

Cryptographic Discovery



Simula
UiB

Types of security

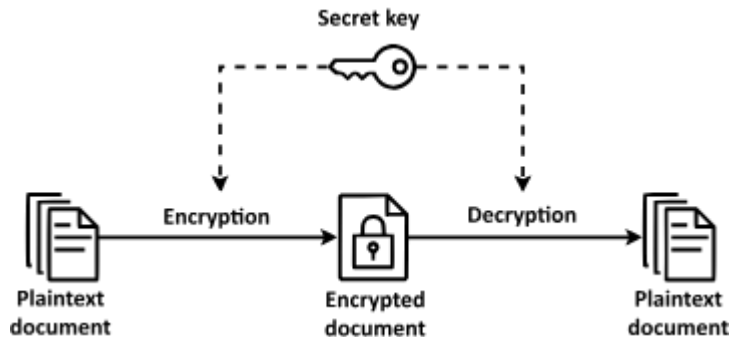
- Cryptography achieves:
 - Confidentiality = information is secret
 - Authenticity = assurance of where information originated
 - Integrity = assurance information has not changed
- Different cryptographic algorithms achieve different types of security



Cryptography for confidentiality

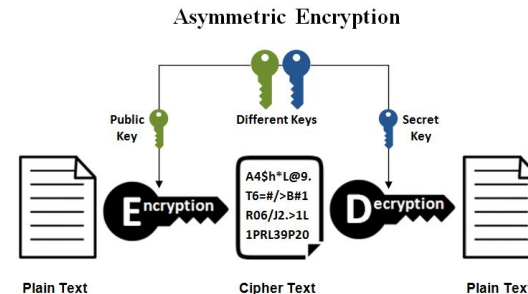
Symmetric encryption

Same **secret key** encrypts **plaintext** and decrypts **ciphertext**



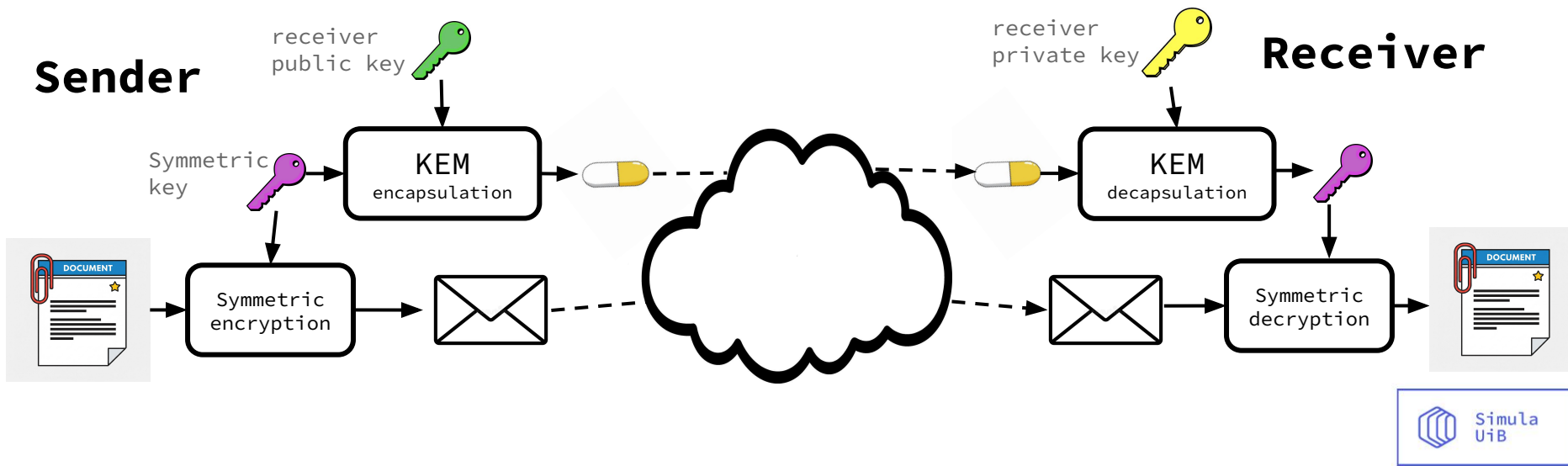
Asymmetric encryption

Public key encrypts **plaintext**,
private key decrypts **ciphertext**



Key Encapsulation Mechanism (KEM)

