On the design of a portable secure filesystem: the crypto bits

by Philip Homburg
<philip@f-src.phicoh.com>

Portable Filesystems

- Defined by a specification (not a single implementation)
- Not done until two independent, interoperable implementations exist

Goals

- Secure
  - Secrecy
  - Integrity
- Robust
- Simple

"A secure FAT32 replacement"

Attack Model

- Single version read-only
- Many versions read-only
- Many versions read/write

Not included:

- Side-channel attacks
- Plausible deniability

Primitives

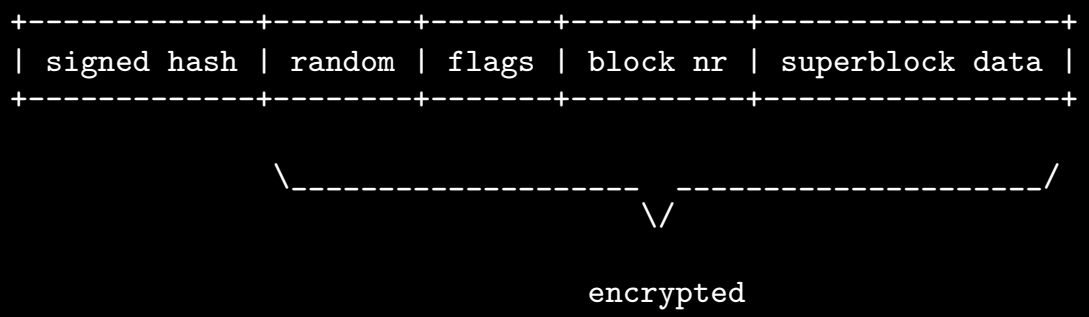
- SHA-256
  - Does not reveal input
  - Requires second pre-image attack
- HMAC-SHA-256
  - Symmetric signature scheme
  - Cannot compute signature without key
- AES-256
  Basic block cipher
- AES-CBC-256
  Chaining method to encrypt more than one AES block (128 bits)
- Argon2
  Password hashing technique

Block Pointer

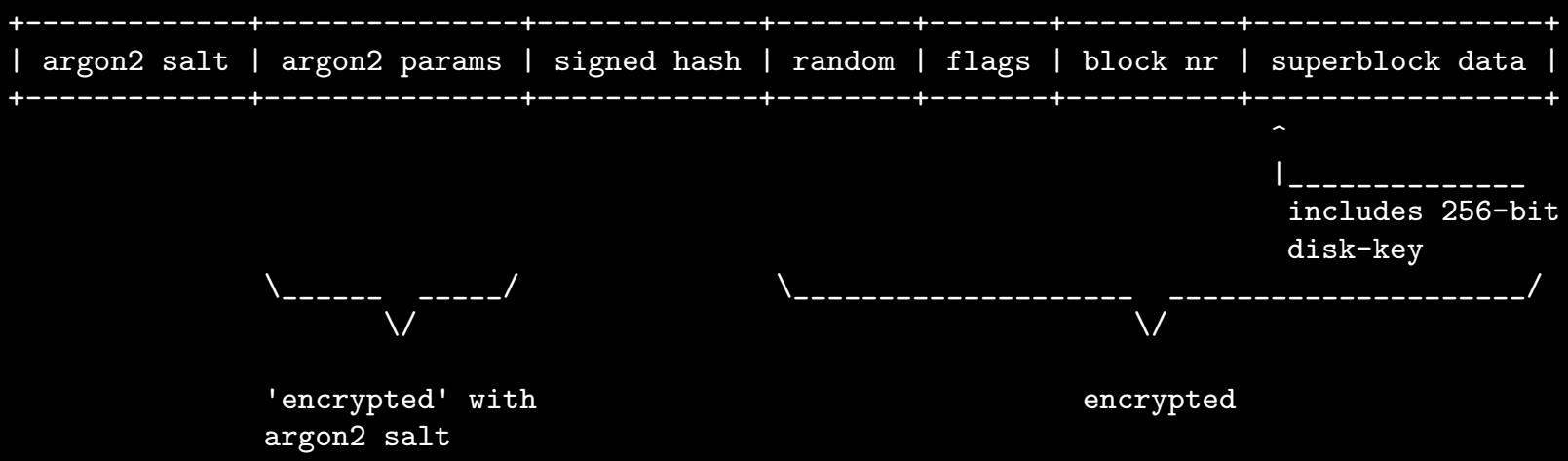
```
typedef struct sef_blkptr
{
        sef_hash_T sebp_hash;
        sef_iv_T sebp_iv;
        sef_ptr_flags_T sebp_flags;
        uint64_T sebp_block;
} sef_blkptr_T;

+------------------+-----------+-------+--------------------+
| 256-bit SHA2 hash | 128-bit IV | flags | 64-bit block number |
+------------------+-----------+-------+--------------------+
                                  ^
                                  | bit 0 of the first byte is the
                                  | big (1) / little (0) endian
                                  | flag
```

