Homework 04: Guidelines for the use of Age

Nicolas Leone
Student ID: 1986354
Cybersecurity

November 16, 2025

Contents

1 Introduction to Age

Age (Another Git Encryptor) is a modern, simple, and fast tool for encrypting and decrypting files from the terminal. Designed as a replacement for GPG for file encryption tasks, Age focuses on simplicity and ease of use while maintaining strong security guarantees. It can be quickly installed on all UNIX-based platforms such as macOS and Linux, and can also be used on Windows systems.

The philosophy behind Age is to provide a minimalist, secure-by-default encryption tool that avoids the complexity and pitfalls that have plagued traditional encryption software. Unlike GPG, which offers numerous configuration options and supports various cryptographic algorithms, Age deliberately restricts its functionality to a carefully selected set of modern, peer-reviewed algorithms. This design choice reduces the attack surface and eliminates the risk of users inadvertently choosing weak or deprecated cryptographic primitives.

The primary use of Age consists of encrypting and decrypting files through two main modes:

- **Symmetric encryption**: Using a passphrase to protect files. This mode is ideal for personal use cases where you encrypt files for yourself or when you can securely share a passphrase with the intended recipient through an out-of-band channel.
- **Public-key encryption**: Using recipient public keys for secure file sharing. This mode enables you to encrypt files for multiple recipients without requiring a shared secret, making it particularly useful in collaborative environments.

Age allows the encryption of input files or text obtained via standard input (keyboard) in either symmetric or asymmetric mode, returning the encrypted data via standard output or, if requested, by creating a file. This flexibility makes Age particularly well-suited for integration into automated workflows, shell scripts, and CI/CD pipelines.

Age can be easily installed on different platforms:

macOS (using Homebrew):

```
brew install age
```

Linux (Ubuntu/Debian):

```
sudo apt install age
```

Windows (using Scoop):

```
scoop install age
```

Windows (using Chocolatey):

```
choco install age.portable
```

2 Cryptographic Algorithms in Age

2.1 Symmetric Encryption: ChaCha20-Poly1305

For symmetric encryption, Age uses the ChaCha20 and Poly1305 algorithms, which together form an AEAD (Authenticated Encryption with Associated Data) scheme providing both confidentiality and authentication.

2.1.1 ChaCha20

ChaCha20 is a stream cipher that uses 256-bit keys for encryption. Its operation is based on producing a continuous keystream of pseudo-random bits, which are XORed with the plaintext to produce the ciphertext.

The keystream generation requires:

- A 96-bit nonce (unique per encryption)
- A 32-bit block counter
- A 256-bit key derived from the user's passphrase using HKDF (HMAC-Based Key Derivation Function)

ChaCha20 executes 20 rounds of transformations on a 4×4 state matrix. The basic operation is the *quarter round*, which performs additions, XORs, and rotations on four 32-bit words:

1. $a \mathrel{+}= b$; $d \mathrel{\oplus}= a$; $d \mathrel{\ll}= 16$;
2. $c \mathrel{+}= d$; $b \mathrel{\oplus}= c$; $b \mathrel{\ll}= 12$;
3. $a \mathrel{+}= b$; $d \mathrel{\oplus}= a$; $d \mathrel{\ll}= 8$;
4. $c \mathrel{+}= d$; $b \mathrel{\oplus}= c$; $b \mathrel{\ll}= 7$;

where "+" denotes addition modulo 2^{32}, "$\oplus$" denotes XOR, and "$\ll n$" denotes left rotation by n bits.

The state is initialized with constants, the key, the counter, and the nonce. After 20 rounds (alternating between column and diagonal operations), the initial state is added back to produce the final keystream block.

2.1.2 Poly1305 Authentication

Poly1305 is a message authentication code (MAC) that produces a 128-bit authentication tag from the message and a 256-bit key. The key is divided into two parts: r (used for polynomial evaluation) and s (used for final masking).

The algorithm operates as follows:

1. Clamp r by clearing specific bits to ensure security
2. Initialize accumulator $\text{Acc} = 0$ and set prime $P = 2^{130} - 5$
3. Process message in 16-byte blocks: $\text{Acc} = ((\text{Acc} + \text{block}) \cdot r) \bmod P$
4. Add s to the accumulator and output the 128 least significant bits as the tag

The combination of ChaCha20 for encryption and Poly1305 for authentication provides a fast, secure AEAD construction that is resistant to both passive and active attacks.

2.2 Public-Key Encryption: X25519

For public-key encryption, Age uses X25519 for key exchange. X25519 is based on elliptic curve Diffie-Hellman (ECDH) operations on Curve25519, defined by:

$$y^2 = x^3 + 486662x^2 + x$$

2.2.1 Key Exchange Mechanism

X25519 leverages the commutative property of scalar multiplication on elliptic curves:

$$k_b \cdot (k_a \cdot P) = k_a \cdot (k_b \cdot P)$$

where k_a and k_b are 256-bit secret keys, and P is a base point on the curve (conventionally $x = 9$).

