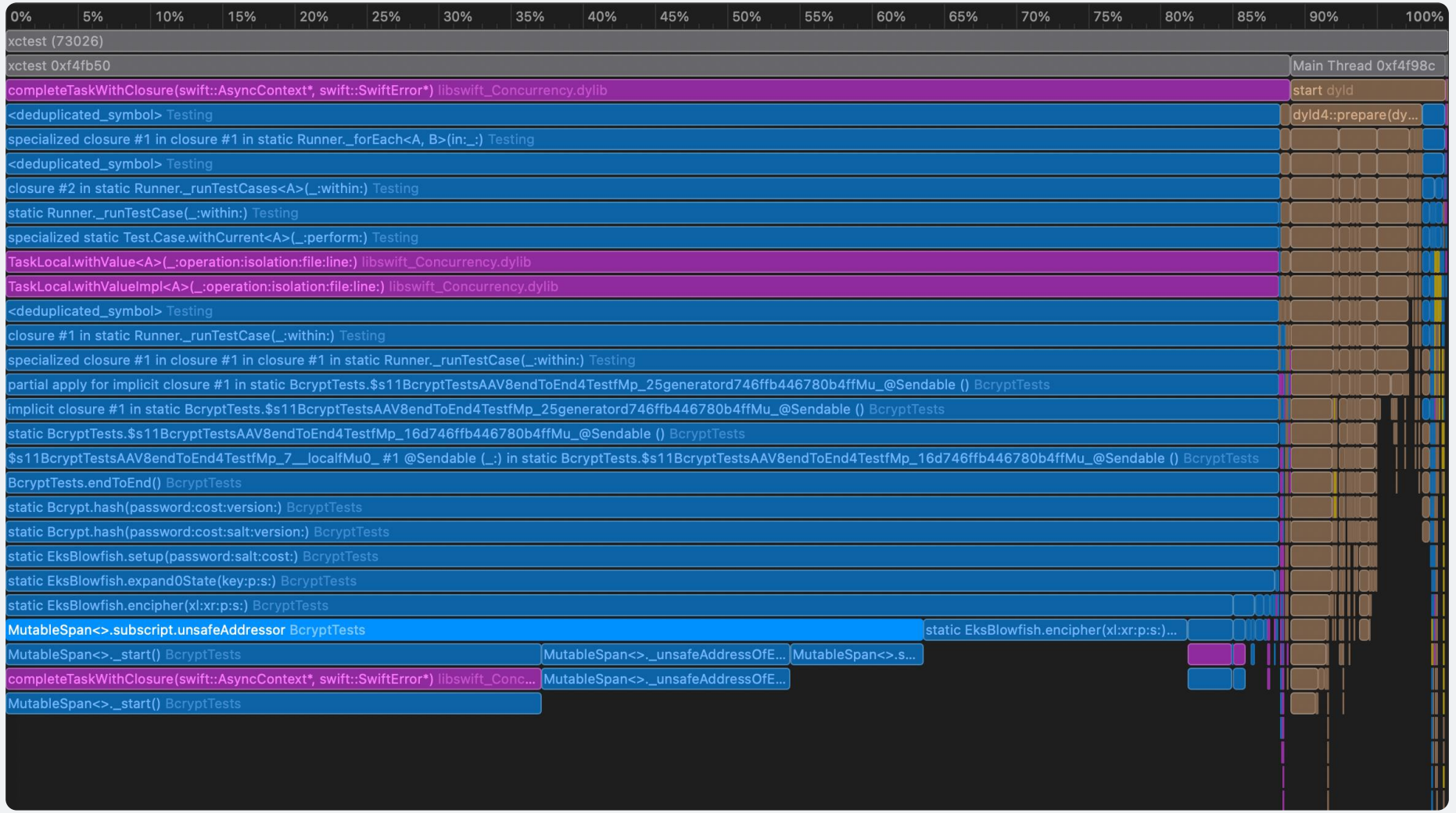
```
@usableFromInline
@inline(__always)
static func encipher(
   xl: inout UInt32,
   xr: inout UInt32,
   p: borrowing MutableSpan<UInt32>,
   s: borrowing MutableSpan<UInt32>
)

p[unchecked: 0]
```

