QCon NYC — 25 JUNE 2019

MAINTAINING THE GO CRYPTO LIBRARIES

Filippo Valsorda

Google

@FiloSottile

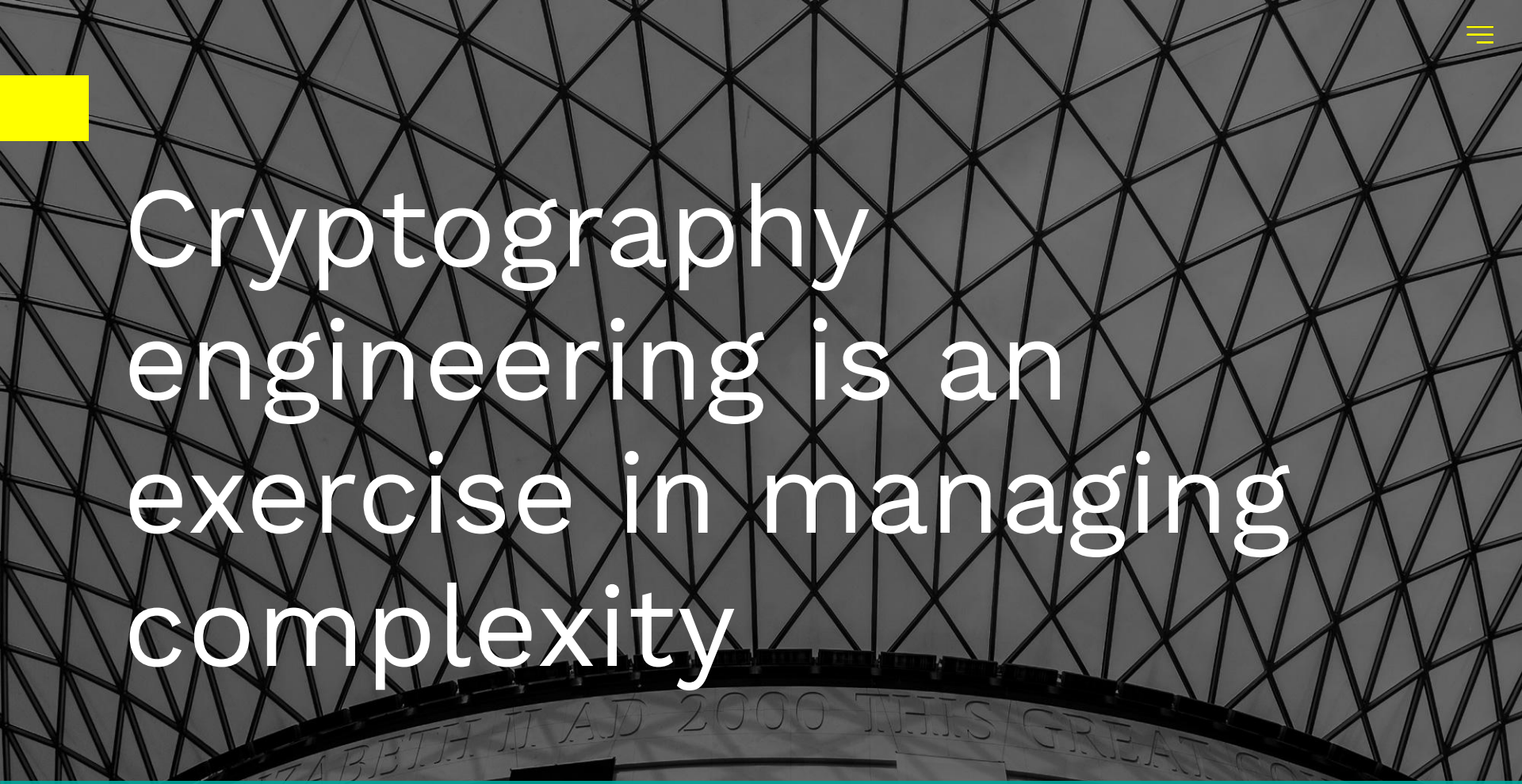
WHO AM I

Go security coordinator

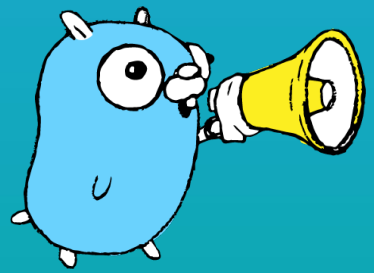
Go crypto/... packages owner and maintainer

