Authoritative

Guide to CBOM

Implement Cryptography Bill of Materials for Post-Quantum Systems and Applications

OWASP

Second Edition

Table of Contents

About the Guide

CycloneDX is a modern standard for the software supply chain. It has been ratified as ECMA-424 by Ecma International.

The content in this guide results from continuous community feedback and input from leading experts in the software supply chain security field. This guide would not be possible without valuable feedback from the CycloneDX Industry Working Group (IWG), the CycloneDX Core Working Group (CWG), the many CycloneDX Feature Working Groups (FWG), Ecma International Technical Committee 54, and a global network of contributors and supporters.

Copyright and License

Second Edition, 21 October 2025

Version	Changes	Updated On	Updated By
Second Edition	Updated for CycloneDX v1.7	2025-10-21	CycloneDX Cryptography Working Group
First Edition	Initial Release	2024-04-09	CycloneDX Cryptography Working Group

Preface

Welcome to the Authoritative Guide series by the OWASP Foundation and OWASP CycloneDX. In this series, we aim to provide comprehensive insights and practical guidance, ensuring that security professionals, developers, and organizations alike have access to the latest best practices and methodologies.

At the heart of the OWASP Foundation lies a commitment to inclusivity and openness. We firmly believe that everyone deserves a seat at the table when it comes to shaping the future of cybersecurity standards. Our collaborative model fosters an environment where diverse perspectives converge to drive innovation and excellence.

In line with this ethos, the OWASP Foundation has partnered with Ecma International to create an inclusive, community-driven ecosystem for security standards development. This collaboration empowers individuals to contribute their expertise and insights, ensuring that standards like CycloneDX reflect the collective wisdom of the global cybersecurity community.

One standout example of this model is OWASP CycloneDX, which has been ratified as an Ecma International standard and is now known as ECMA-424. By leveraging the strengths of both organizations, CycloneDX serves as a cornerstone of security best practices, providing organizations with a universal standard for software and system transparency.

As you embark on your journey through this Authoritative Guide, we encourage you to engage actively with the content and join us in shaping the future of cybersecurity standards. Together, we can build a safer and more resilient digital world for all.

Andrew van der Stock
Executive Director, OWASP Foundation

The Innovative History of OWASP CycloneDX

OWASP CycloneDX has carved a legacy steeped in innovation, collaboration, and a commitment to openness. OWASP continues to advance software and system transparency standards, prioritizing capabilities that facilitate risk reduction.

October 2025

OWASP CycloneDX v1.7

First specification supporting citations that improve traceability, attribution, and auditability, and comprehensive support for patents and patent families to address intellectual property transparency.

June 2024

International Standardization

CycloneDX v1.6 ratified as an Ecma International standard and published as ECMA-424.

April 2024

OWASP CycloneDX v1.6

First specification to support cryptographic assets for Post-Quantum Cryptography (PQC) readiness and first general-purpose attestation specification to digitally transform audit and attestation workflows.

December 2023

Ecma TC54 Established

First working group chartered with holistic supply chain goals of standardizing core data formats, APIs, and algorithms that advance software and system transparency.

June 2023

OWASP CycloneDX v1.5

First specification to support AI Transparency, configuration and data components, and formulation describing how components were created, tested, trained, evaluated, and deployed.

January 2022

OWASP CycloneDX v1.4

First specification to introduce vulnerability sharing and transparency, including Vulnerability Disclosure Reports (VDR) and Vulnerability Exploitability eXchange (VEX).

May 2021

OWASP CycloneDX v1.3

First specification to incorporate support for composition completeness surpassing NTIA's framing of "known unknowns".

May 2020

OWASP CycloneDX v1.2

First specification to incorporate SWID (ISO/IEC 19770-2:2015) and services into inventory including data classifications, providers, and relationships between services and components.

March 2019

OWASP CycloneDX v1.1

First specification with complete pedigree support describing component lineage and the commits, patches, and diffs which make a forked version unique.

March 2018

OWASP CycloneDX v1.0

First general-purpose, security-focused Bill of Materials standard supporting software and hardware components. Introduced the world to Package-URL for software security use cases.

Source: https://tc54.org/history

Introduction

A Cryptography Bill of Materials (CBOM) is an object model to describe cryptographic assets and their dependencies. Support for CBOM is included in CycloneDX v1.6 and higher. Discovering, managing, and reporting on cryptographic assets is necessary as the first step on the migration journey to quantum-safe systems and applications. Cryptography is typically buried deep within components that are used to compose and build systems and applications. It makes sense to minimize this effort through alignment and reuse of concepts and components used to implement the Software Supply Chain Security (SSCS).

Advances in quantum computing introduce the risk of previously-secure cryptographic algorithms becoming compromised faster than ever before. In May of 2022, the White House released a National Security Memorandum outlining the government's plan to secure critical systems against potential quantum threats. This memorandum contains two key takeaways for both agency and commercial software providers: document the potential impact of a breach, and have an alternative cryptography solution ready.

As cryptographic systems evolve from using classical primitives to quantum-safe primitives, there is expected to be more widespread use of cryptographic agility, or the ability to quickly switch between multiple cryptographic primitives. Cryptographic agility serves as a security measure or incident response mechanism when a system's cryptographic primitive is discovered to be vulnerable or no longer complies with policies and regulations.

As part of an agile cryptographic approach, organizations should seek to understand what cryptographic assets they are using and facilitate the assessment of the risk posture to provide a starting point for mitigation. CycloneDX designed CBOM for this purpose.

CBOM Design

The overall design goal of CBOM is to provide an abstraction that allows modeling and representing cryptographic assets in a structured object format. This comprises the following points.

1. Modelling cryptographic assets Cryptographic assets occur in several forms. Algorithms and protocols are most commonly implemented in specialized cryptographic libraries. They may, however, also be 'hardcoded' in software components. Certificates and related cryptographic material like keys, tokens, secrets, or passwords are other cryptographic assets to be modeled.

2. Capturing cryptographic asset properties Cryptographic assets have properties that uniquely define them and that make them actionable for further reasoning. For example, it makes a difference if one knows the algorithm family (e.g. AES) or the specific variant or instantiation (e.g. AES-128-GCM). This is because the security level and the algorithm primitive (authenticated encryption) are only defined by the definition of the algorithm variant. The presence of a weak cryptographic algorithm like SHA1 vs. HMAC-SHA1 also makes a difference. Therefore, the goal of CBOM is to capture relevant cryptographic asset properties.

3. Capturing crypto asset dependencies To understand the impact of a cryptographic asset, it is important to capture its dependencies. Cryptographic libraries 'implement' certain algorithms and protocols, but their implementation alone does not reflect their usage by applications. CBOM, therefore, differentiates between 'provides' and 'uses' dependencies. It is possible to model algorithms or protocols that use other algorithms (e.g., TLS 1.3 uses ECDH/secp256r1), libraries that implement algorithms, and applications that 'use' algorithms from a library.

4. Applicability to various software components CycloneDX supports various components such as applications, frameworks, libraries, containers, operating systems, devices, firmware, and files. CBOM extends this model and can communicate a component's dependency on cryptographic assets.

5. High compatibility to CycloneDX SBOM and related tooling CBOM is native to the CycloneDX standard. It integrates cryptographic assets as an additional component type in the CycloneDX schema and further extends dependencies with the ability to specify dependency usage and implementation details. CBOM data can be present in existing xBOMs or externalized into dedicated CBOMs, thus creating modularity, which may optionally have varying authentication and authorization requirements.

6. Enable automatic reasoning CBOM enables tooling to reason about cryptographic assets and their dependencies automatically. This allows checking for compliance with policies that apply to cryptographic use and implementation.

Use Cases

Cryptography Asset Management

The Cryptography Bill of Materials (CBOM) is a comprehensive inventory of cryptographic assets, encompassing keys, certificates, tokens, and more. This is a requirement of the OMB M-23-02, where such a system is characterized as a [..."software or hardware implementation of one or more cryptographic algorithms that provide one or more of the following services: (1) creation and exchange of encryption keys; (2) encrypted connections; or (3) creation and validation of digital signatures."]

CBOM provides a structured framework for organizations to catalog and track their cryptographic resources, facilitating efficient management and ensuring security and compliance standards are met. By maintaining a detailed record of cryptographic assets, including their usage, expiration dates, and associated metadata, CBOM enables proactive monitoring and streamlined auditing processes. With CBOM, organizations can effectively safeguard their cryptographic infrastructure, mitigate risks associated with unauthorized access or misuse, and maintain the integrity and confidentiality of sensitive data across diverse digital environments.

