Disk encryption in NetBSD

Taylor R Campbell
riastradh@NetBSD.org

BSDCan 2023
Ottawa, Canada
May 20, 2023

Disk encryption in NetBSD

`https://www.NetBSD.org/gallery/presentations/riastradh/bsdcan2023/diskencryption.pdf`

Goal

Present a logical disk device that behaves just like an underlying physical disk device for file systems, swap, and other block storage.

```
    logical disk                    physical disk
+---------------------+     +---------------------+
| 512-byte sector 0   | <-> | 512-byte sector 0   |
+---------------------+     +---------------------+
| 512-byte sector 1   | <-> | 512-byte sector 1   |
:                     :     :                     :
:                     :     :                     :
| 512-byte sector n-1 | <-> | 512-byte sector n-1 |
+---------------------+     +---------------------+
```

Goal

Present a logical disk device that behaves just like an underlying physical disk device for file systems, swap, and other block storage.

```
    logical disk                physical disk
+---------------------+     +---------------------+
| 512-byte sector 0   | <-> | 512-byte sector 0   |
+---------------------+     +---------------------+
| 512-byte sector 1   | <-> | 512-byte sector 1   |
:                     :     :                     :
:                     :     :                     :
| 512-byte sector n-1 | <-> | 512-byte sector n-1 |
+---------------------+     +---------------------+
```

Sector writes must be atomic (if physical disk guarantees this)

Atomicity

- Applications like file systems and databases assume sector writes are atomic (or close to it[1])
- Breaking this can lead to data corruption on power loss

[1] `https://www.sqlite.org/atomiccommit.html`

Threat model

- Theft of laptop

Threat model

- Theft of laptop
- Tampering with laptop

Threat model

- Theft of laptop
- Tampering with laptop

Threat model

- Theft of laptop
- ~~Tampering with laptop~~

Threat model

- Theft of laptop (while powered off or hibernating)
- ~~Tampering with laptop~~

Threat model

- Theft of laptop (while powered off or hibernating)
- ~~Tampering with laptop~~
- Recycling a disk

Not threat model

- Tampering with laptop
  - Border search
  - ‘Evil maid’

  Adversary could modify firmware, install hardware keylogger, *etc.*—can’t be detected/prevented by storage protocol alone
- MITM on network storage devices
  - iSCSI

  Practical limitations with a disk device

Security properties

- (ideal) Adversary can't learn anything about what is stored

Security properties

- (ideal) Adversary can't learn anything about what is stored
- (realistic) Adversary can't learn *much* about what is stored
  - Content of fixed-shape data indistinguishable
  - Different shapes—directory structures, sparse file allocation, write patterns, file systems—may be distinguishable

Caveats

1. Zero-fill
2. Wear-levelling
3. Access patterns on network storage (even with just passive eavesdropper, not active MITM)

Caveats

1. Zero-fill
   - Exposes which sectors written, possibly shape of data
   - Scrubbing disk first hides shape but bad for SSD performance
2. Wear-levelling
3. Access patterns on network storage (even with just passive eavesdropper, not active MITM)

Caveats

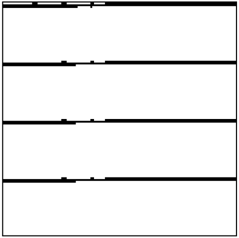
1. Zero-fill
   - Exposes which sectors written, possibly shape of data
   - Scrubbing disk first hides shape but bad for SSD performance
2. Wear-levelling
   - SSD delays costly erasure with virtual sector remapping
   - Adversary may see many snapshots of some sectors
3. Access patterns on network storage (even with just passive eavesdropper, not active MITM)

Caveats

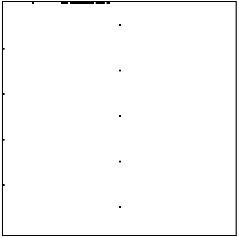
1. Zero-fill
   - Exposes which sectors written, possibly shape of data
   - Scrubbing disk first hides shape but bad for SSD performance
2. Wear-levelling
   - SSD delays costly erasure with virtual sector remapping
   - Adversary may see many snapshots of some sectors
3. Access patterns on network storage (even with just passive eavesdropper, not active MITM)
   - Can't conceal without cooperation of network protocol