SECTION 1

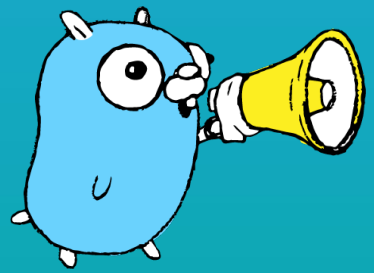
Cryptography is Hard



Cryptography engineering is an exercise in managing complexity



In cryptography engineering a single mistake makes your entire system useless.



In cryptography engineering a single mistake makes your entire system useless.

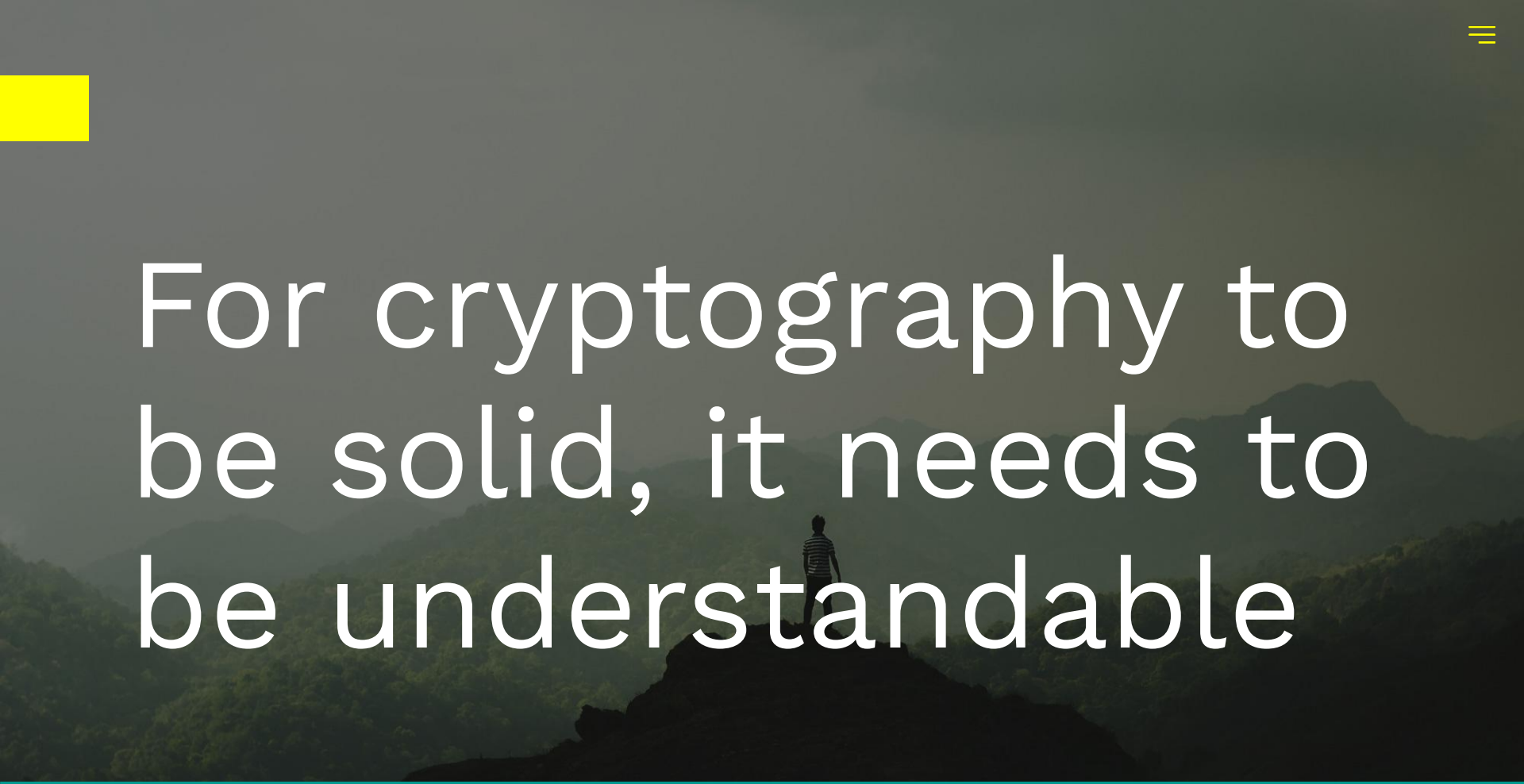
… and tests won’t save you.

