Using secrets safely with agents

Key takeaways

1. Manage a service token as a lifecycle: issue a narrow credential, keep only authenticated ciphertext in Git, expose plaintext to one consumer, and revoke the credential when exposure is possible.

2. SOPS encrypts secret values and authenticates the document; age protects the data key for named recipients. The private age identity belongs outside the repository.

3. Command-scoped environments reduce persistent plaintext and accidental inheritance. They do not constrain the consumer or erase logs and copies it creates.

Common misunderstandings of using encrypted credentials

Which claim would you defend right now? Choose before the evidence.

“Deleting a temporary .env file removes every plaintext copy.”

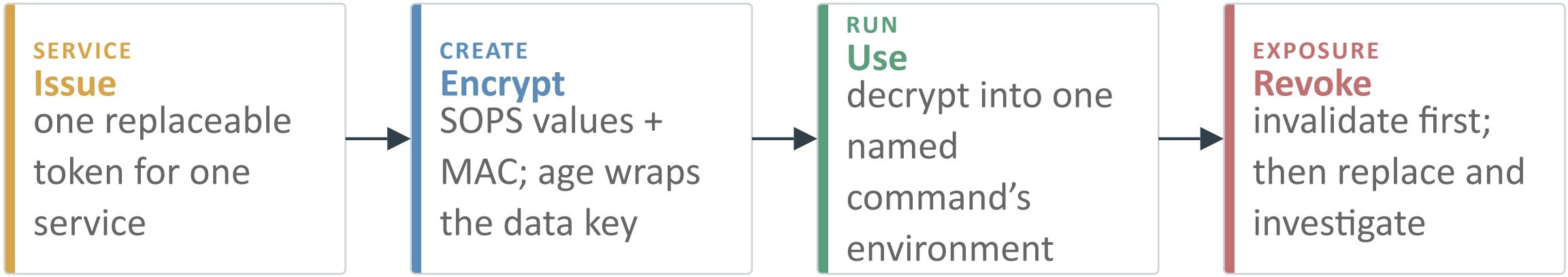
“A wrapper policy can stop a process that already holds the age identity.”

“One encrypted team token is least privilege because its storage is protected.”

Keep your choice. Look for the mechanism or counterexample that would make it fail.

Treat a credential as a lifecycle

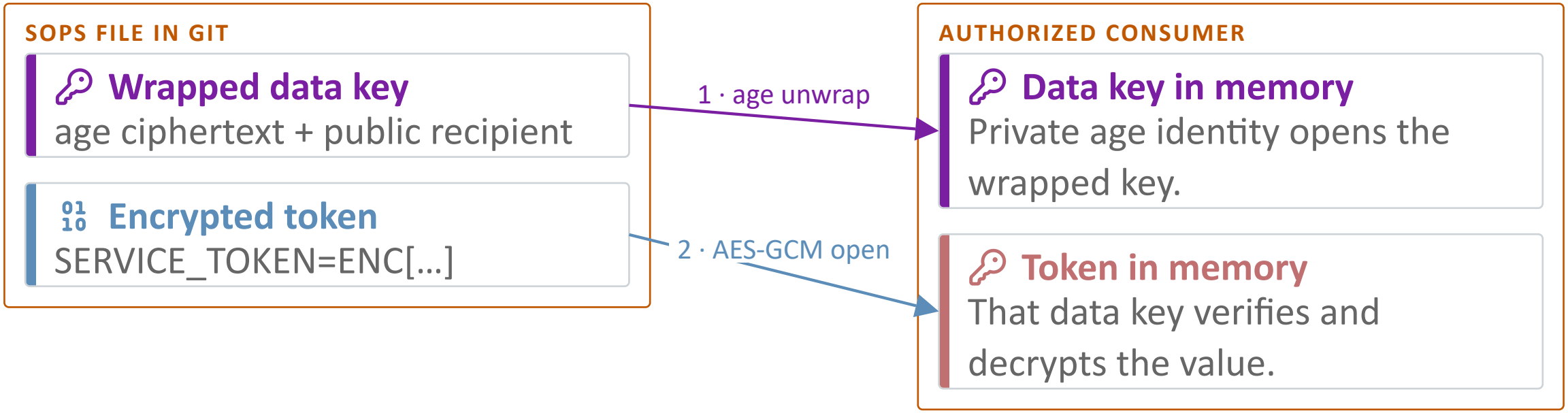
Storage is one beat; issue, use, exposure, and revocation decide the risk



A replaceable, narrowly scoped token makes both normal use and incident response smaller.