- KEM is a special case of asymmetric encryption:
 - Plaintext is always a symmetric key
- Used in combination with symmetric encryption

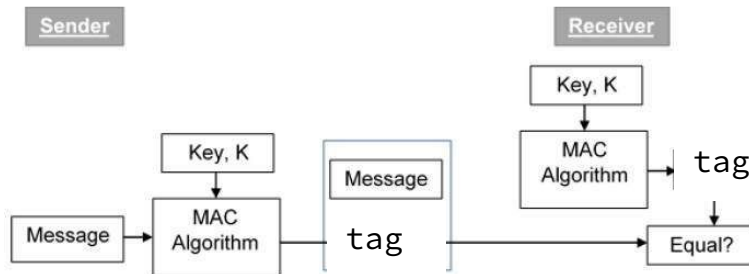


Cryptography for authenticity and integrity

Message authentication Code (MAC) – symmetric

Secret key produces **tag**
from **message**

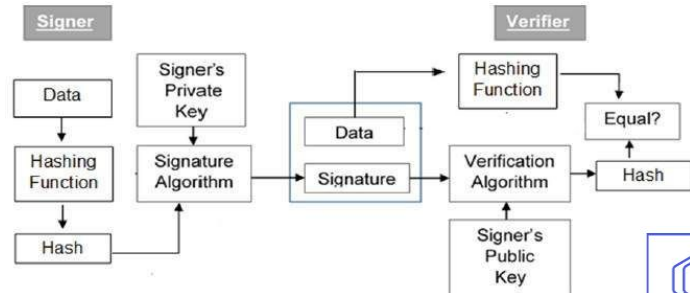
(message,tag) verified using
secret key



Digital signature – asymmetric

Private key produces **digital signature**
from **message**

(message,signature)
verified using **public key**



Cryptographic Inventory

- Important to get a complete overview of all cryptography you have in your IT systems
- Cryptography hides in many places, not easy to get a complete overview
- Tools exist to automate the search
 - Must use multiple tools for complete discovery
- IT environment not static – check, and insist on, PQC compliance for every new system or service to be acquired

Where to search for cryptography

- Source code, disk images (static code analysis)
- Compiled code, executables (binary analysis)
- Network traffic
- System or configuration settings
- Certificate stores
- Cloud services
- Hardware/firmware (IoT or OT)

Static code analysis

- Searches for cryptography in source code, libraries and disk images
- Looks for calls to functions in cryptographic libraries
- Open-source tools exist for scanning Python and Java code
- Limitation: only works for in-house and open source code, not for third-party applications

Dynamic analysis

- Searches for cryptography in running code and executables
- Finds cryptography used at run-time, also for third-party applications
- Limitations
 - must set up real-time monitoring of running applications
 - No open-source tools as of now

Network monitoring

- Scans network traffic, looking for use of cryptography
- Finds cryptography used in network protocols (TLS, SSH, IPsec,...)
- Not disrupting normal operations
- Open source tools for network scanning for cryptography exist
- Limitations
 - Only finds cryptography in network communication
 - If communication is encrypted, only discovers cryptography used at protocol level
 - Might be more crypto used inside encrypted traffic

System or configuration settings

- Scan configuration files (Windows registry, Linux configs,...) looking for policies on cryptography or installed hardware (TPMs or HSMs)
- Scan file systems looking for installation of known cryptographic libraries (OpenSSL, BouncyCastle,...)
- Limitations
 - Only finds what may exist, not what is used
 - Requires privileged access to read system files

Certificate and key stores

- Certificates and public keys readily show the cryptography used for authenticating machines and people
- Easy to parse and document
- Limitation:
 - Only covers certificates and keys found on system, not their usage

Cloud services

- Check your cloud services for cryptography deployed by cloud provider (key management systems, key vaults)
- Tools can scan container images or infrastructure-as-code for cryptographic libraries
- Scan network traffic to and from cloud provider
- Contact cloud provider and ask for PQC compliance
- Limitations:
 - Access to cloud provider solutions may be (is) restricted

Hardware and firmware

- Check what cryptography is running in IoT or OT devices
- Need specialized tools for firmware scanning or hardware introspection
- Limitations
 - Limited API, difficult to test automatically with common tools
 - Probably need to trust information from device vendor

Cryptography to keep or replace

Different types of cryptography: some is fine to keep, some must be replaced

OK to keep

Symmetric ciphers (AES, Chacha, ...)

