Protecting secrets

Key takeaways

1. Encryption protects privacy: a key turns ciphertext back into plaintext. Reading the stored bytes should not reveal the secret.

2. HMAC protects integrity: a secret key authenticates the ciphertext and its metadata. Verify the tag before using the record.

3. Authentic records can still be copied or replayed. Check the intended purpose and compare the version with protected freshness state.

Common misunderstandings of secret protection

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

"Encrypting a token also prevents an attacker from changing which token a program uses."

"A SHA-256 hash stored beside a file detects malicious edits as reliably as HMAC."

"A valid HMAC proves that a credential is intended for this task and is the latest version."

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

Encryption for privacy, HMAC for integrity

Privacy / confidentiality

Unauthorized parties cannot read sensitive data.

Encryption

Turns plaintext into ciphertext. A decryption key recovers the plaintext.

Without the key, reading the file should not reveal the token.

Integrity

Unauthorized parties cannot alter data without prevention or detection.

HMAC

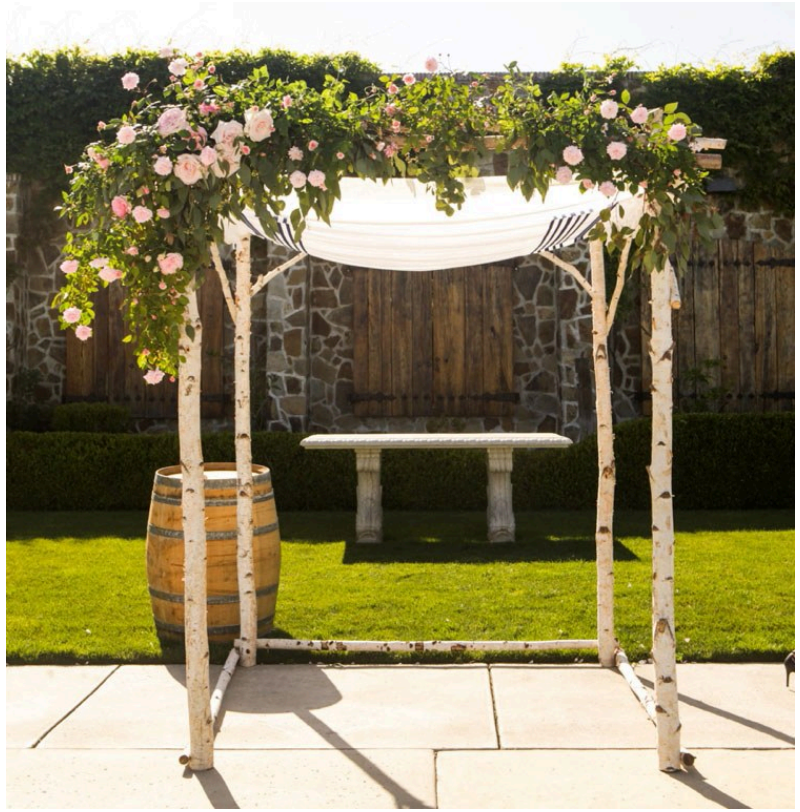
Hash-based message authentication code: computes a tag from a secret key and message.

Verify the tag over ciphertext and metadata before using the record.

Both require key protection. Neither stops copying an entire valid record.

source: NIST CSRC Glossary, confidentiality and data integrity; RFC 2104, HMAC

Trust is difficult in the real world



Encryption protects a secret until you share its plaintext. You must then trust the recipient. Even a trustworthy person can face pressure to reveal it.

- The law can reduce that pressure. In federal criminal cases, a husband or wife may refuse to testify against their spouse.
- Society accepts a cost: courts may lose useful evidence to protect a relationship built on confidence.

source: Trammel v. United States, 445 U.S. 40 (1980)

Authentication and authorization



Principal: an identity the system distinguishes, such as a person, service, or process.

