gosignify – reimplementing OpenBSD's signify
a case study in "C vs. Go"

Frank Braun

March 25, 2015

about me

- computer science background
- 10+ years experience in C
- 2+ years experience in Go
- IT security consulting
- freelancing (coding & secure servers)
- will work for Bitcoin
- main interest: "freedom technology"
- main project: Mute secure messaging system

problems in file distribution

- authentication (proven source)
- integrity (file was not tampered with)

Solution:

- public-key cryptography / digital signatures
- distribute public keys
- distribute signatures

⇒ user can verify authenticity and integrity of files

GnuPG

GnuPG is a common common solution for digital signatures:

- Debian
- git

Some problems:

- large code base (in C)
- does much more than digital signatures
- keys are large

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1

mQINBE3fpicBEACg43JhMrYwb8pp0b4/SP/q2/nBZWAVzAQylxvnoSaDzCcnNZLg
quXaBuyehqsUILNCu0Ysx/yV0Ha8aP652lzVmsaXiMwE+PtJgxb32EXKcBzRMTVw
gq1pgumyiizbFJV3ve7MM8U3s91l82znqKwt/WLINA8CJs0IbSQuasP3YCSPN3DH
Gu7eeufImU3Ij7dbKe8TdFd9UV8RIWJ0AALyHE4S8wHZpwpYP8ACaJo18KbNn0rz
kgYVtL8+2uu4HlIZQyPQ3myLmaXefrsO/OZFjaaWQYPNfTAZTyIUKQwVD56CrVUK
sQPfAUS34isnGDfj1EtcIksm/2PFXy/9IZUB5IE2f7KRKpzy7lf8LGenPgEEq/XV
VSPuXvpM2zJm7eJqhOorqDGaIv2DeRCXphnE5RhBvET0fPaEdVCTI3K4KVHwzbrM
C6htV5QEOH8fIF7z6NvgFIBnDtGoYEkQtn/l/SpYNzJe4Jxxjvl5Pb7momnIcFqZ
6m3Fw9KIF0P5AOhLY1L4I5lk18pQpSPF2Ia2kKigGkgFvqjOPIfML0ZOVX0J5Y6Y
iEW2mpGbsu6l9580h2fQUxWi5M7ZT0evwN6zsuVIAhtEo4S9Cs6scLLD1yS6aH9B
FyZPe3yMvP5myfCIR1qK31VjfFohbXsrD9IBDNx760aoenUwjHCArv0CewARAQAB
tCNGcmFuayBCcmF1biA8ZnJhbmtAY3J5cHRvZ3JvdXAubmV0PokCQAQTAQgAKgIb
AwULCQgHAwUVCgkICwUWAgMBAAIeAQIXgAIZAQUCU4XNjwUJC0nB5QAKCRA1ryrC
zsAOlNnBD/9Pu2vMeCShbNbbov9cE8bK9qSBisEEIfK7UCCR5KV19udPm4AnVFkh
UW2Lo1woGm+MJhsTYo21DhJ1z5TST63dmxs1as/eui1zIEhX9Jm8re+NG/O9/bmu
oHWgxNKziplkDjmexzGg5O0WIde7AGQNW25vb8UoLneVTEITuatI54L8sMOZNDhS
vnC85lPmx+Lbignw2iHy+ZUfOp+KN54IjxWUYbWPyINxZuHLH0l+pTfYETodnd9B
1AAZ0GwqzwHdz6zB6Bjem0LYdo7XsZFf1HMeksvuXwA+jlZd3hMELqNpLUwJMmH8
bxRTif/UZk7tTgQhDxqVu3QrUy6B56PvTqjZ6NV73xXbtKEPb4nKV8Lkq7B4jgU9
Jbg9DBudmiYqdgI73Nu8MxSYe0gzeJrMpYOx+AFLzery0b0JV5p08m1rfMDhZ4V5
35dFNTT7T9e3ASaqYlUngtc3PvYEZIieXjhyluBbESF1fe9K6eFDpi/BgrJEk++/
jCHQVic7jmAh70gCAm/jKDQoGZwm0X3ihhv+BzTbzmMkd3N2cJEq10//eDnzMPAy
3JQhkHlryhAUEU8MD855mIS/LbjyrUrR8g0sZA5yuTP71dK6o6ra8iiuN1rBsJFe
/nd76/HyN32mQ89h4ZilPFHqpUUVWKEOyxSCmL3U9Em3IuI3EoyJX7QhRnJhbmsg
QnJhdW4gPGZyYW5rQHNoYWRvd2xpZmUuY2M+iQI9BBMBCAAnAhsDBQsJCAcDBRUK
CQgLBRYCAwEAAh4BAheABQJThc2VBQkLScHlAAoJEDWvKsLOwA6UDUQQAI5tyAMF
oQkMsuKTjUhqQPP5Zs+Ff6nfir8eEwYaYQGvcjl/LatHctmvWdwfbH1SW4HC3zrY
x15NZB5L5SBl+790IyU/bAt6Gay7dAdlfLIoZQ30qM0Hr8a4J6lG0c0nH6HHa0TD
TWM4l4VC7LH1FmFYHtvbv5qwANetAaAqMDPcy4dd+0atq0Rq/2xS28bYNxt4TGxE
O0KfXf8L6dYmLrRnDcsGzKrh3/PV2ZH/aGkakq2vp/t2oHib9p1e+Saom0z8s0en
1b+vWKyMu3lgGTUUm/q96dBz/8egricAIeiUA3s30FTiRRoZEIsD0WrVq8z0dDad
hU7Mdt5EOZB[illegible]sliz0OzJMTIZ5ek8m6F
```

OpenBSD's signify

- small tool for key-pair generation, signing, and verification
- uses Ed25519 public-key signature system
- elliptic-curve signatures ⇒ small keys!
- base64 encoded keys
- SHA-512 hashing
- `bcrypt_pbkdf`

signify examples

Generate key-pair:

```
$ signify -G -p key.pub -s key.sec
$ cat key.pub
untrusted comment:  signify public key
RWRms08WDt5hdFc8RNgJBhwxfBfS6dA/9JVwPnYPPTj8PaOGlHg78/BM
```

Sign message:

```
$ signify -S -s key.sec -m msg.txt
$ cat msg.txt.sig
untrusted comment:  verify with key.pub
RWRms08WDt5hdIuAyFcjdMdObY9+hQKeK8vvgNmjGEeE5VJKuVcKDhLn
qXHSYYRK4urvHitZ9qdYIjOFsYpkS7+jgi/HUiHzwgE=
```

Verify message:

```
$ signify -V -p key.pub -m msg.txt
Signature Verified
```

reimplementation in Go: why?

- small scope, seems easy enought for a pet project
- personal interest
- good case study in “C vs. Go”
- I want a Gopher!

arc4random in C

```
arc4random_buf(keynum, sizeof(keynum));

...

arc4random_buf(enckey.salt, sizeof(enckey.salt));
```

Problem: arc4random is not portable

⇒ own portability layer is needed

arc4random in Go

```
err := io.ReadFull(rand.Reader, keynum[:])

...

err := io.ReadFull(rand.Reader, enckey.Salt[:])
```

No portability problem: `io.ReadFull` is in stdlib

bzero: simple case in C

```
uint8_t digest[SHA512_DIGEST_LENGTH];

...

SHA512Init(&ctx);
SHA512Update(&ctx, enckey.seckey, sizeof(enckey.seckey))
SHA512Final(digest, &ctx);

...

explicit_bzero(digest, sizeof(digest));
```

bzero: simple case in Go

```
digest := hash.SHA512(privateKey[:])
...
bzero.Bytes(digest)
```

bzero: simple implementation in Go

```go
package bzero

func Bytes(buf []byte) {
  for i := 0; i < len(buf); i++ {
    buf[i] = 0
  }
}
```

bzero: complex case in C

```
struct enckey {
  uint8_t pkalg[2];
  uint8_t kdfalg[2];
  uint32_t kdfrounds;
  uint8_t salt[16];
  uint8_t checksum[8];
  uint8_t keynum[KEYNUMLEN];
  uint8_t seckey[SECRETBYTES];
};

...

struct enckey enckey;

...

explicit_bzero(&enckey, sizeof(enckey));
```

bzero: complex case in Go

```
type enckey struct {
  Pkalg     [2]byte
  Kdfalg    [2]byte
  Kdfrounds [4]byte
  Salt      [16]byte
  Checksum  [8]byte
  Keynum    [KEYNUMLEN]byte
  Seckey    [SECRETBYTES]byte
}

...

var enckey enckey

...

bzero.Struct(&enckey)
```

bzero: complex implementation in Go

```go
package bzero

func Struct(strct interface{}) {
  s := reflect.ValueOf(strct).Elem()
  for i := 0; i < s.NumField(); i++ {
    f := s.Field(i)
    switch k := f.Kind(); k {
    case reflect.Array:
      Bytes(f.Slice(0, f.Len()).Bytes())
    case reflect.Slice:
      Bytes(f.Bytes())
    default:
      panic(fmt.Sprintf("bzero:␣cannot␣zero␣%s", k))
    }
  }
}
```

testing in OpenBSD: solution

Shell script in `regress/usr.bin/signify`:

```
#!/bin/sh
srcdir=$1
pubkey="$srcdir/regresskey.pub"
seckey="$srcdir/regresskey.sec"
orders="$srcdir/orders.txt"
forgery="$srcdir/forgery.txt"
set -e
cat $seckey | signify -S -s - -x test.sig -m $orders
diff -u "$orders.sig" test.sig
signify -V -q -p $pubkey -m $orders
signify -V -q -p $pubkey -m $forgery 2> /dev/null && exit 1
signify -S -s $seckey -x confirmorders.sig -e -m $orders
signify -V -q -p $pubkey -e -m confirmorders
diff -u $orders confirmorders
sha256 $pubkey $seckey > HASH
sha512 $orders $forgery >> HASH
signify -S -e -s $seckey -m HASH
rm HASH
signify -C -q -p $pubkey -x HASH.sig
true
```

testing in OpenBSD: problems

- framework not integrated into C, roll your own
- different language for test
- no coverage analysis!

gosignify testing

```
package signify
func Main(args ...string) error {
  ...
}

package main
func main() {
  if err := signify.Main(os.Args...); err != nil {
    if err != flag.ErrHelp {
      fmt.Fprintf(os.Stderr, ”%s:␣%s\n”, os.Args[0], err
    }
    os.Exit(1)
  }
}
```

⇒ tests in same language, fully integrated

⇒ coverage analysis!

comparison

	C	G
speed	++	+
binary size	++	−−
portability	+[1]	++
testing	−	++
stdlib	−−	++
productivity	−	++

[1] a lot of work!

conclusion

take away:

> *"Only prefer C over Go if you have extreme resource limitiations (e.g., embedded systems), otherwise the productivity and portability advantages of Go are well worth the sacrifices in speed and binary size."*

projects:

- Gosignify: `https://github.com/frankbraun/gosignify`
- Mute: `http://mute.berlin` (register for beta invitation)

contacts:

- `frank@cryptogroup.net` (please use PGP, key on key server)
- `94CC ADA6 E814 FFD5 89D0 48D7 35AF 2AC2 CEC0 0E94`