Hash functions and MACS (SHA3, SHA256, HMAC,...)

Must be replaced

Asymmetric ciphers and key exchange (RSA, ECDH, ...)

Digital signatures (RSA, ECDSA, ...)

Use tools to find cryptography

- Several existing tools for discovering cryptography in an ecosystem
- Different tools have different focus
 - network, source code/libraries, binaries, end users vs. servers,...
- **Expect to use multiple tools for a complete overview**
- Many tools are vendor-specific, some are open-source

Vendor-specific tools

Examples only, none recommended in particular

[All products](#) / [IBM Quantum Safe](#) / [IBM Quantum Safe Explorer](#) / 2.x /

IBM Quantum Safe Explorer

AgileSec™ Analytics

Discover and Build a Complete Inventory of Keys, Certificates, Algorithms, Libraries, and Protocols in Seconds, Not Months!

CipherInsights™
From QUANTUMXCHANGE

**Zero to PQC: Start with
Cryptographic Inventory**

From discovery to remediation,
we take you the whole journey



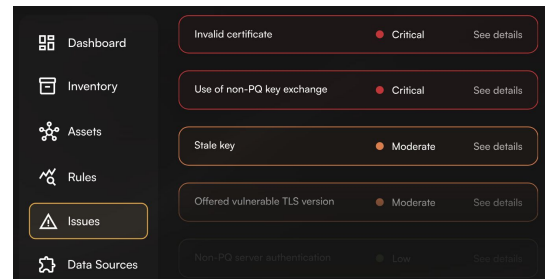
CRYPTONEXT COMPASS

**Inventory all your
cryptography**

QVision

Cryptographic Analytics

Uncover and Safeguard Cryptographic
Assets in Minutes, Not Months



**Palo Alto Networks Announces New Quantum
Security Innovations**

*New Cryptography Inventory Tool, Quantum-Optimized Firewalls and PAN-OS 12.1 Enable
Quantum Readiness*

Open-source tools

Source code/libraries

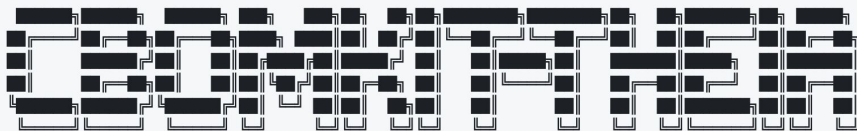
Addressing post-quantum cryptography with CodeQL

Learn how researchers and security experts at GitHub, Microsoft, and Santander came together to address the challenges presented by the post-quantum cryptography world.

Sonar Cryptography Plugin (CBOMkit-hyperion)

license [Apache-2.0](#) release [v1.4.7](#)

This repository contains a SonarQube Plugin that detects cryptographic assets in source code and generates [CBOM](#). It is part of the [CBOMKit](#) toolset.



CBOMkit-theia analyzes cryptographic assets in a container image or directory. It is part of cbomkit (<https://github.com/PQCA/cbomkit>) donated to PQCA by IBM Research.

Network traffic

ppqscan: Open-source post-quantum cryptography scanner

ppqscan is an open-source tool that lets users scan SSH and TLS servers to see which [Post-Quantum Cryptography \(PQC\)](#) algorithms they claim to support. It saves the results in JSON files. You can turn one or more of these files into an HTML report that opens in a web browser.

[README](#) [GPL-3.0 license](#) [Security](#)

[CodeQL](#) [passing](#) [License](#) [GPLv3](#)

CryptoMon

Network Cryptography Monitor - using eBPF, written in python.