Span

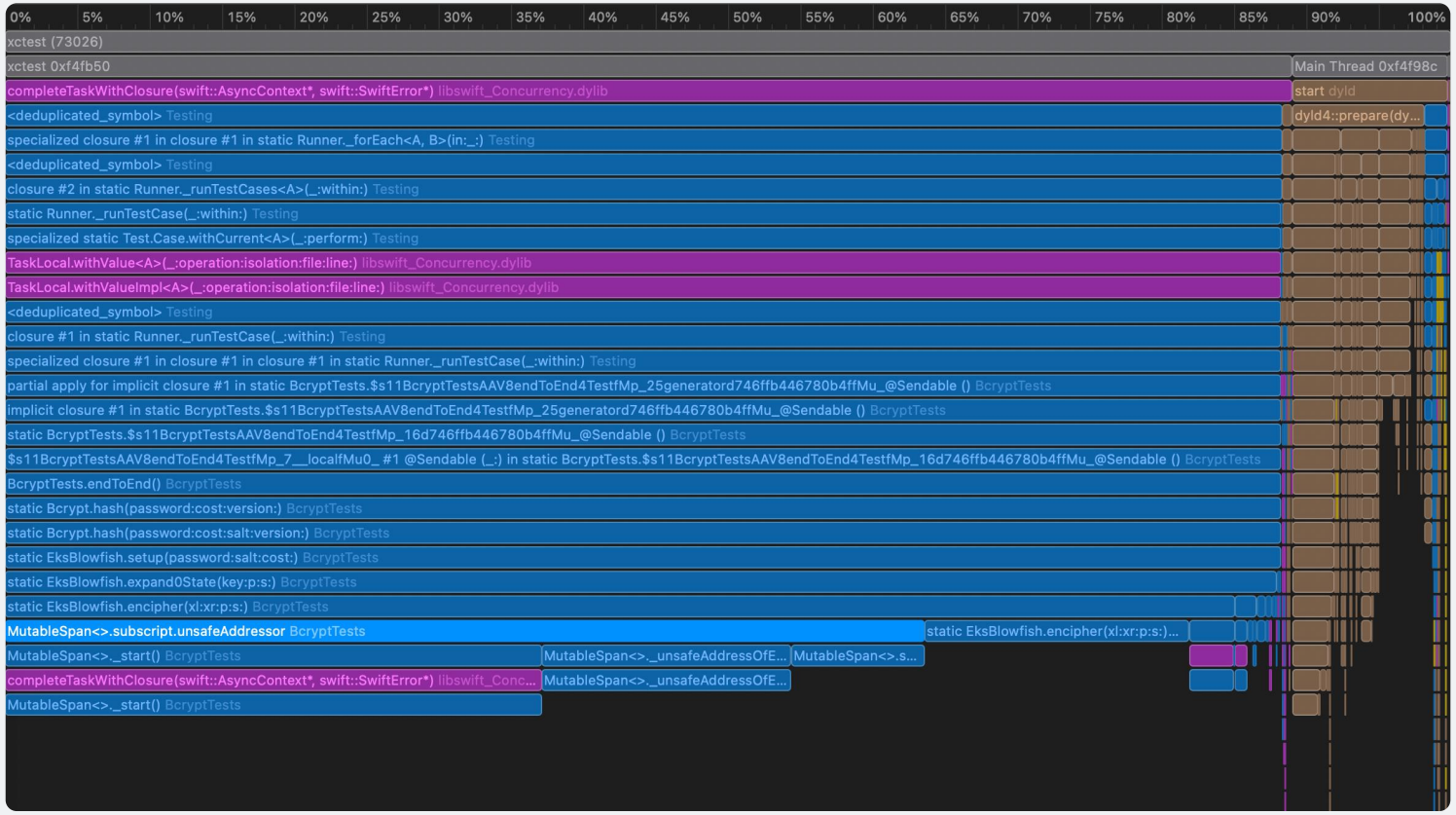


Time: 3420 -> 1010ms

Span

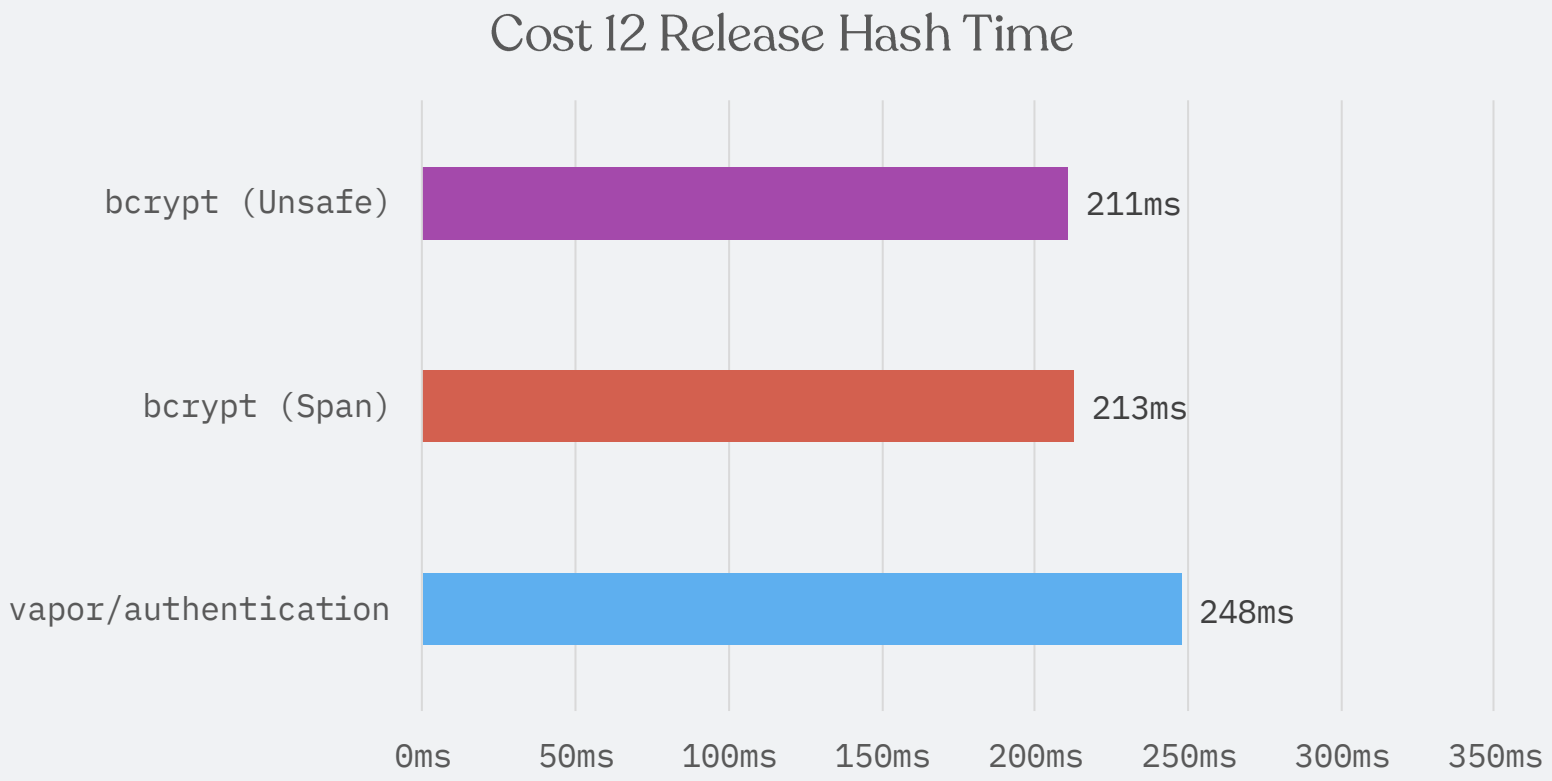

0%
5%
10%
15%
20%
25%
30%
35%
40%
45%
50%
55%
60%
65%
70%
75%
80%
85%
90%
100%
xctest (73026)
xctest 0xf4fb50
Main Thread 0xf4f98c
completeTaskWithClosure(swift::AsyncContext*, swift::SwiftError*) libswift_Concurrency.dylib
start dyld
<deduplicated_symbol> Testing
dyld4::prepare(dy...
specialized closure #1 in closure #1 in static Runner._forEach<A, B>(in:_:) Testing
<deduplicated_symbol> Testing
closure #2 in static Runner._runTestCases<A>(_:within:) Testing
static Runner._runTestCase(_:within:) Testing
specialized static Test.Case.withCurrent<A>(_:perform:) Testing
TaskLocal.withValue<A>(_:operation:isolation:file:line:) libswift_Concurrency.dylib
TaskLocal.withValueImpl<A>(_:operation:isolation:file:line:) libswift_Concurrency.dylib
<deduplicated_symbol> Testing