Complexity that affects users

API surface

Complexity that affects contributors

If users roll their own, what is available is not easy enough


For cryptography to
be solid, it needs to
be understandable

SECTION 2

The Go cryptography libraries

```
crypto/tls          crypto/x509
crypto/ed25519      crypto/ecdsa
crypto/sha256       crypto/sha512
crypto/cipher       crypto/aes
crypto/rsa          crypto/elliptic
crypto/rand         crypto/subtle
crypto/hmac

crypto/des crypto/md5  crypto/dsa
crypto/rc4 crypto/sha1
```

Packages in the Go standard library

THE GO CRYPTOGRAPHY LIBRARIES

```
x/crypto/acme      x/crypto/argon2
x/crypto/bcrypt
x/crypto/blake2[bs]
x/crypto/chacha20poly1305
x/crypto/cryptobyte
x/crypto/curve25519
x/crypto/hkdf      x/crypto/nacl
x/crypto/pbkdf2    x/crypto/scrypt
x/crypto/sha3      x/crypto/ssh
```

Packages in golang.org/x/crypto

... and more

Go is good for cryptography

- Memory safety
- Performance
- Reproducible builds
- Static analysis

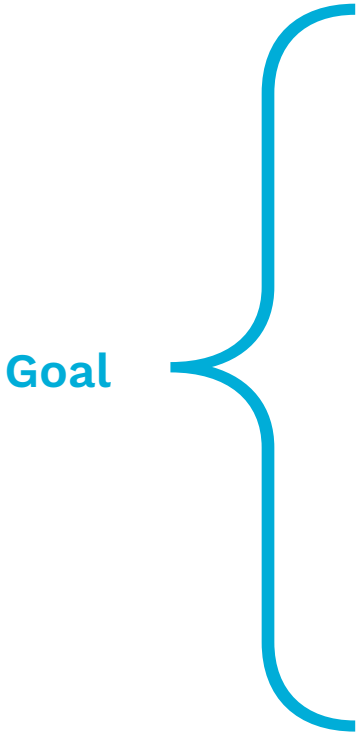
Go is good for cryptography

- Memory safety
- Performance
- Reproducible builds
- Static analysis

- Clarity and explicit control flow
- Easy documentation
- go fmt

Success

Go has a solid, modern, production-ready crypto library.