Shape as proxy for content: newfs

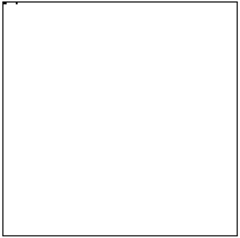
ffs



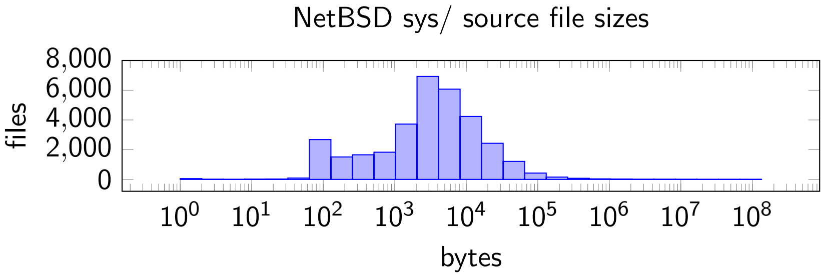
lfs

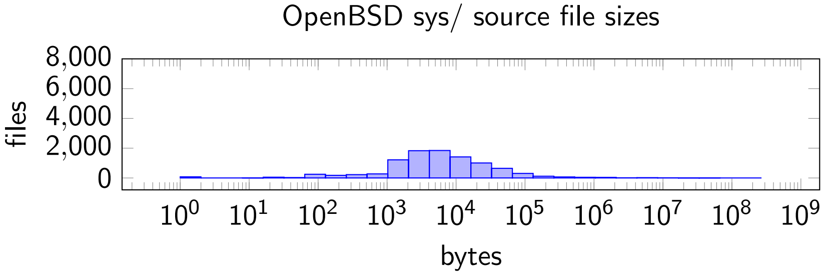


msdos



Shape as proxy for content: NetBSD vs OpenBSD





No ciphertext expansion

- Atomic sector writes means No ciphertext expansion allowed
  - No counter per block for nonce-based ciphers like AES-CTR
  - No authentication tags to detect forgeries

No ciphertext expansion

- Atomic sector writes means No ciphertext expansion allowed
  - No counter per block for nonce-based ciphers like AES-CTR
  - No authentication tags to detect forgeries
- Can work around by adding logging layer

No ciphertext expansion

- Atomic sector writes means No ciphertext expansion allowed
  - No counter per block for nonce-based ciphers like AES-CTR
  - No authentication tags to detect forgeries
- Can work around by adding logging layer
  - . . . at cost of 2x write amplification

No ciphertext expansion

- Atomic sector writes means No ciphertext expansion allowed
  - No counter per block for nonce-based ciphers like AES-CTR
  - No authentication tags to detect forgeries
- Can work around by adding logging layer
  - . . . at cost of 2x write amplification
  - . . . and still won't detect rollback

cgd(4) encryption

- Each logical cgd(4) device has an encryption key for a ‘tweakable block cipher’

$$C = E_k^t(P)$$

- Each sector is encrypted independently with sector number as tweak

cgd(4) encryption

- Each logical cgd(4) device has an encryption key for a ‘tweakable block cipher’ (not quite)

$$C = E_k^t(P)$$

- Each sector is encrypted independently with sector number as tweak

cgd(4) encryption

- Each logical cgd(4) device has an encryption key for a ‘tweakable block cipher’ (not quite)

$$C = E_k^t(P)$$

- Each sector is encrypted independently with sector number as tweak

$$physicalsector_i = E_k^{\mathrm{littleendian}(i)}(logicalsector_i)$$

Not quite tweakable block ciphers

- 512-byte sector broken into 32 16-byte blocks:

  `bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb`

- User writes to some 16-byte blocks in the middle:

  `bbbbbbbbbwbbbbwwbbbbbbbbwbbbbbbb`

Not quite tweakable block ciphers

```
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
bbbbbbbbbwbbbbwwbbbbbbbbwbbbbbbb
```

AES-XTS reveals which 16-byte blocks changed:

```
---------*----**--------*-------
```

AES-CBC reveals which 16-byte block prefixes of disk sectors didn’t change:

```
---------***********************
```

Ideally entire sector is randomized by any change to content:

```
********************************
```

(Can’t use stream ciphers like AES-GCM or ChaCha20/Poly1305 because of multiple snapshots.)

