FOH-1: Front-Only Hardening for Obscuring API Keys in Serverless Frontends

Alwin
Universitas Indonesia

Sahira
Universitas Gadjah Mada

***Abstract*—Placing secrets in the browser is inherently unsafe because every ingredient needed to run—and to reverse—ships to the client. FOH-1 (Front-Only Hardening) formalizes a set of techniques that raise the cost of abuse compared to shipping plaintext secrets, without promising server-grade secrecy. The core consists of build-time sealing (AES-256-GCM), runtime key reconstruction from multiple shards, per-request just-in-time (JIT) decryption, buffer zeroization, light anti-debug friction, and (in this revision) binding the ciphertext to deployment context via Additional Authenticated Data (AAD) together with content-hashed asset names and a manifest. We provide a pure-frontend reference implementation (suitable for GitHub Pages) and a test plan that verifies the claimed properties.**

I. Introduction

Vendor SDKs often forbid client-side secrets, yet certain scenarios require serverless deployments (public demos, PoCs, static sites). FOH-1 is a disciplined hardening approach for those constraints: not perfectly secure, but strictly better than embedding plaintext keys or bundling `.env` values.

Contributions

- A tool-agnostic design describing a clean lifecycle for secrets on the client, including AAD binding and content-hashed assets.
- A reference implementation: a Node sealing script (build time) and a WebCrypto module (runtime) with per-request JIT decryption.
- A frontend-only validation plan (Mocha/Chai) covering static exposure, JIT behavior, zeroization, shard integrity, AAD enforcement, anti-debug friction, and absence of source maps.

II. Threat Model and Objectives

Adversary. End users with DevTools able to patch scripts, hook `fetch`, and read static assets.
Capabilities. Breakpoint JS; capture headers; extract shards and re-derive the key offline.
Non-goals. Preventing man-in-the-browser; hiding headers on the wire; providing server-grade secrecy.
FOH-1 goals.

- Remove plaintext secrets from static artifacts (limited static secrecy).
- Shrink the plaintext's in-memory lifetime (JIT + zeroization).
- Increase reverse-engineering effort (multi-source shards, no source maps, obfuscation, optional Worker/WASM isolation).
- Keep everything reproducible on static hosting (e.g., GitHub Pages).
- Bind sealed data cryptographically to origin/path/version via AES-GCM AAD.

III. FOH-1 Design

1) **Build-time sealing.** Encrypt the API key with AES-256-GCM; output `sealed.hash.json` containing Base64 fields `{iv, ct}` with `ct = ciphertext || tag`. Compute AAD as `origin|pathBase|version` and authenticate it.
2) **Runtime key reconstruction.**

$$K = \mathrm{SHA256}(A) \oplus \mathrm{SHA256}(\texttt{mesh.svg}) \oplus \mathrm{SHA256}(B)$$

   with shard A and C as split constants in code, and shard B as the hash of a static asset `mesh.hash.svg`.
3) **Per-request JIT decryption.** Decrypt only to construct headers; do not retain plaintext in closures; wipe buffers immediately.
4) **Zeroization.** Overwrite intermediate `Uint8Array` buffers. (Strings cannot be deterministically wiped; keep lifetime minimal.)
5) **Anti-debug friction.** Disable production source maps; split strings; allow a small delay when DevTools is heuristically detected (test hook provided).
6) **Content-hashed assets & manifest.** A manifest (`foh-manifest.json`) references hashed filenames for sealed data and the mesh asset, and records the AAD tuple.