September 2025: discovery tools are under development and improving

Output from scans and searches

- Different tools give output in different formats
- Output from search should be given in systemized format
- Cryptography Bill of Materials (CBOM) is a a good format for listing cryptographic assets found in a system
- Take the time to convert output from searches into CBOM format
 - Existing converter: Cryptobom Forge converts output from CodeQL into CBOM

```
$sudo python3 monitoring.py
```

```
Starting sniffing with scapy
```

```
Request to : 57148 -> 142.250.74.142 : https. Supported: [TLS_AES_256_GCM_SHA384, TLS_CHACHA20_POLY1305_SHA256, TLS_AES_128_GCM_SHA256, TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384, TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384, TLS_DHE_RSA_WITH_AES_256_GCM_SHA384, TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256, TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256, TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256, TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256, TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256, TLS_DHE_RSA_WITH_AES_128_GCM_SHA256, TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384, TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384, TLS_DHE_RSA_WITH_AES_256_CBC_SHA256, TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256, TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256, TLS_DHE_RSA_WITH_AES_128_CBC_SHA256, TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA, TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA, TLS_DHE_RSA_WITH_AES_256_CBC_SHA, TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA, TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA, TLS_DHE_RSA_WITH_AES_128_CBC_SHA, TLS_RSA_WITH_AES_256_GCM_SHA384, TLS_RSA_WITH_AES_128_GCM_SHA256, TLS_RSA_WITH_AES_256_CBC_SHA256, TLS_RSA_WITH_AES_128_CBC_SHA256, TLS_RSA_WITH_AES_256_CBC_SHA, TLS_RSA_WITH_AES_128_CBC_SHA, TLS_EMPTY_RENEGOTIATION_INFO_SCSV]
```

```
Request to : 55032 -> 140.82.121.3 : https. Supported: [TLS_AES_256_GCM_SHA384, TLS_CHACHA20_POLY1305_SHA256, TLS_AES_128_GCM_SHA256, TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384, TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384, TLS_DHE_RSA_WITH_AES_256_GCM_SHA384, TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256, TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256, TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256, TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256, TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256, TLS_DHE_RSA_WITH_AES_128_GCM_SHA256, TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384, TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384, TLS_DHE_RSA_WITH_AES_256_CBC_SHA256, TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256, TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256, TLS_DHE_RSA_WITH_AES_128_CBC_SHA256, TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA, TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA, TLS_DHE_RSA_WITH_AES_256_CBC_SHA, TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA, TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA, TLS_DHE_RSA_WITH_AES_128_CBC_SHA, TLS_RSA_WITH_AES_256_GCM_SHA384, TLS_RSA_WITH_AES_128_GCM_SHA256, TLS_RSA_WITH_AES_256_CBC_SHA256, TLS_RSA_WITH_AES_128_CBC_SHA256, TLS_RSA_WITH_AES_256_CBC_SHA, TLS_RSA_WITH_AES_128_CBC_SHA, TLS_EMPTY_RENEGOTIATION_INFO_SCSV]
```

```
^Cend
```

```
$|
```

Parsing output from tools

Strings indicating symmetric crypto, hash functions, macs (OK to keep)

SHA AES Poly1305

HMAC CBC GCM

ChaCha Not exhaustive list...

Strings indicating asymmetric cryptography, digital signatures, key exchange (needs replacement))

RSA DSA EC DH

Ed 25519 P521 DHE

Not exhaustive list...

Exceptions (old and deprecated algorithms)

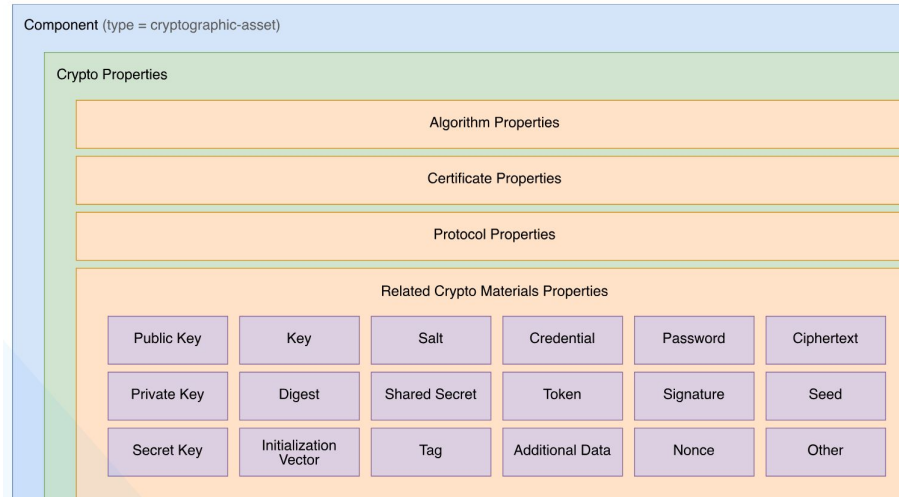
SHA1, MD5: replace with SHA3 or SHA256

DES, 3DES: replace with AES

Cryptographic Bill of Materials

- CBOM: file format describing cryptographic assets
- Part of CycloneDX component model
- Realized as JSON files
- Algorithms, keys, protocols, certificates, and other cryptographic assets can be listed in CBOMs

