Clevis and Tang: Overcoming the unlocking problem

Christoph Biedl

Debian MiniDebconfOnline
May 30th 2020

This is about

- “Clevis is a plugable framework for automated decryption”
- “Tang is a server for binding data to network presence”

Some statements

- Why (hard disk) encryption? — perhaps GDPR?
- How it works: LUKS (2004/2017)
- Not a topic: Ciphers, key lengths
- Remember: `https://m.xkcd.com/538/`

The unlocking problem

How to store and retrieve the LUKS passphrase?

- Manually
  - Interactive, “`dropbear-initramfs`”
  - Does not scale — we want automated unlocking
- Secret on a second medium
  - Entire computer stolen
  - Management
- Key server
  - Key escrow
  - Network infrastructure compromised (MITM)
  - Transport protection, identity management
  - The key server has too much knowledge
  - . . . becomes an attack target

The big picture

Clevis and Tang

- Clevis encrypts/decrypts a message
- . . . with a little help from Tang and/or others
- clevis-luks uses Clevis to encrypt/decrypt a LUKS passphrase
- It's all about automation

Clevis

Clevis does:

- Symmetric encryption
- plaintext (PT) $\mapsto$ `clevis encrypt` $\mapsto$ encrypted (JWE)
- encrypted (JWE) $\mapsto$ `clevis decrypt` $\mapsto$ plaintext (PT)
- Using JOSE, eventually openssl
- How to store the encryption key?
- Clevis uses “pin”s

pin

A pin does:

- plaintext (PT) $\mapsto$ pin $\mapsto$ encrypted (JWE)
- encrypted (JWE) $\mapsto$ pin $\mapsto$ plaintext (PT)
- The pin handles key storage
- Example: Store in the file system

Tang

Tang is an implementation of a pin.

Features:

- The key is not stored(sic!)
- A derived information is stored, irreversible
- . . . unless there's a little help from another instance
- That instance doesn't have the key either

Elgamal encryption

Alice wants to send a secret message K to Bob

- Alice creates keypair A (private), a (public)
- Bob creates keypair B (private), b (public)
- Public keys are public
- Alice creates $k = f(K, A, b)$
- Alice publishes k
- Bob can compute $K = g(k, B, a)$

MacCallum-Relyea Exchange

Alice wants to send a secret message K to . . . herself

- Alice wants to send Bob a secret message K
- Alice creates keypair A (private), a (public)
- Bob creates keypair B (private), b (public)
- Public keys are public
- Alice creates $k = f(K, A, b)$
- Alice keeps k
- Alice creates X, and x (derived from X)
- Alice sends x
- Bob computes $x' = h(x, B)$, and sends x' back
- Alice can compute $K = g'(k, a, X, x')$

Tang, summary

- Key re-creation requires presence of the Tang server
- Partial reachability is a feature
- Tang server
  - is stateless
  - has no database
  - small and cheap

How else to use Clevis

Pins:

- tpm2
- more to come

Combining pins:

- With redundancy and thresholds
- Shamir's Secret Sharing (SSS, 1979)

Other use cases:

- Everything that wants passphrase

Complete LUKS workflow

- Create LUKS partition as usual
- Bind to Clevis:
  - `clevis luks bind -d DEVICE <pin> <configuration>`
  - Clevis creates another LUKS passphrase
  - Clevis encrypts that passphrase using the pin
- Unlocking:
  - `clevis unlock -d DEVICE`
  - Automation available: initramfs, dracut, systemd, udisks2

Status in Debian

- First included in “stable” = Debian 10 (“buster”)
  - All but clevis: OK-ish
  - clevis stable: 11-2 — no initramfs and somewhat buggy
- clevis unstable/testing: 13-2
- Backport is trivial

Who is behind this

- Initially Nathaniel McCallum
- Currently Sergio Correia
- Network-Bound Disk Encryption (NBDE)
- `https://github.com/latchset/...{clevis,tang,jose,luksmeta}`

Finally

- Wanted: Users, and feedback
- Backup your LUKS headers!