IV. Reference Implementation

A. Build-time sealing (`seal.js`, Node 18+)

```
/**
 * seal.js (Node 18+, ESM; package.json: { "type": "
    ↪ module" })
 * USAGE:
```

```
 * node seal.js "<PLAINTEXT_API_KEY>" ./public/mesh.
    ↪ svg "CONST_A" "CONST_B" ./public "https://
    ↪ yourname.github.io" "/repo-name" "v1"
 */
import { readFileSync, writeFileSync, copyFileSync,
    ↪ mkdirSync } from 'fs';
import { createHash, randomBytes, createCipheriv }
    ↪ from 'crypto';
import { join } from 'path';

const [,, API_KEY, meshPath, CONST_A, CONST_B,
    ↪ outDir, aadOrigin, aadPath, aadVersion] =
    ↪ process.argv;
if (!API_KEY || !meshPath || !CONST_A || !CONST_B ||
    ↪  !outDir || !aadOrigin || !aadPath || !
    ↪ aadVersion) {
  console.error('Usage: node seal.js "<API_KEY>" ./
    ↪ public/mesh.svg "CONST_A" "CONST_B" ./public
    ↪ "https://origin" "/pathBase" "vX"');
  process.exit(1);
}

mkdirSync(outDir, { recursive: true });

const sha256 = (buf) => createHash('sha256').update(
    ↪ buf).digest();
const hex8   = (buf) => createHash('sha256').update(
    ↪ buf).digest('hex').slice(0, 8);

const meshBytes = readFileSync(meshPath);
const hA = sha256(Buffer.from(CONST_A, 'utf8'));
const hM = sha256(meshBytes);
const hB = sha256(Buffer.from(CONST_B, 'utf8'));

const key = Buffer.alloc(32);
for (let i = 0; i < 32; i++) key[i] = hA[i] ^ hM[i]
    ↪ ^ hB[i];

// AAD bind: origin | pathBase | version
const aad = `${aadOrigin}|${aadPath}|${aadVersion}`;
const aadBytes = Buffer.from(aad, 'utf8');

// AES-256-GCM
const iv = randomBytes(12);
const cipher = createCipheriv('aes-256-gcm', key, iv
    ↪ );
cipher.setAAD(aadBytes);

const plaintext = Buffer.from(API_KEY, 'utf8');
const encrypted = Buffer.concat([cipher.update(
    ↪ plaintext), cipher.final()]);
const tag = cipher.getAuthTag();
const ctAndTag = Buffer.concat([encrypted, tag]);

// Hashed filenames
const meshName   = `mesh.${hex8(meshBytes)}.svg`;
const sealedName = `sealed.${hex8(ctAndTag)}.json`;

// Write outputs
copyFileSync(meshPath, join(outDir, meshName));
const sealed = { iv: iv.toString('base64'), ct:
    ↪ ctAndTag.toString('base64'), aad };
writeFileSync(join(outDir, sealedName), JSON.
    ↪ stringify(sealed));

const manifest = {
  mesh: meshName,
  sealed: sealedName,
  aadOrigin, aadPath, aadVersion,
  aad
};
writeFileSync(join(outDir, 'foh-manifest.json'),
    ↪ JSON.stringify(manifest, null, 2));
console.log('FOH-1 sealed:', { meshName, sealedName,
    ↪  aad });
```