Enabling a secure ecosystem

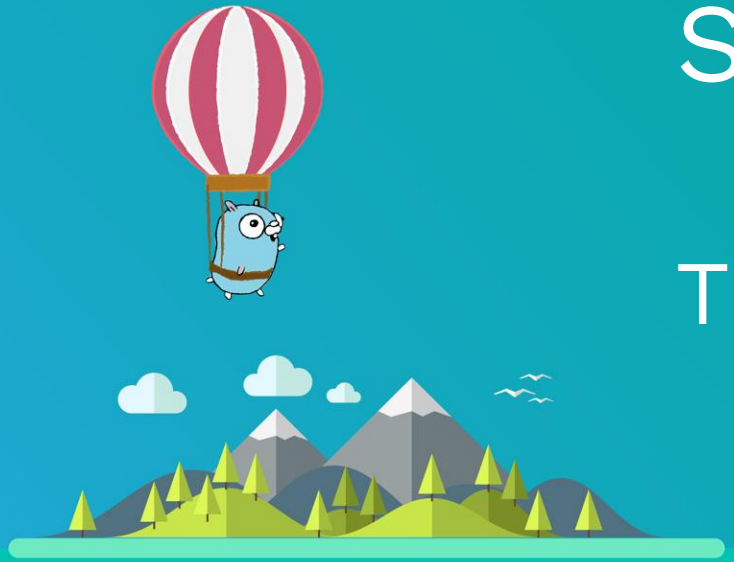
The Go Crypto Principles

Secure, safe, practical, modern

https://golang.org/design/cryptography-principles

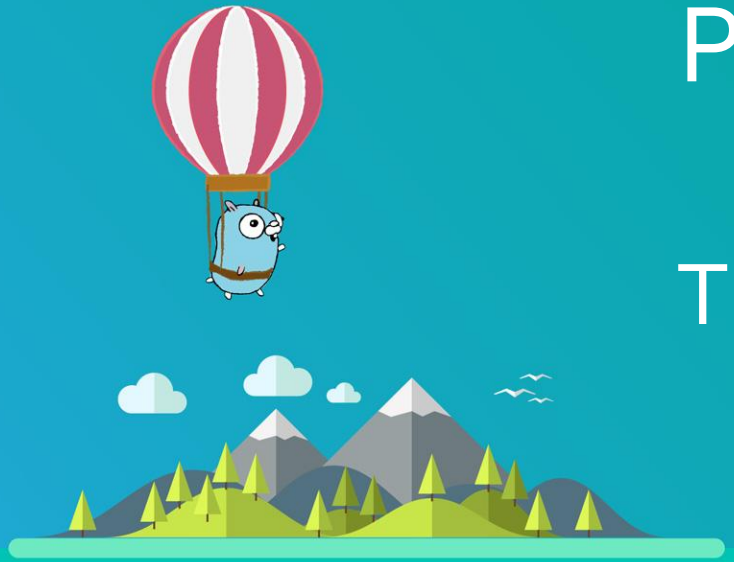
Secure

The obvious one



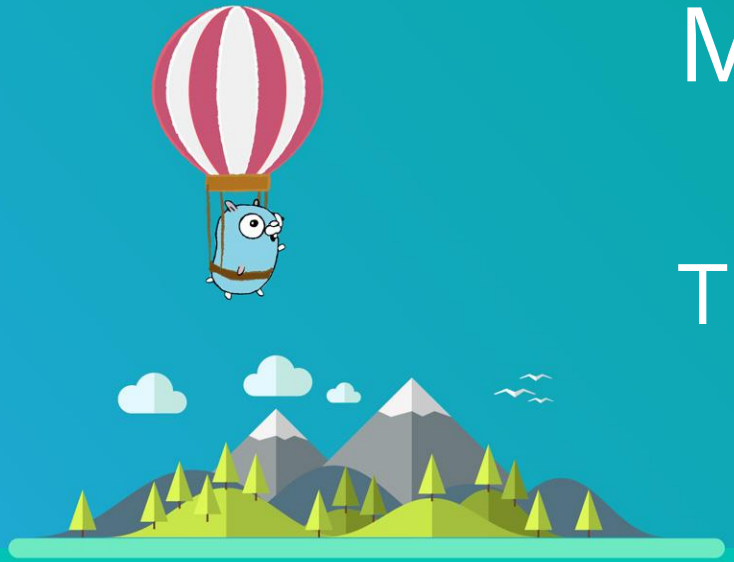
Safe

The overlooked one



Practical

The dangerous one



Modern

The aspirational one

SECTION 3

How the Go cryptography libraries are different

Not a priority

Maximum performance

Universal support

Uncommon use cases

A priority

Readability

Safe defaults

Good guidance, docs and examples

CIPHERSUITES SUPPORTED BY OPENSSL

TLS_AES_256_GCM_SHA384
TLS_CHACHA20_POLY1305_SHA256
TLS_AES_128_GCM_SHA256
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
TLS_DHE_DSS_WITH_AES_256_GCM_SHA384
TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256
TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256
TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256
TLS_ECDHE_ECDSA_WITH_AES_256_CCM_8
TLS_ECDHE_ECDSA_WITH_AES_256_CCM
TLS_DHE_RSA_WITH_AES_256_CCM_8
TLS_DHE_RSA_WITH_AES_256_CCM
TLS_ECDHE_ECDSA_WITH_ARIA_256_GCM_SHA384
TLS_ECDHE_RSA_WITH_ARIA_256_GCM_SHA384
TLS_DHE_DSS_WITH_ARIA_256_GCM_SHA384
TLS_DHE_RSA_WITH_ARIA_256_GCM_SHA384
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
TLS_DHE_DSS_WITH_AES_128_GCM_SHA256
TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8
TLS_ECDHE_ECDSA_WITH_AES_128_CCM
TLS_DHE_RSA_WITH_AES_128_CCM_8
TLS_DHE_RSA_WITH_AES_128_CCM