Identifying Weak Cryptographic Algorithms

CBOM enables organizations to conduct thorough assessments and discover weak algorithms or flawed implementations that could compromise security. Through analysis of CBOM data, including cryptographic algorithms, key management practices, and usage patterns, organizations can pinpoint areas of concern and prioritize remediation efforts. CBOM facilitates proactive identification of weaknesses and vulnerabilities, allowing organizations to enhance the resilience of their cryptographic infrastructure and mitigate the risk of exploitation, thereby bolstering overall cybersecurity posture and safeguarding sensitive data against potential threats.

Post-Quantum Cryptography (PQC) Readiness

CBOM is crucial in preparing applications and systems for an impending post-quantum reality, aligning with guidance from the National Security Agency (NSA) and the National Institute of Standards and Technology (NIST). As quantum computing advancements threaten the security of current cryptographic standards, CBOM provides a structured approach to inventorying cryptographic assets and evaluating their resilience against quantum threats.

Most notably, public key algorithms like RSA, DH, ECDH, DSA or ECDSA are considered not quantum-safe. These algorithms occur in various components and may be hardcoded in applications but are more commonly and preferably used via dedicated cryptographic libraries or services. Developers often don't directly interact with cryptographic algorithms such as RSA or ECDH but use them via protocols like TLS 1.3 or IPsec, by using certificates, keys, or other tokens. With upcoming cryptographic agility it becomes less common to put in stone (or software) the algorithms that will be used. Instead, they are configured during deployment or negotiated in each network protocol session. CBOM is designed with these considerations in mind and to allow insight into the classical and quantum security level of cryptographic assets and their dependencies.

By cataloging cryptographic algorithms and their respective parameters, CBOM enables organizations to identify vulnerable or weak components that require mitigation or replacement with quantum-resistant alternatives recommended by NSA and NIST. Through comprehensive analysis and strategic planning facilitated by CBOM, organizations can proactively transition to post-quantum cryptographic primitives, ensuring the long-term security and integrity of their systems and applications.

Assess Cryptographic Policies and Advisories

A cryptographic inventory in machine-readable form like CBOM brings benefits if one wants to check for compliance with cryptographic policies and advisories. An example of such an advisory is CNSA 2.0, which was announced by NSA in September of 2022. CNSA 2.0 states, among other things, that National Security Systems (NSS) for firmware and software signing needs to support and prefer CNSA 2.0 algorithms by 2025 and exclusively use them by 2030. The advised algorithms are the stateful hash-based signature schemes LMS and XMSS from NIST SP 800-208. With a CBOM inventory that documents the use of LMS and XMSS by such systems, compliance with CNSA 2.0 can be assessed in an automated way.

Identify Expiring and Long-Term Cryptographic Material

An RSA certificate expiring in one week poses less cryptographic risk than the same certificate expiring in 20 years. Service downtime due to an expired certificate is another risk to be considered. Therefore, we argue that an inventory that captures the life cycle of cryptographic material as allowed by CBOM gives context to an inventory that is instrumental for managing cryptographic risk.

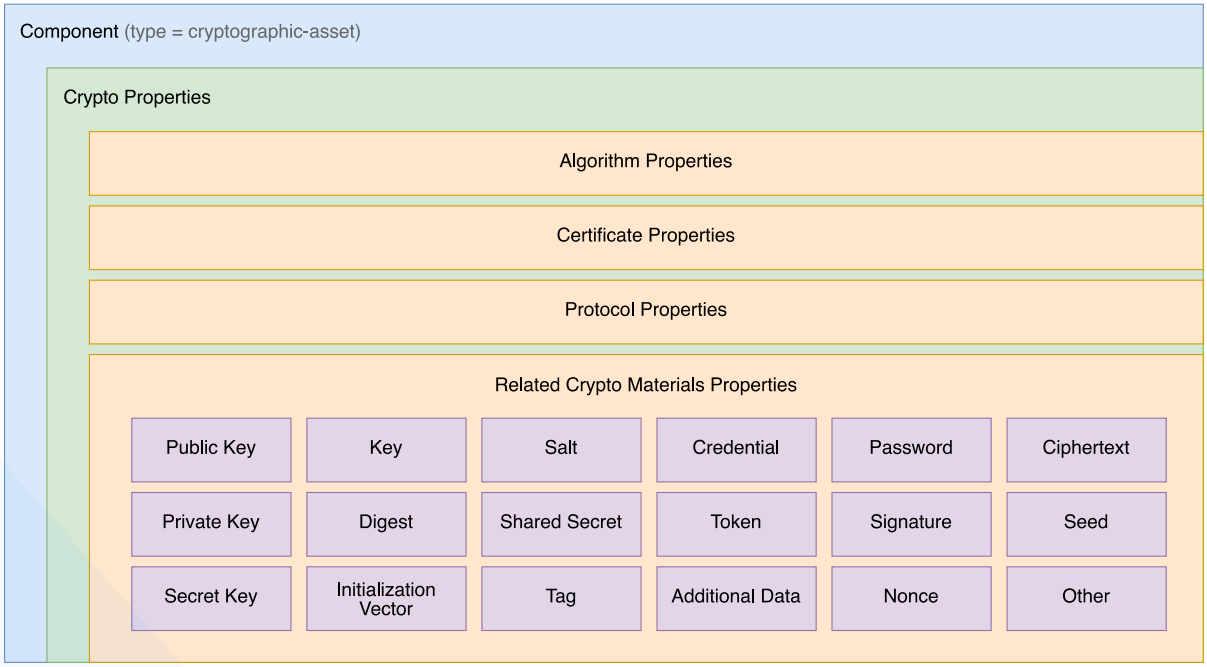
Ensure Cryptographic Certifications

Higher cryptographic assurance is provided by certifications such as FIPS 140-3 (levels 1 to 4) or Common Criteria (EAL1 to 7). To obtain these certifications, cryptographic modules need to undergo certification processes. For regulated environments such as FedRAMP, such certifications are important requirements. CBOM allows the capture of certification levels of cryptographic assets so that this property can be easily identified.

Anatomy of a CBOM

The Cryptography Bill of Materials (CBOM) presents a structured approach to inventory management of cryptographic assets, leveraging the CycloneDX component model to comprehensively represent a diverse array of cryptographic assets. CBOM is implemented within the CycloneDX object model, facilitating the standardized representation of cryptographic assets, including algorithms, keys, protocols, and certificates. The CycloneDX object model provides a flexible and extensible framework that accommodates the complexity and diversity of cryptographic infrastructure.

Structure and Cryptographic Asset Types



CycloneDX can represent the following types of cryptographic assets:

Type	Description
algorithm	Cryptographic function commonly used for data encryption, authentication, and digital signatures or other primitives.
certificate	An electronic document that provides the identity or validates a public key.
protocol	A set of rules and guidelines that govern the behavior and communication with each other.
private-key	The confidential key of a key pair used in asymmetric cryptography.
public-key	The non-confidential key of a key pair used in asymmetric cryptography.
secret-key	A key used to encrypt and decrypt messages in symmetric cryptography.

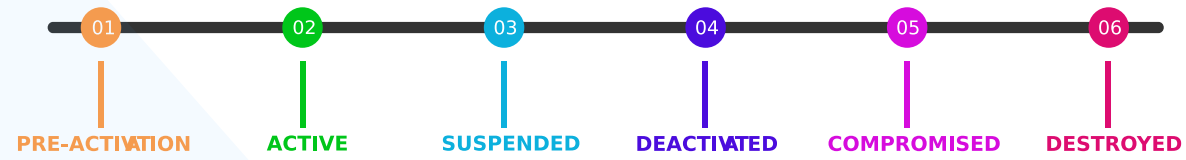
Type	Description
key	A piece of information, usually an octet string, which, when processed through a cryptographic algorithm, processes cryptographic data.
ciphertext	The result of encryption performed on plaintext using an algorithm (or cipher).
signature	A cryptographic value that is calculated from the data and a key known only by the signer.
digest	The output of the hash function.
initialization-vector	A fixed-size random or pseudo-random value used as an input parameter for cryptographic algorithms.
nonce	A random or pseudo-random number that can only be used once in a cryptographic communication.
seed	The input to a pseudo-random number generator. Different seeds generate different pseudo-random sequences.
salt	A value used in a cryptographic process, usually to ensure that the results of computations for one instance cannot be reused by an attacker.
shared-secret	A piece of data known only to the parties involved, in a secure communication.
tag	A message authentication code (MAC), sometimes known as an authentication tag, is a short piece of information used for authenticating and integrity-checking a message.
additional-data	An unspecified collection of data with relevance to cryptographic activity.
password	A secret word, phrase, or sequence of characters used during authentication or authorization.
credential	Establishes the identity of a party to communication, usually in the form of cryptographic keys or passwords.
token	An object encapsulating a security identity.
other	Another type of cryptographic asset.
unknown	The type of cryptographic asset is not known.

