In code we trust: Secure multiparty code reviews with signatures and hash chains

Frank Braun

@thefrankbraun

2018-05-19

Reflections on Trusting Trust

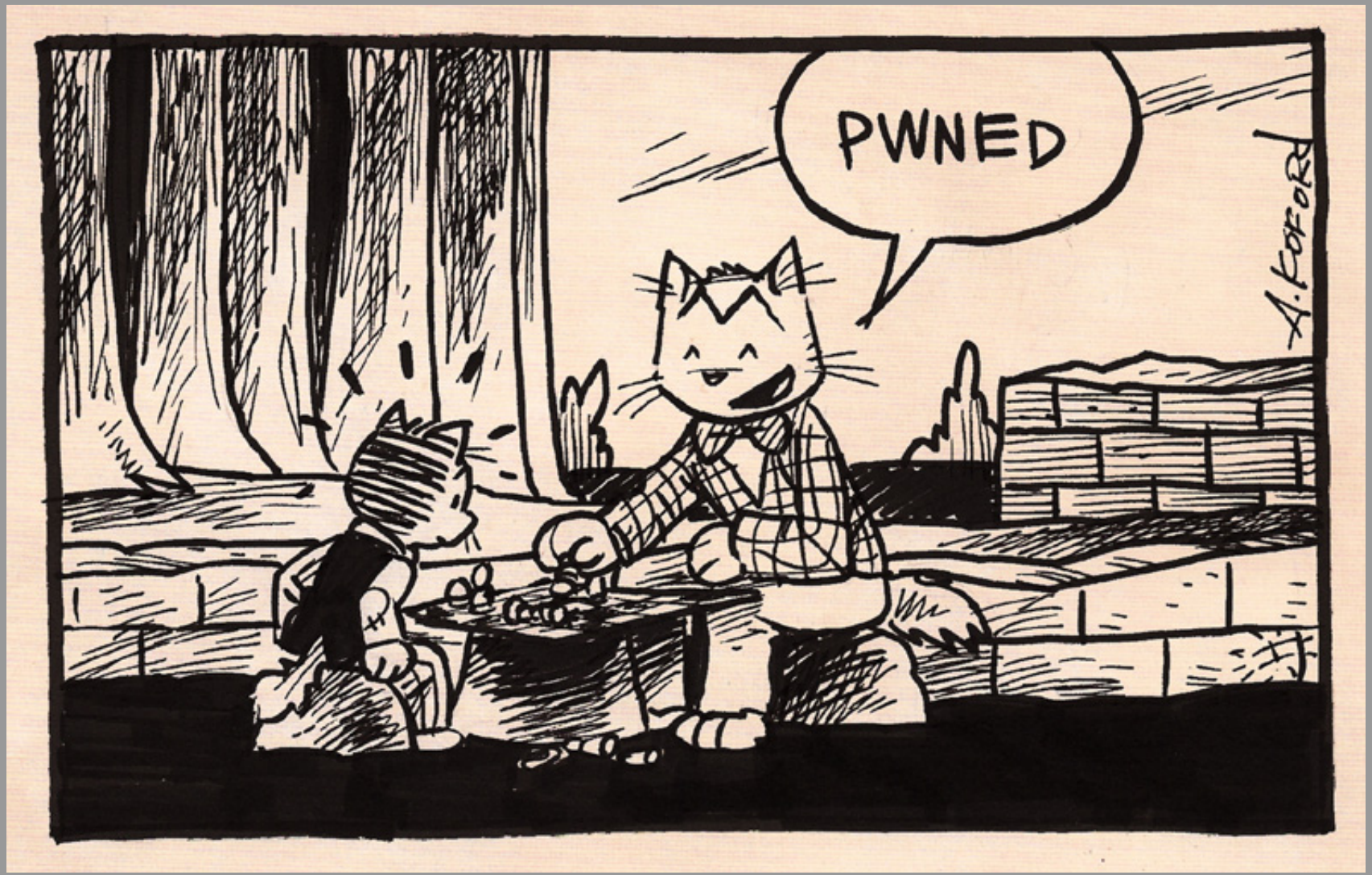
> *"To what extend should one trust a statement that a program is free of Trojan horses? Perhaps it is more important to trust the people who wrote the software."*
> *— Ken Thompson, Turing Award Lecture, 1984*

questions:

- how can we trust the people who wrote the software?
- how can we make sure we actually run the code they wrote?