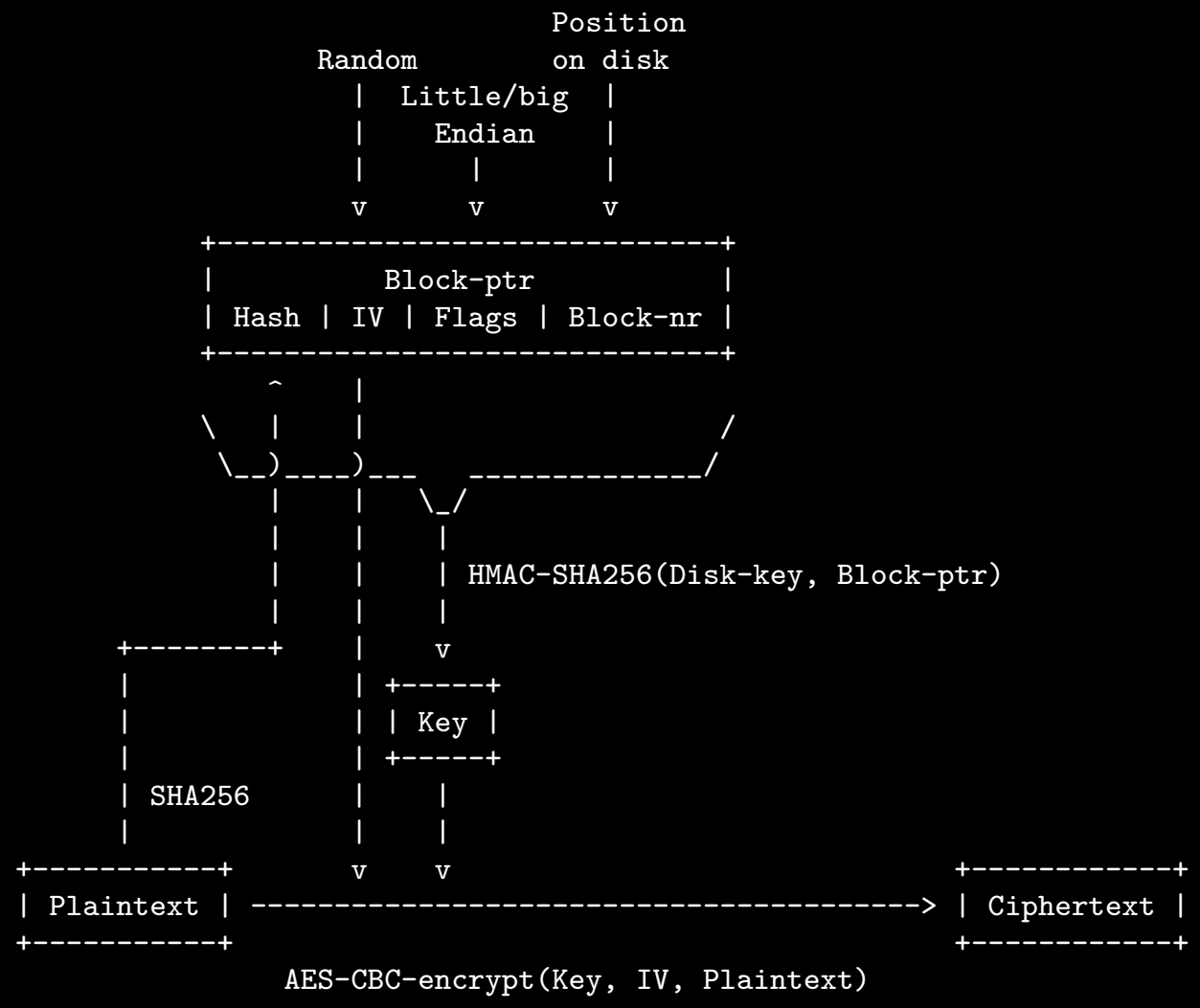
Checkpoint


signed hash
random
flags
block nr
superblock data
encrypted

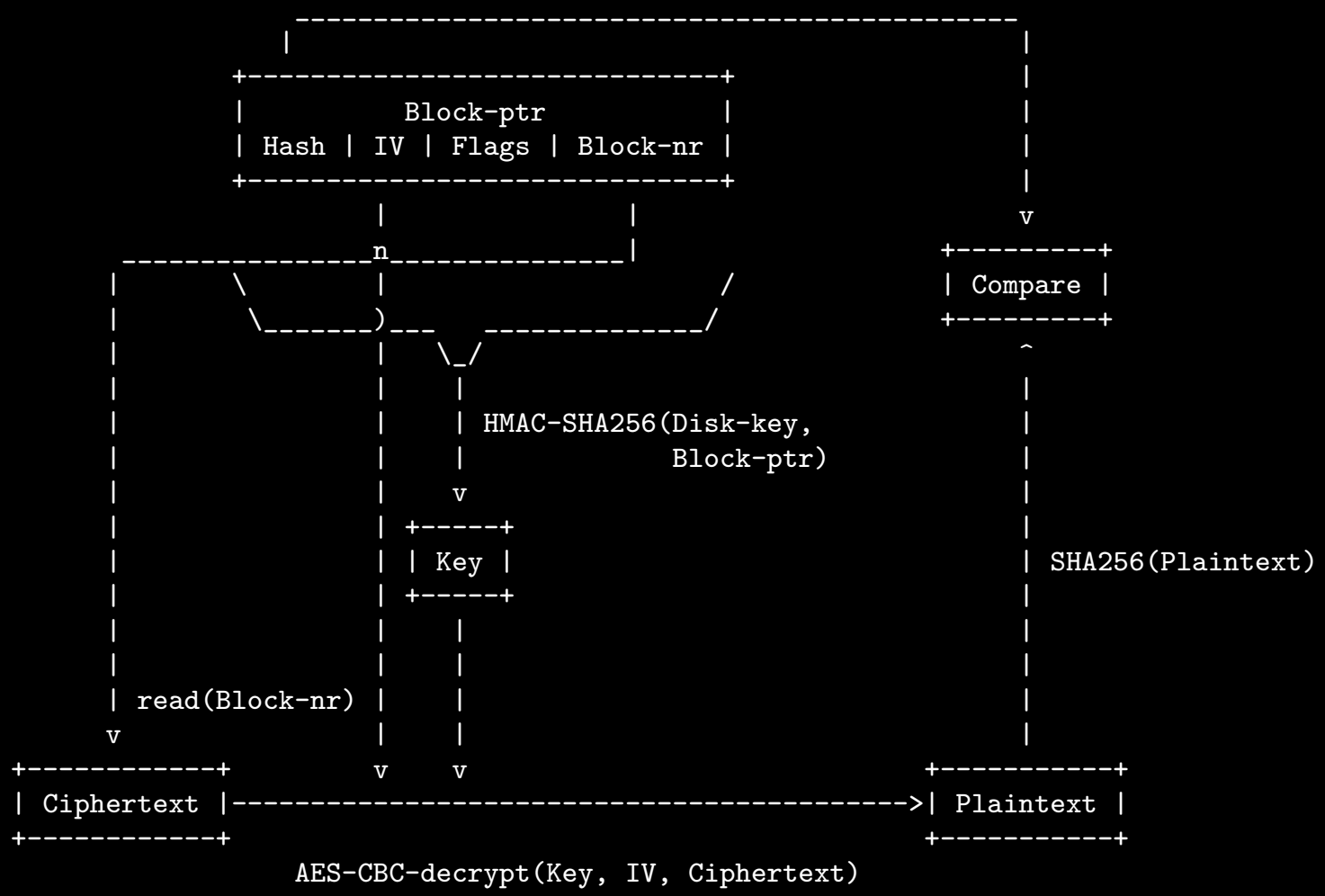
Superblock


argon2 salt
argon2 params
signed hash
random
flags
block nr
superblock data
includes 256-bit
disk-key
'encrypted' with
argon2 salt
encrypted