Key Management

NIST Special Publication 800-57 outlines key management guidelines, including various key states and their purposes. The key states defined in SP 800-57 include:

1. **Pre-activation**: Keys in this state are generated but not yet placed into active use. They are securely stored until needed, awaiting activation for cryptographic operations.
2. **Active**: Keys in the active state are currently in use for cryptographic operations such as encryption, decryption, or digital signatures. They play a fundamental role in securing data and communications within the system.
3. **Suspended**: Keys in the suspended state are temporarily inactive, typically due to security concerns or operational issues. While suspended, they are not utilized for cryptographic operations but can be reinstated if deemed appropriate.
4. **Deactivated**: Keys in the deactivated state are intentionally rendered inactive, often as part of routine maintenance or security protocols. Once deactivated, they are no longer used for cryptographic operations but may be retained for archival or auditing purposes.
5. **Compromised**: Keys in the compromised state are deemed untrustworthy due to known or suspected security breaches. Immediate action is required to revoke and replace compromised keys to prevent unauthorized access and safeguard the integrity of cryptographic operations.
6. **Destroyed**: Keys in the destroyed state are permanently eradicated from the system, typically to prevent any potential misuse or unauthorized access. Destruction ensures that the keys are irrecoverable and cannot be compromised, thus enhancing overall security.

These key states play a critical role in the lifecycle management of cryptographic keys, ensuring that keys are properly managed, secured, and monitored throughout their operational lifespan, thereby maintaining the integrity and confidentiality of sensitive data and systems.



CycloneDX fully supports key states as detailed in the example below:

```
"cryptoProperties": {
  "assetType": "related-crypto-material",
  "relatedCryptoMaterialProperties": {
    "type": "private-key",
    "state": "compromised",
    ...
  }
}
```

Intersection of Key Management States and Lifecycle Phases

As outlined in the CycloneDX Authoritative Guide to SBOM, CycloneDX supports lifecycles that encompass those in the Software Development Life Cycle (SDLC) and those commonly used in Software Asset Management (SAM). CycloneDX lifecycle phases can be used together with the key management states defined by NIST to help identify and prevent risks associated with compromised keys and operational anomalies.

CycloneDX defines the following phases:

Phase	Description
Design	BOM produced early in the development lifecycle containing an inventory of components and services that are proposed or planned to be used. The inventory may need to be procured, retrieved, or resourced prior to use.
Pre-build	BOM consisting of information obtained prior to a build process and may contain source files, development artifacts, and manifests. The inventory may need to be resolved and retrieved prior to use.
Build	BOM consisting of information obtained during a build process where component inventory is available for use. The precise versions of resolved components are usually available at this time as well as the provenance of where the components were retrieved from.
Post-build	BOM consisting of information obtained after a build process has completed and the resulting components(s) are available for further analysis. Built components may exist as the result of a CI/CD process, may have been installed or deployed to a system or device, and may need to be retrieved or extracted from the system or device.
Operations	BOM produced that represents inventory that is running and operational. This may include staging or production environments and will generally encompass multiple SBOMs describing the applications and operating system, along with HBOMs describing the hardware that makes up the system. Operations Bill of Materials (OBOM) can provide a full-stack inventory of runtime environments, configurations, and additional dependencies.
Discovery	BOM consisting of information observed through network discovery providing point-in-time enumeration of embedded, on-premise, and cloud-native services such as server applications, connected devices, microservices, and serverless functions.
Decommission	BOM containing inventory that will be or has been retired from operations.

Key management states, as defined in NIST SP 800-57, offer a structured framework for managing cryptographic keys throughout their lifecycle. These states—such as pre-activation, active, suspended, deactivated, compromised, and destroyed—align closely with the lifecycle phases defined in CycloneDX. By integrating key management states with CycloneDX lifecycle phases, organizations gain a comprehensive understanding of the security posture of their cryptographic assets and can identify potential risks more effectively.

Manage Keys From Inception

In the CycloneDX lifecycle, the "Design" and "Pre-build" phases correspond to the pre-activation state in key management. During development, cryptographic keys are generated but not yet put into active use. By linking pre-activation with these phases, organizations can ensure that cryptographic keys are securely managed from their inception, minimizing the risk of unauthorized access or compromise before deployment.

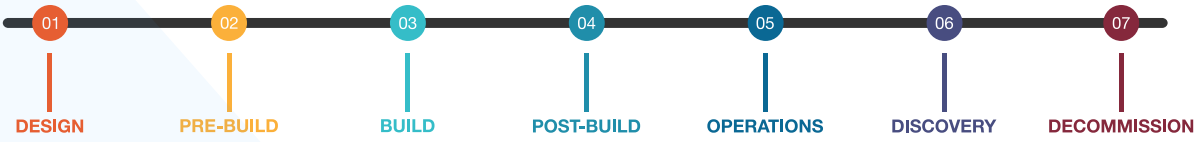
Anomaly Detection

As software progresses to the "Build" and "Post-build" phases in CycloneDX, cryptographic keys transition into the active state, where they are utilized for encryption, decryption, or digital signatures. Integration with key management states allows organizations to monitor the active keys' security status and detect anomalies that may indicate compromise or unauthorized usage, thereby mitigating potential risks during the software development lifecycle.

During the "Operations" phase in CycloneDX, cryptographic keys may undergo transitions such as suspension or deactivation if security concerns arise. For example, if a vulnerability is discovered in a cryptographic algorithm, keys associated with that algorithm may be suspended or deactivated to prevent exploitation. By correlating these transitions with the CycloneDX lifecycle, organizations can promptly respond to security incidents and mitigate associated risks, ensuring the integrity and confidentiality of cryptographic operations across deployment environments.

Furthermore, the compromised and destroyed states in key management align with the "Decommission" phase in CycloneDX. When cryptographic keys reach the end of their operational lifespan, they are revoked, destroyed, or securely decommissioned to prevent unauthorized access. CycloneDX facilitates the documentation of these actions, providing a comprehensive audit trail of key management activities and ensuring regulatory compliance.

Any deviances from expected lifecycle mappings to key management states should be investigated.



Prevention of Compromised Keys During Build or Deployment

The compromised key management state is crucial in preventing software supply chain attacks when integrated into the "Build" lifecycle phase in CycloneDX. During the build phase, cryptographic keys are utilized to sign software artifacts, ensuring their authenticity and integrity. By leveraging the "compromised" state within CycloneDX, organizations can detect compromised keys promptly and prevent unauthorized usage during the build process. For instance, if a key used for code signing is compromised, CycloneDX metadata can flag the key as compromised, triggering automated processes to revoke its access privileges and prevent its use in signing software artifacts. This proactive approach mitigates the risk of adversaries leveraging compromised keys to inject malicious code into the software supply chain, enhancing the security and trustworthiness of software builds distributed to consumers.

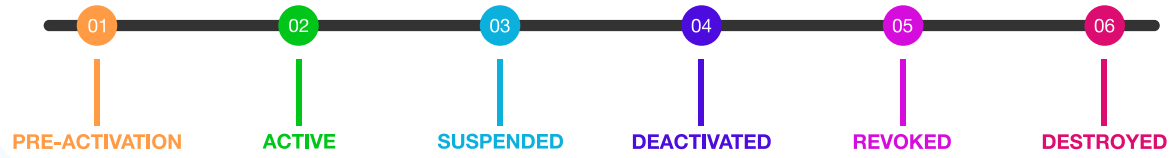
By leveraging key management states within the CycloneDX lifecycle phases, organizations can enhance their risk management practices, proactively identify security vulnerabilities, and maintain the integrity of cryptographic operations throughout the software development lifecycle. This integrated approach enables organizations to safeguard sensitive data and mitigate potential risks associated with cryptographic assets.

Certificate Management

Certificate lifecycle management documents the state of certificates associated with cryptographic assets. The BOM schema exposes a certificateState property which is an array of state objects. Each state object can be either a pre-defined state (recommended) or a custom state.