Structure and Cryptographic Asset Types



--	SHA256	HASH Hash Function	ElytronHmacTest.java:41	↗
--	SHA256	HASH Hash Function	ElytronSignatureAlgTest.java:40	↗
--	SHA256	HASH Hash Function	ElytronSignatureAlgTest.java:39	↗
?	AES128	OTHER	AesKeyWrapAlgorithmProvider.java:37	↗
?	AES128	OTHER	AesKeyWrapAlgorithmProvider.java:44	↗
?	AES128	OTHER	ElytronEcdhEsAlgorithmProvider.java:80	↗
?	AES128	OTHER	ElytronEcdhEsAlgorithmProvider.java:113	↗
?	AES128	OTHER	ElytronEcdhEsAlgorithmProvider.java:115	↗
!	RSASSA-PSS	SIGNATURE Signature	WildFlyElytronProvider.java:184	↗
!	RSASSA-PSS	SIGNATURE Signature	ElytronSignatureAlgTest.java:30	↗
!	RSASSA-PSS	SIGNATURE Signature	ElytronSignatureAlgTest.java:36	↗
!	RSASSA-PSS	SIGNATURE Signature	ElytronSignatureAlgTest.java:38	↗
?	PUBLIC-KEY@F51F3DD8-6494-4603-82F7-44FC1C9547FE	Unspecified	BCFIPSECDSCryptoProvider.java:85	↗
?	KEY@3851A5CA-5945-4929-B79F-D13D83A63108	Unspecified	WildFlyElytronProvider.java:138	↗
?	PRIVATE-KEY@EF1B8C64-6FF7-4728-9846-B68DFAD6D25E	Unspecified	ElytronSignatureAlgTest.java:36	↗
?	KEY@30DA5AD3-5230-4964-AE9A-106EC7CC1454	Unspecified	ElytronOCSPProviderTest.java:62	↗



PQCC Inventory Workbook

- PQC Coalition has made workbook to helping to get overview of PQC inventory -> <https://pqcc.org/pqc-inventory-workbook/>



The workbook helps organizations:

1. Identify and define systems or assets to track.
2. Categorize assets by priority (e.g., high, medium, or low).
3. Populate the inventory tab with available information.
4. Update the inventory as new data or systems are identified.

Cryptographic discovery in example company



Simula
UiB

Cryptography discovery process

Example of cryptographic discovery process in a fictitious medium-sized business embarking on its PQC migration



- NordICT AS: IT consultancy, operating in the Nordics
- IT consultancy, operating in the Nordics, with around 1000 employees
- ~25% are remote workers.



NordICT profile



- Around 1000 employees (~ 250 remote)
- All employees receive a company laptop (organisationally managed)
- Proprietary company data stored in internal servers + cloud services
- Internal development team: create software and services for clients and NordICT internal processes.
- NordICT has a well-established cybersecurity posture
- Embarking on a multi-year, phased PQC migration, initiated by CISO

Discovery strategy



Separation of discovery task:

- 1) identify vulnerable cryptography in codebases developed for clients
- 2) identify cryptography on internal systems and devices managed by its IT department.
 - operating systems, communication servers/controllers for wired and wireless networks, 3rd-party and in-house application software, firewalls, key management systems, etc
- 3) identify vulnerable cryptography usage by cloud services used by the company (but not managed at NordICT)
 - method: capture network traffic to cloud providers

Discovery strategy



Approach:

- automate discovery as much as possible using discovery tools, leveraging existing enterprise systems if possible.
- goal: create a cryptography inventory that will help NordICT to develop its migration roadmap.
- PQC migration team tasked with performing market research on automated tools, both commercial and open-source.