The key exchange process works as follows:

1. Each party generates a random 256-bit secret key
2. Each computes their public key by scalar multiplication with the base point
3. Public keys are exchanged
4. Each party computes the shared secret using their secret key and the other party's public key

In Age's implementation, a symmetric session key is generated using a CSPRNG, then this session key is encrypted using the shared secret derived from X25519. This approach allows multiple recipients: the session key is encrypted separately for each recipient's public key, while the actual file content is encrypted only once using the session key with ChaCha20-Poly1305.

Age refers to public keys as **RECIPIENT** and private keys as **IDENTITY**, making the command-line interface intuitive and self-documenting.

3 Usage Guidelines for Engineering Students

3.1 Symmetric Encryption with Passphrase

Symmetric encryption is useful when you want to encrypt files for your own use or when you can securely share a passphrase with the intended recipient.

3.1.1 Encrypting a File

To encrypt a file using a passphrase:

```
# Encrypt a file with a passphrase
age --passphrase --output secure_document.txt.age document.txt

# Age will prompt you to enter and confirm the passphrase
```

Listing 1: Symmetric encryption with Age

Annotated example:

- `-passphrase`: Enables symmetric encryption mode
- `-output secure_document.txt.age`: Specifies the output encrypted file
- `document.txt`: The input file to encrypt
- Age will interactively ask for a passphrase (not shown in command history for security)

3.1.2 Decrypting a File

To decrypt a file encrypted with a passphrase:

```
# Decrypt a file
age --decrypt --output document.txt secure_document.txt.age

# Age will prompt for the passphrase
```

Listing 2: Symmetric decryption with Age

Note: Age will automatically detect that the file was encrypted with a passphrase and prompt you accordingly.

3.1.3 Using Standard Input/Output

Age can work with pipes and standard streams:

```
# Encrypt text from standard input
echo "Secret message" | age --passphrase > message.age

# Decrypt to standard output
age --decrypt message.age

# Encrypt multiple files into a tar archive
tar czf - ~/Documents | age --passphrase --output backup.tar.gz.age
```

Listing 3: Encryption using pipes

3.2 Public-Key Encryption

Public-key encryption is ideal for sharing encrypted files with others without needing to exchange a passphrase. Each user has a key pair (private and public), and files are encrypted with the recipient's public key.

3.2.1 Generating a Key Pair

First, generate your own key pair:

```
# Generate a new key pair
age-keygen -o ~/.age/key.txt

# The output will show:
# Public key:
   age1qlrfls9rlkfe4kmgxvph5n2mpdz9pj9k6cr5ewqlc62gnv96jkvs4syy
# (This is an example, your key will be different)
```

Listing 4: Key generation

The generated file contains both the private key (starting with `AGE-SECRET-KEY-`) and the public key (starting with `age1`).

Example key file content:

```
# created: 2024-11-17T12:00:00Z
# public key:
   age1qlrfls9rlkfe4kmgxvph5n2mpdz9pj9k6cr5ewqlc62gnv96jkvs4syy
AGE-SECRET-KEY-1
   PXEWMFV4RG98ZQ2JY4LF98VCZGELM3GQL76CT45MST4F5X6CR4Q4M2F7JU
```

Important security practices:

- Keep the private key file (`key.txt`) secure and confidential
- Set appropriate file permissions: `chmod 600 /.age/key.txt`
- Share only the public key (the line starting with `age1`) with others
- Consider backing up the private key in a secure location

3.2.2 Encrypting for a Recipient

To encrypt a file for someone else, you need their public key:

```
# Encrypt a file for a single recipient
age --recipient
   age1qlrfls9rlkfe4kmgxvph5n2mpdz9pj9k6cr5ewqlc62gnv96jkvs4syy \
    --output project_report.pdf.age project_report.pdf

# Encrypt for multiple recipients
age --recipient
   age1qlrfls9rlkfe4kmgxvph5n2mpdz9pj9k6cr5ewqlc62gnv96jkvs4syy \
    --recipient
       age1t90nl05gngh942zdkwwk3wrkgz87k2fctk5qcyxhka7ycucza3uqg39xrr
        \
```

```
    --output shared_file.txt.age shared_file.txt
```

Listing 5: Public-key encryption

Annotated example:

- `-recipient`: Specifies a recipient's public key (can be used multiple times)
- When encrypting for multiple recipients, each can decrypt the file with their own private key
- The file size increases minimally with additional recipients

3.2.3 Decrypting a File

To decrypt a file sent to you:

```
# Decrypt using your identity file
age --decrypt --identity ~/.age/key.txt \
    --output project_report.pdf project_report.pdf.age

# Decrypt from standard input
cat encrypted_file.age | age --decrypt --identity ~/.age/key.txt >
   decrypted.txt
```