Pre-defined states:

- **Pre-activation**: The certificate has been issued by the issuing certificate authority (CA) but has not been authorized for use.
- **Active**: The certificate may be used to cryptographically protect information, cryptographically process previously protected information, or both.
- **Suspended**: The use of a certificate may be suspended for several possible reasons.
- **Deactivated**: Certificates in the deactivated state shall not be used to apply cryptographic protection but, in some cases, may be used to process cryptographically protected information.
- **Revoked**: A revoked certificate is a digital certificate that has been invalidated by the issuing certificate authority (CA) before its scheduled expiration date.
- **Destroyed**: The certificate has been destroyed.



Each pre-defined state may optionally include a reason string to explain rationale (for example: "key compromise", "decommissioned", "policy update").

Custom states (use the name attribute) are supported when an organization has domain-specific lifecycle stages that do not map to the pre-defined set. Custom states should include a description and may include a reason.

The following example defines both pre-defined as well as custom certificate states.

```
{
  "certificateProperties": {
    "certificateState": [
      {
        "state": "active"
      },
      {
        "state": "revoked",
        "reason": "private key compromise detected 2024-08-01"
      },
      {
        "name": "archived",
        "description": "Moved to long-term archive after rotation",
        "reason": "rotation completed"
      }
    ]
  }
}
```

Recommendations

- Prefer the pre-defined state values to maximize interoperability.
- Use reason to capture context for status changes (who, when, why).
- Use custom states sparingly, document custom meanings in your BOM or supporting policy.
- Record lifecycle timestamps in the certificate's metadata using the corresponding properties:
    - pre-activation → certificateProperties.creationDate
    - active → certificateProperties.activationDate
    - deactivated → certificateProperties.deactivationDate
    - revoked → certificateProperties.revocationDate
    - destroyed → certificateProperties.destructionDate

Practical Examples

Examples of assets typically cataloged within a CBOM include algorithms, keys, protocols, and certificates, each with associated metadata crucial for effective management and security. Algorithms refer to the mathematical functions utilized for encryption, hashing, and digital signatures, while keys encompass cryptographic keys used for encryption, decryption, and authentication. Protocols delineate the rules and procedures governing secure communication between entities, and certificates authenticate the identities of entities within a cryptographic system. Associated CBOM metadata includes details such as algorithm versions, key lengths, protocol configurations, and certificate attributes, providing comprehensive insights necessary for the management and security of cryptographic assets.

Algorithm

A cryptographic algorithm is added in the components array of the BOM. The examples below list the algorithm AES-128-GCM-128-12 and RSA-PKCS1-1.5-SHA-512-2048.

```
"components": [
  {
    "type": "cryptographic-asset",
    "name": "AES-128-GCM-128-12",
    "cryptoProperties": {
      "assetType": "algorithm",
      "algorithmProperties": {
        "algorithmFamily": "AES",
        "primitive": "ae",
        "parameterSetIdentifier": "128",
        "mode": "gcm",
        "executionEnvironment": "software-plain-ram",
        "implementationPlatform": "x86_64",
        "certificationLevel": [ "none" ],
        "cryptoFunctions": [ "keygen", "encrypt", "decrypt", "tag" ],
        "classicalSecurityLevel": 128,
        "nistQuantumSecurityLevel": 1
      },
      "oid": "2.16.840.1.101.3.4.1.6"
    }
  },
  {
    "name": "RSA-PKCS1-1.5-SHA-512-2048",
    "type": "cryptographic-asset",
    "cryptoProperties": {
      "assetType": "algorithm",
      "algorithmProperties": {
        "algorithmFamily": "RSASSA-PKCS1",
        "primitive": "signature",
        "parameterSetIdentifier": "512",
        "executionEnvironment": "software-plain-ram",
        "implementationPlatform": "x86_64",
        "certificationLevel": [ "none" ],
        "cryptoFunctions": [ "sign", "verify" ],
        "nistQuantumSecurityLevel": 0
      },
      "oid": "1.2.840.113549.1.1.13"
    }
  }
]
```

A complete example can be found at https://cyclonedx.org/shortcut/example/algorithm

An example with the QSC Signature algorithm ML-DSA-44 is listed below.

```
"components": [ {
    "name": "ML-DSA-44",
    "type": "cryptographic-asset",
    "cryptoProperties": {
      "assetType": "algorithm",
      "algorithmProperties": {
        "algorithmFamily": "ML-DSA",
        "primitive": "signature",
        "executionEnvironment": "software-plain-ram",
        "implementationPlatform": "x86_64",
        "certificationLevel": [ "none" ],
        "cryptoFunctions": ["keygen", "sign", "verify"],
        "nistQuantumSecurityLevel": 1
      },
      "oid": "2.16.840.1.101.3.4.3.17"
    }
  } ]
```

An example with a hybrid scheme combining ML-KEM-1024 and ECDH-secp521r1 is listed below. The relation between the combiner draftietftlshybriddesign13 and the algorithms is expressed as dependency.

```
"components": [
  {
    "name": "ECDH-secp521r1",
    "bom-ref": "ecdhsecp521r1",
    "type": "cryptographic-asset",
    "cryptoProperties": {
      "assetType": "algorithm",
      "algorithmProperties": {
        "algorithmFamily": "ECDH",
        "ellipticCurve": "secg/secp521r1",
        "primitive": "key-agree",
        "executionEnvironment": "software-plain-ram",
        "implementationPlatform": "x86_64",
        "certificationLevel": [ "none" ],
        "cryptoFunctions": [ "keygen", "keyderive" ],
        "nistQuantumSecurityLevel": 0
      },
      "oid": "1.3.132.0.35"
    }
  },
  {
    "name": "ML-KEM-1024",
    "bom-ref": "mlkem1024",
    "type": "cryptographic-asset",
    "cryptoProperties": {
      "assetType": "algorithm",
      "algorithmProperties": {
        "algorithmFamily": "ML-KEM",
        "primitive": "kem",
        "executionEnvironment": "software-plain-ram",
        "implementationPlatform": "x86_64",
        "certificationLevel": [ "none" ],
        "cryptoFunctions": [ "keygen", "keyderive" ],
        "nistQuantumSecurityLevel": 5
```

```
        },
        "oid": "2.16.840.1.101.3.4.1.48"
      }
    },
    {
      "name": "draft-ietf-tls-hybrid-design-13",
      "bom-ref": "draftietftlshybriddesign13",
      "type": "cryptographic-asset",
      "cryptoProperties": {
        "assetType": "algorithm",
        "algorithmProperties": {
          "primitive": "combiner",
          "executionEnvironment": "software-plain-ram",
          "implementationPlatform": "x86_64",
          "certificationLevel": [ "none" ],
          "cryptoFunctions": [ "keygen", "keyderive" ],
          "nistQuantumSecurityLevel": 0
        },
        "oid": "1.3.101.110"
      }
    }
  ],
  "dependencies": [
    {
      "ref": "draftietftlshybriddesign13",
      "dependsOn": ["mlkem1024", "ecdhsecp521r1"]
    }
  ]
```

Key

The following example demonstrates how an RSA-2048 public key can be included in a CBOM.

```
"components": [ {
    "name": "RSA-2048",
    "type": "cryptographic-asset",
    "bom-ref": "crypto/key/rsa-2048@1.2.840.113549.1.1.1",
    "cryptoProperties": {
      "assetType": "related-crypto-material",
      "relatedCryptoMaterialProperties": {
        "type": "public-key",
        "id": "2e9ef09e-dfac-4526-96b4-d02f31af1b22",
        "state": "active",
        "size": 2048,
        "algorithmRef": "crypto/algorithm/rsa-2048@1.2.840.113549.1.1.1",
        "creationDate": "2024-01-01T00:00:00.000Z",
        "activationDate": "2024-01-02T00:00:00.000Z",
        "updateDate": "2024-01-03T00:00:00.000Z",
        "expirationDate": "2026-01-01T00:00:00.000Z",
        "size": 2048,
        "format": "PKCS#8",
        "securedBy": {
          "mechanism": "Software",
          "algorithmRef": "crypto/algorithm/aes-128-gcm@2.16.840.1.101.3.4.1.6"
        },
        "creationDate": "2016-11-21T08:00:00Z",
        "activationDate": "2016-11-21T08:20:00Z"
      },
      "oid": "1.2.840.113549.1.1.1"
```

```
    }
  }, {
    "name": "RSA-PSS-SHA-256-32-2048",
    "type": "cryptographic-asset",
    "bom-ref": "crypto/algorithm/rsa-2048@1.2.840.113549.1.1.1",
    "cryptoProperties": { ... }
  }, {
    "name": "AES-128-GCM-16-12",
    "type": "cryptographic-asset",
    "bom-ref": "crypto/algorithm/aes-128-gcm@2.16.840.1.101.3.4.1.6",
    "cryptoProperties": { ... }
  }
]
```