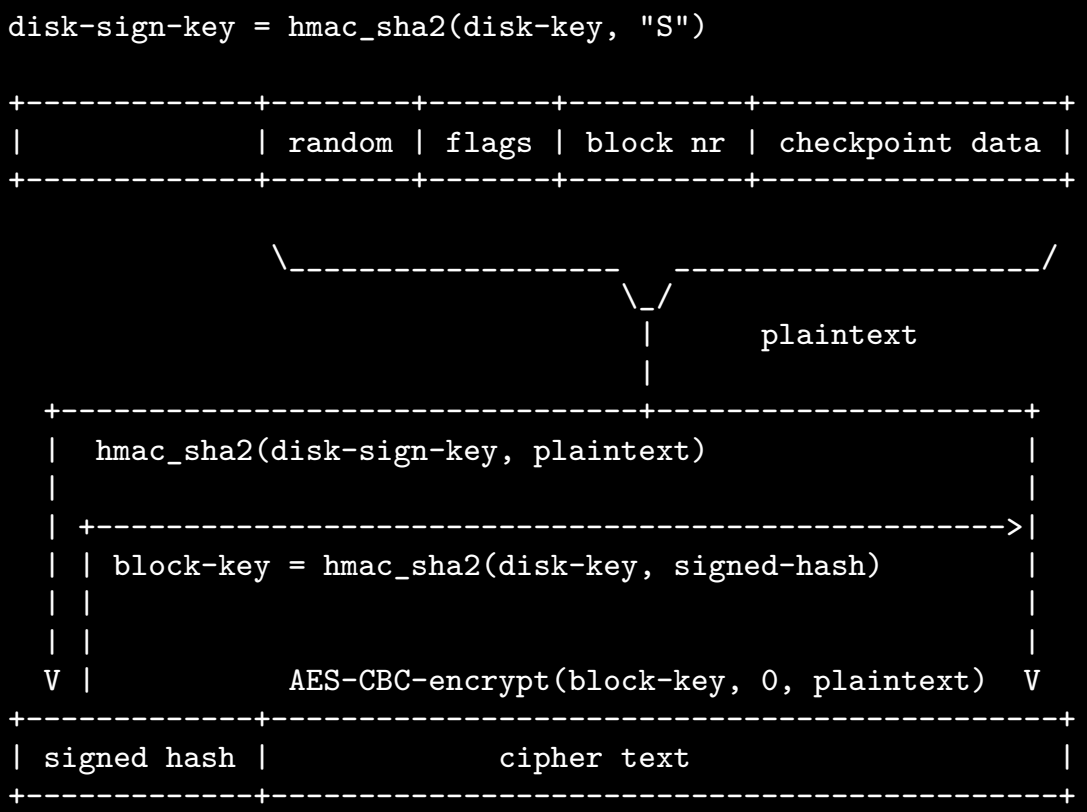
Encrypt Block



Decrypt Block



Encrypt Checkpoint



Decrypt Checkpoint

Encrypt Superblock

```
super-key = argon2(iter, mem, par, super-key-len, salt, password)
super-sign-key = hmac_sha2(super-key, "S")
```