Authentication

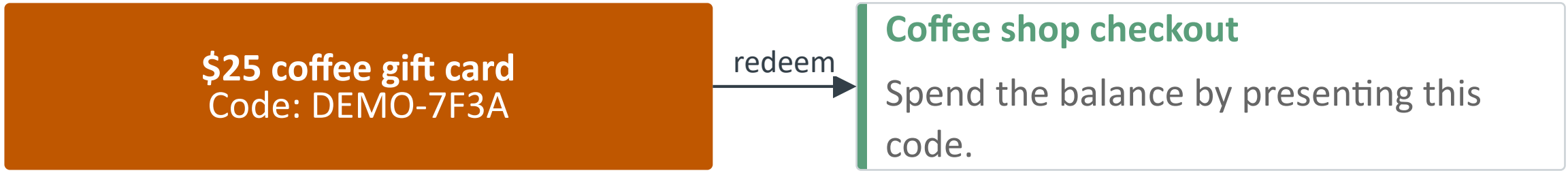
Establishes the claimed identity. An ID check connects a traveler to a name.

Authorization

Checks the permitted action. A boarding pass permits a specific flight.

source: RFC 4949, Internet Security Glossary, Version 2

Bearer tokens authorize whoever holds them



- Suppose a coffee shop accepts a gift-card code without checking your ID. Whoever presents the code can spend its remaining balance.
- A screenshot copies the credential, not the money. Another person can spend the balance before you do.
- An API bearer token works similarly: its holder can exercise its permissions. Unlike a gift-card balance, those permissions are not consumed when they are used.

source: RFC 6750, OAuth 2.0 Bearer Token Usage, §§1.2 and 5

A copied token lets an attacker use the API

A GitHub bot posts test results using a token with Issues: write permission.

GitHub REST API syntax; substitute a repository and token

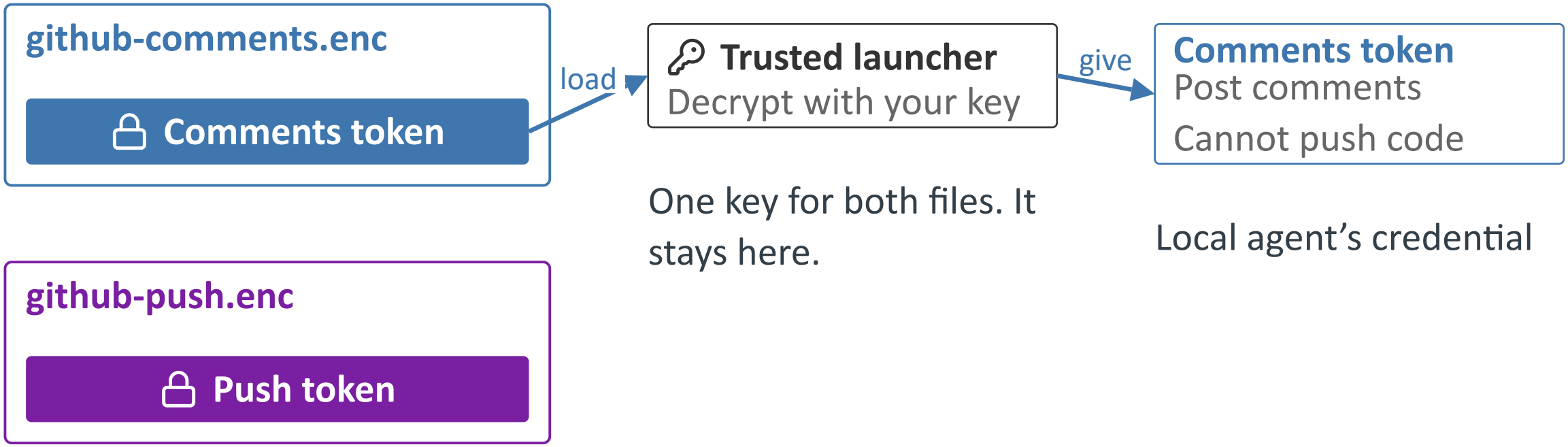
```
curl -X POST \
  https://api.github.com/repos/OWNER/REPO/issues/17/comments \
  -H "Authorization: Bearer github_pat_REDACTED" \
  -H "Content-Type: application/json" \
  -d '{"body":"The tests passed."}'
```

- Bearer names the authentication scheme: possession of this token authorizes the request. Only the string after Bearer is secret.
- Copy the token, post a false result as the bot. HTTPS protects transport; it does not distinguish the holders.

source: GitHub Docs, REST API: Create an issue comment; RFC 6750, §2.1

Encrypt stored tokens to protect privacy

Task: post issue comments. Select the blue credential.