Key management

- cgdconfig(8) userland tool configures cgd(4) with:
  - physical disk
  - cipher
  - key
  - verification method
- Driven by parameters file, e.g. `/etc/cgd/wd0e`:

```
algorithm aes-cbc;
iv-method encblkno1;
keylength 256;
verify_method ffs;
keygen pkcs5_pbkdf2/sha1 {
        iterations 39361;
        salt AAAAgMoHiYonye6KogdYJAobCHE=;
};
```

Key derivation

- Can derive key from:
  - key stored in parameters file
  - random key derived from /dev/random or /dev/urandom
  - shell command
  - password using stored salt and cost
- `cgdconfig -g` calibrates timing
- Can combine multiple keygen blocks—key combined with xor

Verification and 2-factor authentication

- Recall: zero ciphertext expansion
- Even with password-based key derivation, nothing in cgd(4) ciphertext helps to guess password without also guessing salt
- Given key (e.g., derived from password and salt), `verify_method` checks for a known pattern like ffs or gpt, or just re-entering password, to verify password entry

Verification and 2-factor authentication

- Recall: zero ciphertext expansion
- Even with password-based key derivation, nothing in cgd(4) ciphertext helps to guess password without also guessing salt
- Given key (e.g., derived from password and salt), `verify_method` checks for a known pattern like ffs or gpt, or just re-entering password, to verify password entry
- 2FA: Store cgd parameters file on separate USB flash drive

Verification and 2-factor authentication

- Recall: zero ciphertext expansion
- Even with password-based key derivation, nothing in cgd(4) ciphertext helps to guess password without also guessing salt
- Given key (e.g., derived from password and salt), `verify_method` checks for a known pattern like ffs or gpt, or just re-entering password, to verify password entry
- 2FA: Store cgd parameters file on separate USB flash drive
- Use `cgdconfig -G` to back up key in another parameters file with no password—offline in a safe place

AES risk: side channels

- Table-based AES software leaks keys through cache timing
- CVE-2005-1797
- Demonstrated in practice against Linux dm-crypt[2]
- Requires arbitrary code execution to trigger disk I/O

[2] Dag Arne Osvik, Adi Shamir, and Eran Tromer, 'Cache Attacks and Countermeasures: The Case of AES'. Topics in Cryptology—CT-RSA 2006, pp. 1–20. `https://link.springer.com/chapter/10.1007/11605805_1`

AES risk: side channels

- Table-based AES software leaks keys through cache timing
- CVE-2005-1797
- Demonstrated in practice against Linux dm-crypt[2]
- Requires arbitrary code execution to trigger disk I/O
  - …like JavaScript in a web browser

[2] Dag Arne Osvik, Adi Shamir, and Eran Tromer, 'Cache Attacks and Countermeasures: The Case of AES'. Topics in Cryptology—CT-RSA 2006, pp. 1–20. https://link.springer.com/chapter/10.1007/11605805_1

Solution: Don't do AES that way

- All table-based AES software ripped out of NetBSD 10 kernel
- Replaced by:
  - AES-NI on newer x86
  - ARMv8.0-AES on newer Arm
  - AES Padlock on VIA x86
  - Vector permutation AES on older x86 (SSSE3), Arm
  - Vectorized bitsliced AES on much older x86 (SSE2)
  - Portable C bitsliced AES from BearSSL

Solution: Don't do AES that way

- All table-based AES software ripped out of NetBSD 10 kernel
- Replaced by:
  - AES-NI on newer x86
  - ARMv8.0-AES on newer Arm
  - AES Padlock on VIA x86
  - Vector permutation AES on older x86 (SSSE3), Arm
  - Vectorized bitsliced AES on much older x86 (SSE2)
  - Portable C bitsliced AES from BearSSL
  - ...got an exotic vector unit like SPARC or MIPS? Happy to help adapt it to that!