Decrypt Superblock

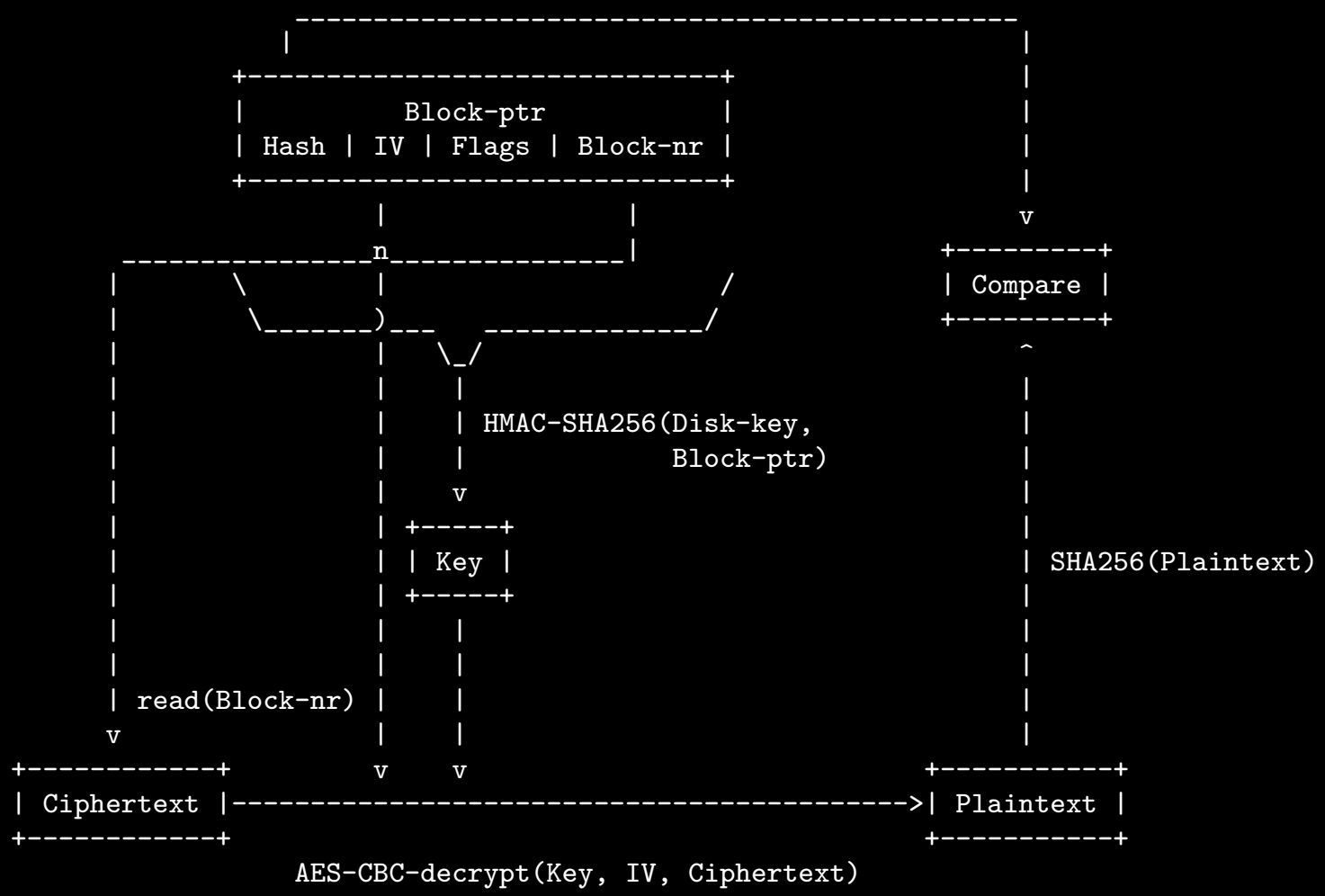
Attacks

Evaluate the three different attacks (single version read-only, many versions read-only, many versions read/write) on the three types of blocks (data block, checkpoint block, super block)

Single version read-only, data block

- Assume known plaintext attack
- brute force 256-bit AES encryption key
- reconstruct block pointer
- brute force 256-bit HMAC to find disk key

Decrypt Block



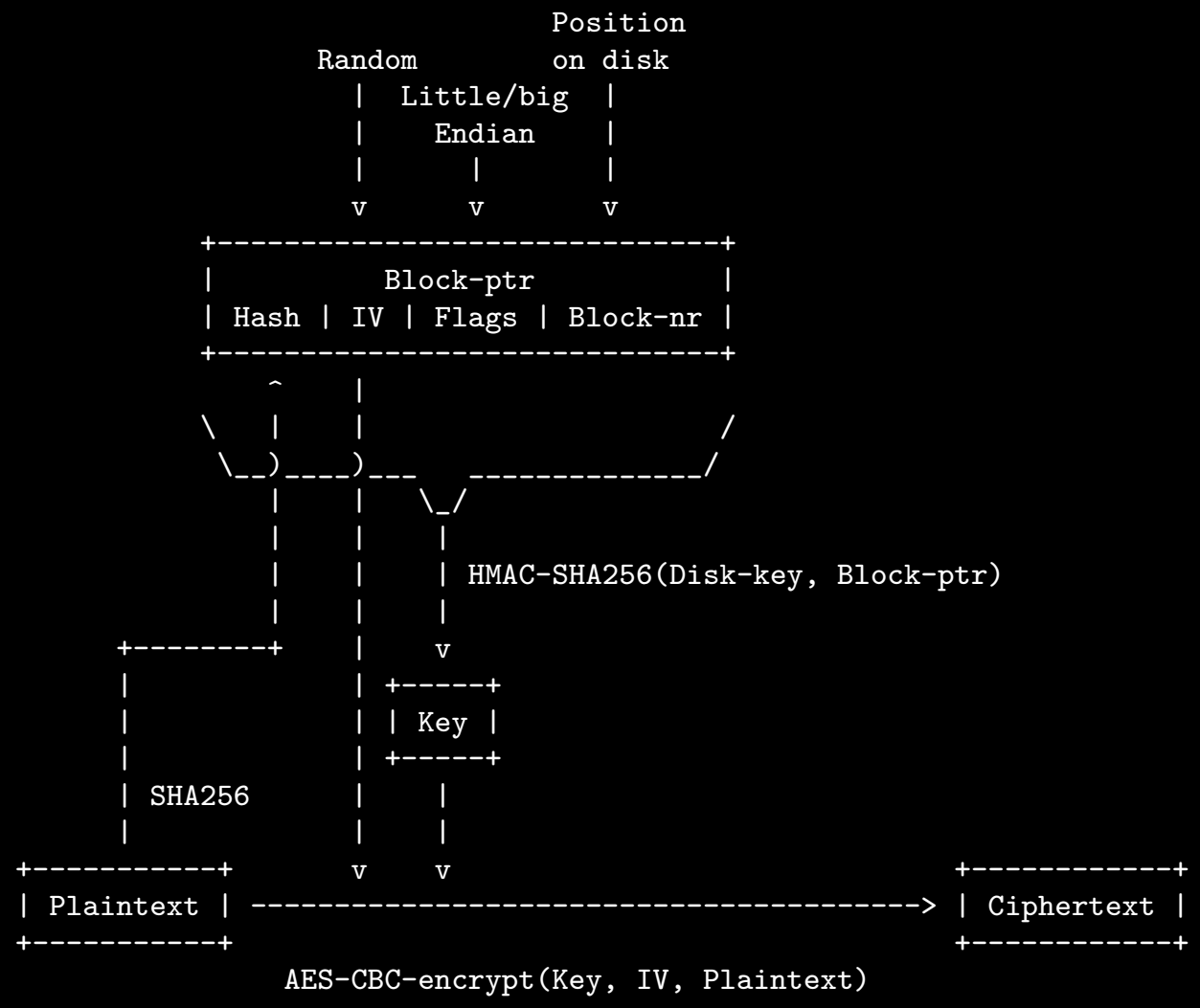
Many versions read-only, data block

For two block writes, the encryption key is different if:

- The contents of the blocks is different
- The blocks are written in different locations
- The IVs are different

Conclusion: encryption doesn't leak meta-data. Be careful about the filesystem layout.