closure #1 in static Runner._runTestCase(_:within:) Testing
specialized closure #1 in closure #1 in closure #1 in static Runner._runTestCase(_:within:) Testing
partial apply for implicit closure #1 in static BcryptTests.$s11BcryptTestsAAV8endToEnd4TestfMp_25generatord746ffb446780b4ffMu_@Sendable () BcryptTests
implicit closure #1 in static BcryptTests.$s11BcryptTestsAAV8endToEnd4TestfMp_25generatord746ffb446780b4ffMu_@Sendable () BcryptTests
static BcryptTests.$s11BcryptTestsAAV8endToEnd4TestfMp_16d746ffb446780b4ffMu_@Sendable () BcryptTests
$s11BcryptTestsAAV8endToEnd4TestfMp_7__localfMu0_ #1 @Sendable (_:) in static BcryptTests.$s11BcryptTestsAAV8endToEnd4TestfMp_16d746ffb446780b4ffMu_@Sendable () BcryptTests
BcryptTests.endToEnd() BcryptTests
static Bcrypt.hash(password:cost:version:) BcryptTests
static Bcrypt.hash(password:cost:salt:version:) BcryptTests
static EksBlowfish.setup(password:salt:cost:) BcryptTests
static EksBlowfish.expand0State(key:p:s:) BcryptTests
static EksBlowfish.encipher(xl:xr:p:s:) BcryptTests
MutableSpan<>.subscript.unsafeAddressor BcryptTests
static EksBlowfish.encipher(xl:xr:p:s:)...
MutableSpan<>._start() BcryptTests
MutableSpan<>._unsafeAddressOfE...
MutableSpan<>.s...
completeTaskWithClosure(swift::AsyncContext*, swift::SwiftError*) libswift_Conc...
MutableSpan<>._unsafeAddressOfE...
MutableSpan<>._start() BcryptTests

Span



`MutableSpan<>.subscript.unsafeAddressor`

Span


Cost 12 Release Hash Time
bcrypt (Unsafe)
211ms
bcrypt (Span)
213ms
vapor/authentication
248ms
0ms
50ms
100ms
150ms
200ms
250ms
300ms
350ms

Span: what we learned

- Great: solves the unsafety problem
- Debug performance is not quite there yet
- Usability can be improved
- bcrypt will use pointers for now

What about you?