A complete example can be found at https://cyclonedx.org/shortcut/example/key

Protocol

The following example lists an instance of the TLS v1.2 protocol with a number of cipher suites.

```
"components": [
  {
    "name": "google.com",
    "type": "cryptographic-asset",
    "bom-ref": "c9c7ac91-2115-45e8-ae13-7d0e1dec74be",
    "cryptoProperties": {
      "assetType": "certificate",
      "certificateProperties": {
        "serialNumber": "1234567890ABCDEF",
        "subjectName": "CN = www.google.com",
        "issuerName": "C = US, O = Google Trust Services LLC, CN = GTS CA 1C3",
        "notValidBefore": "2016-11-21T08:00:00Z",
        "notValidAfter": "2017-11-22T07:59:59Z",
        "certificateFormat": "X.509",
        "certificateFileExtension": "crt",
        "fingerprint": {
          "alg": "SHA-256",
          "content": "1e15e0fbd3ce95bde5945633ae96add551341b11e5bae7bba12e98ad84a5beb4"
        },
        "certificateState": [
          {
            "state": "active",
            "reason": "Certificate is currently valid and in use"
          }
        ],
        "creationDate": "2016-11-21T07:30:00Z",
        "activationDate": "2016-11-21T08:00:00Z",
        "relatedCryptographicAssets": [
          {
            "type": "algorithm",
            "ref": "6b00f384-6c39-420f-91eb-94de0f7be569RR"
          },
          {
            "type": "publicKey",
            "ref": "ceb37320-8239-40e8-ab77-8798dbd98773"
          }
        ]
      },
      "oid": "2.5.4.3"
```

```
    }
   },
   {
    "name": "SHA512withRSA",
    "type": "cryptographic-asset",
    "bom-ref": "6b00f384-6c39-420f-91eb-94de0f7be569",
    "cryptoProperties": {
     "assetType": "algorithm",
     "algorithmProperties": {
      "primitive": "signature",
      "executionEnvironment": "software-plain-ram",
      "implementationPlatform": "x86_64",
      "certificationLevel": [
       "none"
      ],
      "padding": "pkcs1v15",
      "cryptoFunctions": [
       "sign",
       "verify"
      ]
     },
     "oid": "1.2.840.113549.1.1.13"
    }
   },
   {
    "name": "RSA-2048",
    "type": "cryptographic-asset",
    "bom-ref": "ceb37320-8239-40e8-ab77-8798dbd98773",
    "cryptoProperties": {
     "assetType": "related-crypto-material",
     "relatedCryptoMaterialProperties": {
      "type": "public-key",
      "id": "2e9ef09e-dfac-4526-96b4-d02f31af1b22",
      "state": "active",
      "size": 2048,
      "format": "PEM",
      "value": "-----BEGIN PUBLIC KEY-----\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA...\n-
---END PUBLIC KEY-----",
      "creationDate": "2016-11-21T08:00:00Z",
      "activationDate": "2016-11-21T08:20:00Z",
      "updateDate": "2016-11-21T08:00:00Z",
      "expirationDate": "2017-11-22T07:59:59Z",
      "securedBy": {
       "mechanism": "None"
      },
      "fingerprint": {
       "alg": "SHA-256",
       "content": "a1b2c3d4e5f67890123456789012345678901abcdef123456789012abcdef123456"
      },
      "relatedCryptographicAssets": [
       {
        "type": "algorithm",
        "ref": "a154af0a-0dca-4ed5-b611-2405a3a6ae47"
       }
      ]
     },
     "oid": "1.2.840.113549.1.1.1"
    }
   },
   {
    "name": "RSA-2048",
```

```
    "type": "cryptographic-asset",
    "bom-ref": "a154af0a-0dca-4ed5-b611-2405a3a6ae47",
    "cryptoProperties": {
      "assetType": "algorithm",
      "algorithmProperties": {
        "primitive": "pke",
        "algorithmFamily": "RSAES-OAEP",
        "parameterSetIdentifier": "2048",
        "executionEnvironment": "software-plain-ram",
        "implementationPlatform": "x86_64",
        "certificationLevel": [
          "none"
        ],
        "padding": "oaep",
        "cryptoFunctions": [
          "encrypt",
          "decrypt"
        ]
      },
      "oid": "1.2.840.113549.1.1.1"
    }
  },
  {
    "name": "TLS 1.3 Protocol",
    "type": "cryptographic-asset",
    "bom-ref": "a3553dc1-f376-43d1-89dc-87bb71981c0c",
    "cryptoProperties": {
      "assetType": "protocol",
      "protocolProperties": {
        "type": "tls",
        "version": "1.3",
        "cipherSuites": [
          {
            "name": "TLS_AES_256_GCM_SHA384",
            "algorithms": [
              "1977d71b-8981-4292-b40d-842a019c2229",
              "422fa336-b401-42b7-89b8-8966aa30bca0"
            ],
            "identifiers": [
              "0x13,0x02"
            ]
          },
          {
            "name": "TLS_CHACHA20_POLY1305_SHA256",
            "algorithms": [
              "1af4fc08-5d0d-436e-8058-eeef921983d0",
              "6af3066b-ab66-4593-975f-d9ba2c623a89"
            ],
            "identifiers": [
              "0x13,0x03"
            ]
          }
        ]
      },
      "oid": "1.3.6.1.5.5.7.3.1"
    }
  }
]
```


A complete example can be found at https://cyclonedx.org/shortcut/example/protocol