Encrypt Block

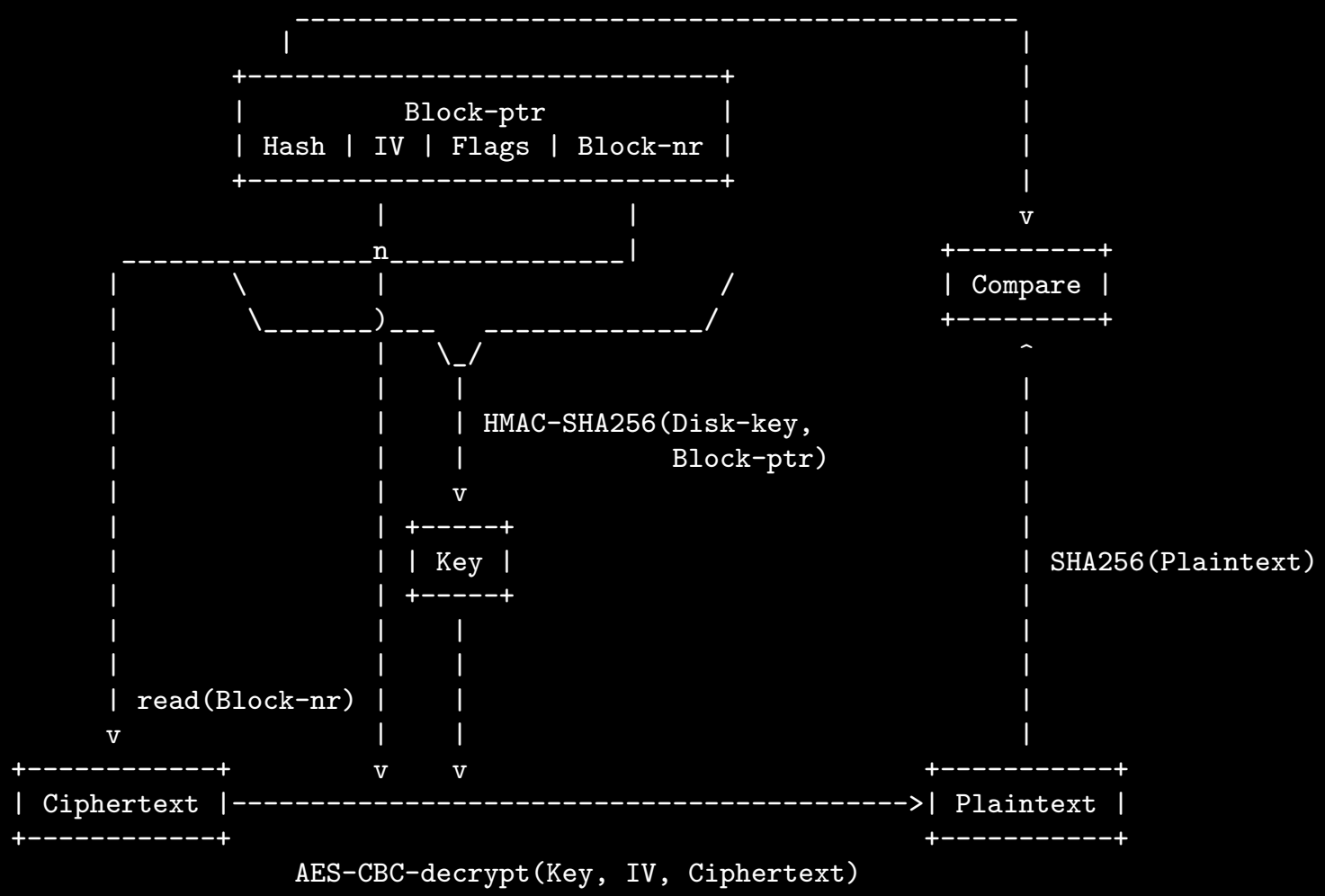


Many versions read/write, data block

Two possible attacks:

- Second pre-image attack on SHA-256. This results in the same hash.
- Modify the hash in the block pointer. This ultimately requires changing a checkpoint.
  Note that this results in a different encryption key.

Decrypt Block



Single version read-only, checkpoint

We can assume a known plaintext attack (plenty of redundancy in the checkpoint). There are two options:

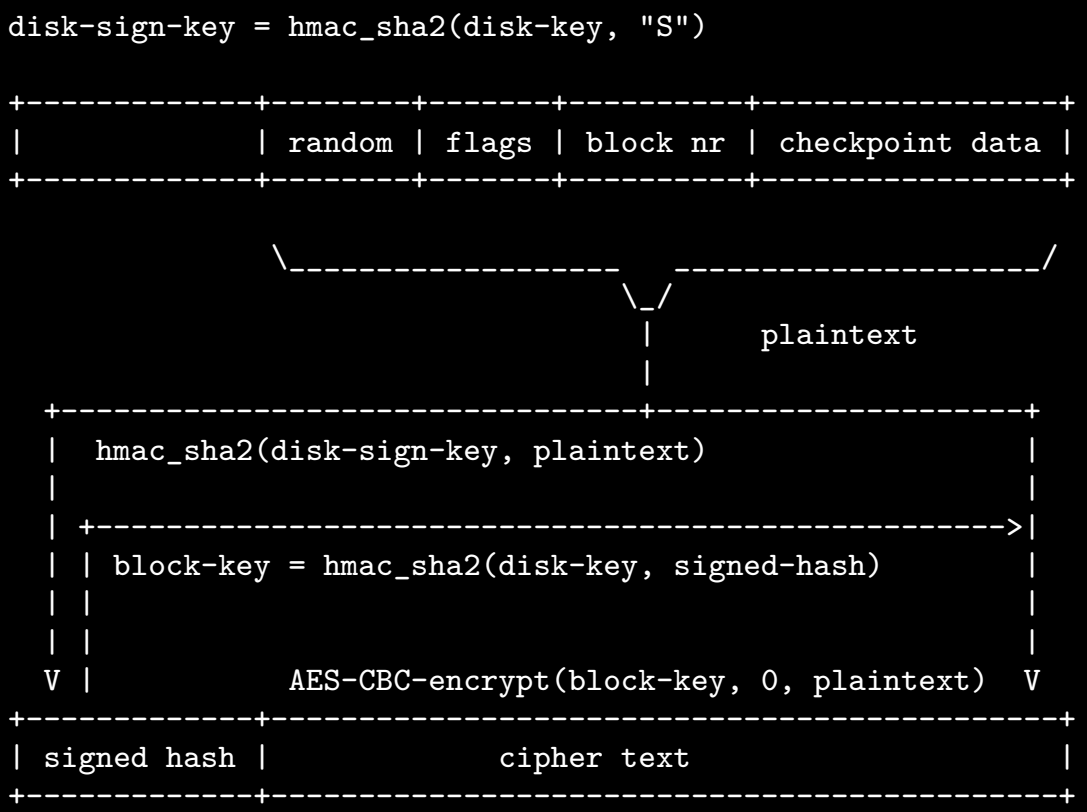
- The same strategy as for a data block (brute force AES and brute force the HMAC
- Brute force the combination of HMAC and AES. In general this is the same as the previous option. However, if the disk key is weak, it may provide an advantage.