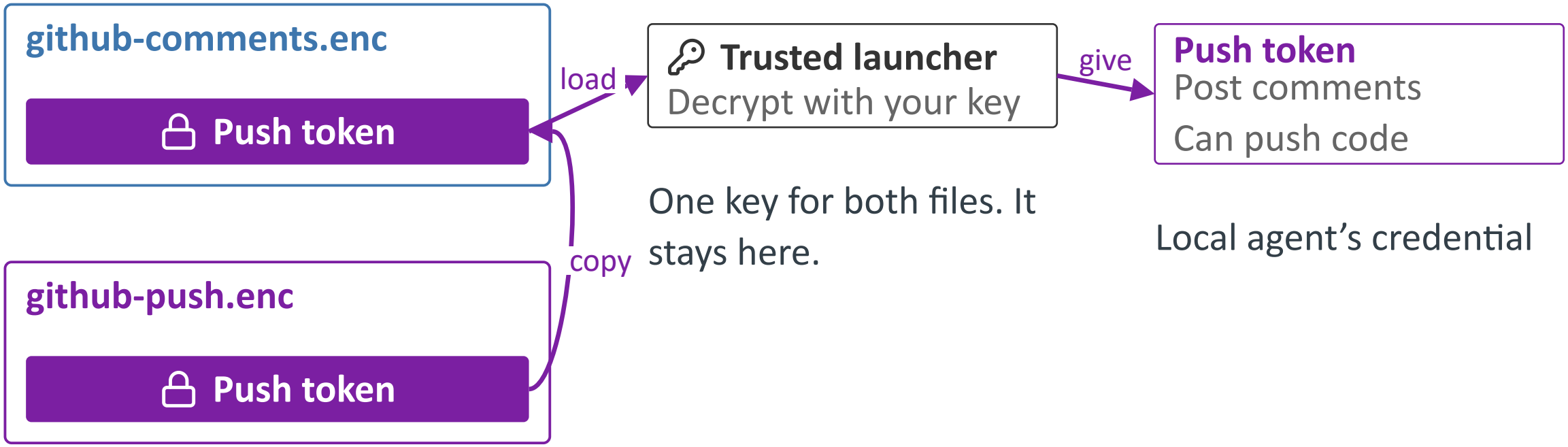


Both tokens can post issue comments; only purple can change repository code.

Attacker: can read and replace these files, but cannot read the key or change the launcher.

Copying the token file grants push access

Integrity attack: overwrite the comments file with the push file.



The attacker writes encrypted bytes without learning the token. The launcher must authenticate the record and check that its purpose matches the task.

source: Context-binding pattern: Tink, Bind ciphertext to its context

Instapoll

Instapoll · Review · Oct 20 A

A service encrypts an API token before storing it. Someone who cannot decrypt the token can still replace bytes in the stored file. What must the consumer check before using the decrypted value?

- A. The encryption key has remained private
- B. The stored file still has the original name
- C. An authenticated integrity check covering the protected data succeeds
- D. The ciphertext differs from the plaintext

HMAC protects the integrity of a record

Trusted launcher configuration: expected purpose = “comments”

```
# nonce: public; unique for each encryption under one key
message = encode(purpose, version, nonce, ciphertext)
tag = hmac.digest(mac_key, message, "sha256")
```

Intact push record

purpose = “push” · version = 42

nonce: 000102… · ciphertext: f709ca…

Stored tag: 239ef6229785…

Recomputed: 239ef6229785…

Tag matches, but push ≠ comments → REJECT

Purpose changed by attacker

purpose = “comments” · version = 42

nonce: 000102… · ciphertext: f709ca…

Stored tag: 239ef6229785…

Recomputed: 4bf683372a3f…

Stored ≠ recomputed → REJECT

source: RFC 2104, HMAC; RFC 8152, §9.1

What makes a hash cryptographic?