Certificate

The following example details an advanced X.509 certificate in a CBOM.


```
"components": [
    {
      "name": "revoked-internal-ca.example.com",
      "type": "cryptographic-asset",
      "bom-ref": "840ADC47-55CD-44C6-A306-B37A9149B066",
      "cryptoProperties": {
        "assetType": "certificate",
        "certificateProperties": {
          "serialNumber": "ABCDEF1234567890FEDCBA",
          "subjectName": "CN = internal-ca.example.com, OU = IT Security, O = Example Corp, C = US",
          "issuerName": "CN = Example Root CA, O = Example Corp, C = US",
          "notValidBefore": "2023-01-01T00:00:00Z",
          "notValidAfter": "2025-12-31T23:59:59Z",
          "certificateFormat": "X.509",
          "certificateFileExtension": "pem",
          "fingerprint": {
            "alg": "SHA-256",
            "content": "9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08"
          },
          "certificateState": [
            {
              "state": "revoked",
              "reason": "Certificate was compromised due to private key exposure in security incident #2024-001"
            }
          ],
          "creationDate": "2022-12-15T10:00:00Z",
          "activationDate": "2023-01-01T00:00:00Z",
          "revocationDate": "2024-01-10T15:45:30Z",
          "certificateExtensions": [
            {
              "commonExtensionName": "basicConstraints",
              "commonExtensionValue": "CA:TRUE, pathlen:2"
            },
            {
              "commonExtensionName": "keyUsage",
              "commonExtensionValue": "Certificate Sign, CRL Sign, Digital Signature"
            },
            {
              "commonExtensionName": "extendedKeyUsage",
              "commonExtensionValue": "TLS Web Server Authentication, TLS Web Client Authentication"
            },
            {
              "commonExtensionName": "subjectAlternativeName",
              "commonExtensionValue": "DNS:internal-ca.example.com, DNS:ca.internal.example.com,
IP:192.168.1.100"
            },
            {
              "commonExtensionName": "authorityKeyIdentifier",
              "commonExtensionValue": "keyid:01:02:03:04:05:06:07:08:09:0A:0B:0C:0D:0E:0F:10:11:12:13:14"
            },
            {
              "commonExtensionName": "subjectKeyIdentifier",
              "commonExtensionValue": "A1:B2:C3:D4:E5:F6:07:08:09:0A:0B:0C:0D:0E:0F:10:11:12:13:14"
            },
            {
              "commonExtensionName": "crlDistributionPoints",
```

```
            "commonExtensionValue": "URI:http://crl.example.com/root-ca.crl"
          },
          {
            "commonExtensionName": "authorityInformationAccess",
            "commonExtensionValue": "OCSP - URI:http://ocsp.example.com, CA Issuers -
URI:http://certs.example.com/root-ca.crt"
          },
          {
            "commonExtensionName": "certificatePolicies",
            "commonExtensionValue": "Policy: 1.2.3.4.5.6.7.8.1, CPS: http://www.example.com/cps"
          },
          {
            "customExtensionName": "organizationalSecurityLevel",
            "customExtensionValue": "HIGH"
          },
          {
            "customExtensionName": "incidentTrackingId",
            "customExtensionValue": "SEC-2024-001"
          },
          {
            "customExtensionName": "complianceFramework",
            "customExtensionValue": "SOX, PCI-DSS Level 1, ISO 27001"
          }
        ],
        "relatedCryptographicAssets": [
          {
            "type": "algorithm",
            "ref": "2A0DA4D2-BBCA-4515-9BCD-C870A3EA4CE8"
          },
          {
            "type": "publicKey",
            "ref": "ACCAF8BC-5F73-4869-A3FE-1C64E8D96408"
          }
        ]
      },
      "oid": "2.5.4.3"
    }
  },
  {
    "name": "active-server.example.com",
    "type": "cryptographic-asset",
    "bom-ref": "4497B977-4D07-4245-9457-C2CF37FF399A",
    "cryptoProperties": {
      "assetType": "certificate",
      "certificateProperties": {
        "serialNumber": "1122334455667788AABBCCDD",
        "subjectName": "CN = server.example.com, OU = Web Services, O = Example Corp, C = US",
        "issuerName": "CN = Example Intermediate CA, O = Example Corp, C = US",
        "notValidBefore": "2024-01-01T00:00:00Z",
        "notValidAfter": "2025-01-01T23:59:59Z",
        "certificateFormat": "X.509",
        "certificateFileExtension": "crt",
        "fingerprint": {
          "alg": "SHA-256",
          "content": "9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08"
        },
        "certificateState": [
          {
            "name": "monitored",
            "description": "Certificate is under enhanced monitoring due to recent security incidents",
            "reason": "Proactive monitoring following organizational security policy updates"
```

```
          }
        ],
        "creationDate": "2023-12-20T09:00:00Z",
        "activationDate": "2024-01-01T00:00:00Z",
        "certificateExtensions": [
          {
            "commonExtensionName": "keyUsage",
            "commonExtensionValue": "Digital Signature, Key Encipherment"
          },
          {
            "commonExtensionName": "extendedKeyUsage",
            "commonExtensionValue": "TLS Web Server Authentication"
          },
          {
            "commonExtensionName": "subjectAlternativeName",
            "commonExtensionValue": "DNS:server.example.com, DNS:www.server.example.com,
DNS:api.server.example.com"
          },
          {
            "commonExtensionName": "signedCertificateTimestamp",
            "commonExtensionValue": "Log ID: ABCD1234..., Timestamp: 2024-01-01T00:00:00Z, Signature:
3045022100..."
          },
          {
            "customExtensionName": "deploymentEnvironment",
            "customExtensionValue": "PRODUCTION"
          },
          {
            "customExtensionName": "businessCriticality",
            "customExtensionValue": "CRITICAL"
          },
          {
            "customExtensionName": "autoRenewalEnabled",
            "customExtensionValue": "true"
          }
        ],
        "relatedCryptographicAssets": [
          {
            "type": "algorithm",
            "ref": "14478B86-9306-45B5-BA2A-1660B723244C"
          },
          {
            "type": "publicKey",
            "ref": "F1F3D902-0A1B-4C0C-9F6A-F36E041B0B7D"
          }
        ]
      },
      "oid": "2.5.4.3"
    }
  },
  {
    "name": "RSA-SHA256",
    "type": "cryptographic-asset",
    "bom-ref": "2A0DA4D2-BBCA-4515-9BCD-C870A3EA4CE8",
    "cryptoProperties": {
      "assetType": "algorithm",
      "algorithmProperties": {
        "primitive": "signature",
        "executionEnvironment": "software-plain-ram",
        "implementationPlatform": "x86_64",
        "padding": "pkcs1v15",
```

```
          "cryptoFunctions": ["sign", "verify"]
        },
        "oid": "1.2.840.113549.1.1.11"
      }
    },
    {
      "name": "ECDSA-P256",
      "type": "cryptographic-asset",
      "bom-ref": "14478B86-9306-45B5-BA2A-1660B723244C",
      "cryptoProperties": {
        "assetType": "algorithm",
        "algorithmProperties": {
          "primitive": "signature",
          "curve": "secp256r1",
          "executionEnvironment": "software-plain-ram",
          "implementationPlatform": "x86_64",
          "certificationLevel": ["fips140-3-l1"],
          "cryptoFunctions": ["sign", "verify"]
        },
        "oid": "1.2.840.10045.4.3.2"
      }
    },
    {
      "name": "RSA-4096-Revoked-CA-Key",
      "type": "cryptographic-asset",
      "bom-ref": "ACCAF8BC-5F73-4869-A3FE-1C64E8D96408",
      "cryptoProperties": {
        "assetType": "related-crypto-material",
        "relatedCryptoMaterialProperties": {
          "type": "public-key",
          "id": "revoked-ca-key-2024",
          "state": "compromised",
          "size": 4096,
          "format": "PEM",
          "value": "-----BEGIN PUBLIC KEY-----\nMIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEA...\n---
--END PUBLIC KEY-----",
          "creationDate": "2022-12-15T10:00:00Z",
          "activationDate": "2023-01-01T00:00:00Z",
          "updateDate": "2024-01-10T15:45:30Z",
          "securedBy": {
            "mechanism": "HSM",
            "algorithmRef": "aes-256-gcm-ref"
          },
          "fingerprint": {
            "alg": "SHA-256",
            "content": "9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08"
          },
          "relatedCryptographicAssets": [
            {
              "type": "algorithm",
              "ref": "2A0DA4D2-BBCA-4515-9BCD-C870A3EA4CE8"
            }
          ]
        },
        "oid": "1.2.840.113549.1.1.1"
      }
    },
    {
      "name": "ECDSA-P256-Server-Key",
      "type": "cryptographic-asset",
      "bom-ref": "F1F3D902-0A1B-4C0C-9F6A-F36E041B0B7D",
```

```
      "cryptoProperties": {
        "assetType": "related-crypto-material",
        "relatedCryptoMaterialProperties": {
          "type": "public-key",
          "id": "server-key-2024",
          "state": "active",
          "size": 256,
          "format": "PEM",
          "value": "-----BEGIN PUBLIC KEY-----\nMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAE...\n-----END
PUBLIC KEY-----",
          "creationDate": "2023-12-20T09:00:00Z",
          "activationDate": "2024-01-01T00:00:00Z",
          "expirationDate": "2025-01-01T23:59:59Z",
          "securedBy": {
            "mechanism": "HSM",
            "algorithmRef": "aes-256-gcm-ref"
          },
          "fingerprint": {
            "alg": "SHA-256",
            "content": "d4e5f67890123456789abcdef0123456789abcdef0123456789abcdef0123456"
          },
          "relatedCryptographicAssets": [
            {
              "type": "algorithm",
              "ref": "14478B86-9306-45B5-BA2A-1660B723244C"
            }
          ]
        },
        "oid": "1.2.840.10045.2.1"
      }
    }
  ]
```

Cryptographic configuration with CBOM and OBOM

The following example presents an Operations Bill of Materials (OBOM) in which cryptographic assets are configured through an OpenSSL 3 configuration file.

```
"components": [
  {
    "name": "ML-KEM-768",
    "type": "cryptographic-asset",
    "cryptoProperties": {
      "assetType": "algorithm",
      "algorithmProperties": {
        "algorithmFamily": "ML-DSA",
        "primitive": "kem",
        "executionEnvironment": "software-plain-ram",
        "cryptoFunctions": ["keygen", "encapsulate", "decapsulate"],
        "nistQuantumSecurityLevel": 3
      }
    }
  },
  {
    "name": "x25519",
    "type": "cryptographic-asset",
    "cryptoProperties": {
      "assetType": "algorithm",
      "algorithmProperties": {
```

```
        "algorithmFamily": "ECDH",
        "primitive": "key-agree",
        "executionEnvironment": "software-plain-ram",
        "cryptoFunctions": ["keygen", "keyderive"],
        "nistQuantumSecurityLevel": 0
      }
    }
  },
  {
    "name": "ECDH-P-256",
    "type": "cryptographic-asset",
    "cryptoProperties": {
      "assetType": "algorithm",
      "algorithmProperties": {
        "algorithmFamily": "ECDH",
        "primitive": "key-agree",
        "executionEnvironment": "software-plain-ram",
        "cryptoFunctions": ["keygen", "keyderive"],
        "nistQuantumSecurityLevel": 0
      }
    }
  },
  {
    "name": "openssl-config",
    "type": "data",
    "data": {
      "bom-ref": "config-001",
      "type": "configuration",
      "url": "/etc/openssl/ssl/openssl.cnf",
      "contents": {
        "attachment": {
          "contentType": "text/plain",
          "encoding": "utf8",
          "content": "DEFAULT_GROUPS     = x25519:SecP256r1MLKEM768"
        }
      }
    }
  },
  "dependencies": [
    {
      "ref": "ML-KEM-768",
      "dependsOn": ["openssl-config"]
    },
    {
      "ref": "x25519",
      "dependsOn": ["openssl-config"]
    },
    {
      "ref": "ECDH-P-256",
      "dependsOn": ["openssl-config"]
    }
  ]
]
```