Decrypt Checkpoint

Many versions read-only, checkpoint

There is no reason to assume that two checkpoints will ever have the same contents. Even if that would happen, the random number would cause a different encryption key.
Note that checkpoint will most likely show up in disk write patterns, so an attacker will be able to figure out which blocks are checkpoint blocks.

Encrypt Checkpoint



Many versions read/write, checkpoint

Two possible attacks:

- Second pre-image attack on SHA-256. This results in the same hash.
- Attack the signed hash. This requires brute forcing the disk signing key.
  Note that this results in a different encryption key.

Decrypt Checkpoint

Single version read-only, superblock

We can assume a known plaintext attack (plenty of redundancy in the superblock). The best option is to brute force the password:

- For each guess, hash the password with argon2 and then HMAC followed by AES.

Decrypt Superblock

Many versions read-only, superblock

This is not a realistic attack. The superblock only changes to change the password, or to resize the filesystem.

Many versions read/write, superblock

This is basically the same attack as changing a checkpoint. Note that bruteforcing the password is within all likelyhood the best option.

Robustness

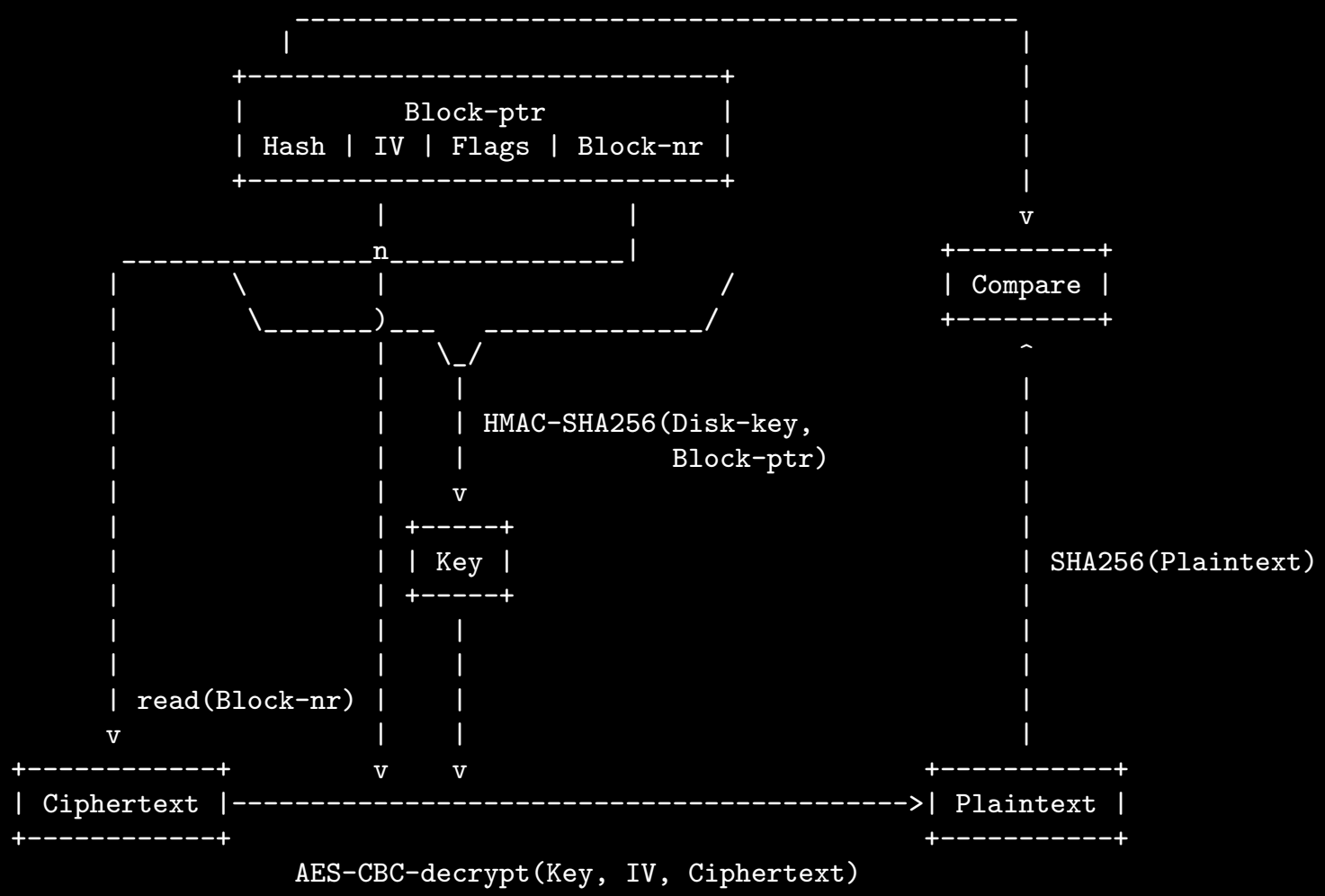
What about broken random number generators?

- Predictible but unique numbers
- Always the same number (9 “Dilbert”, 4 “xkcd”)

Single version read-only, data block

The encryption key is derived from the contents of the block and the position on disk. So even without randomness, this does not leak information.