TLS_ECDHE_ECDSA_WITH_ARIA_128_GCM_SHA256
TLS_ECDHE_RSA_WITH_ARIA_128_GCM_SHA256
TLS_DHE_DSS_WITH_ARIA_128_GCM_SHA256
TLS_DHE_RSA_WITH_ARIA_128_GCM_SHA256
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
TLS_DHE_DSS_WITH_AES_256_CBC_SHA256
TLS_ECDHE_ECDSA_WITH_CAMELLIA_256_CBC_SHA384
TLS_ECDHE_RSA_WITH_CAMELLIA_256_CBC_SHA384
TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA256
TLS_DHE_DSS_WITH_CAMELLIA_256_CBC_SHA256
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
TLS_DHE_DSS_WITH_AES_128_CBC_SHA256
TLS_ECDHE_ECDSA_WITH_CAMELLIA_128_CBC_SHA256
TLS_ECDHE_RSA_WITH_CAMELLIA_128_CBC_SHA256
TLS_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA256
TLS_DHE_DSS_WITH_CAMELLIA_128_CBC_SHA256
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
TLS_DHE_RSA_WITH_AES_256_CBC_SHA
TLS_DHE_DSS_WITH_AES_256_CBC_SHA
TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA
TLS_DHE_DSS_WITH_CAMELLIA_256_CBC_SHA

TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
TLS_DHE_RSA_WITH_AES_128_CBC_SHA
TLS_DHE_DSS_WITH_AES_128_CBC_SHA
TLS_DHE_RSA_WITH_SEED_CBC_SHA
TLS_DHE_DSS_WITH_SEED_CBC_SHA
TLS_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA
TLS_DHE_DSS_WITH_CAMELLIA_128_CBC_SHA
TLS_RSA_WITH_AES_256_GCM_SHA384
TLS_RSA_WITH_AES_256_CCM_8
TLS_RSA_WITH_AES_256_CCM
TLS_RSA_WITH_ARIA_256_GCM_SHA384
TLS_RSA_WITH_AES_128_GCM_SHA256
TLS_RSA_WITH_AES_128_CCM_8
TLS_RSA_WITH_AES_128_CCM
TLS_RSA_WITH_ARIA_128_GCM_SHA256
TLS_RSA_WITH_AES_256_CBC_SHA256
TLS_RSA_WITH_CAMELLIA_256_CBC_SHA256
TLS_RSA_WITH_AES_128_CBC_SHA256
TLS_RSA_WITH_CAMELLIA_128_CBC_SHA256
TLS_RSA_WITH_AES_256_CBC_SHA
TLS_RSA_WITH_CAMELLIA_256_CBC_SHA
TLS_RSA_WITH_AES_128_CBC_SHA
TLS_RSA_WITH_SEED_CBC_SHA
TLS_RSA_WITH_CAMELLIA_128_CBC_SHA
TLS_RSA_WITH_IDEA_CBC_SHA

```
openssl ciphers -stdname -s ALL
```

CIPHERSUITES SUPPORTED BY CRYPTO/TLS

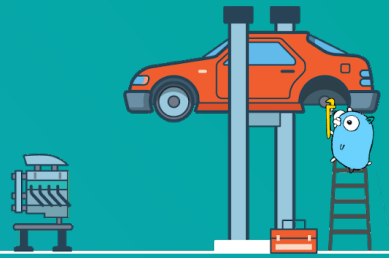
```
TLS_RSA_WITH_RC4_128_SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA
TLS_RSA_WITH_AES_128_CBC_SHA
TLS_RSA_WITH_AES_256_CBC_SHA
TLS_RSA_WITH_AES_128_CBC_SHA256
TLS_RSA_WITH_AES_128_GCM_SHA256
TLS_RSA_WITH_AES_256_GCM_SHA384
TLS_ECDHE_ECDSA_WITH_RC4_128_SHA
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
TLS_ECDHE_RSA_WITH_RC4_128_SHA
TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305
TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305

TLS_AES_128_GCM_SHA256
TLS_AES_256_GCM_SHA384
TLS_CHACHA20_POLY1305_SHA256
```