Hardware BOM with cryptographic assets

The following example describes a Hardware Security Module (HSM) with dependencies to cryptographic assets.

```
{
    "bomFormat": "CycloneDX",
    "specVersion": "1.7",
    "version": 1,
    "metadata": {
        "timestamp": "2025-08-27T09:21:22.948466Z",
        "component": {
            "type": "device",
            "name": "Hardware Security Module",
            "version": "1.0",
            "bom-ref": "hsm-001"
        }
    },
    "components": [
        {
          "type": "device",
          "bom-ref": "crypto-processor-001",
          "name": "cryptoprocessor",
          "supplier": {
            "name": "CryptoSuperSecure Inc."
          },
          "version": "3.2",
          "description": "Cryptographic Co-processor, Post-Quantum Enabled",
          "properties": [
            {
              "name": "cdx:device:quantity",
              "value": "4"
            },
            {
              "name": "cdx:device:function",
              "value": "coprocessor"
            },
            {
              "name": "cdx:device:location",
              "value": "mainboard"
            },
            {
              "name": "cdx:device:deviceType",
              "value": "smd"
            },
            {
              "name": "cdx:device:lotNumber",
              "value": "1234-4567"
            }
          ]
        },
        {
          "type": "device",
          "bom-ref": "secure-memory-001",
          "name": "memory",
          "version": "1.0",
          "description": "Secure Memory Module",
          "properties": [
            {
              "name": "cdx:device:quantity",
              "value": "4"
            },
            {
              "name": "cdx:device:function",
              "value": "memory"
```

```
        },
        {
          "name": "cdx:device:location",
          "value": "mainboard"
        },
        {
          "name": "cdx:device:deviceType",
          "value": "smd"
        }
      ]
    },
    {
      "type": "device",
      "bom-ref": "tamper-detect-001",
      "name": "tamper",
      "version": "2.1",
      "description": "Tamper Detection Circuit",
      "properties": [
        {
          "name": "cdx:device:quantity",
          "value": "1"
        },
        {
          "name": "cdx:device:function",
          "value": "tamper-detection"
        },
        {
          "name": "cdx:device:location",
          "value": "mainboard"
        },
        {
          "name": "cdx:device:deviceType",
          "value": "smd"
        }
      ]
    },
    {
      "type": "device",
      "bom-ref": "hrng-001",
      "name": "trng",
      "version": "2.1",
      "description": "Hardware true random number generator",
      "properties": [
        {
          "name": "cdx:device:quantity",
          "value": "1"
        },
        {
          "name": "cdx:device:function",
          "value": "trng"
        },
        {
          "name": "cdx:device:location",
          "value": "mainboard"
        },
        {
          "name": "cdx:device:deviceType",
          "value": "smd"
        }
      ]
    },
```

```
      {
        "name": "ML-DSA-87",
        "type": "cryptographic-asset",
        "cryptoProperties": {
          "assetType": "algorithm",
          "algorithmProperties": {
            "algorithmFamily": "ML-DSA",
            "primitive": "signature",
            "executionEnvironment": "hardware",
            "implementationPlatform": "other",
            "certificationLevel": [ "fips140-3-l3" ],
            "cryptoFunctions": ["keygen", "sign", "verify"],
            "nistQuantumSecurityLevel": 5
          },
          "oid": "2.16.840.1.101.3.4.3.19"
        }
      },
      {
        "name": "SLH-DSA-SHA2-256s",
        "type": "cryptographic-asset",
        "cryptoProperties": {
          "assetType": "algorithm",
          "algorithmProperties": {
            "algorithmFamily": "SLH-DSA",
            "primitive": "signature",
            "executionEnvironment": "hardware",
            "implementationPlatform": "other",
            "certificationLevel": [ "fips140-3-l3" ],
            "cryptoFunctions": ["keygen", "sign", "verify"],
            "nistQuantumSecurityLevel": 5
          },
          "oid": "2.16.840.1.101.3.4.3.39"
        }
      },
      {
        "name": "AES-256-Wrap-PAD",
        "type": "cryptographic-asset",
        "cryptoProperties": {
          "assetType": "algorithm",
          "algorithmProperties": {
            "algorithmFamily": "AES",
            "primitive": "key-wrap",
            "executionEnvironment": "hardware",
            "implementationPlatform": "other",
            "certificationLevel": [ "fips140-3-l3" ],
            "cryptoFunctions": ["encrypt", "decrypt", "other"],
            "nistQuantumSecurityLevel": 5
          }
        }
      }

    ],
    "dependencies": [
    {
      "ref": "crypto-processor-001",
      "provides": ["ML-DSA-87", "SLH-DSA-SHA2-256s", "AES-256-Wrap-PAD"]
    }
  ]
}
```

A complete example can be found at https://cyclonedx.org/shortcut/example/cert

Dependencies

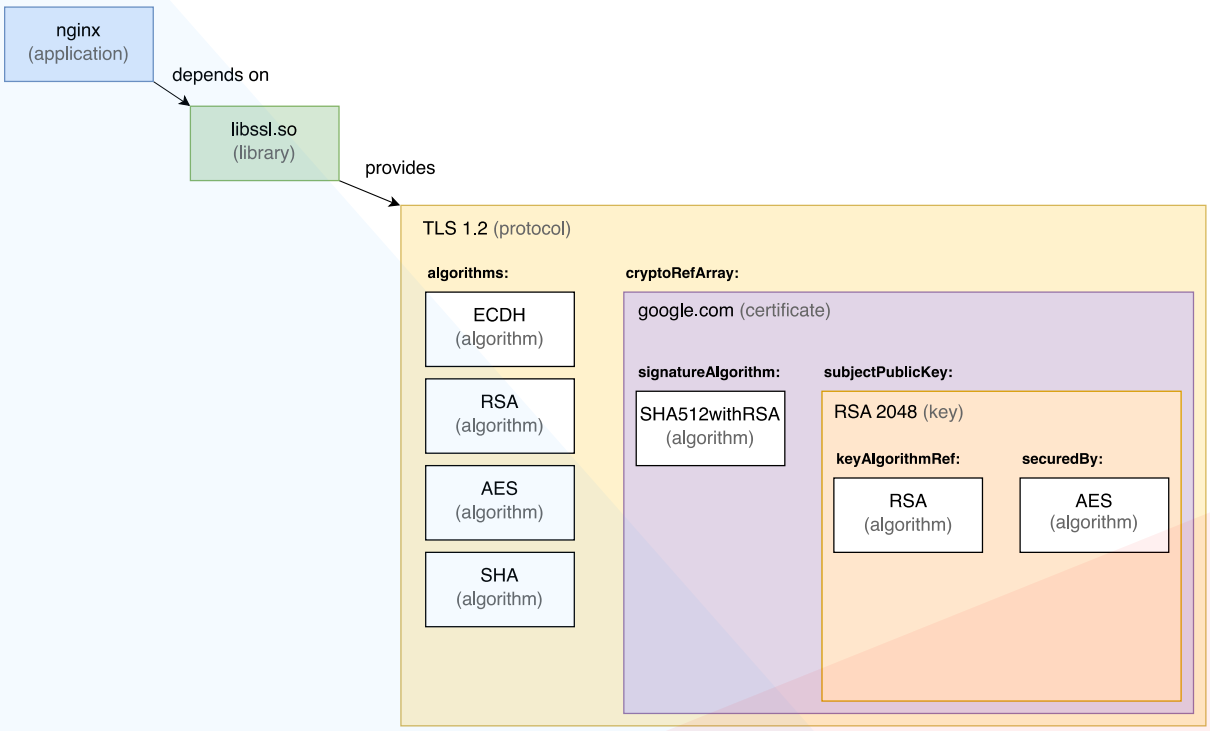
CycloneDX provides the ability to describe components and their dependency on other components. This relies on a component's bom-ref to associate the component with the dependency element in the graph. The only requirement for bom-ref is that it is unique within the BOM. Package-URL (PURL) is an ideal choice for bom-ref as it will be both unique and readable. If PURL is not an option or not all components represented in the BOM contain a PURL, then UUID is recommended. A dependency graph is capable of representing both direct and transitive relationships. In CycloneDX representation dependencies, a dependency graph SHOULD be codified to be one node deep, meaning no nested child graphs. All relations are on the same level.