SOPS and age form an encryption envelope

The repository holds a protected data key; a private identity elsewhere unlocks it



Creation runs the other way: a fresh data key encrypts the token; the public age recipient wraps that key.

The private age identity stays outside Git. It unlocks the data key, which unlocks the token.

source: SOPS documentation: encryption model and age recipients

Ciphertext should still look recognizable

Names stay reviewable; values, wrapped key material, and the MAC do not reveal the token

```
SERVICE_TOKEN=ENC[AES256_GCM,data:V1p...,iv:Q7m...,tag:N4k...,type:str]
sops_age__list_0__map_recipient=age1example...
sops_age__list_0__map_enc=-----BEGIN AGE ENCRYPTED FILE-----\n...
sops_mac=ENC[AES256_GCM,data:X9r...,iv:K2c...,tag:T8a...,type:str]
sops_version=3.13.3
```

Visible record	Meaning
SERVICE_TOKEN=ENC[…]	encrypted value, IV, and authentication tag
age recipient + enc	public recipient and age-wrapped data key
sops_mac=ENC[…]	integrity check for the document tree

source: SOPS documentation: dotenv format and document MAC

Instapoll

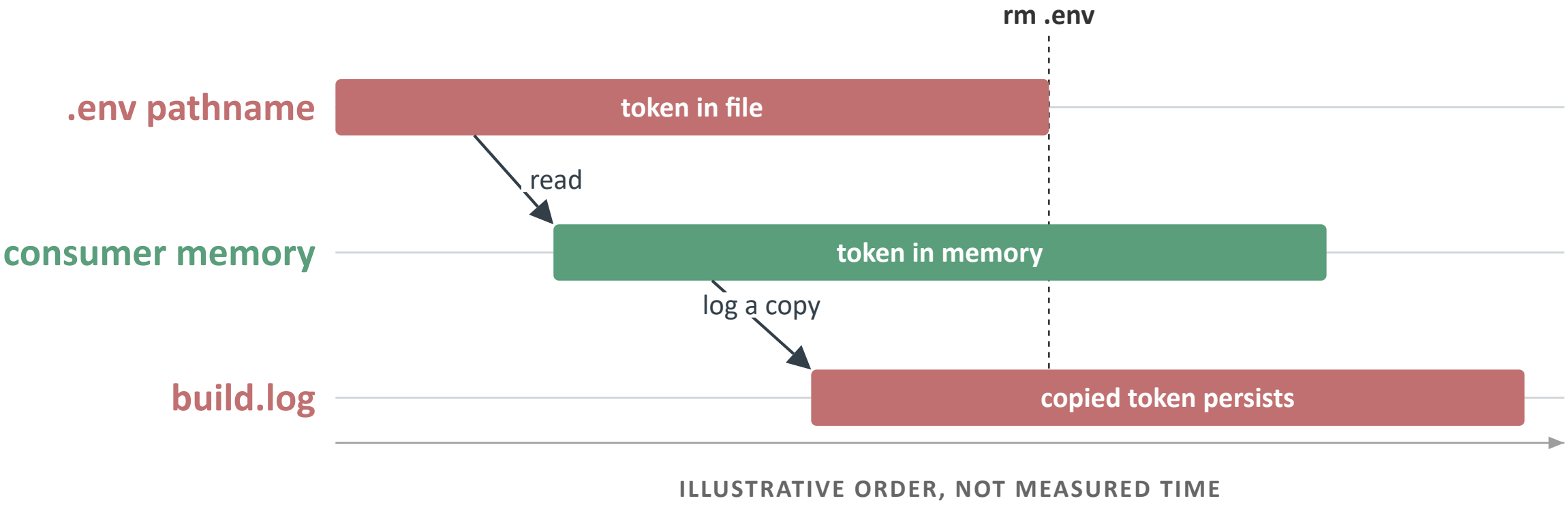
Instapoll · Review · Oct 29 B

An agent runs a build that decrypts a token to a mode-0600 .env file, uses it, then deletes the file. Which statement about the token’s exposure is justified?

