Cryptsetup Benchmark

Christian Külker

2024-02-07

Contents

1 Introduction

Cryptsetup is a utility used in Linux systems for setting up encrypted storage using the Linux Unified Key Setup (LUKS) format. It offers robust encryption options for securing data on hard drives, partitions, or even removable media. One of the key features of Cryptsetup is its 'benchmark' command, which helps users evaluate the performance of various encryption algorithms on their hardware.

2 Installation

Before you can use the ‘cryptsetup benchmark’ command, you need to ensure that Cryptsetup is installed on your system. For Debian you can install it by following these steps:

```
aptitude install cryptsetup
```

3 Benchmark

The ‘cryptsetup benchmark’ command is a powerful tool for testing the performance of different cryptographic algorithms on your system. This is especially useful when deciding which algorithm to use for encrypting your data. Here’s how to use it as **root**:

1. Run the command inside a terminal as **root**:

```
cryptsetup benchmark
```

2. Examples:

Debian 12 Bookworm Machine 1:

```
cryptsetup benchmark

# Tests are approximate using memory only (no storage IO).
PBKDF2-sha1       858081 iterations per second for 256-bit key
PBKDF2-sha256    1138519 iterations per second for 256-bit key
PBKDF2-sha512     983654 iterations per second for 256-bit key
PBKDF2-ripemd160  560735 iterations per second for 256-bit key
PBKDF2-whirlpool  346751 iterations per second for 256-bit key
argon2i       4 iterations, 911848 memory, 4 parallel threads (CPUs) for\
 256-bit key (requested 2000 ms time)
argon2id      4 iterations, 904283 memory, 4 parallel threads (CPUs) for\
 256-bit key (requested 2000 ms time)
#     Algorithm |       Key |      Encryption |      Decryption
        aes-cbc        128b       521.9 MiB/s      1648.5 MiB/s
    serpent-cbc        128b        76.2 MiB/s       305.0 MiB/s
    twofish-cbc        128b       179.9 MiB/s       234.9 MiB/s
        aes-cbc        256b       395.8 MiB/s      1377.0 MiB/s
    serpent-cbc        256b        86.4 MiB/s       306.1 MiB/s
    twofish-cbc        256b       185.8 MiB/s       235.0 MiB/s
        aes-xts        256b      1466.4 MiB/s      1472.3 MiB/s
    serpent-xts        256b       255.3 MiB/s       277.7 MiB/s
    twofish-xts        256b       217.4 MiB/s       219.7 MiB/s
```

```
        aes-xts        512b      1215.9 MiB/s      1215.5 MiB/s
    serpent-xts        512b       271.4 MiB/s       278.7 MiB/s
    twofish-xts        512b       212.9 MiB/s       219.7 MiB/s
```

Debian 12 Bookworm machine 2 (slower)

```
cryptsetup benchmark

PBKDF2-sha1       423495 iterations per second for 256-bit key
PBKDF2-sha256     597819 iterations per second for 256-bit key
PBKDF2-sha512     473184 iterations per second for 256-bit key
PBKDF2-ripemd160  318522 iterations per second for 256-bit key
PBKDF2-whirlpool  189959 iterations per second for 256-bit key
argon2i       4 iterations, 303407 memory, 4 parallel threads (CPUs) for\
 256-bit key (requested 2000 ms time)
argon2id      4 iterations, 307680 memory, 4 parallel threads (CPUs) for\
 256-bit key (requested 2000 ms time)
#     Algorithm |       Key |      Encryption |      Decryption
        aes-cbc        128b        99.6 MiB/s       107.8 MiB/s
    serpent-cbc        128b        50.2 MiB/s        65.1 MiB/s
    twofish-cbc        128b       116.0 MiB/s       132.9 MiB/s
        aes-cbc        256b        82.0 MiB/s        85.3 MiB/s
    serpent-cbc        256b        56.6 MiB/s        65.1 MiB/s
    twofish-cbc        256b       124.5 MiB/s       132.9 MiB/s
        aes-xts        256b       110.4 MiB/s       108.8 MiB/s
    serpent-xts        256b        58.7 MiB/s        63.5 MiB/s
    twofish-xts        256b       123.3 MiB/s       129.9 MiB/s
        aes-xts        512b        87.7 MiB/s        86.2 MiB/s
    serpent-xts        512b        64.3 MiB/s        63.6 MiB/s
    twofish-xts        512b       130.8 MiB/s       129.6 MiB/s
```

Debian 11 Bullseye machine 3 (slowest - Raspberry Pi 4)

```
cryptsetup benchmark

# Tests are approximate using memory only (no storage IO).
PBKDF2-sha1       347210 iterations per second for 256-bit key
PBKDF2-sha256     579964 iterations per second for 256-bit key
PBKDF2-sha512     468951 iterations per second for 256-bit key
PBKDF2-ripemd160  293225 iterations per second for 256-bit key
PBKDF2-whirlpool  121138 iterations per second for 256-bit key
argon2i       4 iterations, 278106 memory, 4 parallel threads (CPUs) for\
 256-bit key (requested 2000 ms time)
```

```
argon2id      4 iterations, 284125 memory, 4 parallel threads (CPUs) for\
 256-bit key (requested 2000 ms time)
#     Algorithm |       Key |      Encryption |      Decryption
        aes-cbc        128b        23.2 MiB/s        79.1 MiB/s
    serpent-cbc        128b        35.9 MiB/s        38.3 MiB/s
    twofish-cbc        128b        59.0 MiB/s        61.8 MiB/s
        aes-cbc        256b        17.4 MiB/s        60.0 MiB/s
    serpent-cbc        256b        37.0 MiB/s        38.4 MiB/s
    twofish-cbc        256b        59.8 MiB/s        61.9 MiB/s
        aes-xts        256b        88.1 MiB/s        77.3 MiB/s
    serpent-xts        256b        36.3 MiB/s        38.7 MiB/s
    twofish-xts        256b        62.2 MiB/s        62.1 MiB/s
        aes-xts        512b        66.6 MiB/s        58.3 MiB/s
    serpent-xts        512b        38.0 MiB/s        38.4 MiB/s
    twofish-xts        512b        63.9 MiB/s        62.0 MiB/s
```