Cryptography discovery tools



The team's research concludes that many tools may be required, creating a challenge for the IT team with limited resources.

- use of different tools will require the normalisation of the different reporting formats into a common format, to integrate into existing security event reporting tools.

NordICT adopts a risk-based approach, to prioritise systems to be migrated first after inventory is complete

- Note: prioritisation can only be done if key processes and assets for their business have already been identified and mapped to operational system and services

Cryptography discovery process at NordICT



The discovery process of vulnerable cryptographic algorithms will be broken into the three following use-cases:

1. vulnerable cryptography used in code, compiled binaries, or dependencies during a continuous integration/continuous delivery (CI/CD) development pipeline
2. vulnerable cryptography used in assets on end-user systems and servers, including applications and associated libraries
3. vulnerable cryptography used in network protocols, enabling traceability to specific systems using active scanning and historical traffic captures

1. discovery: protecting code development pipeline



The process can be related to NIST's DevSecOps project¹, which in turn leverages the Secure Software Development Framework (SSDF)²

- SSDF defines secure development practices organized into four practice groups

Practice Group	Description
Prepare the Organization (PO)	Organizations should ensure that their people, processes, and technology are prepared to perform secure software development at the organization level. Many organizations will find some PO practices to also be applicable to subsets of their software development, like individual development groups or projects.
Protect the Software (PS)	Organizations should protect all components of their software from tampering and unauthorized access.
Produce Well-Secured Software (PW)	Organizations should produce well-secured software with minimal security vulnerabilities in its releases.
Respond to Vulnerabilities (RV)	Organizations should identify residual vulnerabilities in their software releases and respond appropriately to address those vulnerabilities and prevent similar ones from occurring in the future.

1: <https://www.nccoe.nist.gov/sites/default/files/2022-11/dev-sec-ops-project-description-final.pdf>

2: <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-218.pdf>

Table source: <https://www.nccoe.nist.gov/sites/default/files/2023-12/pqc-migration-nist-sp-1800-38b-preliminary-draft.pdf>

1. discovery: protecting code development pipeline



NordICT migration team chooses to align migration tasks within the framework provided by the SSDF, mapping tasks to PQC migration characteristics.

- approach will lead to creation of CBOM

SSDF Task ID	Task Description	Migration to PQC Characteristic
PS.3.2	Collect, safeguard, maintain, and share provenance data for all components of each software release (e.g., in a software bill of materials [SBOM]).	Creation of a cryptography bill of materials (CBOM)
PW.6.1	Use compiler, interpreter, and build tools that offer features to improve executable security.	Inspection of source code, dependent libraries, and containerized software via continuous integration actions
PW.7.2	Perform the code review and/or code analysis based on the organization's secure coding standards, and record and triage all discovered issues and recommended remediations in the development team's workflow or issue tracking system.	Inspection of source code via continuous integration actions
RV.1.2	Review, analyze, and/or test the software's code to identify or confirm the presence of previously undetected vulnerabilities.	Inspection of existing source code in a repository

Table source: <https://www.nccoe.nist.gov/sites/default/files/2023-12/pqc-migration-nist-sp-1800-38b-preliminary-draft.pdf>

2. discovery: operational systems and applications



Typical cryptographic assets in this category can be divided into two classes of files:

1. **executable**: runnable software, e.g.
 - a. application binaries
 - b. Cryptographic libraries
 - c. Java archives
2. **non-executable**: filesystem objects referenced by executables, e.g.
 - a. Keystores (PKCS#12, Java Keystores, etc)
 - b. Other key formats (OpenPGP keys, OpenSSH keys, PKCS#1, etc)

3. discovery: transport protocols and network services



Identify cryptographic protocols used in network communications, both internally and externally

Most common protocols:

- TLS (originally to protect HTTP, but also used for securing SMTP, IMAPS, RDP, etc)
- SSH (for secure remote shell access)
- IPsec (basis of most VPN implementations)

Exercise

Download CBOM files from <https://tinyurl.com/bdmny3tf>

Analyze files on <https://www.zurich.ibm.com/cbom/>

Task: Mark all cryptography to be replaced

SHA ChaCha HMAC

AES CBC GCM

Poly1305

Ed 25519 RSA DSA

EC DH DHE P521