HMAC uses a cryptographic hash: a public, deterministic, efficient map from input bytes to a fixed-size digest. SHA-256 returns 256 bits (32 bytes).

SHA-256 of UTF-8 “comments” (no quotes or newline)

```
2b4244fb2663714ee7241914de64ede5288e4cb2921611e2a41193f11de11922
```

Preimage resistance	Given a digest h, finding any input x with H(x) = h is infeasible.
Second-preimage resistance	Given x, finding a different y with H(y) = H(x) is infeasible.
Collision resistance	Finding any two different inputs with the same digest is infeasible.

Collisions exist because the output is finite; finding one should be infeasible.

source: NIST CSRC Glossary: cryptographic hash function; FIPS 180-4

One bit spreads across the hash output

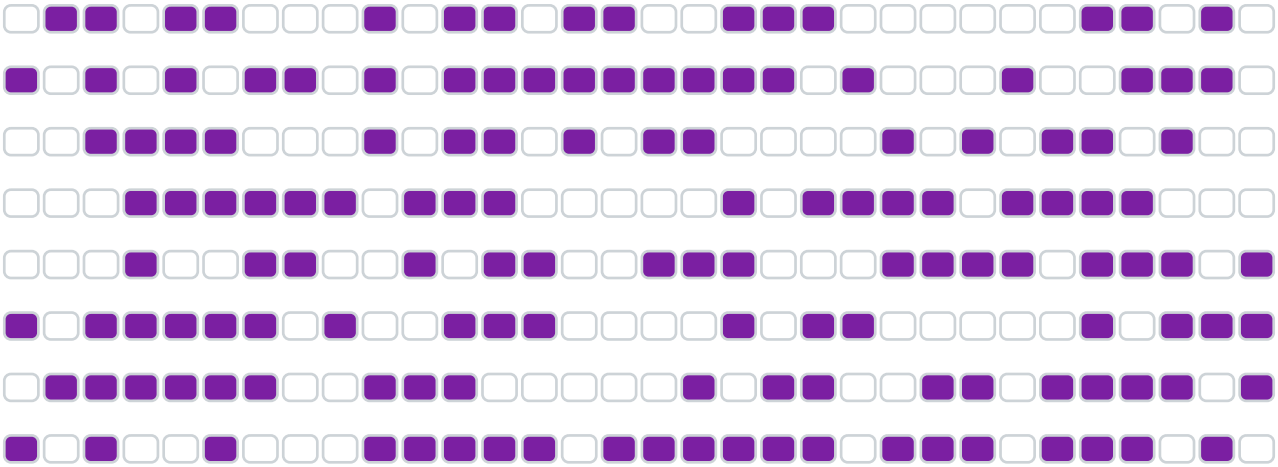
Avalanche: each output bit should flip with probability ½.

“comments” → “commentS”

Only one bit in the last byte changes:

s	0	1	**1**	1	0	0	1	1
S	0	1	**0**	1	0	0	1	1

SHA-256: one cell per output bit



Purple: 142 changed · white: 114 unchanged

Avalanche alone is not security. Anyone can recompute a hash; HMAC uses a secret key.

source: Motara & Irwin, SHA-1 and the Strict Avalanche Criterion (2016); FIPS 180-4

HMAC cannot detect replay of an intact record

You replace an overly broad token with a comments-only token.