Supported algorithms

Ciphers:

- `blowfish-cbc`
- `3des-cbc`
- `aes-cbc`

Password-based key derivation:

- `pkcs5_pbkdf2/sha1`

Supported algorithms

Ciphers:

- ~~blowfish-cbc~~ (still supported, but don't use it)
- ~~3des-cbc~~ (still supported, but don't use it)
- aes-cbc

Password-based key derivation:

- pkcs5_pbkdf2/sha1

Supported algorithms

Ciphers:

- ~~`blowfish-cbc`~~ (still supported, but don't use it)
- ~~`3des-cbc`~~ (still supported, but don't use it)
- `aes-cbc`
- `aes-xts`
- `adiantum`

Password-based key derivation:

- `pkcs5_pbkdf2/sha1`
- `argon2id`

New cipher: AES-XTS

- Tweakable 16-byte block cipher based on AES
- IEEE Std 1619–2007
- NIST SP 800–38E
- Faster than AES-CBC encryption
- Comparable to AES-CBC decryption
- Not a tweakable wide-block cipher

New cipher: AES-XTS

- Tweakable 16-byte block cipher based on AES
- IEEE Std 1619–2007
- NIST SP 800–38E
- Faster than AES-CBC encryption
- Comparable to AES-CBC decryption
- Not a tweakable wide-block cipher
  - Leaks slightly more than AES-CBC

New cipher: Adiantum

- Based on ChaCha, Poly1305, NH, and AES
  - One AES call per block (disk sector), so not a bottleneck
- Designed by Paul Crowley and Eric Biggers at Google
- Well-understood design with comfortable security bounds proven relative to security of components[3]
- Suited for CPUs without hardware AES acceleration
- Tweakable wide-block cipher (arbitrary size $\geq$16-byte)

[3] Paul Crowley and Eric Biggers, 'Adiantum: length-preserving encryption for entry-level processors'. IACR Transactions on Symmetric Cryptology, 2018(4), pp. 39–61. https://doi.org/10.13154/tosc.v2018.i4.39-61

New cipher: Adiantum

- Based on ChaCha, Poly1305, NH, and AES
  - One AES call per block (disk sector), so not a bottleneck
- Designed by Paul Crowley and Eric Biggers at Google
- Well-understood design with comfortable security bounds proven relative to security of components[3]
- Suited for CPUs without hardware AES acceleration
- Tweakable wide-block cipher (arbitrary size $\geq$16-byte)
  - Best disk encryption security of all choices

[3] Paul Crowley and Eric Biggers, 'Adiantum: length-preserving encryption for entry-level processors'. IACR Transactions on Symmetric Cryptology, 2018(4), pp. 39–61. `https://doi.org/10.13154/tosc.v2018.i4.39-61`

New key derivation: Argon2

- PBKDF2-SHA1 can only use single-threaded CPU time before you get bored to raise adversary's costs
- Argon2 can use memory and parallelism too
- Especially at boot time: memory is free, CPUs are idle

```
algorithm       adiantum;
iv-method       encblkno1;
keylength       256;
verify_method   gpt;
keygen argon2id {
        iterations 32;
        memory 5214;
        parallelism 2;
        version 19;
        salt AAAAgLZ5QgleU2m/Ib6wiPYxz98=;
};
```

Configuring multiple disks

```
Configuring CGD devices.
/dev/dk1's passphrase:
```

Configuring multiple disks

```
Configuring CGD devices.
/dev/dk1's passphrase:
re-enter device's passphrase:
```

Configuring multiple disks

```
Configuring CGD devices.
/dev/dk1's passphrase:
re-enter device's passphrase:
/dev/wd0e's passphrase:
```