B. Runtime module (public/secret.js, WebCrypto, JIT)

```
const CONST_A_PARTS = ['9w^v', 'Yk!p', 'Qz'];
const CONST_B_MIX   = { a: 'mA3', b: 'Lr#0', c: '2_f' };

// Heuristic DevTools detector + deterministic test hook
const DEVTOOLS_TRIPPED = (() => {
  let tripped = false;
  const check = () => {
    const w = window;
    if ((w.outerWidth - w.innerWidth > 200) || (w.
     ↪ outerHeight - w.innerHeight > 200)) tripped = true;
    if (w.__FORCE_DEVTOOLS__ === true) tripped = true;
  };
  check(); window.addEventListener('resize', check);
  return () => tripped;
})();

const enc = new TextEncoder();
const dec = new TextDecoder();

async function sha256(bufLike) {
  const buf = bufLike instanceof Uint8Array ? bufLike : enc.
     ↪ encode(bufLike);
  const hash = await crypto.subtle.digest('SHA-256', buf);
  return new Uint8Array(hash);
}
const zero = (b) => { if (b?.fill) b.fill(0); };

function xor32(a, b, c) {
  const out = new Uint8Array(32);
  for (let i = 0; i < 32; i++) out[i] = a[i] ^ b[i] ^ c[i];
  return out;
}

function b64ToBytes(b64) {
  const bin = atob(b64);
  const bytes = new Uint8Array(bin.length);
  for (let i = 0; i < bin.length; i++) bytes[i] = bin.
     ↪ charCodeAt(i);
  return bytes;
}

async function fetchJSON(path) {
  const r = await fetch(path, { cache: 'no-store' });
  if (!r.ok) throw new Error(`Fetch failed: ${path}`);
  return r.json();
}

async function fetchBytes(path) {
  const r = await fetch(path, { cache: 'no-store' });
  if (!r.ok) throw new Error(`Fetch failed: ${path}`);
  const ab = await r.arrayBuffer();
  return new Uint8Array(ab);
}

/**
 * initSecret
 * @param {object} opts
 *   - manifestPath: path to manifest (default 'foh-manifest
     ↪ .json' relative to page)
 *   - aadOriginOverride: override runtime origin (testing
     ↪ AAD mismatch)
 */
export async function initSecret({
  manifestPath = 'foh-manifest.json',
  aadOriginOverride
} = {}) {
  if (DEVTOOLS_TRIPPED()) {
    await new Promise(r => setTimeout(r, 700 + Math.random()
     ↪  * 900));
  }

  const manifest = await fetchJSON(manifestPath);
  const { mesh, sealed, aadOrigin, aadPath, aadVersion } =
     ↪ manifest;

  // Compose runtime AAD
  const runtimeOrigin = aadOriginOverride || window.location
     ↪ .origin;
  const aadCandidate = `${runtimeOrigin}|${aadPath}|${
     ↪ aadVersion}`;

  // Early-fail if context doesn't match
  if (aadCandidate !== manifest.aad) {
    throw new Error('FOH-1 AAD mismatch: this bundle is not
     ↪ sealed for this origin/path/version.');
  }

  const [meshBytes, sealedObj] = await Promise.all([
    fetchBytes(mesh),
    fetchJSON(sealed)
  ]);

  // Reconstruct key
  const CONST_A = enc.encode(CONST_A_PARTS.join(''));
  const CONST_B = enc.encode(CONST_B_MIX.a + CONST_B_MIX.b +
     ↪  CONST_B_MIX.c);
  const [hA, hM, hB] = await Promise.all([sha256(CONST_A),
     ↪ sha256(meshBytes), sha256(CONST_B)]);
  const rawKey = xor32(hA, hM, hB);
  zero(hA); zero(hM); zero(hB);

  const key = await crypto.subtle.importKey('raw', rawKey, {
     ↪  name: 'AES-GCM' }, false, ['decrypt']);
  zero(rawKey);

  const iv = b64ToBytes(sealedObj.iv);
  const ct = b64ToBytes(sealedObj.ct);
  const aadBytes = enc.encode(aadCandidate);

  // JIT decrypt per request
  async function authHeaderOnce() {
    const buf = await crypto.subtle.decrypt(
      { name: 'AES-GCM', iv, tagLength: 128, additionalData:
     ↪  aadBytes },
      key,
      ct
    );
    const u8  = new Uint8Array(buf);
    try {
      const token = dec.decode(u8); // strings cannot be
     ↪ zeroed
      const prefix = 'Bea' + 'rer';
      return { 'Authorization': `${prefix} ${token}` };
    } finally {
      zero(u8);
    }
  }

  return {
    withAuthFetch: async (url, init = {}) => {
      const headers = new Headers(init.headers || {});
      const h = await authHeaderOnce();
      for (const k of Object.keys(h)) headers.set(k, h[k]);
      return fetch(url, { ...init, headers });
    }
  };
}
```

C. Assets and manifest

```
<!-- public/mesh.<hash>.svg -->
<svg xmlns="http://www.w3.org/2000/svg" width="32"
     ↪ height="32" viewBox="0 0 32 32">
  <path d="M1 16 C4 2, 28 2, 31 16 C28 30, 4 30, 1
     ↪ 16 Z" fill="none" stroke="black" />
</svg>
```

Example manifest (public/foh-manifest.json):

```
{
  "mesh": "mesh.7f3a1cde.svg",
  "sealed": "sealed.2a9b0f61.json",
  "aadOrigin": "https://yourname.github.io",
  "aadPath": "/repo-name",
  "aadVersion": "v1",
  "aad": "https://yourname.github.io|/repo-name|v1"
}
```

D. Usage example (public/app.js)

```
import { initSecret } from './secret.js';

(async () => {
  const { withAuthFetch } = await initSecret();
  // Demo request (same-origin). Replace with your
     ↪ real endpoint.
  const res = await withAuthFetch('foh-manifest.json
     ↪ ', { method: 'GET' });
  console.log('Demo status:', res.status);
})();
```

