Engineer

Subresource Integrity

a hash a browser checks a file against

What it is

Subresource Integrity lets a page state in advance exactly which bytes a script or stylesheet must contain, and has the browser discard the file if what arrives is different. It is usually shortened to SRI. The page writes a cryptographic digest of the file into an `integrity` attribute on the `script` or `link` element that loads it. When the response comes in, the browser runs the same hash function over the bytes it received and compares the result with the digest on the element. If the two are equal, the file runs or styles the page as usual. If they differ, the browser treats the response as a network error: the script never executes and the stylesheet never applies.

The value is the name of a hash function, a hyphen, and the digest in base64. One command over the finished file produces it, and running it twice shows how little it takes to change the answer:

```
$ printf "alert('Hello, world.');" > hello.js
$ openssl dgst -sha384 -binary hello.js | openssl base64 -A
H8BRh8j48O9oYatfu5AZzq6A9RINhZO5H16dQZngK7T62em8MUt1FLm52t+eX6xO
$ echo >> hello.js
$ openssl dgst -sha384 -binary hello.js | openssl base64 -A
qu8P7hCLFcBpas8BOnGmaANCkfXJIpFttDEaxOj0Pq9H6Q8NYjsMb+PVqwuJygVe
```

With `sha384-` in front, the first output is the attribute’s value for that one-line script. One newline added at the end gives a digest with nothing in common with the first, and a browser holding the first would refuse the second file.

The specification recognises three hash functions, SHA-256, SHA-384 and SHA-512, all from the SHA-2 family, and calls SHA-384 a good baseline at the time of writing. An attribute may list several digests separated by spaces. The browser

picks the strongest function among them and accepts the file when it matches any digest made with that function. That lets a page move to a stronger algorithm, or accept two releases of a library during a switch-over, without a window in which the page carries only wrong digests.

The attribute exists on `script` and `link`, and the same value can ride on a `Link` response header that preloads a file. The specification's current text is its second edition, a W3C Working Draft from the Web Application Security Working Group; the first edition became a W3C Recommendation in 2016.

Use the concept in a decision

Put "What is Subresource Integrity?" to work in a real decision. We can help relate the concept to the system you are building or buying. Accounts, passwords and permissions arranged so people hold exactly the access their job needs, and leavers lose it that day.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. Wrote 1,000 RBAC rules to secure Linux servers — 1,000 access rules written to close a security gap

Wrote 1,000 RBAC rules to secure Linux servers Get in touch →

Why it matters

A page that loads a library from a content delivery network, or from any host its owners do not run, hands that host the power to run code on the page. HTTPS protects the file on its way across the network, but it says nothing about what the host chose to send in the first place. If the CDN's storage is broken into, an account that publishes to it is taken over, or the file behind a familiar address is quietly replaced, every page that includes it runs the new code with the page's full authority. The page's own code has no way to tell the difference.

SRI moves that decision back to the page. The digest is computed by the site's owner over a file they have read or built, and it travels inside the page, from the site's own server. A host that serves anything else has its file refused instead of run. The page loses a feature until somebody looks, a visible fault with an obvious cause, instead of quietly running somebody else's code.

The digest also pins a version. An address such as `/lib/latest.js` can start serving a new release under a page at any moment; the digest cannot follow it. An upgrade the owner never chose becomes a refusal on one specific element, reported through an error event.

It is worth being exact about what SRI does not do. It checks a file against a digest; it does not judge whether the file is safe. A digest of a library that already contains a hostile line protects that hostile line faithfully. It leaves inline script alone, which is the work of a Content Security Policy. And it trusts the page that carries the digest, so the page itself still has to arrive intact.

What it looks like in practice

This site uses it on its own stylesheet. The security page lists Subresource Integrity on the stylesheet: the bundle is fingerprinted and hashed when the site is built, so the reader's browser can verify that it arrived unmodified. No executable JavaScript is published and there are no third-party requests. A reader who counts the hosts in the browser's network panel finds one, the site's own.