Configuring multiple disks

```
Configuring CGD devices.
/dev/dk1's passphrase:
re-enter device's passphrase:
/dev/wd0e's passphrase:
/dev/ld0a's passphrase:
```

Configuring multiple disks

```
Configuring CGD devices.
/dev/dk1’s passphrase:
re-enter device’s passphrase:
/dev/wd0e’s passphrase:
/dev/ld0a’s passphrase:
re-enter device’s passphrase:
```

Configuring multiple disks

```
Configuring CGD devices.
/dev/dk1's passphrase:
re-enter device's passphrase:
/dev/wd0e's passphrase:
/dev/ld0a's passphrase:
re-enter device's passphrase:
/dev/dk5's passphrase:
```

Configuring multiple disks

```
Configuring CGD devices.
/dev/dk1’s passphrase:
re-enter device’s passphrase:
/dev/wd0e’s passphrase:
/dev/ld0a’s passphrase:
re-enter device’s passphrase:
/dev/dk5’s passphrase:
i’m hungry please feed me more passphrases:
```

Shared key derivation

/etc/cgd/dk1

```
algorithm adiantum;
...
keygen argon2id { iterations 32; memory 5214; ...
        shared "my laptop" \
            algorithm hkdf-hmac-sha256 \
            subkey AAAAQEGELNr3bj3I;
};
```

/etc/cgd/wd0e

```
algorithm aes-xts;
...
keygen argon2id { iterations 32; memory 5214; ...
        shared "my laptop" \
            algorithm hkdf-hmac-sha256 \
            subkey AAAAQHSC15pr1Pe4;
};
```

Configuring multiple disks from a shared key

```
Configuring CGD devices.
/dev/dk1’s passphrase:
```

Configuring multiple disks from a shared key

```
Configuring CGD devices.
/dev/dk1’s passphrase:
swapctl: setting dump device to /dev/dk12
Starting file system checks:
Loaded entropy from /var/db/entropy-file.
Setting tty flags.
...
```

Generating shared-key parameters files

Generate a parameter file for use with shared key:

```
cgdconfig -g -S -k argon2id -o /etc/cgd/dk1 \
        -V gpt adiantum
```

Generate a parameter file for another disk using same shared key:

```
cgdconfig -g -S -P /etc/cgd/dk1 -o /etc/cgd/wd0e \
        -V gpt aes-cbc 256
```

fidocrypt—‘storing’ keys with U2F/FIDO

`https://github.com/riastradh/fidocrypt`

- fidocrypt(1) tool stores a secret in a cryptfile
- Can be opened only with an enrolled U2F/FIDO device
- No cryptfile, or no enrolled U2F/FIDO device? No secret

Manage U2F/FIDO devices enrolled in a cryptfile

```
$ fidocrypt enroll -n yubi5nano /etc/cgd.crypt
```

Manage U2F/FIDO devices enrolled in a cryptfile

```
$ fidocrypt enroll -n yubi5nano /etc/cgd.crypt
tap key to enroll; waiting...
```

Manage U2F/FIDO devices enrolled in a cryptfile

```
$ fidocrypt enroll -n yubi5nano /etc/cgd.crypt
tap key to enroll; waiting...
tap key again to verify; waiting...
$
```

Manage U2F/FIDO devices enrolled in a cryptfile

```
$ fidocrypt enroll -n yubi5nano /etc/cgd.crypt
tap key to enroll; waiting...
tap key again to verify; waiting...
$ fidocrypt list /etc/cgd.crypt
```

Manage U2F/FIDO devices enrolled in a cryptfile

```
$ fidocrypt enroll -n yubi5nano /etc/cgd.crypt
tap key to enroll; waiting...
tap key again to verify; waiting...
$ fidocrypt list /etc/cgd.crypt
1 yubi5nano
```

Get secret out of cryptfile with U2F/FIDO device

```
$ fidocrypt get /etc/cgd.crypt
```

(For illustration only—don't put your secrets anywhere visible!)