Refer to the CycloneDX Authoritative Guide to SBOM for additional details.

In the context of cryptographic dependencies, CycloneDX provides some additional capabilities. As of CycloneDX v1.6, there are two types of dependencies: dependsOn and provides.

Dependency Type	Description
dependsOn	The bom-ref identifiers of the components or services that are dependencies of this dependency object.
provides	The bom-ref identifiers of the components or services that define a given specification or standard, which are provided or implemented by this dependency object. For example, a cryptographic library that implements a cryptographic algorithm. A component that implements another component does not imply that the implementation is in use.

The dependency type, dependsOn, is leveraged by classic SBOMs to define a complete graph of direct and transitive dependencies. However, for cryptographic and similar assets, "provides" allows for many additional use cases.



The example shows an application (nginx) that uses the libssl cryptographic library. This library implements the TLSv1.2 protocol. The relationship between the application, the library and the protocol can be expressed by using the dependencies properties of the SBOM standard.

Since a TLS protocol supports different cipher suites that include multiple algorithms, there should be a way to represent these relationships as part of the CBOM. Compared to adding the algorithms as "classic" dependencies to the protocol, we defined special property fields that allow referencing the deployment with additional meaning. The protocolProperties allows adding an array of algorithms to a cipher suite as part of the cipher suite array. By modeling and then referencing these algorithms, we can still have only one classical component at the SBOM level but a subtree of crypto dependencies within the crypto asset components.

The following example illustrates a simple application with a dependency on a cryptographic library, which, in turn, implements AES-128-GCM. The cryptographic library also depends on another library.

```
"dependencies": [
  {
    "ref": "acme-application",
    "dependsOn": ["crypto-library"]
  },
  {
    "ref": "crypto-library",
    "provides": ["aes128gcm"],
    "dependsOn": ["some-library"]
  }
]
```

Decoupling CBOM From SBOM With BOM-Link

With CycloneDX, it is possible to reference a BOM, or a component, service, or vulnerability inside a BOM from other systems or other BOMs. This deep-linking capability is referred to as BOM-Link and is a formally registered URN, governed by IANA, and compliant with RFC-8141.

Syntax:

```
urn:cdx:serialNumber/version#bom-ref
```

Examples:

```
urn:cdx:f08a6ccd-4dce-4759-bd84-c626675d60a7/1
urn:cdx:f08a6ccd-4dce-4759-bd84-c626675d60a7/1#componentA
```

Field	Description
serialNumber	The unique serial number of the BOM. The serial number MUST conform to RFC-4122.
version	The version of the BOM. The default version is 1.
bom-ref	The unique identifier of the component, service, or vulnerability within the BOM.

There are many use cases that BOM-Link supports. Two common scenarios are:

- Reference one BOM from another BOM
- Reference a specific component or service in one BOM from another BOM

Linking an SBOM to a CBOM

In CycloneDX, external references point to resources outside the object they're associated with and may be external to the BOM, or may refer to resources within the BOM. External references can be applied to individual components, services, or to the BOM itself.

The following example illustrates how an application in an SBOM can reference an external CBOM:

```
"components": [
  {
    "type": "application",
    "name": "Acme Application",
    "version": "1.0.0",
    "externalReferences": [
      {
        "type": "bom",
        "url": "https://example.com/bom/acme-application-1.0.0-cbom.cdx.json",
        "hashes": [ {
            "alg": "SHA-256",
            "content": "708f1f53b41f11f02d12a11b1a38d2905d47b099afc71a0f1124ef8582ec7313"
        } ]
      }
    ]
  }
]
```

Attestations

CycloneDX Attestations is a modern standard for security compliance. CycloneDX Attestations enable organizations with a machine-readable format for communication about security standards, claims and evidence about security requirements, and attestations to the veracity and completeness of those claims. You can think of Attestations as a way to manage "compliance as code."

Cryptography Standards

Organizations can declare the cryptography standards they follow, such as NIST or FIPS, in a CycloneDX Attestation. This helps ensure that all parties involved in the software development and deployment process are aware of the required cryptography standards.

By providing evidence such as test results, code reviews or other documents that prove that their software meets the cryptography requirements, they enable automatic verification of compliance with the requirements. For example, it can be verified that only approved cryptography algorithms are used and implemented correctly.

CycloneDX Attestations can also be used to manage compliance with cryptography requirements over time. As new weaknesses are discovered or standards change, organizations can update their applications, and therefore their attestations, to reflect the changes and ensure ongoing compliance.

Refer to the CycloneDX Authoritative Guide to Attestations for additional details.

Cryptography Definitions

CycloneDX provides a list of cryptographic algorithms definitions for use in a CBOM. It defines and organizes cryptographic algorithms by their primitive types—such as digital signatures, hash functions, symmetric encryption, and public-key encryption.

Each section corresponds to a specific cryptographic primitive and groups algorithms by family. Within each family, algorithm variants are described using patterns that capture key characteristics such as parameter sets, modes of operation, or key lengths. These definitions supports consistent identification, classification, and reporting of cryptographic algorithms in a CBOM.

Algorithm Family Definition Structure

The format for each algorithm entry is:

```
AlgorithmFamily: Pattern[-{optionalParameter}]
```

Where:

- AlgorithmFamily is the algorithm family
- Pattern shows how the algorithm should be referenced
- Elements in [] are optional parameters
- Elements in () indicate choices
- The | symbol indicates alternative choices
- Elements in {} indicate placeholders

Example

The following example defines the RSASSA-PKCS1 algorithm family, a signature scheme that may be used with different digest (hash) algorithms and key sizes. It is standardized in RFC 8017 and in IEEE 1363.

```
"algorithms": [
    {
      "family": "RSASSA-PKCS1",
      "standard": [
        {"name": "RFC8017", "url": "https://doi.org/10.17487/RFC8017"},
        {"name": "IEEE1363", "url": "https://doi.org/10.1109/IEEESTD.2000.92290"}
      ],
      "variant": [
        {
          "pattern": "RSA-PKCS1-1.5[-{digestAlgorithm}][-{keyLength}]",
          "primitive": "signature"
        }
      ]
    }
]
```

A cryptographic asset can reference the pattern in the component name. The following example defines RSASSA-PKCS1 using SHA-256 and a 2048 bit key length.

```
"components": [
    {
      "type": "cryptographic-asset",
      "bom-ref": "asset-1",
      "name": "RSA-PKCS1-1.5-SHA-256-2048",
      "cryptoProperties": {
        "assetType": "algorithm",
        "algorithmProperties": {
          "primitive": "signature",
          "algorithmFamily": "RSASSA-PKCS1",
        }
      }
    }
]
```

Elliptic Curve Definitions

The cryptography definitions also define a list of elliptic curves. The definitions include the standardization body (e.g., NIST), a description, Object Identifier (OID), form (e.g., Weierstrass). Since the same elliptic curve might be standardized by different bodies under different names, the curve also contains aliases. The source data is obtained from neuromancer.sk.

Elliptic Curve Example

The following example defines the P-256 NIST curve, along with its aliases secp256r1 and prime256v1.

```
"ellipticCurves": [
    {
      "name": "nist",
      "description": "RECOMMENDED ELLIPTIC CURVES FOR FEDERAL GOVERNMENT USE  July 1999",
      "curves": [
        {
          "name": "P-256",
          "description": null,
          "oid": "1.2.840.10045.3.1.7",
          "form": "Weierstrass",
          "aliases": [
            {
              "category": "secg",
              "name": "secp256r1"
            },
            {
              "category": "x962",
              "name": "prime256v1"
            }
          ]
        }
      ]
    }
]
```

All elliptic curves are defined in the enum cryptography-defs.schema.json#/definitions/algorithmFamiliesEnum.

A cryptographic asset can reference an elliptic curve with the ellipticCurve property.