$\Rightarrow$ this talk is not about making sure the code you execute is right, but making sure you execute the right code!

what happens if you execute the wrong code?



first approach

blindly executing something downloaded from the Internet is "problematic" (npm)

most common solution:

- developer hashes package
- developer signs hash
- developer publishes package, hash, and signature
- user downloads package, hash, and signature
- user verifies package hash
- user verifies signature

$\Rightarrow$ no key managment, user has to know pubkey of the developer

secure APT (Debian) packages

that's the model employed by apt in Debian and related distros

> *"By adding a key to apt's keyring, you're telling apt to trust everything signed by the key, and this lets you know for sure that apt won't install anything not signed by the person who possesses the private key."*—`https://wiki.debian.org/SecureApt`

reverse conclusion:
apt trusts everything signed by the person's private key

- `dpkg` has support for verifying GPG signatures of Debian package files, but this verification is disabled by default
- only repository metadata is verified!

Git version control system

- data integrity via Git's data structure (Merkle trees)
- Git allows to sign tags and commits with GPG

Git: signing & verifying tags

```
$ git tag -s v1.5 -m 'my signed 1.5 tag'
$ git tag -v v1.5
```

problem:

- tags are not unmodifiable

Git: signing & verifying commits

```
$ git commit -a -S -m 'signed commit'
$ git merge --verify-signatures signed-branch
```

only merging "fast-forwarding" branches gives some protection against regression (given on knows the `HEAD`)

problem:

- every commit needs to be signed
- user has to trust all developer keys

$\Rightarrow$ hard to deploy in practice

threat model / possible attacks

- key compromise
- developer coercion / wrench attack
- regression / suppressing updates

A developer being forced to give up his signing key would be a Black Swan event.

a possible solution

1. multiparty signatures (two-men rule), helps to mitigate:
   - key compromise
   - developer coercion / $5 wrench attack
2. key rotation, helps to mitigate:
   - key compromise
3. distribution of an unmodifiable code history, helps to mitigate:
   - regression / suppressing updates

but 3. requires a single source of truth (SSOT)

Codechain goals

- signed multiparty code reviews
- easy & built-in key rotation
- protection against $5 wrench attack
- regression protection, unmodifiable history
- minimal usable implementation written in Go ASAP
- focus on **source** distribution, not binary

out-of-scope:

- source code management (just use Git)
- single source of truth
- code distribution
- reproducible builds

Codechain design

(joined work with Jonathan Logan)

- the "unit" of code are directory trees
- the hash of a directory tree is a tree hash
- the code history is a sequence of unique tree hashes, starting from the hash of the empty tree
- the sequence of tree hashes and their signatures are recorded in a hash chain file
- the signatures contributes towards a m-of-n threshold
- code is distributed as a set of patch files which transform a directory tree *a* into a directory tree *b*
- patch files are named after the outgoing tree hash *a*

tree hashes

```
$ cd $GOPATH/src/github.com/frankbraun/codechain/doc/helloproject

$ codechain treehash -l
f ab81f3080f71a034c90dc0ca64b62295d3a75a23ec1b0f498dfda4a34325ae3a README.md
f ad125cc5c1fb680be130908a0838ca2235db04285bcdd29e8e25087927e7dd0d hello.go

$ codechain treehash
d844cbe6f6c2c29e97742b272096407e4d92e6ac7f167216b321c7aa55629716

$ codechain treehash -l | sha256sum
d844cbe6f6c2c29e97742b272096407e4d92e6ac7f167216b321c7aa55629716
```

patch files

```
codechain patchfile version 1
treehash e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495
+ f ab81f3080f71a034c90dc0ca64b62295d3a75a23ec1b0f498dfda4a34
dmppatch 2
@@ -0,0 +1,45 @@
+## Example project for Codechain walkthrough%0A
+ f ad125cc5c1fb680be130908a0838ca2235db04285bcdd29e8e2508792
dmppatch 2
@@ -0,0 +1,78 @@
+package main%0A%0Aimport (%0A%09%22fmt%22%0A)%0A%0Afunc main
%7B%0A%09fmt.Println(%22hello world!%22)%0A%7D%0A
treehash d844cbe6f6c2c29e97742b272096407e4d92e6ac7f167216b321
```

hash chain format

- a hash chain is stored in a simple newline separated text file
- each hash chain entry corresponds to a single line of the form:

```
hash-of-previous current-time type type-fields ...
```

where:

- hash-of-previous is the SHA256 hash of the previous line (without newline)
- the fields are separated by single white spaces
- the current-time is encoded as an ISO 8601 string in UTC
- all hashes in a hash chain are SHA256 hashes encoded in hex notation
- hex encodings have to be lowercase
- all public keys are Ed25519 keys and they and their signatures are encoded in base64 (URL encoding without padding)
- comments are arbitrary UTF-8 sequences (without newlines)

hash chain types

there are six different types of hash chain entries:

```
cstart
source
signtr
addkey
remkey
sigctl
```

- a hash chain must start with a `cstart` entry
- that is the only line where this type must appear

type `cstart`

A `cstart` entry starts a new hash chain.

```
hash-of-prev cur-time cstart pubkey nonce signature [comment]
```

type `source`

Marks a new source tree state for publication (from developer).

```
hash-of-prev cur-time source source-hash pubkey sig [comment]
```

Signature is over the source-hash and the optional comment.

type `signtr`

Signs a previous entry and approves all code changes and changes to the set of signature keys and `m` up to that point.

```
hash-of-prev cur-time signtr hash-of-chain-entry pubkey sig
```

It does not have to sign the previous line ($\rightarrow$ detached signatures).

type addkey

Marks a pubkey for addition to the list of approved signature keys.

```
hash-of-prev cur-time addkey w pubkey sig [comment]
```

The weight (towards `m`) is denoted by `w`.

type `remkey`

A `remkey` entry marks a signature pubkey for removal.

```
hash-of-prev cur-time remkey pubkey
```

type `sigctl`

Denotes an update of `m`, the minimum number of necessary signatures to approve state changes (the threshold).

```
hash-of-prev cur-time sigctl m
```

start Codechain walkthrough with example project

```
$ cd doc/hellproject
$ ls

hello.go README.md
```

let's generate a key pair for Alice

```
$ codechain keygen

passphrase:
confirm passphrase:
comment (e.g., name; can be empty):
Alice <alice@example.com>
secret key file created:
/home/frank/.config/codechain/secrets/
KDKOGoY8ErjOnbDQb4k8SZFMvWdAIb-x6FGKKCRby70
public key with signature and optional comment:
KDKOGoY8ErjOnbDQb4k8SZFMvWdAIb-x6FGKKCRby70
JNBIdjLOu20He3c-Dn7sjpspO8bmKFxTIOItfZkqieb8h218t3g-QooD
'Alice <alice@example.com>'
```

let's start using Codechain for our example project

```
$ codechain start -s ~/.config/codechain/secrets/KDKOGoY8ErjOnbDQb4k8SZFMvWdAIb-x6FGKKCR
passphrase:

e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
2018-05-19T00:07:02Z cstart KDKOGoY8ErjOnbDQb4k8SZFMvWdAIb-x6FGKKCRby70
sVnVenzHyCOV6nLUkCKg6ARIlkYsTV-n
0UmUcDFZ2j3WWnqzEdxX-wzofWIhF3O0Rm1tT6qMUwLu8a1R5MwbK5zDongYZKccpA37Vp6Sp3m0xSreGskzCg
Alice <alice@example.com>
```

let's add Bob (who already has a key) as reviewer

```
$ codechain addkey 91HOu2fvkjHd5S0LtAWTl6dYBk5cqB-NWiJqc0c_7Gc
Xsr_L-1_5_B56vocve8s3Pb3vJoc-jpa2-tzIQhEjuoytYfcAiONu3er6RnVNMcsPuZFeqWCQKBwka-F-c13Ag
'Bob <bob@example.com>'

40c7e5ca4be98e9cae6931afa4ac09e11ecb1ce20fa18d0faaabfac7e8fad071
2018-05-19T00:09:44Z addkey 1 91HOu2fvkjHd5S0LtAWTl6dYBk5cqB-NWiJqc0c_7Gc
Xsr_L-1_5_B56vocve8s3Pb3vJoc-jpa2-tzIQhEjuoytYfcAiONu3er6RnVNMcsPuZFeqWCQKBwka-F-c13Ag
Bob <bob@example.com>
```