- A. Mode 0600 prevents tools launched by the agent from reading the file
- B. Deleting the file revokes the token even if another process copied it
- C. A process allowed to read the file could copy the token before deletion, and that copy may survive
- D. Keeping the encrypted source in Git prevents plaintext from appearing in build output

Decryption moves the trust boundary

Illustrative leak: a permitted reader copies the token into a log before rm .env



Removing the original pathname does not remove an existing copy or revoke the credential.

Create ciphertext through stdin

Bash: fail closed, disable tracing, and write only SOPS output

```
# Bash script: fail on errors, unset variables, or failed pipeline stages.
set -euo pipefail; set +x           # disable command tracing
read -rsp 'New service token: ' token; printf '\n'
trap 'unset token' EXIT
target=secrets/service.env
printf 'SERVICE_TOKEN=%s\n' "$token" |
  sops encrypt --filename-override "$target" \
    --input-type dotenv --output-type dotenv --output "$target"
unset token; trap - EXIT
```

- -r reads literally, -s disables terminal echo, and -p prints the prompt; the token never appears in shell history or command arguments.
- The EXIT trap clears the shell variable on success or failure. --filename-override selects the configured .sops.yaml rule; --output writes only after encryption succeeds.
- Inspect ENC[…] and sops_mac before staging; never print decrypted output as a check.

source: SOPS advanced usage: stdin encryption and --filename-override

Replace one value, or accept an edit window

Use the narrow stdin path by default; interactive editing deliberately widens trust

Primary: replace one value

```
set -euo pipefail
set +x
read -rsp 'Replacement: ' token
printf '\n'
trap 'unset token' EXIT
printf '%s' "$token" | jq -Rs . |
  sops set --value-stdin \
    secrets/service.env \
    '["SERVICE_TOKEN"]'
unset token
trap - EXIT
```

Alternative: edit document

```
sops edit secrets/service.env

# SOPS creates a private
# plaintext temporary file,
# launches the editor,
# then re-encrypts and cleans up
```

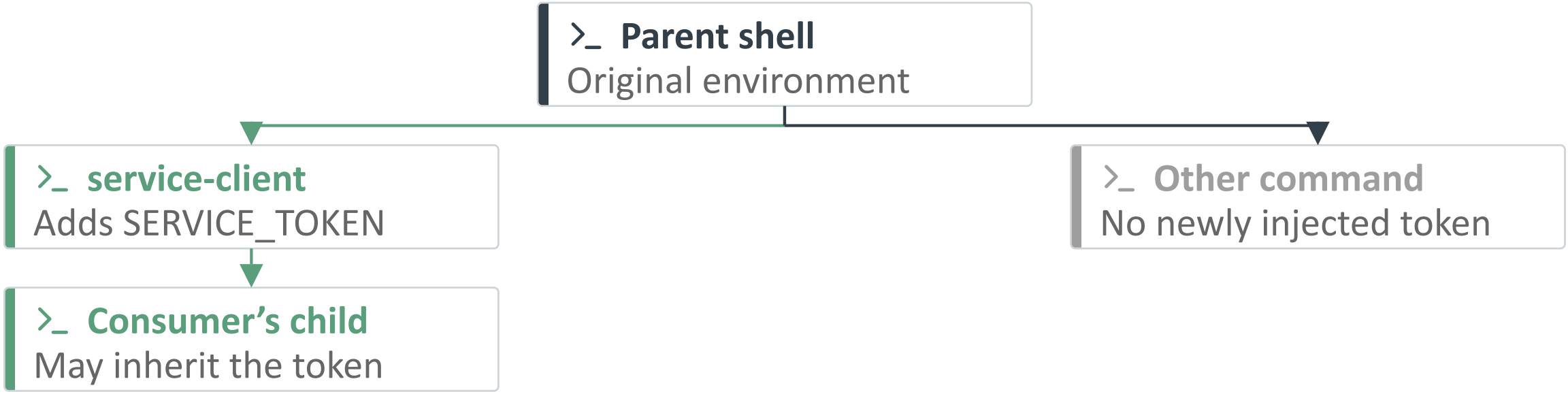
- Interactive edit is convenient for several fields, but the editor, its plugins, swap files, and crash recovery join the trust boundary.
- Neither workflow makes a compromised consumer safe; both only manage how plaintext reaches it.

source: SOPS edit implementation and set command documentation

with-secrets scopes the injected environment

Arrows are process launches; the selected command's children may inherit the token

```
with-secrets secrets/service.env -- service-client sync
```



Existing values remain; later files override earlier ones.