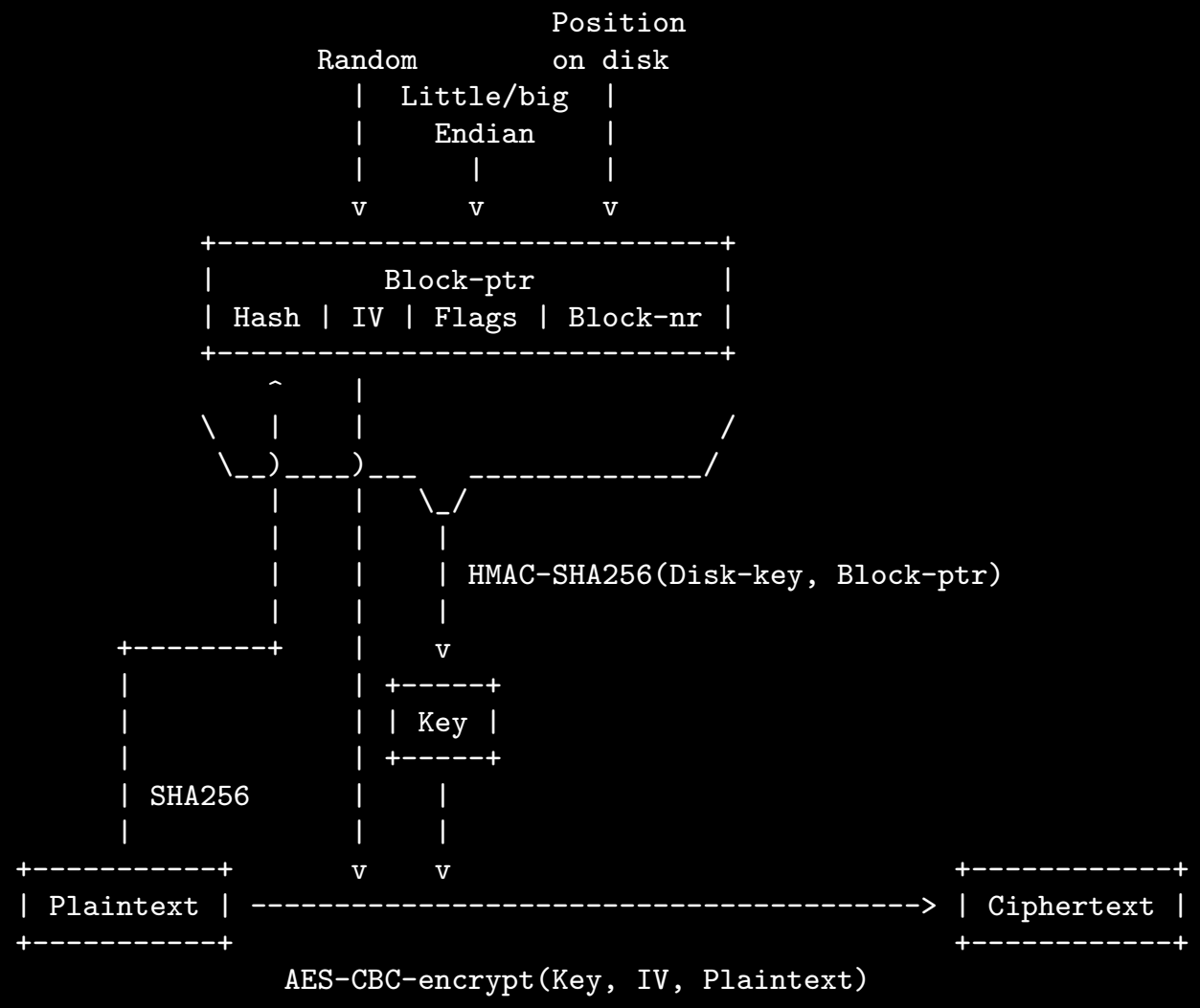
Decrypt Block



Many versions read-only, data block

- With a unique but predictible IV, all blocks will have different encryption keys. So no data is leaked.
- With a constant IV, blocks with identical content and written to the same position on disk will get the same encryption key. In this case the system will leak some meta-data.