History of github-comments.enc — purpose stays comments



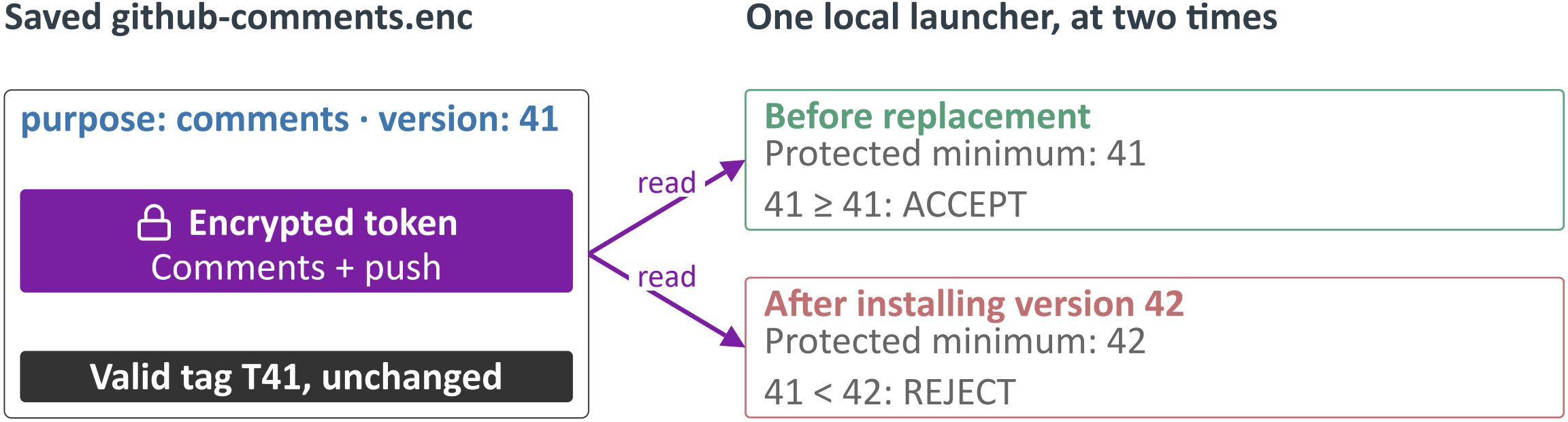
Old token was never revoked at GitHub
Replacing the local file did not invalidate the old credential.

Reader: valid tag, no remembered version → accept old token with push access.

source: RFC 5116, §1.2: authentication does not provide anti-replay

A protected minimum version rejects rollback

Keep the minimum accepted version outside replaceable data.



The reader keeps its minimum outside these files and advances it on replacement. The old file's version, nonce, and tag can all be replayed.

source: Matetic et al., ROTE (USENIX Security 2017); RFC 5116, §1.2

Replay defenses need a freshness reference

Stored-file rollback	Compare with a protected version or digest. Local protected storage can suffice; a separate server is optional.
Replayed requests	Authenticate each operation with a fresh, single-use challenge. The verifier remembers the pending challenge, not a version history.
Expired messages	An authenticated expiry and a trusted clock limit age. Replays inside the time window still need duplicate detection.

If the attacker restores both the data and every freshness reference, the reader cannot distinguish the old state from the current one.

source: NIST SP 800-63B-4, Replay Resistance; RFC 4120, §3.2.3; ROTE (2017)

Key Takeaways

1. Encrypt the token to protect its privacy. Verify HMAC over the ciphertext and metadata to detect unauthorized changes before use.

2. A valid HMAC is only the first check. Match the authenticated purpose to the chosen task, then reject replay using a protected minimum version.

3. A stolen bearer token retains its grant. Keep plaintext out of logs and limit the grant with scope, expiration, and revocation.

Match the attack to the defense

10 min

- In pairs, match attacks to defenses. Use each defense once. The attacker has no keys.

Attack	Defense
A. Read a token from a stolen file	1. Match authenticated purpose to the task
B. Edit sealed purpose: push → comments	2. Compare version to a protected minimum
C. Overwrite comments file with intact push file	3. Encrypt the token before storing it
D. Restore old comments file with push access	4. Revoke the old token at GitHub
E. Use old plaintext token after replacement	5. Verify HMAC over the record's fields

Matches: A → __ B → __ C → __ D → __ E → __

- Which attacks preserve a valid HMAC? Why does HMAC alone not stop them?
- Would replacing HMAC with a public SHA-256 hash still stop B? Explain.