The credits page describes the same file from the build's side, as one CSS bundle concatenated and fingerprinted at build time and served with an SRI hash. Fingerprinting puts a hash of the file's contents into its name, so a changed stylesheet gets a new address and a browser never mistakes an old cached copy for the current one, while the integrity value lets the browser check the bytes behind that address. The build produces both from the same bytes in the same step, so every release ships the stylesheet and its digest together, and the commonest failure described below, a file that changes after it was hashed, has very little room to happen.

Where it goes wrong

Something between the build and the browser rewrites the file. A CDN that minifies scripts, a proxy that injects into response content, or a deploy step that rewrites a file after its digest was taken changes the content bytes, and the browser refuses the result even though nobody attacked anything. Content-preserving gzip or Brotli recompression does not cause that mismatch: the check uses the decoded response body, not the compressed bytes sent over the network. The specification puts the duty on content-modifying intermediaries: they must keep the digest in step with what they serve, or deliver the original untouched, and the origin should send `Cache-Control: no-transform` to ask them to leave the file alone. The dependable order is to take the digest last, over exactly the decoded content bytes the browser will check.

A cross-origin file arrives without CORS. The specification requires an integrity check on a file from another origin to use CORS, and calls an attempt without it a logical error. For example, the element can use `crossorigin="anonymous"` and the host must answer with a header that shares the file. `crossorigin="use-credentials"` also uses CORS, with server headers that explicitly allow the requesting origin and credentials. Without a CORS request and matching server policy, the browser blocks the file rather than load it unchecked. The rule protects other sites: without it, an attacker could load a private response with a guessed digest and learn from the outcome, for example, whether the reader is logged in there.

The digest arrives over plain HTTP. SRI protects a page against a compromised file host, never against someone on the network path. On a page served without HTTPS, whoever can rewrite the file in transit can rewrite or delete the digest in the page as well, so the specification recommends delivering integrity metadata only to a secure context.

The hash function is one the browser skips. The specification recognises only SHA-256, SHA-384 and SHA-512 and recommends that browsers refuse known-weak functions such as MD5 and SHA-1. The surprise is what happens to a token outside that list: the browser ignores it, and an attribute made only of ignored tokens counts as carrying no metadata at all, so the file loads unchecked. A build tool that can fingerprint with MD5, as Hugo's can, makes this easy to ship by accident.

A refused file takes the page down with it. When the check fails, the browser fires an error event on the element and the script or stylesheet is simply missing. For a library the page cannot run without, the specification suggests catching the event and loading a fallback copy from a trusted, if slower, source such as the site's own server. Without one, the protection works as designed and the reader still gets a broken page.

Only the elements that carry a digest are checked. An integrity attribute on one script says nothing about the next one somebody adds without it, and reviewing the scripts that have one will never reveal the one that lacks it. The draft adds a response header that covers this from the server side. `Integrity-Policy` with `blocked-destinations=(script)` blocks every external script that arrives without integrity metadata, and `Integrity-Policy-Report-Only` lists them first, so they can be fixed before anything is blocked.

Further reading

- Subresource Integrity — the current draft, with the verification algorithm, the `Integrity-Policy` headers and the notes on proxies, plain HTTP and cross-origin leaks.
- Subresource Integrity, W3C Recommendation 2016 — the first edition, the stable Recommendation the draft grew from.
- Fetch Standard — where a request's integrity metadata is checked against the response body, and the CORS protocol a cross-origin check depends on.
- Hugo: resources.Fingerprint — how a static site generator fingerprints a file at build time and hands back the matching integrity value.
- FIPS 180-4, Secure Hash Standard — the federal standard that defines SHA-256, SHA-384 and SHA-512.

Glossary

Choose for your actual constraints

Bring the deployment path, the operational risks and the people on call. We can scope a change with checks, a recovery route and clear ownership.

See what it costs Get in touch →

Make the terminology practical

Ask how "What is Subresource Integrity?" applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/sri/