E. Optional: CSP meta (defense-in-depth)

```
<meta http-equiv="Content-Security-Policy"
      content="default-src 'self'; script-src 'self
     ↪ '; connect-src 'self';
               img-src 'self' data:; style-src 'self
     ↪ ' 'unsafe-inline';
               object-src 'none'; base-uri 'none';
     ↪ frame-ancestors 'none';
               require-trusted-types-for 'script'">
```

V. Validation and Testing

A. Properties under test

- K1: No static API key or Bearer <token> in JS artifacts.
- K2: No credentials on the global scope.
- K3: JIT behavior—plaintext exists only while constructing a request; buffers are zeroized.
- K4: Shard integrity—changing the mesh asset breaks AES-GCM (invalid tag).
- K4b: AAD mismatch prevents decryption.
- K5: Anti-debug friction adds measurable delay.
- K6: No production source maps are shipped.

B. Test page (public/test.html)

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8"/>
  <title>FOH-1 Tests</title>
  <link rel="stylesheet" href="https://unpkg.com/
     ↪ mocha/mocha.css"/>
</head>
<body>
  <div id="mocha"></div>
  <script src="https://unpkg.com/mocha/mocha.js"></
     ↪ script>
  <script src="https://unpkg.com/chai/chai.js"></
     ↪ script>
  <script src="https://unpkg.com/chai-as-promised"><
     ↪ /script>
  <script>mocha.setup('bdd');</script>
  <script type="module" src="/tests/foh.spec.js"></
     ↪ script>
  <script>mocha.run();</script>
</body>
</html>
```

C. Spec (`public/tests/foh.spec.js`)

```
/* global chai, chaiAsPromised */
const { expect } = chai;
chai.use(chaiAsPromised);

import { initSecret } from '/secret.js';

const readText = (u) => fetch(u, { cache:'no-store' }).then(
    ↪ r => r.text());

function sameOriginScriptPaths() {
  const set = new Set(
    performance.getEntriesByType('resource')
      .filter(e => e.initiatorType === 'script')
      .map(e => new URL(e.name, location.href))
      .filter(u => u.origin === location.origin)
      .map(u => u.pathname)
  );
  Array.from(document.scripts).filter(s => s.src)
    .forEach(s => { const u = new URL(s.src, location.href);
    ↪  if (u.origin === location.origin) set.add(u.
    ↪ pathname); });
  Array.from(document.querySelectorAll('link[rel="
    ↪ modulepreload"][href]'))
    .forEach(l => { const u = new URL(l.href, location.href)
    ↪ ; if (u.origin === location.origin) set.add(u.
    ↪ pathname); });
  set.add('/secret.js');
  return [...set];
}

describe('FOH-1', function () {
  this.timeout(20000);

  it('K1: No static API key patterns across same-origin JS',
    ↪  async () => {
    const paths = sameOriginScriptPaths();
    const contents = await Promise.all(paths.map(p =>
    ↪ readText(p)));
    const whole = contents.join('\n');
    expect(whole).to.not.match(/sk_(live|test)_[A-Za-z0
    ↪ -9]+/);
    expect(whole).to.not.match(/Authorization["']?\s*:\s*["'
    ↪ ]Bearer\s+[A-Za-z0-9_\-+=/.]{20,}["']/);
  });

  it('K2 & K3: JIT decrypt; no credentials on global scope',
    ↪  async () => {
    const { withAuthFetch } = await initSecret();
    let seenAuth = null;

    const origFetch = window.fetch;
    window.fetch = async (url, init = {}) => {
      const res = await origFetch(url, init);
      if (init && init.headers) {
        const h = new Headers(init.headers);
        seenAuth = h.get('Authorization');
      }
      return res;
    };

    await withAuthFetch('foh-manifest.json'); // dummy
    ↪ request
    expect(seenAuth).to.be.a('string').and.match(/^Bearer\s
    ↪ +\S+/);

    const globals = Object.getOwnPropertyNames(window).join(
    ↪ '\n');
    expect(globals).to.not.match(/apiKey/i);

    window.fetch = origFetch;
  });

  it('K4: Shard integrity -- switching mesh breaks AES-GCM (
    ↪ forged manifest)', async () => {
    const origManifest = await (await fetch('foh-manifest.
    ↪ json', {cache:'no-store'})).json();
    const forged = { ...origManifest, mesh: 'mesh_bad.svg'
    ↪ }; // ensure this file exists in /public