CIPHERSUITES SUPPORTED BY CRYPTO/TLS

~~TLS_RSA_WITH_RC4_128_SHA~~
TLS_RSA_WITH_3DES_EDE_CBC_SHA
TLS_RSA_WITH_AES_128_CBC_SHA
TLS_RSA_WITH_AES_256_CBC_SHA
~~TLS_RSA_WITH_AES_128_CBC_SHA256~~
TLS_RSA_WITH_AES_128_GCM_SHA256
TLS_RSA_WITH_AES_256_GCM_SHA384
~~TLS_ECDHE_ECDSA_WITH_RC4_128_SHA~~
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
~~TLS_ECDHE_RSA_WITH_RC4_128_SHA~~
TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA

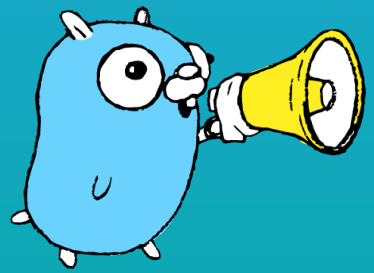
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
~~TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256~~
~~TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256~~
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305
TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305

TLS_AES_128_GCM_SHA256
TLS_AES_256_GCM_SHA384
TLS_CHACHA20_POLY1305_SHA256



Most of the value of the Go cryptography libraries is in what they **don’t** ship.

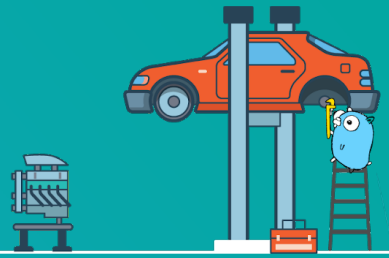
No knobs. Curated selection of features.



Maintaining a cryptography library is an exercise in **resisting** complexity.

SECTION 4

How the Go cryptography libraries **stay** different

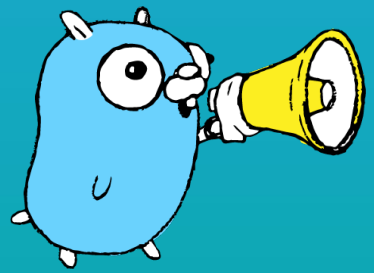


The maintainer asymmetry:

reviewing cryptographic code can take
10 times the time it takes to write it.

```
elliptic benchmarks:
name             old time/op    new time/op    delta
BaseMult           1.40ms ± 0%    1.44ms ± 0%   +2.66%  (p=0.029 n=4+4)
BaseMultP256        317µs ± 0%      50µs ± 0%  -84.14%  (p=0.029 n=4+4)
ScalarMultP256      854µs ± 2%     214µs ± 0%  -74.91%  (p=0.029 n=4+4)

ecdsa benchmarks:
name             old time/op    new time/op    delta
SignP256           377µs ± 0%     111µs ± 0%  -70.57%  (p=0.029 n=4+4)
SignP384          6.55ms ± 0%    6.48ms ± 0%   -1.03%  (p=0.029 n=4+4)
VerifyP256        1.19ms ± 0%    0.26ms ± 0%  -78.54%  (p=0.029 n=4+4)
KeyGeneration      319µs ± 0%      52µs ± 0%  -83.56%  (p=0.029 n=4+4)
```

	Commit message		
A	src/crypto/elliptic/p256_asm_ppc64le.s	+2517	-0
M	src/crypto/elliptic/p256_generic.go	+1	-1
A	src/crypto/elliptic/p256_ppc64le.go	+505	-0
		+3023	-1



“Secure” is relative to maintainer resources.