increase number of necessary signers to two

```
$ codechain sigctl --m 2

34cd10effd93e67ba96fefb29ea751d013459a6de11cc117cf1deacd77d6b7be
2018-05-19T00:10:25Z sigctl 2
```

publish first release

```
$ codechain publish

opening keyfile: /home/frank/.config/codechain/secrets/KDKOGoY8ErjOnbDQb4k8SZFMvWdAIb-x6
passphrase:
publish patch? [y/n]: y
comment describing code change (can be empty):
first release

92d2fc6687b0d36d045adaf34a1615e513ef0e2dc60384cfe19863e9753567f8
2018-05-19T00:11:44Z source d844cbe6f6c2c29e97742b272096407e4d92e6ac7f167216b321c7aa5562
KDKOGoY8ErjOnbDQb4k8SZFMvWdAIb-x6FGKKCRby70
r5aZCYGwWCFppaMDV7XSOHoyCl3qbUKGiSuYzjsTl4C0W9n0tCa0MXDy_fOwspV9f4_o0kMcb6XZS706ml3FAQ
first release
```

review changes

```
$ codechain review

opening keyfile: /home/frank/.config/codechain/secrets/KDKOGoY8ErjOnbDQb4k8SZFMvWdAIb-x6
passphrase:
signer/sigctl changes:
0 addkey 1 91HOu2fvkjHd5S0LtAWTl6dYBk5cqB-NWiJqc0c_7Gc Bob <bob@example.com>
0 sigctl 2
confirm signer/sigctl changes? [y/n]: y
patch 1/1
first release
developer: KDKOGoY8ErjOnbDQb4k8SZFMvWdAIb-x6FGKKCRby70
Alice <alice@example.com>
review patch (no aborts)? [y/n]: y
sign patch? [y/n]: y
d258ce20943beeed2d483096702a1449447f112dec7d907d50c285c649c17a24
2018-05-19T00:12:48Z signtr
d258ce20943beeed2d483096702a1449447f112dec7d907d50c285c649c17a24 KDKOGoY8ErjOnbDQb4k8SZF
HKlLKnYSCVzc4b-erETK50EN5gKRKZQsT16grv7eFBklFqXBFoSXSmcY99HLWhAP9BJcA6c3Px1trNBns3KkDA
```

see current status of project

```
$ codechain status

no signed releases yet

signers (2-of-2 required):
1 91HOu2fvkjHd5S0LtAWTI6dYBk5cqB-NWiJqc0c_7Gc Bob <bob@example.com>
1 KDKOGoY8ErjOnbDQb4k8SZFMvWdAIb-x6FGKKCRby70 Alice <alice@example.com>

unsigned entries:
1 source d844cbe6f6c2c29e97742b272096407e4d92e6ac7f167216b321c7aa55629716 first release

head:
2e34e23ee293e8c0ed174639d325eb3e30f5337d5c5846380367724e93cb619e

tree matches d844cbe6f6c2c29e97742b272096407e4d92e6ac7f167216b321c7aa55629716
```

let's build distribution for Bob to review first release

```
$ codechain createdist -f /tmp/dist.tar.gz
```

as Bob, apply the distribution in an empty directory

```
$ cd ~/helloproject
$ codechain apply -f /tmp/dist.tar.gz
$ find . -type f

./.codechain/hashchain
./.codechain/patches/e3b0c44298fc1c149afbf4c8996fb92427ae
```