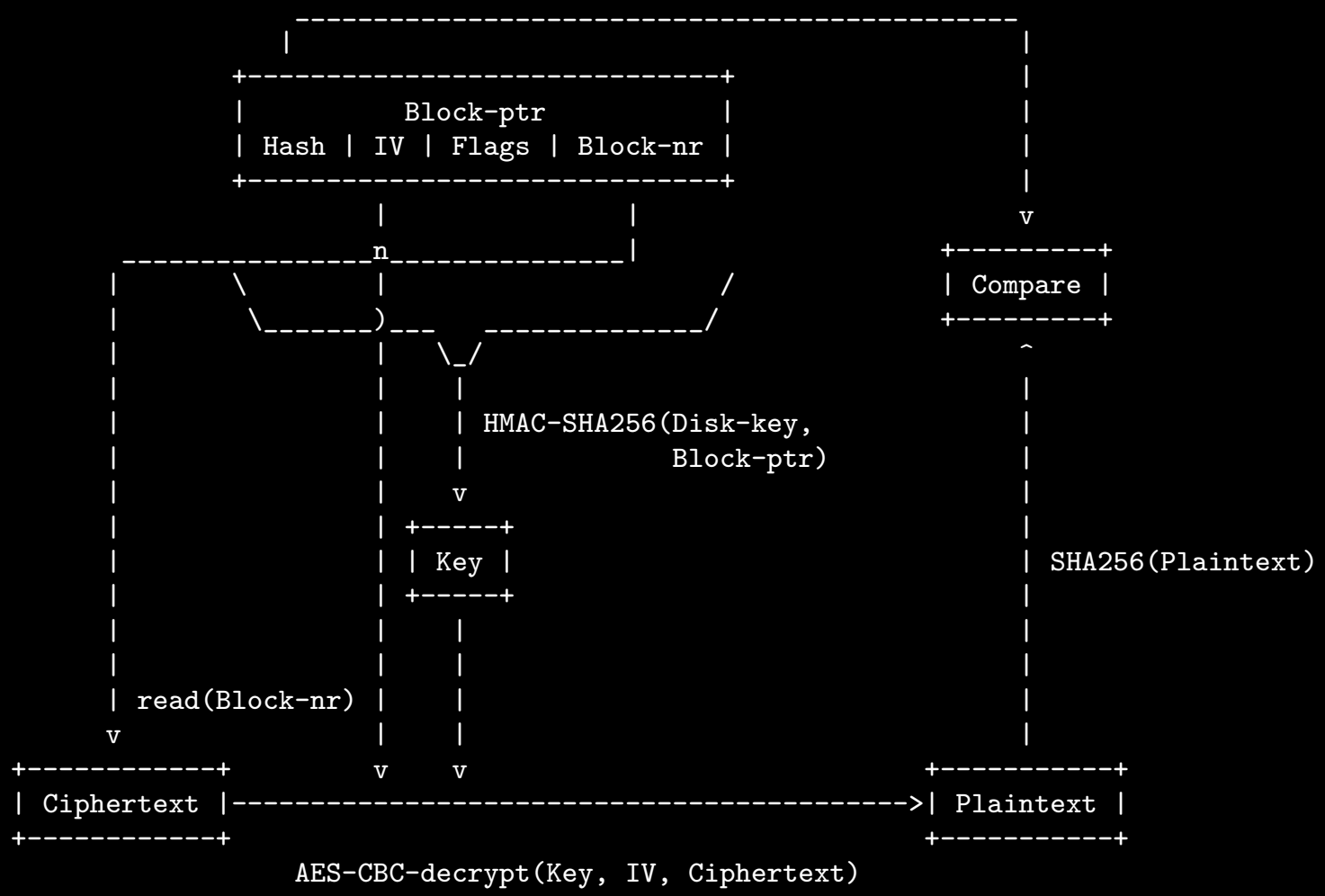
Encrypt Block



Many versions read/write, data block

Randomness has no effect on modification attacks.

Decrypt Block



Single version read-only, checkpoint

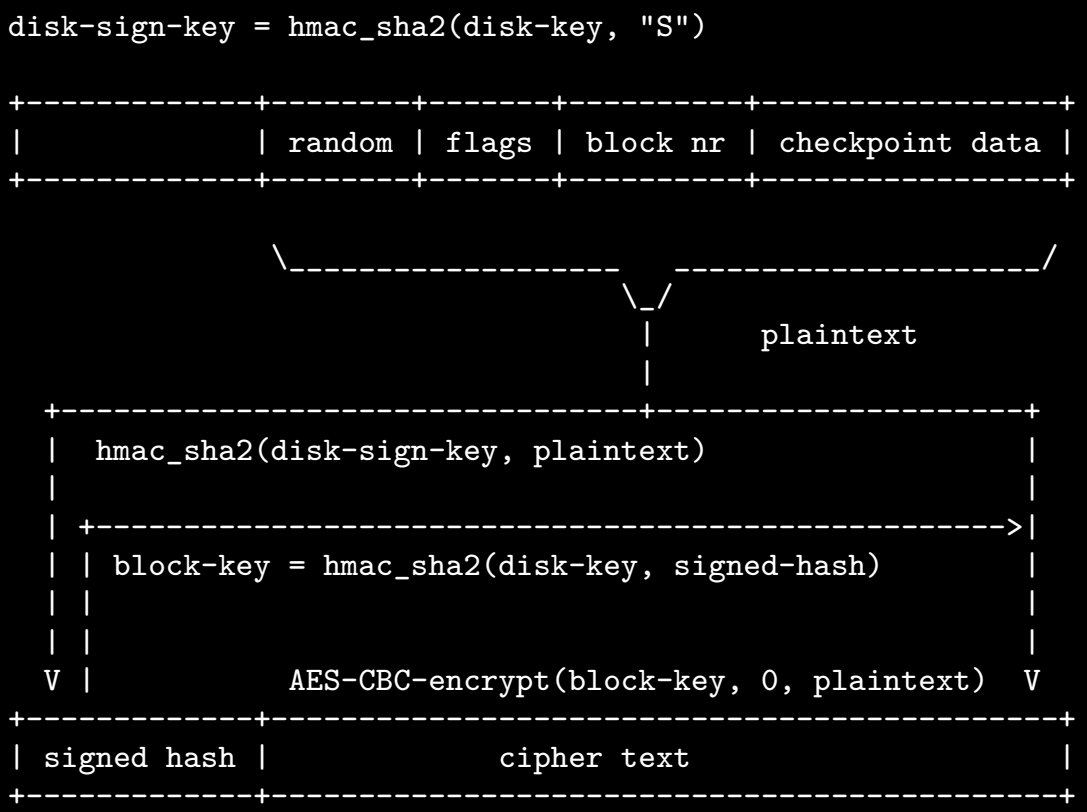
Same as for a data block, the contents of the checkpoint and the position on disk are used to compute the encryption key. No randomness is required.

Decrypt Checkpoint

Many versions read-only, checkpoint

- A unique “random” value will ensure a unique encryption key. So no information is leaked.
- In general, all checkpoints are different. A contant “random” value will result in re-use of a encryption key only if the contents of the checkpoints is also identical (as well as the position). Even if that situation occurs, it is not clear if that leaks any meta-data.

Encrypt Checkpoint



Many versions read/write, checkpoint

Modification attacks do not depend on randomness.

Decrypt Checkpoint

Single version read-only, superblock

No randomness is required to encrypt a superblock.

Decrypt Superblock

Many versions read-only, superblock

The superblock only changes to change the password, or to resize the filesystem. Lack of randomness does not leak anything which cannot be observed directly by looking at changing superblocks.

Many versions read/write, superblock

This is the same as before, modification does not depend on randomness.

Sorry, I cheated

The disk encryption key, which is stored in the superblock, is also a random numner. If this key is predictible, it is game over!
Solution: hash the random number with the super-key.

```
super-key = argon2(iter, mem, par, super-key-len, salt, password)

disk-key= hmac_sha2(super-key, random)
```

Questions?

- Questions?
- Remarks?
- Is it secure?
- Missing features?

Mail: <philip@f-src.phicoh.com>