Listing 6: Public-key decryption

3.3 Advanced Use Cases

3.3.1 Storing Recipients in a File

For frequently used recipients, create a recipients file:

```
# Create a recipients file
cat > team_recipients.txt << EOF
# Engineering Team Public Keys
age1qlrfls9rlkfe4kmgxvph5n2mpdz9pj9k6cr5ewqlc62gnv96jkvs4syy  #
   Alice
age1t90nl05gngh942zdkwwk3wrkgz87k2fctk5qcyxhka7ycucza3uqg39xrr  #
   Bob
age1w5t2n7jqd47y5dmwn6mcgkhv6ck6x9v8yutqvv5zqm5wj7fqt79q8cq8rx  #
   Charlie
EOF

# Encrypt for all recipients in the file
age --recipients-file team_recipients.txt \
    --output confidential.txt.age confidential.txt
```

Listing 7: Using a recipients file

3.3.2 Encrypting Directories

Age works on files, but can be combined with archiving tools:

```
# Encrypt a directory using tar
tar czf - ~/my_project | age --recipient age1qlr... \
    --output my_project_backup.tar.gz.age

# Decrypt and extract
age --decrypt --identity ~/.age/key.txt my_project_backup.tar.gz.
   age \
    | tar xzf -
```

Listing 8: Encrypting directories

3.3.3 Encrypting to SSH Keys

Age can use existing SSH keys for encryption:

```
# Encrypt using an SSH public key
age --recipient "$(ssh-keygen -Y find-principals -s allowed_signers
    \
    -f ~/.ssh/id_ed25519.pub)" --output file.age file.txt

# Or use the ssh-rsa format directly
age --recipient "$(cat ~/.ssh/id_rsa.pub)" --output file.age file.
   txt

# Decrypt using SSH private key
age --decrypt --identity ~/.ssh/id_ed25519 file.age > file.txt
```

Listing 9: Using SSH keys

3.4 Integration with Version Control

Age can be integrated with Git for encrypting sensitive files in repositories:

```
# Encrypt secrets before committing
age --recipient age1qlr... --output secrets.env.age secrets.env
git add secrets.env.age
git commit -m "Add encrypted secrets"

# Add to .gitignore to avoid committing plaintext
echo "secrets.env" >> .gitignore
```

Listing 10: Git integration example

4 Best Practices and Security Considerations

4.1 Passphrase Security

When using symmetric encryption:

- Use strong, unique passphrases (at least 20 characters)
- Consider using a passphrase manager or diceware method
- Never reuse passphrases across different files containing sensitive data
- Age uses scrypt for key derivation, which is resistant to brute-force attacks

4.2 Key Management

For public-key encryption:

- Store private keys securely with appropriate file permissions (`chmod 600`)
- Consider hardware security tokens (e.g., YubiKey) for high-value keys
- Maintain secure backups of private keys
- Regularly rotate keys for long-term use cases
- Verify public key fingerprints through a separate secure channel

4.3 File Handling

- Securely delete plaintext files after encryption using tools like `shred` or `srm`
- Be aware that Age does not compress files; compress before encrypting if needed
- Use `.age` extension for encrypted files to indicate their status
- Consider the metadata implications: file sizes and timestamps may leak information

4.4 Operational Security

- Age does not hide metadata (file size, modification time)
- Use encrypted containers or full-disk encryption for additional protection
- When using pipes, be mindful of shell history recording sensitive commands
- Verify the integrity of Age binaries through checksums or signatures
- Keep Age updated to benefit from security patches

5 Conclusions

Age provides a modern, streamlined approach to file encryption that is particularly well-suited for engineering students and professionals who need secure file handling without the complexity of traditional tools like GPG.

5.1 Key Takeaways

- Age offers two encryption modes: symmetric (passphrase-based) and asymmetric (public-key)
- The cryptographic foundation is solid: ChaCha20-Poly1305 for symmetric encryption and X25519 for key exchange
- The tool emphasizes simplicity and ease of use without sacrificing security
- Integration with existing workflows (pipes, SSH keys, version control) is straightforward
- The plugin system allows for extended functionality like hardware token support

5.2 When to Use Age

Age is ideal for:

- Encrypting files for personal use or sharing with colleagues
- Creating encrypted backups
- Protecting sensitive configuration files in version control
- Quick, command-line based encryption tasks
- Situations where GPG's complexity is unnecessary

Age may not be suitable for scenarios requiring digital signatures, key revocation infrastructure, or compatibility with existing GPG-based workflows.

5.3 Further Resources

For more detailed information:

- Official documentation: `https://age-encryption.org`
- Man pages: `man age`, `man age-keygen`
- RFC 7539: ChaCha20 and Poly1305 specification
- Age specification: `https://github.com/C2SP/C2SP/blob/main/age.md`

References

1. Filippo Valsorda. *Age - A simple, modern and secure file encryption tool.* `https://github.com/C2SP/C2SP/blob/main/age.md`

2. Filippo Valsorda. *Age Manual Pages.* `https://github.com/FiloSottile/age/blob/main/doc/age.1.html`

3. IBM. *HMAC-Based Extract-then-Expand Key Derivation Function (HKDF).* `https://www.ibm.com/docs/en/semeru-runtime-ce-z/11`

4. RFC 7539. *ChaCha20 and Poly1305 for IETF Protocols.* `https://datatracker.ietf.org/doc/html/rfc7539`

5. xarg. *X25519 Key Exchange.* `https://x25519.xargs.org`