3. The output will display a list of algorithms along with their encryption and decryption speeds. It typically includes algorithms like AES, Serpent, Twofish, etc.
4. Look at the results to determine which algorithm provides a good balance between security and performance for your needs.

3.1 Understanding the Benchmark Output

The output of 'cryptsetup benchmark' are spitted in two sections. The first measure hashing in 'iterations per second' (the bigger the number the better) and the second measure different algorithms and includes several columns:

- Algorithm: The encryption algorithm (e.g., aes, serpent).
- Key: The size of the key used by the algorithm.
- Encryption Speed: The speed at which data can be encrypted.
- Decryption Speed: The speed at which data can be decrypted.

Higher speeds indicate better performance. However, remember that the most performant algorithm may not always be the most secure, so balance is key if performance is an issue.

3.2 Tips for Using Cryptsetup

- Choose the Right Algorithm: Use the benchmark results to choose an encryption algorithm that suits your balance of security and performance.
- Backup Keys: Always backup your encryption keys in a secure location.

- Regular Updates: Keep Cryptsetup and your system updated for the latest security patches. Make backups of all you data at least before a major cryptsetup update.

3.3 Conclusion

The ‘cryptsetup benchmark’ command in Debian 12 Bookworm is an invaluable tool for anyone looking to secure their data with encryption. By understanding and utilizing this command, you can make informed decisions about the encryption algorithms best suited for your storage hardware and security requirements. Remember, while performance is important, it should not compromise the security of your encrypted data.

4 Understand Existing Setups

1. Chose the correct partition

```
lsblk
NAME                MAJ:MIN RM   SIZE RO TYPE  MOUNTPOINTS
sda                   8:0    0 119.2G  0 disk
├─sda1                8:1    0   487M  0 part  /boot
├─sda2                8:2    0     1K  0 part
└─sda5                8:5    0 118.8G  0 part
  └─sda5_crypt      254:0    0 118.7G  0 crypt
    ├─z2--vg-root   254:1    0 117.8G  0 lvm   /
    └─z2--vg-swap_1 254:2    0   976M  0 lvm   [SWAP]
```

2. In the above setup `/dev/sda5` is the correct one. Then dump the existing values. In this output some keys are removed. So your output should show keys like ‘Salt’, ‘Digest’, …

```
cryptsetup luksDump /dev/sda5

LUKS header information
Version:               2
Epoch:                 3
Label:                 (no label)
Subsystem:             (no subsystem)
Flags:               (no flags)

Data segments:
  0: crypt
        cipher: aes-xts-plain64
```

```
Keyslots:
  0: luks2
        Key:        512 bits
        Priority:   normal
        Cipher:     aes-xts-plain64
        Cipher key: 512 bits
        PBKDF:      argon2id
        AF hash:    sha256
Tokens:
Digests:
  0: pbkdf2
        Hash:       sha256
```

3. The output will display detailed information about the LUKS container, including the version of LUKS, the used key size, the cipher (encryption algorithm), and the hash used.

- Look for the Cipher name to identify the encryption algorithm (e.g., aes).
- The Hash spec field shows the hashing algorithm used (e.g., sha256).

5 Missing

Cryptsetup is a versatile tool with a range of functionalities beyond just checking encryption algorithms and hashing. Here are some additional aspects and capabilities of Cryptsetup that are worth noting and that are not covered in this document:

- **Key Management:** LUKS supports multiple key slots, allowing multiple passphrases to unlock the same volume. This feature is useful for both personal and shared environments, where different users can have their own passphrase.
- **Encryption of Swap and Temporary Files:** Cryptsetup can be used to encrypt swap partitions and other temporary file storage, which is crucial for maintaining security, especially in systems that handle sensitive data.
- **Integration with System Boot Process:** Cryptsetup can integrate with the system's boot process for full disk encryption, including encrypting the root partition. This can be configured to require a passphrase at boot time, enhancing security. Tools like `dropbear` can be used to do this over SSH.
- **Header Backup and Restoration:** It's possible to backup and restore the LUKS header using Cryptsetup. This is a critical step in data recovery scenarios, as damage to the header can render the encrypted data inaccessible.
- **Compatibility with Other Tools:** Cryptsetup is compatible with other Linux tools and utilities, such as Logical Volume Manager (LVM), making it suitable for complex

storage setups.

- **Support for Different Cryptographic Backends:** Cryptsetup can use different cryptographic backends like OpenSSL, offering flexibility in cryptographic implementations. However this needs to be done at compile time. For OpenSSL it is done with the configure option `--with-crypto_backend=openssl`.

6 Packages

Debian	#	cryptsetup
Bookworm	12	2:2.6.1-4~deb12u1
Bullseye	11	2:2.3.7-1+deb11u1

7 History

Version	Date	Notes
0.1.0	2024-02-07	Initial release

8 Disclaimer of Warranty

9 Limitation of Liability