golang / go

Watch 3,339 | Unstar 59,854 | Fork 8,286

Code | Issues 4,553 | Pull requests 107 | Wiki | Security | Insights | Settings

AssemblyPolicy

Edit | New Page

Bryan C. Mills edited this page on Jun 27, 2018 · 9 revisions

- Minimize assembly
- Explain why it's needed
- Comment it well
- Test units individually

golang / go

Watch 3,339 | Unstar 59,854 | Fork 8,286

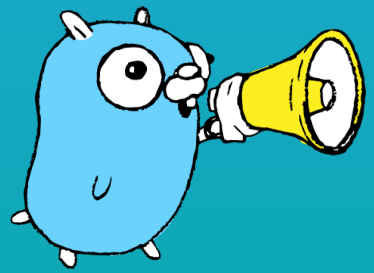
Code | Issues 4,553 | Pull requests 107 | Wiki | Security | Insights | Settings

AssemblyPolicy

Edit | New Page

Bryan C. Mills edited this page on Jun 27, 2018 · 9 revisions

- Minimize assembly
- Explain why it's needed
- Comment it well
- Test units individually
- Use code generation
- Write small testable units
- Write a reference Go implementation
- Document why the Go is slow
- Use test tooling and fuzzing



Policies need to be relative to maintainer resources, too!

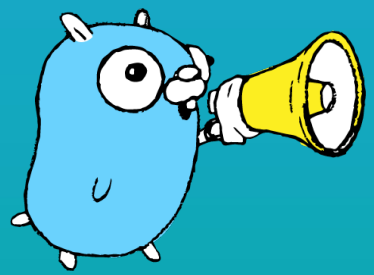
func **Mul64** 1.12

```
func Mul64(x, y uint64) (hi, lo uint64)
```

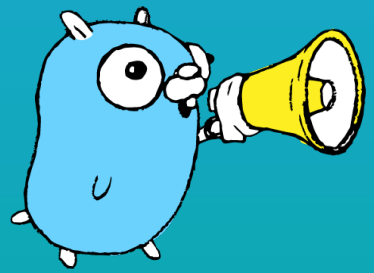
Mul64 returns the 128-bit product of x and y: (hi, lo) = x * y with the product bits' upper half returned in hi and the lower half returned in lo.

Tools to even the ground

- Fuzzing (oss-fuzz)
- Mutation testing (soon!)
- Reusable tests (golang.org/x/crypto/cryptotest)



Everyone wants their proposal accepted...



Everyone wants their proposal accepted...

... and everyone else's rejected.

proposal: x/crypto: deprecate unused, legacy and problematic packages #30141

Closed **FiloSottile** opened this issue on Feb 8 · 82 comments

- blowfish
- bn256
- cast5
- md4
- ripemd160
- tea
- twofish

proposal: x/crypto/curve25519: new X25519 API #32670

Open **FiloSottile** opened this issue 7 days ago · 4 comments

FiloSottile commented 7 days ago — Member

Proposed API

```
package curve25519

// Basepoint is the canonical Curve25519 generator.
var Basepoint []byte

// X25519 returns the result of the scalar multiplication (scalar * point),
// according to RFC 7748, Section 5. scalar, point and the return value
// are slices of 32 bytes.
//
// If point is Basepoint (but not if it's a different slice with the same contents)
// a precomputed implementation might be used to speed up the computation.
func X25519(scalar, point []byte) ([]byte, error)
```

proposal: crypto/tls: remove SSLv3 support #32716

Open **FiloSottile** opened this issue 5 days ago · 1 comment

FiloSottile commented 5 days ago • edited

Member

SSLv3 has been irreparably broken since the POODLE attack 5 years ago.

Go 1.12 Release Notes

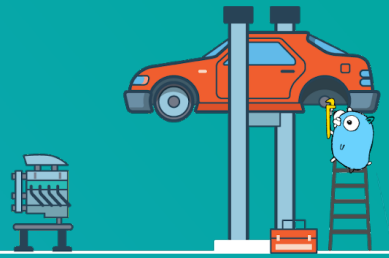
Core library

TLS 1.3

Go 1.12 adds opt-in support for TLS 1.3 in the `crypto/tls` package as specified by RFC 8446. It can be enabled by adding the value `tls13=1` to the `GODEBUG` environment variable. It will be enabled by default in Go 1.13.

SECTION 5

Conclusion



Every project has a **complexity budget**.

Whether you acknowledge it or not.

You should actively manage it.

Thank you!

Filippo Valsorda

Google

@FiloSottile