    const blob = new Blob([JSON.stringify(forged)], { type:
    ↪ 'application/json' });
    const url  = URL.createObjectURL(blob);
    try {
      await expect((async () => {
        const { withAuthFetch } = await initSecret({
    ↪ manifestPath: url });
        await withAuthFetch('foh-manifest.json'); // dummy
      })()).to.be.rejected; // invalid GCM tag
    } finally {
      URL.revokeObjectURL(url);
    }
  });

  it('K4b: AAD mismatch causes decrypt failure', async () =>
    ↪  {
    await expect((async () => {
      const { withAuthFetch } = await initSecret({
    ↪ aadOriginOverride: 'https://evil.invalid' });
      await withAuthFetch('foh-manifest.json');
    })()).to.be.rejected;
  });

  it('K5: Anti-debug friction introduces measurable delay',
    ↪ async () => {
    window.__FORCE_DEVTOOLS__ = true;
    const t0 = performance.now();
    const { withAuthFetch } = await initSecret();
    await withAuthFetch('foh-manifest.json');
    const dt = performance.now() - t0;
    expect(dt).to.be.greaterThan(600);
  });

  it('K6: No production source maps shipped', async () => {
    const paths = sameOriginScriptPaths();
    const texts = await Promise.all(paths.map(p => readText(
    ↪ p)));
    const whole = texts.join('\n');
    expect(whole).to.not.match(/[#@]\s*sourceMappingURL\s
    ↪ *=/);
  });
});
```

VI. Expected Results

- K1: No common key patterns or hard-coded `Bearer` tokens appear in JS resources.
- K2–K3: Authorization exists only at request time; no `apiKey`-like globals.
- K4: Changing the mesh asset results in a WebCrypto `OperationError` (invalid GCM tag).
- K4b: AAD mismatch fails fast before or during decryption.
- K5: A consistent extra latency (≈0.6–1.6 s) appears when the DevTools test hook is active.
- K6: No `sourceMappingURL` markers are present in production artifacts.

VII. Security Discussion

Strengthened. No plaintext secrets in static assets; key assembled from multiple inputs; per-request JIT decryption shortens plaintext lifetime; GCM tag and AAD enforce integrity and context binding.

Residual risks. Attackers can hook `fetch` and copy headers at use time. Strings cannot be deterministically wiped. DevTools detection is heuristic and bypassable.

Operational notes. Repeated *decryption* with the same IV is safe; IV reuse is hazardous for *encryption*, not for reading sealed data. Cross-origin requests with `Authorization` trigger CORS preflight.

VIII. Limitations & Compliance

Many providers forbid client-side secrets. Prefer public/sandbox keys or narrowly scoped and rate-limited tokens. FOH-1 is not suitable for high-value secrets or regulated contexts (e.g., PCI/HIPAA). For production systems, employ an edge/server signer issuing short-lived tokens.

IX. Recommended Practices

Keep token scope minimal; rotate regularly. Enforce rate limits and origin allowlists on the provider side. Consider moving derivation and decrypt into a Web Worker or WASM for isolation. Use content-hashed filenames for all sealed assets to lock cache behavior.

X. Reproducibility & Deployment (GitHub Pages)

Directory layout

```
/public
  |-- index.html
  |-- app.js
  |-- secret.js
  |-- foh-manifest.json
  |-- mesh.<hash>.svg
  |-- mesh_bad.svg          # for K4 tests only
  |-- sealed.<hash>.json
  |-- tests/
      |-- foh.spec.js
  |-- test.html
seal.js
obfuscator.config.json
package.json
```

Local / CI flow

```
# 1) Seal whenever the key/asset changes (adjust
    ↪ origin/path/version):
node seal.js "<PLAINTEXT_API_KEY>" ./public/mesh.svg
    ↪  "CONST_A" "CONST_B" ./public \
  "https://yourname.github.io" "/repo-name" "v1"

# 2) Obfuscate (optional, for production):
npx javascript-obfuscator public --output build --
    ↪ config obfuscator.config.json

# 3) Deploy to gh-pages (use build or public):
npx gh-pages -d build

# 4) Open /test.html on GitHub Pages; all tests
    ↪ should pass.
```