20 March 2025

Modern Algorithms in the Web Cryptography API

Filip Skokan
Staff Engineer at Okta
https://github.com/panva/jose (JOSE for Node.js, Browser, Cloudflare Workers, Deno, Bun, and other Web-interoperable runtimes) ~100M monthly downloads

Modern Algorithms in the Web Cryptography API

where: https://twiss.github.io/webcrypto-modern-algos/ soon at WICG (Web Platform Incubator Community Group)

who: Daniel Huigens, current Editor of Web Cryptography API

what: *This specification defines a number of modern cryptographic algorithms for the Web Cryptography API, namely ML-KEM, ML-DSA, SLH-DSA, AES-OCB, ChaCha20-Poly1305, SHA-3, cSHAKE, KMAC, and Argon2.*

IETF

Modern Algorithms in the Web Cryptography API

why?

“This is about representation and packaging” - Mike Jones @ COSE circa 20 hours ago

- public/private/secret CryptoKey import and export in JWK format
- JWK “alg” is checked during key import *when present*
- JWK “alg” is included in exported keys *whenever possible*

Modern Algorithms in the Web Cryptography API

Briefly on Web Cryptography API and its “Algorithms”

- Referred to as WebCrypto, WebCryptoAPI, or SubtleCrypto, it’s not the most convenient API* but it’s the only one we’ve got that’s available in most ~~Java~~EcmaScript runtimes
- Keys are generated as `CryptoKey` instances. These have a type either secret, public, or private.
- Keys are imported from key formats (raw, spki, pkcs8, jwk) as `CryptoKey` instances.
- Keys may be exported from `CryptoKey` instances in key formats (raw, spki, pkcs8, jwk).
- `CryptoKey` instances have an “Algorithm”
  - e.g.“Ed25519”; or
  - { name: “RSA-PSS”, hash: { name: “SHA-256” } }
  - { name: “ECDSA”, namedCurve: “P-256” }
- `CryptoKey` instances are used as input for methods e.g. `.sign()`, `.encrypt()`
- Methods accept an “Algorithm” too
  - e.g.“Ed25519”; or
  - { name: “RSA-PSS”, saltLength: 32 }
  - { name: “ECDSA”, hash: { name: “SHA-256” }
- Method Algorithm dictionaries are not always the same as CryptoKey Algorithm dictionaries

** The `SubtleCrypto` interface is named "SubtleCrypto" to reflect the fact that many of these algorithms have subtle usage requirements in order to provide the required algorithmic security guarantees.*

IETF

Modern Algorithms in the Web Cryptography API

Web Cryptography API's intersection with JOSE is the JSON Web Key format

IETF

Modern Algorithms in the Web Cryptography API

ML-DSA

- all definitions in https://datatracker.ietf.org/doc/draft-ietf-cose-dilithium/
- new JWK Algorithm Key Pair (AKP) key type
  - note: “alg” is required
  - pending update to align private key representation with LAMPS

Modern Algorithms in the Web Cryptography API

ML-KEM

- new SubtleCrypto methods introduced to encapsulate/decapsulate
- JWA definitions in https://datatracker.ietf.org/doc/draft-ietf-jose-pqc-kem/
  - but there's no JOSE Key representation in this document, no mention of the use of Algorithm Key Pair (AKP) key type

Modern Algorithms in the Web Cryptography API

SLH-DSA

- definitions in https://datatracker.ietf.org/doc/draft-ietf-cose-sphincs-plus/
  - uses Algorithm Key Pair (AKP) key type
    - note: “alg” is required
  - SLH-DSA-SHA2-128s, SLH-DSA-SHAKE-128s, and SLH-DSA-SHA2-128f
    - Q: this is a subset of the SLH-DSA parameter sets, meaning the full set cannot be represented as JWK

IETF

Modern Algorithms in the Web Cryptography API

AES-OCB

Authenticated encryption and decryption using AES in OCB mode, as described in RFC7253.

🤷 WWJD?

- JWK "kty" oct
- JWK "alg" "A128OCB", "A192OCB", "A256OCB"
- Algorithm Usage Location: JWK (not alg)
- JOSE Implementation Requirements: Prohibited???

Follows existing registration conventions for modes we don't have JOSE JWA for.

Modern Algorithms in the Web Cryptography API

ChaCha20-Poly1305

Authenticated encryption and decryption using AEAD_CHACHA20_POLY1305, as described in RFC8439.

🤷 WWJD?

- JWK "kty" oct
- JWK "alg" "C20P"
- Algorithm Usage Location: JWK (not alg)
- JOSE Implementation Requirements: Prohibited???

"C20P" was previously used in an I-D https://datatracker.ietf.org/doc/draft-amringer-jose-chacha/ some libraries use this identifier.

IETF

Modern Algorithms in the Web Cryptography API

SHA-3

SHA-3 family of hash functions as described in FIPS-202.

note: “digest” algorithms apply to all WebCryptoAPI Algorithms that have a “hash” in their dictionary

🤷 WWJD?

- Interaction with HMAC, i.e. HS256 (HMAC using SHA-256)
- Interaction with RSASSA-PKCS1-v1_5, i.e. RS256 (RSASSA-PKCS1-v1_5 using SHA-256
- Interaction with RSASSA-PSS, i.e. PS256 (RSASSA-PSS using SHA-256)
- Interaction with ECDSA, i.e. ES256 (ECDSA using P-256 and SHA-256)
- Interaction with RSA-OAEP, i.e. RSA-OAEP-256 (RSAES OAEP using SHA-256 and MGF1 with SHA-256)

Modern Algorithms in the Web Cryptography API

cSHAKE (part 1/2)

cSHAKE128 and cSHAKE256 as described in NIST-SP800-185. When customizations are empty this is just SHAKE128 and SHAKE256, these are XOF

note: “digest” algorithms apply to all WebCryptoAPI Algorithms that have a “hash” in their dictionary

Same interactions as with SHA-3 but more complicated because of the variable length output

Modern Algorithms in the Web Cryptography API

cSHAKE (part 2/2)

Same interactions as with SHA-3 but more complicated because of the variable length output

🤷 WWJD?

- Interaction with HMAC, i.e. HS256 (HMAC using SHA-256)
- Interaction with RSASSA-PKCS1-v1_5, i.e. RS256 (RSASSA-PKCS1-v1_5 using SHA-256
- Interaction with RSASSA-PSS, i.e. PS256 (RSASSA-PSS using SHA-256)
  - as per RFC8702
- Interaction with ECDSA, i.e. ES256 (ECDSA using P-256 and SHA-256)
  - as per RFC8702
- Interaction with RSA-OAEP, i.e. RSA-OAEP-256 (RSAES OAEP using SHA-256 and MGF1 with SHA-256)

IETF

Modern Algorithms in the Web Cryptography API

KMAC

KMAC128 and KMAC256, as specified by NIST-SP800-185, these are XOF

🤷 WWJD?

- JWK "kty" oct
- JWK "alg" "K128", “K256”
- Algorithm Usage Location: JWK (not alg)
- JOSE Implementation Requirements: Prohibited???

Lacking a JWA Algorithm definition that’s probably the most we can do?