Bob reviews the changes and creates a detached signature

```
$ codechain review -d

opening keyfile: /home/frank/bob.bin
passphrase:
patch 1/1
first release
developer: KDKOGoY8ErjOnbDQb4k8SZFMvWdAIb-x6FGKKCRby70
Alice <alice@example.com>
review patch (no aborts)? [y/n]: y
sign patch? [y/n]: y
2e34e23ee293e8c0ed174639d325eb3e30f5337d5c5846380367724e93cb619e
91HOu2fvkjHd5S0LtAWTl6dYBk5cqB-NWiJqc0c_7Gc
xffZultos-MCbI4cNzAzAoccuDSnpL2nq_BsQanIruYM3RXoD9kdC6WiPEUkxrphKdG742IgBWlB3LwY0i1ZCw
```

now Alice can add the detached signature

```
$ codechain review -a 2e34e23ee293e8c0ed174639d325eb3e30f5337d5c5846380367724e93cb619e
91HOu2fvkjHd5S0LtAWTI6dYBk5cqB-NWiJqc0c_7Gc xffZultos-MCbI4cNzAzAoccuDSnpL2nq_BsQanIruYM3

2e34e23ee293e8c0ed174639d325eb3e30f5337d5c5846380367724e93cb619e
2018-05-19T00:34:51Z signtr 2e34e23ee293e8c0ed174639d325eb3e30f5337d5c5846380367724e93cb
91HOu2fvkjHd5S0LtAWTI6dYBk5cqB-NWiJqc0c_7Gc
xffZultos-MCbI4cNzAzAoccuDSnpL2nq_BsQanIruYM3RXoD9kdC6WiPEUkxrphKdG742IgBWlB3LwY0i1ZCw
```

which gives us our first signed release

```
$ codechain status

signed releases:
d844cbe6f6c2c29e97742b272096407e4d92e6ac7f167216b321c7aa55629716 first release

signers (2-of-2 required):
1 91HOu2fvkjHd5S0LtAWTI6dYBk5cqB-NWiJqc0c_7Gc Bob <bob@example.com>
1 KDKOGoY8ErjOnbDQb4k8SZFMvWdAIb-x6FGKKCRby70 Alice <alice@example.com>

no unsigned entries

head:
9f97737b292f66e52c06027871be328006f125a9d86fbe1fc4f03ff98303e36f

tree matches d844cbe6f6c2c29e97742b272096407e4d92e6ac7f167216b321c7aa55629716
```

we can publish the first release now

```
$ codechain createdist -f /tmp/helloproject.tar.gz
```

users can apply it & verify the hash chain contains the head

```
$ cd ~/helloproject
$ codechain apply -f /tmp/helloproject.tar.gz -head 9f97737b292f66e52c06027871be328006f
```

the tree hash now matches the first signed release

```
$ codechain treehash

d844cbe6f6c2c29e97742b272096407e4d92e6ac7f167216b321c7aa
```

show the complete hash chain (shortened hashes)

```
$ codechain status -p

e3b0c44298fc1c149afbf4c8996fb924 2018-05-19T00:07:02Z cstart KDKO
40c7e5ca4be98e9cae6931afa4ac09e1 2018-05-19T00:09:44Z addkey 1 91H
34cd10effd93e67ba96fefb29ea751d0 2018-05-19T00:10:25Z sigctl 2
92d2fc6687b0d36d045adaf34a1615e5 2018-05-19T00:11:44Z source d844c
d258ce20943beeed2d483096702a1449 2018-05-19T00:12:48Z signtr d258c
2e34e23ee293e8c0ed174639d325eb3e 2018-05-19T00:34:51Z signtr 2e34e
```

Codechain

- Codechain beta is available now

→ `https://github.com/frankbraun/codechain`

- minimal code base, Go only, cross-platform (tested on Linux)
- ≈6000 lines of code (plus vendored dependencies)
- public domain (`http://unlicense.org/`)
- Codechain depends on the `git` binary (for `git diff`), but that's optional
- Codechain is reviewed and signed with Codechain (2-of-2)

current head of Codechain's hash chain:

`e16029c49f470e7f007d03bb3271ff7f1a2375bc548313d2a2aa2bf9ac66595a`

conclusion

While it is good that code signing is widely deployed now it doesn't solve important attack vectors.

Codechain mitigates:

- key compromise (with multiparty signatures & key rotation)
- developer coercion (with multiparty signatures & key rotation)
- somewhat mitigates regression / suppression of updates

$\Rightarrow$ a solution is available now which improves upon the status quo

future work: build a single source of truth (SSOT) system

acknowledgments: Jonathan Logan of Cryptohippie, Inc.

contacts:

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

Slides: `http://frankbraun.org/in-code-we-trust.pdf`

thank you very much for your attention! questions?