Get secret out of cryptfile with U2F/FIDO device

```
$ fidocrypt get /etc/cgd.crypt
fidocrypt: specify an output format (-F)
Usage: fidocrypt get -F <format> <cryptfile>
$
```

(For illustration only—don't put your secrets anywhere visible!)

Get secret out of cryptfile with U2F/FIDO device

```
$ fidocrypt get /etc/cgd.crypt
fidocrypt: specify an output format (-F)
Usage: fidocrypt get -F <format> <cryptfile>
$ fidocrypt get -F base64 /etc/cgd.crypt
```

(For illustration only—don't put your secrets anywhere visible!)

Get secret out of cryptfile with U2F/FIDO device

```
$ fidocrypt get /etc/cgd.crypt
fidocrypt: specify an output format (-F)
Usage: fidocrypt get -F <format> <cryptfile>
$ fidocrypt get -F base64 /etc/cgd.crypt
tap key; waiting...
```

(For illustration only—don't put your secrets anywhere visible!)

Get secret out of cryptfile with U2F/FIDO device

```
$ fidocrypt get /etc/cgd.crypt
fidocrypt: specify an output format (-F)
Usage: fidocrypt get -F <format> <cryptfile>
$ fidocrypt get -F base64 /etc/cgd.crypt
tap key; waiting...
yTpyXp1Hk3F48Wx3Mp7B2gNOChPyPW0VOH3C7l5AM9A=
```

(For illustration only—don't put your secrets anywhere visible!)

Enroll another U2F/FIDO device

```
$ fidocrypt enroll -n redsolokey cgd.crypt
```

Enroll another U2F/FIDO device

```
$ fidocrypt enroll -n redsolokey cgd.crypt
tap a key that’s already enrolled; waiting...
```

Enroll another U2F/FIDO device

```
$ fidocrypt enroll -n redsolokey cgd.crypt
tap a key that’s already enrolled; waiting...
tap key to enroll; waiting...
```

Enroll another U2F/FIDO device

```
$ fidocrypt enroll -n redsolokey cgd.crypt
tap a key that’s already enrolled; waiting...
tap key to enroll; waiting...
tap key again to verify; waiting...
$
```

Enroll another U2F/FIDO device

```
$ fidocrypt enroll -n redsolokey cgd.crypt
tap a key that's already enrolled; waiting...
tap key to enroll; waiting...
tap key again to verify; waiting...
$ fidocrypt list /etc/cgd.crypt
```

Enroll another U2F/FIDO device

```
$ fidocrypt enroll -n redsolokey cgd.crypt
tap a key that’s already enrolled; waiting...
tap key to enroll; waiting...
tap key again to verify; waiting...
$ fidocrypt list /etc/cgd.crypt
2 redsolokey
```

Enroll another U2F/FIDO device

```
$ fidocrypt enroll -n redsolokey cgd.crypt
tap a key that's already enrolled; waiting...
tap key to enroll; waiting...
tap key again to verify; waiting...
$ fidocrypt list /etc/cgd.crypt
2 redsolokey
1 yubi5nano
```

Hook it up to cgd(4)

```
algorithm adiantum;
...
keygen argon2id {
        ...
};
keygen shell_cmd {
        cmd "fidocrypt get -F raw /etc/cgd.crypt";
};
```

Note: Two-factor—password and U2F/FIDO device!

TODO

- Import fidocrypt(1) into base
  - wip/fidocrypt-git in pkgsrc for now
- Integration with sysinst to configure cgd with fidocrypt
- Combine cgd(4) and login password

TODO

- Import fidocrypt(1) into base
  - wip/fidocrypt-git in pkgsrc for now
- Integration with sysinst to configure cgd with fidocrypt
- Combine cgd(4) and login password
  - . . . maybe via more general system keyring or key derivation mechanism

Disk encryption in NetBSD

Questions?

`https://www.NetBSD.org/gallery/presentations/riastradh/bsdcan2023/diskencryption.pdf`