source: your dotfiles kit: bin/with-secrets

Know what the wrapper does and does not constrain

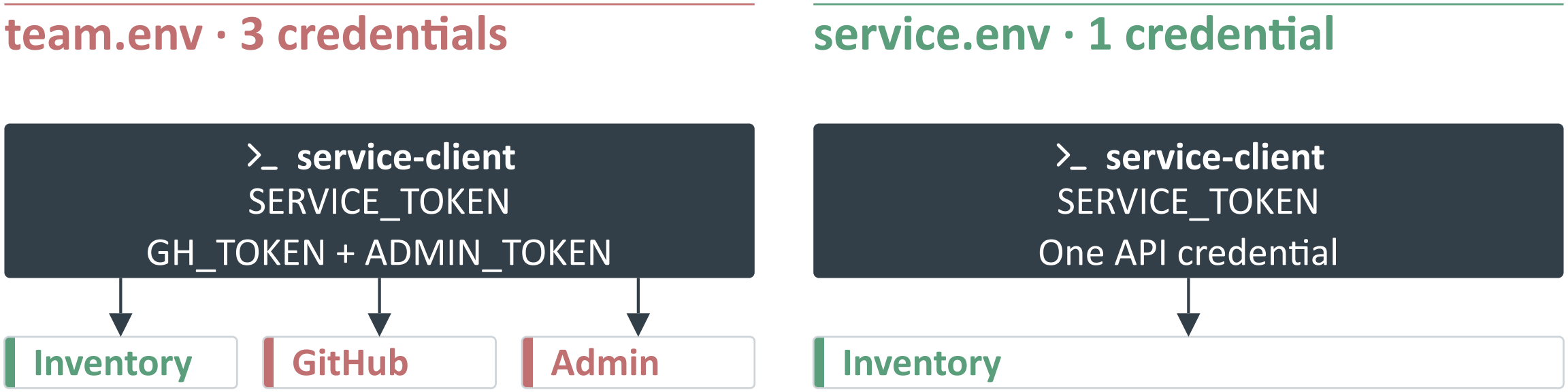
The useful boundary is an environment handoff, not a process sandbox

It reduces	It cannot prevent
plaintext files and their backup or layer residue	the consumer printing, logging, transmitting, or saving the token
unrelated commands inheriting one long-lived shell environment	children intentionally spawned by the consumer inheriting its environment
manual parsing and export code in each project	direct SOPS use by code that can read the private age identity

Actual process sandbox and file permissions still matter; with-secrets supplies variables only to the selected command tree.

Give each consumer only its service credential

Both files are encrypted; arrows show the authority added by their credentials




```
with-secrets secrets/service.env -- service-client sync
```

Restrict allowed actions at the issuer; remove unrelated inherited credentials.

Exposure changes the first action

Contain the credential before cleaning the copies

- 1 · Revoke or rotate at the issuer
  - Invalidate the exposed capability; deleting a file does not disable a token.
- 2 · Bound the exposure
  - Inspect Git history, logs, artifacts, images, backups, forks, transcripts, and audit events.
- 3 · Replace and verify
  - Create a newly scoped token, encrypt it through stdin, and test the intended consumer.
- 4 · Remove the plaintext path
  - Fix the workflow that produced the exposure; cleanup alone leaves the failure repeatable.

Assume possible exposure is real until the issuer has made the old token unusable.

Who owns the credential lifecycle?

Your Responsibility
Choose issuer scope, recipients, authorized consumers, and rotation policy
Decide whether any plaintext window is acceptable
Revoke promptly when exposure is possible

Agent's Responsibility
Use only the named encrypted file and command
Keep values out of arguments, output, logs, and files
Stop when the task needs broader scope or a new plaintext destination

Shared: inspect staged ciphertext, test the narrow consumer, and record how to rotate

Key Takeaways

1. Follow one token through its whole lifecycle: issue a narrow, replaceable credential; keep only authenticated ciphertext in Git; expose plaintext to one named consumer; revoke first when exposure is possible.

2. SOPS encrypts values and authenticates the document, while age protects the data key for named recipients. Keep the private age identity outside the repository, because possession of it enables decryption regardless of wrapper policy.

3. Prefer stdin for creation and one-value replacement, and use a command-scoped environment for consumption. These choices reduce persistent plaintext and accidental inheritance; the consumer, its children, logs, and copies remain inside the trust boundary.